METAHEURíSTICAS HíBRIDAS PARALELAS PARA PROBLEMAS INDUSTRIALES DE CORTE, EMPAQUETADO Y OTROS RELACIONADOS

Presentada para cumplir con los
requerimientos del grado de
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN
en la
UNIVERSIDAD NACIONAL DE SAN LUIS
SAN LUIS, ARGENTINA
OCTUBRE 2009

Autor:
Carolina Salto

Asesores:
Dr. Enrique Alba
Dr. Guillermo Leguizamón

UNIVERSIDAD NACIONAL DE SAN LUIS
DEPARTMENTO DE INFORMÁTICA

Los abajo firmantes certifican que han leído y recomiendan a la Facultad de Ciencias Físico, Matemáticas y Naturales aceptar la tesis titulada "**Metaheurísticas híbridas paralelas para problemas industriales de corte, empaquetado y otros relacionados**" por D. **Carolina Salto** en cumplimiento parcial de los requerimientos para el grado de **Doctor en Ciencias de la Computación**.

Fecha: Octubre 2009

Asesor Científico: _____
Dr. Enrique Alba

Co-Asesor Científico: _____
Dr. Guillermo Leguizamón

A Alberto, Tomás y Martín.

Índice general

Índice de tablas

Índice de figuras

Agradecimientos

Muchas son las dificultades que surgen para la elaboración de un trabajo de tesis. Felizmente, existen personas que siempre están dispuestas a brindar su ayuda, por lo tanto quiero expresar mi sincero reconocimiento a todos los que, de distintas formas y posiciones, han contribuido para la elaboración de este trabajo.

Es un gran placer para mí poder agradecer a mi director de tesis, Dr. Enrique Alba, por la paciencia y atención que me ha brindado para el desarrollo de este trabajo y por su disponibilidad para responder inquietudes a pesar de las múltiples actividades que desarrolla. Un sincero agradecimiento al Dr. Guillermo Leguizamón por el tiempo que ha dado, su respaldo y estímulo. No quisiera dejar de agradecer al Dr. Juan Miguel Molina por sus sugerencias y soporte durante las primeras etapas de este proyecto.

No puedo olvidar a mis compañeros de trabajo Gabriela, Hugo, Natalia, Alina, Carlos y Javier, por crear un ambiente donde es posible el intercambio de ideas y por generar un espacio de trabajo distendido que hacen posibles trabajos de este tipo. Gracias por aguantarme y escucharme. A Paco por su disponibilidad para atender mis pedidos de disponibilidad de máquinas.

En forma muy especial a *toda mi familia* que me apoyó en esta etapa brindándome su fuerza y que siempre me acompaña en los momentos más importantes de mi vida. A Alberto, Tomás y Martín, mis gorditos, que gracias a su incansable apoyo moral y afecto me han brindado el estímulo necesario para la concreción de este proyecto, a pesar de mis ausencias y horas pasadas frente a la compu.

Finalmente debo un especial reconocimiento a la Facultad de Ingeniería de la UNLPam, al CONICET por concederme una beca de Postgrado de Tipo II para la finalización del doctorado y a la Universidad de Málaga por los recursos cedidos para concretar la parte experimental de este trabajo.

Carolina Salto
General Pico, La Pampa, Argentina
Octubre de 2009

Capítulo 1

Introducción

1.1. Planteamiento

Los problemas de corte y empaquetado (C&P) consisten, por lo general, en el corte de materias primas para obtener un conjunto de elementos minimizando el desperdicio de material generado o en el empaquetado de un conjunto de artículos en el menor número de contenedores. Esta clase de problemas cae dentro de la categoría de problemas de optimización combinatoria. Usualmente, se presentan en muchas aplicaciones industriales, tales como:

- vidrio, papel y corte de acero,
- carga de contenedores y camiones,
- diseño de circuitos integrados,
- optimización de portfolio
- y muchas otras.

La mayoría de los problemas de optimización combinatoria, y por consiguiente los problemas de corte y empaquetado, son, en general, difíciles de resolver en la práctica. Estos problemas están incluidos en la clase de problemas $\mathcal{NP}$-duros [92], ya que no se conocen

algoritmos exactos con complejidad polinómica que permitan resolverlos. Debido a su intratabilidad, se han diseñado una gran cantidad de métodos aproximados, los cuales encuentran buenas soluciones en tiempos computacionales razonables. En esta clase de problemas, la búsqueda de una solución requiere una exploración organizada a través del espacio de búsqueda: una búsqueda sin guía es extremadamente ineficiente.

Durante la década de los sesenta se han diseñado diversos métodos aproximados, conocidos como heurísticos, capaces de encontrar soluciones de buena calidad, y que en muchos casos son muy cercanas a la solución óptima o mejor conocida. Gran parte de estos métodos fueron concebidos inspirándose en la resolución de problemas de fácil representación, pero de muy difícil solución, como lo son el problema del vendedor viajero, el problema de la mochila, etc. Debido a la variada naturaleza de estos problemas, los métodos eran útiles apenas para el problema para el cual habían sido inspirados.

En los últimos 20 años han aparecido una nueva clase de algoritmos, basados en la combinación de métodos heurísticos básicos en un marco de alto nivel para explorar en forma eficiente y eficaz el espacio de búsqueda. Esos métodos son comúnmente denominados metaheurísticas [25, 33, 100]. El surgimiento de las técnicas metaheurísticas plantea un cambio importante en el desarrollo de técnicas alternativas frente a heurísticos *ad hoc*, ya que permiten extender de manera estructurada su aplicación a una amplia gama de problemas de optimización representativos del mundo real. Este hecho, junto con su notable eficacia, actuaron como los principales atractivos de la atención de la comunidad científica.

Como mecanismo para mejorar la eficiencia de las metaheurísticas, se han propuesto en los últimos años distintas versiones paralelas de las mismas [11]. Otros enfoques diferentes toman en cuenta el diseño de algoritmos de híbridos, la creación de operaciones especializadas para el problema en cuestión, etc. Sin embargo, utilizar algoritmos paralelos es una forma de aliviar los problemas vinculados a tiempos intensivos de ejecución e importantes requerimientos de memoria para resolver instancias complejas de interés actual. Varias estrategias de paralelización se pueden aplicar a las metaheurísticas. Entre ellas, existen modelos paralelos donde varios hilos de búsqueda simultáneamente exploran el espacio de soluciones. Si cada uno usa diferentes procedimientos de búsqueda, obtenemos una metaheurística

heterogénea paralela. La utilización de múltiples hilos de búsqueda usando diferentes estrategias y valores de los parámetros permite una mayor diversidad y una más profunda exploración del espacio de búsqueda, lo que podría llevar a soluciones más precisas.

Nuestro planteamiento se basa en aplicar técnicas metaheurísticas a problemas de corte y empaquetado. En particular, este trabajo de tesis está enfocado a una de las variantes más conocidas del problema de C&P como lo es el problema de *strip packing*, en el cual la característica más importante es que la plancha de material, donde se deben asignar las distintas piezas rectangulares, tiene una de las dimensiones libres (por lo general la altura). Para tal fin, hemos diseñado distintas metaheurísticas, ya sea desde su configuración como también en la información del problema que pueden incorporar, para sacar máximo partido a dichas técnicas y ofrecer así soluciones de gran calidad. También hemos planteado distintas versiones paralelas de las mismas y propuesto versiones paralelas heterogéneas, basadas en el comportamiento estudiado previamente, para desarrollar metaheurísticas más robustas.

1.2. Objetivos y fases

En esta tesis proponemos aplicar metaheurísticas secuenciales y paralelas a problemas de corte, analizar los resultados para comprender el comportamiento de estos algoritmos y proponer nuevos métodos para resolver los problemas de una manera más eficaz y eficiente. Los objetivos específicos que se esperan alcanzar con esta tesis son:

- Diseño, implementación y evaluación de modelos algorítmicos que incluyan conocimiento del problema en cuestión en el dominio de corte y empaquetado y posiblemente otros dominios afines. Como subobjetivo inicial será necesario un extenso estudio de la literatura existente en este campo y la definición de los problemas concretos de análisis.

- Definición y estudio de extensiones paralelas de los modelos desarrollados, de manera que no sólo se aproveche la potencia numérica inherente a las técnicas descentralizadas sino que también se puedan obtener ganancias en tiempo real al utilizar un conjunto de computadoras para resolver el mismo problema.

- Inclusión de heterogeneidad en modelos descentralizados para los problemas designados. Es de interés que las técnicas diseñadas puedan extenderse para contener componentes distintos que se complementen en el sentido diversificación/intensificación, que actualmente se considera la piedra angular de la optimización moderna.
- Desarrollo de nuevas técnicas de acuerdo a las técnicas modernas de Ingeniería del software (orientación a objetos, bibliotecas estructuradas en componentes, modularidad, corrección, etc.).

Para lograr estos objetivos, inicialmente revisamos la literatura para determinar las características de los problemas de C&P tipo 2SPP con los que trabajaremos, así como también las distintas propuestas metodológicas utilizadas para resolver esta clase de problemas por otros investigadores. Esto permite proponer y evaluar inicialmente modelos para orientar y centrar las propuestas presentadas en este trabajo de tesis. Las técnicas seleccionadas son metaheurísticas poblacionales, a saber algoritmos evolutivos, en particular algoritmos genéticos, y optimización basada en colonias de hormigas. Finalmente, dado el elevado costo computacional asociado a estos problemas también propondremos una extensión utilizando técnicas paralelas. Aplicamos estas metaheurísticas, y sus extensiones, al problema de empaquetado y analizamos los resultados obtenidos. Estas fases están resumidas en la Figura 1.1.

1.3. Aportes

En esta sección enunciamos en forma resumida y esquemática los aportes realizados con la presente tesis, a saber:

- Desarrollo de un nuevo modelo matemático para el problema de *strip packing* considerando las restricciones impuestas en este trabajo de tesis.
- Desarrollo de un operador de recombinación que incorpora conocimiento del problema (BILX).

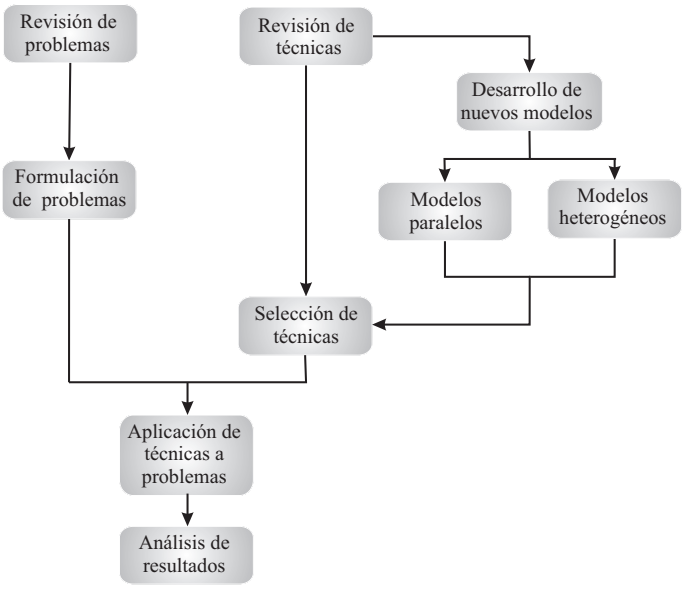


Figura 1.1: Fases seguidas durante la elaboración de esta tesis

- Desarrollo de reglas de construcción para generar una población inicial más especializada para el problema en cuestión.
- Propuesta de operadores de relocalización basado en heurísticas *ad hoc* bien conocidas para el problema (MFF_Adj y MBF_Adj).
- Adaptación del algoritmo optimización basada en colonias de hormigas para resolver el problema *strip packing*, en lo que respecta a la definición del rastro de feromonas, la conformación del vecindario alcanzable por una hormiga y la hibridación con búsqueda local.
- Diseño y evaluación de algoritmos heterogéneos, donde las islas están conformadas por algoritmos que implementan distintos mecanismos de búsqueda.
- Implementación de un generador de instancias de complejidad variable para el problema de C&P, debido a que las instancias encontradas en la bibliografía [29, 31, 160] resultan de desigual complejidad para la prueba de los algoritmos.

De cada uno de los puntos anteriormente enumerados, hemos realizado una importante labor de divulgación, tanto a nivel nacional como internacional, en conferencias y revistas de impacto (véase Apéndice A).

1.4. Organización de la tesis

Este trabajo de tesis se organiza en tres partes principales. En la primera, se presentan los conceptos de metaheurísticas y se caracterizan los problemas de corte y empaquetado. En la segunda parte se presentan las propuestas algorítmicas para resolver el problema como así también los resultados obtenidos con estas propuestas al resolver el problema de *strip packing*. En la tercera y última parte esbozamos las conclusiones a las que hemos arribado al realizar este trabajo, junto con las líneas de investigación que se abren a partir de nuestro estudio. A continuación mostramos en detalle el contenido de los distintos capítulos que conforman esta memoria.

- **Parte I: Fundamentos de las metaheurísticas y los problemas de corte y empaquetado**

  En el **Capítulo 2** ofrecemos una introducción sobre el campo de las metaheurísticas. Posteriormente se dan detalles sobre los mecanismos avanzados para los algoritmos metaheurísticos que se usan en esta tesis: paralelismo y heterogeneidad. El capítulo concluye con una descripción genérica de las metaheurísticas utilizadas en este trabajo de tesis.

  En el **Capítulo 3** presentamos los problemas de corte y empaquetado, resaltando las características de los mismos, para luego describir en particular el problema de *strip packing* en dos dimensiones que se ha resuelto con técnicas metaheurísticas en la presente tesis. En la parte final del capítulo se dan detalles de las instancias del problema utilizadas en los capítulos posteriores.

  En el **Capítulo 4** realizamos una revisión de la literatura existente resumiendo las distintas soluciones existentes para el problema de problemas de *strip packing* usando heurísticas y metaheurísticas.

- **Parte II: Resolución del problema de corte y empaquetado**

  En los **Capítulos 5 y 6** resolvemos el problema de *strip packing* por medio de algoritmos genéticos y algoritmos de optimización basados en colonia de hormigas, respectivamente. Se detallan en cada caso los algoritmos utilizados junto con las modificaciones propuestas para introducir mejoras. Además, se presentan los resultados obtenidos con estos algoritmos y se discuten los mismos.

  En el **Capítulo 7** abordamos la resolución de este problema por medio de la inclusión de técnicas paralelas. Por consiguiente describimos las mismas y seguidamente analizamos los resultados ofrecidos. Finalizamos mostrando las principales conclusiones que se pueden extraer de todo este proceso.

  En el **Capítulo 8** describimos las distintas propuestas heterogéneas planteadas y se comparan sus resultados desde un punto de vista de eficacia y de paralelismo.

- **Parte III: Conclusiones**

  En el **Capítulo 9** finalizamos este trabajo de tesis resumiendo el trabajo presentado y dando una mirada hacia las posibles tareas futuras y desarrollos.

Parte I

Fundamentos de metaheurísticas y problemas de corte y empaquetado

Capítulo 2

Metaheurísticas

En este capítulo presentamos las bases necesarias de los algoritmos de optimización, utilizados posteriormente en este trabajo de tesis, para resolver el problema propuesto. Inicialmente en la Sección 2.1 se define un problema de optimización en forma general, se especifican las distintas formas de plantear su resolución, para pasar luego en la Sección 2.2 al concepto de metaheurística. Se establece su clasificación en basadas en trayectoria y basadas en población, especificando por cada una las distintas características. En la actualidad, uno de los mecanismos que ha probado ser exitoso en el diseño de metaheurísticas eficientes para la resolución de problemas de optimización es el paralelismo. Utilizar algoritmos paralelos es una forma de aliviar los problemas vinculados a tiempos intensivos de ejecución e importantes requerimientos de memoria para resolver instancias complejas de interés actual. Por lo tanto, la Sección 2.4 se corresponde con las estrategias más importantes para paralelizar las metaheurísticas. Una de ellas consiste de múltiples hilos de búsqueda que exploran en forma concurrente el espacio de búsqueda. Cada uno de los hilos de búsqueda puede utilizar distinto procedimiento de búsqueda, ya sea a nivel algorítmico o a nivel de configuración de parámetros, obteniéndose *metaheurísticas paralelas heterogéneas*, las cuales se describen en la Sección 2.5. Esto permite una mayor diversidad y mejor exploración del espacio de búsqueda, lo cual producirá soluciones más exactas. Por lo tanto, se brinda una clasificación de las metaheurísticas heterogéneas que trata de abarcar los algoritmos heterogéneos más importantes. Terminamos este capítulo con la Sección 2.6 presentando las características más importantes de los algoritmos evolutivos y de optimización basados en

colonia de hormigas, por ser las dos metaheurísticas utilizadas en este trabajo de tesis para resolver el problema de empaquetado propuesto.

2.1. Problemas de optimización

La optimización combinatoria estudia problemas caracterizados por un número finito de soluciones factibles. Un área de aplicación importante y difunda se refiere al uso eficiente de recursos escasos con el fin de incrementar la productividad. Como ejemplos de esta clase de problemas se pueden mencionar: problemas de corte y de empaquetado, el problema del viajante de comercio, problemas de planificación de tareas, de asignación, entre otros problema de índole práctica.

Según Papadimitriou y Steiglitz [177], un problema de optimización se define de la siguiente manera:

Definición 2.1.1 (Problema de optimización). Un problema de optimización se formaliza como un par (S, f) donde $S \neq \emptyset$ representa el espacio de soluciones (o de búsqueda) del problema, mientras que f es una función denominada *función objetivo* o *función de fitness*, que se define como: $f : S \rightarrow \mathbf{R}$. Así, resolver un problema de optimización consiste en encontrar una solución, $i^* \in S$, que satisfaga la siguiente desigualdad:

$$f(i^*) \leq f(i), \forall i \in S. \tag{2.1.1}$$

Asumir el caso de maximización o minimización no restringe la generalidad de los resultados, puesto que se puede establecer una igualdad entre tipos de problemas de maximización y minimización del siguiente forma [25, 100]:

$$max\{f(i)|i \in S\} \equiv min\{-f(i)|i \in S\}. \tag{2.1.2}$$

En función del dominio al que pertenezca S, podemos definir problemas de optimización binaria ($S \subseteq \mathbf{B}^*$), entera ($S \subseteq \mathbf{N}^*$), continua ($S \subseteq \mathbf{R}^*$) o mixtas ($S \subseteq (\mathbf{B} \cup \mathbf{N} \cup \mathbf{R})^*$).

Aunque en principio la solución óptima para tales problemas se puede hallar por una simple enumeración, en la práctica esto es con frecuencia imposible, especialmente para problemas con tamaños reales, donde el número de soluciones factibles puede ser muy alto.

Para problemas que presentan un alto grado combinatorio es difícil generar un modelo basado en programación matemática que represente exactamente una situación real. Durante la década de los setenta se han diseñado diversos métodos, conocidos como heurísticos *ad hoc*, capaces de encontrar soluciones de buena calidad y, que en muchos casos, son muy cercanas a la solución óptima o mejor conocida. Gran parte de estos métodos fueron concebidos inspirándose en la resolución de problemas de fácil representación, pero de muy difícil solución, como lo son el problema del vendedor viajero (TSP), el problema de la mochila, etc. Debido a la variada naturaleza de estos problemas, los métodos eran útiles apenas para el problema en el cual habían sido inspirados.

Con el desarrollo de la teoría de complejidad a principios de los setenta, se vuelve claro que, como la mayoría de los problemas de optimización eran $\mathcal{NP}$-duros, había poca esperanza de hallar procedimientos eficientes de solución exacta. Esto enfatizó el rol de las heurísticas para resolver problemas de optimización que fueran encontrados en aplicaciones de la vida real, fueran o no $\mathcal{NP}$-duros. Mientras se desarrollaban y se experimentaba con muchas opciones, las más populares fueron las técnicas de búsqueda local. Estas técnicas son un procedimiento de búsqueda iterativo que comenzando con una solución inicial factible, progresivamente la va mejorando al aplicar un serie de modificaciones locales o *movimientos*. En cada iteración, la búsqueda se mueve a una solución factible mejorada que difiere sólo un poco de la solución actual. La búsqueda finaliza cuando encuentra un óptimo local con respecto de las transformaciones que considera. Por lo general, este óptimo local es una solución bastante mediocre, transformándose en una limitación del método. En búsqueda local, la calidad de las soluciones obtenidas y los tiempos de computación empleados son altamente dependientes de la "bondad" del conjunto de transformaciones consideradas en cada iteración de la heurística *ad hoc*.

Finalmente, en los años ochenta el surgimiento de las técnicas *metaheurísticas* plantea un cambio importante en el desarrollo de técnicas alternativas frente a heurísticas *ad hoc*. La idea básica de las metaheurísticas era combinar diferentes métodos heurísticos a un nivel más alto, para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. El término metaheurística fue introducido por primera vez por Glover [95]. Antes de que

el término fuese aceptado completamente por la comunidad científica, estas técnicas eran denominadas heurísticas modernas [189]. Esta clase de algoritmos incluye técnicas como colonias de hormigas, algoritmos evolutivos, búsqueda local iterada, enfriamiento simulado y búsqueda tabú. Se pueden encontrar revisiones de metaheurísticas en [33, 97].

De las diferentes descripciones de metaheurísticas que se encuentran en la literatura se pueden destacar ciertas propiedades fundamentales que caracterizan a este tipo de métodos:

- Las metaheurísticas son estrategias o plantillas generales que guían el proceso de búsqueda.
- El objetivo es una exploración eficiente del espacio de búsqueda para encontrar soluciones (casi) óptimas.
- Las metaheurísticas son algoritmos no exactos y generalmente son no deterministas.
- Las metaheurísticas pueden incorporar mecanismos para evitar regiones no prometedoras del espacio de búsqueda.
- El esquema básico de cualquier metaheurística tiene una estructura predefinida.
- Las metaheurísticas pueden usar conocimiento del problema que se trata de resolver en forma de heurísticos específicos que son controlados por la estrategia de más alto nivel.

Resumiendo estos puntos, se puede acordar que una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar el espacio de búsqueda. Resulta de especial importancia el correcto equilibrio (generalmente dinámico) entre diversificación e intensificación. El término diversificación se refiere a la evaluación de soluciones en regiones distantes del espacio de búsqueda (de acuerdo a una distancia previamente definida entre soluciones); también se conoce como exploración del espacio de búsqueda. El término intensificación, por otro lado, se refiere a la evaluación de soluciones en regiones acotadas y pequeñas con respecto al espacio de búsqueda centradas en el vecindario de soluciones concretas (explotación del espacio de búsqueda). El equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente las

regiones prometedoras del espacio de búsqueda global y, por otro lado, no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Dentro de las metaheurísticas podemos distinguir dos tipos de estrategias de búsqueda. Por un lado, tenemos las extensiones "inteligentes" de los métodos de búsqueda local (metaheurísticas basadas en trayectoria). El objetivo de estas estrategias es evitar de alguna forma los mínimos locales y moverse a otras regiones prometedoras del espacio de búsqueda. Este tipo de estrategia es el seguido por la búsqueda tabú, la búsqueda local iterada, la búsqueda con vecindario variable y el enfriamiento simulado. Estas metaheurísticas trabajan sobre una o varias estructuras de vecindario impuestas por el espacio de búsqueda. Una estrategia distinta es la seguida por los algoritmos basados en colonias de hormigas o los algoritmos evolutivos; éstos incorporan un componente de aprendizaje en el sentido de que, de forma implícita o explícita, intentan aprender la correlación entre las variables del problema para identificar las regiones del espacio de búsqueda con soluciones de alta calidad. Estos métodos realizan, en este sentido, un muestreo sesgado del espacio de búsqueda. Estas metaheurísticas se denominan basadas en población.

2.2. Definición de metaheurística

Más formalmente, una metaheurística se define como una tupla de elementos que, dependiendo de cómo se definan, dan lugar a una técnica concreta u otra. Esta definición formal ha sido desarrollada en [157] y posteriormente extendida en [51].

Definición 2.2.1 (Metaheurística). Una metaheurística $\mathcal{M}$ es una tupla formada por los siguientes ocho componentes:

$$\mathcal{M} = \langle \mathcal{T}, \Xi, \mu, \lambda, \Phi, \sigma, \mathcal{U}, \tau \rangle \tag{2.2.1}$$

donde:

- $\mathcal{T}$ es el conjunto de elementos que manipula la metaheurística. Este conjunto contiene al espacio de búsqueda y en la mayoría de los casos coincide con él.

- $\Xi = \{(\xi_1, D_1), (\xi_2, D_2), \ldots, (\xi_v, D_v)\}$ es un conjunto de v pares. Cada par está formado por una variable de estado de la metaheurística y el dominio de dicha variable.
- μ es el número de soluciones con las que trabaja $\mathcal{M}$ en un paso.
- λ es el número de nuevas soluciones generadas en cada iteración de $\mathcal{M}$.
- $\Phi : \mathcal{T}^\mu \times \prod_{i=1}^{v} D_i \times \mathcal{T}^\lambda \to [0,1]$ representa el operador que genera nuevas soluciones a partir de las existentes. Esta función debe cumplir para todo $x \in \mathcal{T}^\mu$ y para todo $t \in \prod_{i=1}^{v} D_i$,

$$\sum_{y \in \mathcal{T}^\lambda} \Phi(x,t,y) = 1 \tag{2.2.2}$$

- $\sigma : \mathcal{T}^\mu \times \mathcal{T}^\lambda \times \prod_{i=1}^{v} D_i \times \mathcal{T}^\mu \to [0,1]$ es una función que permite seleccionar las soluciones que serán manipuladas en la siguiente iteración de $\mathcal{M}$. Esta función debe cumplir para todo $x \in \mathcal{T}^\mu$, $z \in \mathcal{T}^\lambda$ y $t \in \prod_{i=1}^{v} D_i$,

$$\sum_{y \in \mathcal{T}^\mu} \sigma(x,z,t,y) = 1, \tag{2.2.3}$$

$$\begin{aligned} &\forall y \in \mathcal{T}^\mu, \sigma(x,z,t,y) = 0 \vee \sigma(x,z,t,y) > 0 \wedge \\ &\wedge (\forall i \in \{1,\ldots,\mu\} \bullet (\exists j \in \{1,\ldots,\mu\}, y_i = x_j) \vee (\exists j \in \{1,\ldots,\lambda\}, y_i = z_j)) \end{aligned} \tag{2.2.4}$$

- $\mathcal{U} : \mathcal{T}^\mu \times \mathcal{T}^\lambda \times \prod_{i=1}^{v} D_i \times \prod_{i=1}^{v} D_i \to [0,1]$ representa el procedimiento de actualización de las variables de estado de la metaheurística. Esta función debe cumplir para todo $x \in \mathcal{T}^\mu, z \in \mathcal{T}^\lambda$ y $t \in \prod_{i=1}^{v} D_i$,

$$\sum_{\mu \in \prod_{i=1}^{v} D_i} \mathcal{U}(x,z,t,u) = 1, \tag{2.2.5}$$

- $\tau : \mathcal{T}^\mu \times \prod_{i=1}^{v} D_i \to [falso, verdadero]$ es una función que decide la terminación del algoritmo.

La definición dada reúne el comportamiento estocástico típico de las técnicas metaheurísticas. En concreto, las funciones Φ, σ y $\mathcal{U}$ deben interpretarse como probabilidades

condicionadas. Por ejemplo, el valor de $\Phi(x, t, y)$ se interpreta como la probabilidad de que se genere el vector de hijos $y \in \mathcal{T}^\lambda$, dado que actualmente el conjunto de individuos con el que la metaheurística trabaja es $x \in \mathcal{T}^\mu$ y su estado interno viene definido por las variables de estado $t \in \prod_{i=1}^{v} D_i$. Puede observarse que las restricciones que se le imponen a las funciones Φ, σ y $\mathcal{U}$ permite considerarlas como funciones que devuelven estas probabilidades condicionadas.

Definición 2.2.2 (Estado de una metaheurística). Sea $\mathcal{M} = \langle \mathcal{T}, \Xi, \mu, \lambda, \Phi, \sigma, \mathcal{U}, \tau \rangle$ una metaheurística y $\Theta = \{\theta_1, \theta_2, \ldots, \theta_\mu\}$ el conjunto de variables que almacenarán las soluciones con las que trabaja la metaheurística. Utilizaremos la notación $first(\Xi)$ para referirnos al conjunto de las variables de estado de la metaheurística, $\{\xi_1, \xi_2, \ldots, \xi_v\}$. Un estado s de la metaheurística es un par de funciones $s = (s_1, s_2)$ con

$$s_1 : \Theta \to \mathcal{T}, \tag{2.2.6}$$

$$s_2 : first(\Xi) \to \bigcup_{i=1}^{v} D_i \tag{2.2.7}$$

donde s_2 cumple

$$s_2(\xi_i) \in D_i \forall \xi_i \in first(\Xi). \tag{2.2.8}$$

Denotamos con $\mathcal{S}_\mathcal{M}$ el conjunto de todos los estados de una metaheurística $\mathcal{M}$.

Por último, una vez definido el estado de la metaheurística, podemos definir su dinámica.

Definición 2.2.3 (Dinámica de una metaheurística). Sea $\mathcal{M} = \langle \mathcal{T}, \Xi, \mu, \lambda, \Phi, \sigma, \mathcal{U}, \tau \rangle$ una metaheurística y $\Theta = \{\theta_1, \theta_2, \ldots, \theta_\mu\}$ el conjunto de variables que almacenarán las soluciones con las que trabaja la metaheurística. Denotaremos con $\overline{\Theta}$ a la tupla $(\theta_1, \theta_2, \ldots, \theta_\mu)$ y con $\overline{\Xi}$ a la tupla $(\xi_1, \xi_2, \ldots, \xi_v)$. Extenderemos la definición de estado para que pueda aplicarse a tuplas de elementos, esto es, definimos $\overline{s} = (\overline{s}_1, \overline{s}_2)$ donde

$$\overline{s}_1 : \Theta^n \to \mathcal{T}^n, \tag{2.2.9}$$

$$\overline{s}_2 : first(\Xi)^n \to \Big(\bigcup_{i=1}^{v} D_i \Big)^n, \tag{2.2.10}$$

y además

$$\overline{s}_1(\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_n}) = (s_1(\theta_{i_1}), s_1(\theta_{i_2}), \ldots, s_1(\theta_{i_n})) \tag{2.2.11}$$

$$\overline{s}_2(\xi_{j_1}, \xi_{j_2}, \ldots, \xi_{j_n}) = (s_2(\xi_{j_1}), s_2(\xi_{j_2}), \ldots, s_2(\xi_{j_n})) \tag{2.2.12}$$

para $n \geq 2$. Diremos que r es un estado sucesor de s si $\exists t \in \mathcal{T}^\lambda$ tal que $\Phi(\overline{s}_1(\overline{\Theta}), \overline{s}_2(\overline{\Xi}), t) > 0$ y además

$$\sigma(\overline{s}_1(\overline{\Theta}), t, \overline{s}_2(\overline{\Xi}), \overline{r}_1(\overline{\Theta})) > 0 \qquad y \tag{2.2.13}$$

$$\mathcal{U}(\overline{s}_1(\overline{\Theta}), t, \overline{s}_2(\overline{\Xi}), \overline{r}_2(\overline{\Xi})) > 0 \tag{2.2.14}$$

Denotaremos con $\mathcal{F}_M$ la relación binaria "ser sucesor de" definida en el conjunto de estados de una metaheurística $\mathcal{M}$. Es decir, $\mathcal{F}_M \subseteq \mathcal{S}_M \times \mathcal{S}_M$, y $\mathcal{F}_M(s, r)$ si r es un estado sucesor de s.

Definición 2.2.4 (Ejecución de una metaheurística). Una ejecución de una metaheurística $\mathcal{M}$ es una secuencia finita o infinita de estados, $s_0, s_1, \ldots$, en la que $\mathcal{F}_M(s_i, s_{i+1})$ para todo $i \geq 0$ y además:

- si la secuencia es infinita se cumple $\tau(s_i(\overline{\Theta}), s_i(\overline{\Xi})) = falso$ para todo $i \geq 0$ y
- si la secuencia es finita se cumple $\tau(s_k(\overline{\Theta}, s_k(\overline{\Xi})) = verdadero$ para el último estado s_k y, además, $\tau(s_i(\overline{\Theta}), s_i(\overline{\Xi})) = falso$ para todo $i \geq 0$ tal que $i < k$.

2.3. Clasificación de las metaheurísticas

Las técnicas metaheurísticas se pueden clasificar y describir de distinta maneras, dependiendo de las características seleccionadas para diferenciarlas [33]. Es posible clasificarlas en metaheurísticas inspiradas en la naturaleza versus las no inspiradas, basadas en memoria o sin ella, o en métodos que usen funciones objetivo estáticas o dinámicas, etc. Una de las clasificaciones más populares, hoy en día, las describen teniendo en cuenta si utilizan en cada paso un único elemento del espacio de búsqueda, denominadas *metaheurísticas basadas en trayectoria*, o si trabajan con un conjunto de puntos o población, en este caso referidas

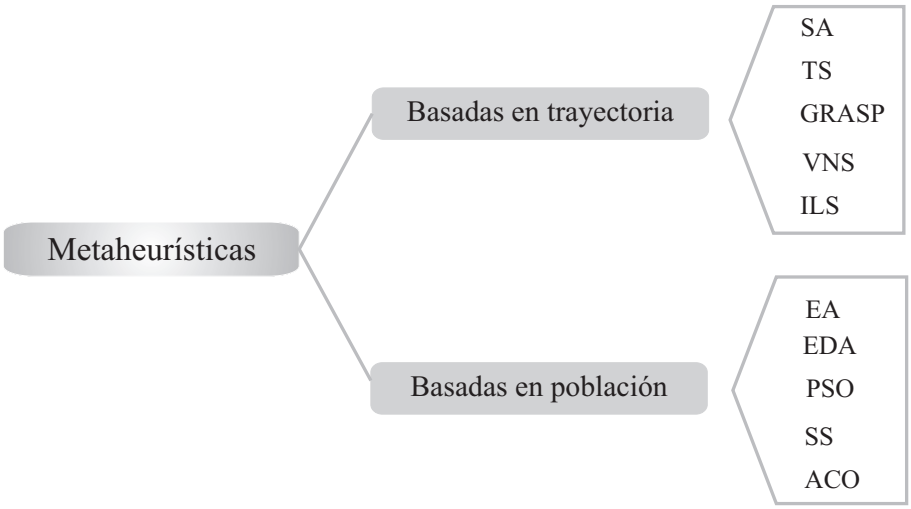


Figura 2.1: Clasificación de las metaheurísticas

como *metaheurísticas basadas en población*. La Figura 2.1 presenta la taxonomía adoptada y además las distintas metaheurísticas dentro de cada clase. En las siguientes secciones se presentarán los aspectos más destacados de cada una de las metaheurísticas.

2.3.1. Métodos basados en trayectoria

Esta sección está dedicada a distinguir las metaheurísticas más importantes basadas en trayectoria y a explicar en forma sintética su funcionamiento. En general las distintas metaheurísticas que caen bajo esta clasificación se caracterizan por partir de una única solución. La actualización de esta solución se realiza examinando el vecindario, de esta manera se genera una trayectoria. Según la notación de la Definición 2.2.1, esto se formaliza con $\mu = 1$. La mayoría de estos algoritmos surgen como extensiones de los métodos simples de búsqueda local a los que se les añade algún mecanismo para escapar de los mínimos locales, debido a que su rendimiento es poco satisfactorio. Los criterios de parada deben ser más complejos que alcanzar un mínimo local; lo que habitualmente se utiliza es alcanzar un tiempo máximo de procesamiento o un número máximo de iteraciones, encontrar una solución s de determinada calidad o detectar que no se produjeron mejoras en las últimas iteraciones.

Enfriamiento Simulado (SA)

La técnica de *enfriamiento simulado o Simulated Annealing* (SA) se considera como una de las pioneras entre los métodos metaheurísticos y una de las primeras en explicitar una estrategia para escapar del óptimos locales. SA fue presentado como un algoritmo de búsqueda por Kirkpatrick et al. en 1983 [133] y Cerney en 1985 [49], en forma independiente. El algoritmo se origina de un mecanismo estadístico, denominado "metrópolis" [162]. Los conceptos de SA fueron inspirados en el proceso de enfriamiento físico de metales. En cada iteración del algoritmo, se comparan los valores objetivos de la solución actual s y una solución s' del vecindario $N(s)$. Se acepta s' si su valor de función de *fitness* es mejor que el *fitness* de s, pasando a ser ahora la solución actual para la nueva iteración. Por otro lado, si s' tiene valor de *fitness* inferior al valor de s, tiene probabilidad de ser aceptada con la esperanza de escapar de un mínimo local en la búsqueda del óptimo global. La probabilidad de aceptar una solución de inferior calidad depende del parámetro temperatura T y de la diferencia *fitness* $f(s') - f(s')$ entre ambas soluciones; típicamente T se decrementa con cada iteración del algoritmo. Este algoritmo ha sido aplicado a una gran variedad de problemas de optimización combinatoria, tales como problemas de asignación cuadrática (QAP) [56], problemas de asignación de tareas [139], problema del vendedor viajero, diseño de circuitos a gran escala de integración [22, 186, 199], entre otros.

Búsqueda tabú (TS)

La *búsqueda tabú o Tabú Search* (TS) es una de las metaheurísticas más citadas y usadas para resolver problemas de optimización combinatoria. La idea básica fue introducida por Glover [95] y una descripción formal del método se puede encontrar en [98].

En cada iteración, TS explora el espacio de búsqueda al moverse desde la solución actual s a una vecina, donde el vecindario de la solución actual $N(s)$ son transformaciones permitidas de la solución s realizadas por una heurística *ad hoc* específica (también denominada *operador de vecindario*). Cuando se encuentra un óptimo local, la búsqueda se mueve a la mejor solución del vecindario, la cual no es necesariamente mejor que s. Para evitar ciclos, se mantiene una historia de la búsqueda en una memoria de corto plazo (*lista tabú*). Los

movimientos que podrían llevar a la trayectoria de búsqueda a soluciones recientemente visitadas se denominan *movimientos tabú* y no son permitidos, a menos que reúnan ciertas condiciones (*criterio de aspiración*). El criterio de aspiración más ampliamente usado es permitir soluciones cuyo *fitness* sea mejor que el de la mejor solución encontrada hasta el momento. La lista tabú, por lo general, almacena sólo una cantidad limitada y fija de información. Una opción rara vez usada es registrar soluciones completas, porque requiere una cantidad importante de almacenamiento y es costoso chequear si un movimiento potencial es tabú o no. La opción más usada consiste en registrar las últimas transformaciones realizadas de la solución actual y prohibir transformaciones en reversa; otras están basadas en características claves de las soluciones o de los movimientos. Este algoritmo se ha utilizado para resolver el QAP [206], el problema MAXSAT [27], el problema de ruteo (VRP) [97], entre otros.

GRASP

El *procedimiento de búsqueda miope aleatorizado y adaptativo o Gready Randomized Adaptive Search Procedure* (GRASP) presenta un diseño relativamente simple [84, 191], el cual combina una heurística *ad hoc* constructiva aleatoria-voraz con una búsqueda local. Un algoritmo GRASP es un procedimiento de multi reinicio. La fase de construcción genera una solución factible s_p, añadiendo paso a paso diferentes componentes c a la solución parcial s_p, inicialmente vacía. El conjunto de *elementos candidatos* está formado por aquellos elementos c que se pueden agregar a s_p en construcción sin alterar su factibilidad. Una función voraz mide el beneficio local de incluir un elemento candidato en la solución parcial s_p. Los elementos candidatos con un valor de función *greedy* al menos tan bueno que un umbral especificado se agregan a la *lista restringida de candidatos* (RCL). El próximo elemento que incluir en la solución se selecciona de RCL en forma aleatoria. Esta inclusión en la solución altera la función *greedy* y el conjunto de elementos candidatos para determinar la próxima RCL. El proceso de construcción finaliza cuando el conjunto de elementos candidatos está vacío. Luego se investiga su vecindario hasta hallar un mínimo local durante la fase de búsqueda local. La mejor solución es el resultado de la búsqueda. Esta metaheurística ha

sido aplicada a una amplia variedad de problemas de optimización como por ejemplo: *job shop scheduling* [7], problemas de asignación [8], problema MAX-CUT [85], VRP [192], etc.

VNS

La *búsqueda con vecindario variable o Variable Neighborhood Search* (VNS), propuesta por Hansen et al. [110], consiste en cambiar de forma sistemática la estructura de entorno durante la búsqueda. En primer lugar se debe definir la estructura del conjunto de vecindarios. La elección de la estructura se puede realizar de distintas maneras. Luego se genera una solución inicial s, se inicia el índice de vecindario k, y el algoritmo itera hasta que se alcance la condición de terminación. Cada iteración consiste de tres fases: la elección del candidato, la búsqueda local y el movimiento. En la primera fase, se elige aleatoriamente un vecino s' de s usando el $k-$ésimo vecindario. En la segunda fase, esta solución s' es el punto de partida de la búsqueda local. Cuando termina el proceso de mejora, se compara la nueva solución s'' con la original s. Si es mejor, s'' se convierte en la solución actual y se reinicializa el contador de vecindarios ($k \leftarrow 1$); si no es mejor, se repite el proceso pero utilizando el siguiente vecindario ($k \leftarrow k+1$). La búsqueda local es el paso de intensificación del método y el cambio de vecindario puede considerarse como el paso de diversificación. Con VNS se han abordado distintos problemas entre los que se encuentran los problemas de empaquetado [86], de localización [112, 221] y VRP [40, 42, 111, 166].

Búsqueda local iterativa (ILS)

La *Búsqueda local iterativa o Iterated Local Search* (ILS) es una técnica potente, simple de implementar, robusta y altamente eficiente [154, 153]. Propone un esquema en el que se incluye una heurística *ad hoc* base para mejorar los resultados de la repetición de dicha heurística. En cada iteración, se perturba la solución actual y, a esta nueva solución, se le aplica un método de búsqueda local para mejorarla. El mínimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa una prueba de aceptación. La importancia del proceso de perturbación es obvia: si es demasiado pequeño puede que el algoritmo no sea capaz de escapar del mínimo local; por otro lado, si es demasiado grande,

la perturbación puede hacer que el algoritmo sea como un método de búsqueda local con un reinicio aleatorio. Por lo tanto, el método de perturbación debe generar una nueva solución que sirva como inicio a la búsqueda local, pero que no debe estar muy lejos de la actual para que no sea una solución aleatoria. El criterio de aceptación actúa como contra-equilibrio, ya que filtra la aceptación de nuevas soluciones dependiendo de la historia de búsqueda y de las características del nuevo mínimo local. La historia del proceso de búsqueda puede ser explotado en la forma de memoria de corto o largo término. Esta metaheurística ha sido usada para resolver el problema del viajante de comercio [203], problema SAT [121], entre otros.

2.3.2. Métodos basados en población

Los métodos basados en población trabajan en cada iteración con un conjunto de soluciones, usualmente denominado *población*; es decir, generalmente $\mu > 1$ y/o $\lambda > 1$. De esta forma, los algoritmos basados en población proveen una forma natural e intrínseca de explorar el espacio de búsqueda.

Algoritmos evolutivos (EAs)

Los *algoritmos evolutivos o Evolutionary Algorithms* (EAs) están inspirados en la capacidad natural de evolucionar de los seres vivos bien adaptados a su ambiente. Los algoritmos evolutivos son algoritmos estocásticos cuyos métodos de búsqueda modelan un fenómeno natural: la herencia genética y la rivalidad darwiniana para la supervivencia. Operan sobre una población de soluciones, denominadas *individuos*. Por lo general, la población inicialmente se genera en forma aleatoria, pero también se puede usar alguna heurística *ad hoc* de construcción.

El esquema general de un EA está formado por tres pasos: seleccionar, alterar (reproducción) y reemplazar; los cuales se repiten hasta que se alcanza un punto de terminación, normalmente después de un número dado de iteraciones. Cada solución se evalúa para dar alguna medida de su aptitud (*fitness*). Se seleccionan algunos miembros de la población

(paso seleccionar) para someterlos a transformaciones (paso alterar) por medio de *operadores genéticos* para formar nuevas soluciones. Hay transformaciones de orden superior (tipo crossover), las cuales crean nuevos individuos al combinar varias partes de dos o más individuos y transformaciones unarias (tipo mutación), las cuales crean nuevos individuos al producir pequeños cambios en un único individuo. Luego, se forma una nueva población al seleccionar los individuos más idóneos (paso reemplazo) de la población actual y/o los mejores individuos generados, dando lugar a la siguiente generación del algoritmo. Tras algún número de generaciones, el algoritmo converge; es de esperar que el mejor individuo represente una solución cercana a la óptima.

Estos algoritmos establecen un equilibrio entre la explotación de buenas soluciones (fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (fase de reproducción), basados sobre el hecho que la política de reemplazo permite la aceptación de nuevas soluciones que no mejoran necesariamente las existentes. En la literatura se han propuesto diferentes algoritmos basados en este esquema general. Básicamente, estas propuestas se pueden clasificar en tres categorías que fueron desarrolladas de forma independiente. Estas categorías son: la *programación evolutiva o Evolutionary Programming* (EP) desarrollada por Fogel [87, 88], las *estrategias evolutivas o Evolution Strategies* (ES) propuestas por Rechenberg en [187] y los algoritmos genéticos o Genetic Algorithms (GA) introducidos por Holland en [120] (consultar para mayores referencias a [100], [165], [188] y [217]).

Algoritmos de estimación de distribuciones (EDAs)

Los *algoritmos de estimación de distribuciones o Estimation of Distribution Algorithms* (EDAs) [169] fueron creados para reducir las desventajas de los operadores de recombinación de los EAs, los cuales tienden romper buenos bloques constructivos. Estos algoritmos tienen una fundamentación teórica en la teoría de probabilidad. Como los EAs, trabajan con una población que evoluciona a medida que progresa la búsqueda. Una vez generada la población inicial, se repite el siguiente ciclo hasta alcanzar el punto de parada. Se selecciona una parte de las mejores soluciones pertenecientes a la población actual, de las cuales se calcula una *distribución de probabilidades* del espacio de búsqueda. Esta distribución de probabilidades

es la muestra para producir la población de la próxima iteración. El campo de EDAs es aún muy nuevo. Se pueden encontrar aplicaciones al problema de la mochila, problema de planificación de tareas y otros problemas de optimización en [141].

Optimización por cúmulo de partículas (PSO)

La *optimización por cúmulo de partículas o Particle Swarm Optimization* (PSO) [132] está inspirada en el comportamiento social del vuelo de aves o el movimiento de los bancos de peces. El algoritmo inicia con una población de soluciones aleatorias y busca el óptimo al actualizar generaciones. Las soluciones potenciales del problema, denominadas *partículas*, vuelan a través del espacio del problema al seguir las partículas óptimas actuales. PSO tiene dos operadores primarios: actualización de velocidad y *actualización de posiciones*. Durante cada generación cada partícula es acelerada hacia la partícula mejor posicionada en la iteración previa y hacia la mejor posicionada global. A cada iteración se calcula un nuevo valor de velocidad para cada partícula en función de su velocidad actual, la distancia entre la posición actual y la mejor conocida por esa partícula (componente cognitivo), y distancia entre la posición actual y la mejor posición del vecindario (componente social). El valor de la nueva velocidad se usa para calcular la próxima posición de la partícula en el espacio de búsqueda. Este proceso es repetido una cantidad de veces o hasta que se logre un error mínimo. PSO se ha aplicado con éxito en diferentes campos de investigación. Algunos ejemplos son: optimización de funciones numéricas [224], entrenamiento de redes neuronales [106], problema del viajante de comercio [176], entre otros.

Búsqueda dispersa (SS)

La *búsqueda dispersa o Scatter Search* (SS) [96, 99] opera sobre un *conjunto de referencia de soluciones* (S_{ref}) formado por soluciones factibles al problema bajo consideración. Este conjunto de referencia es generado e iterativamente actualizado, intentando intensificar y diversificar la búsqueda. Tras combinar las soluciones en S_{ref}, se aplica un procedimiento de búsqueda local para mejorar las soluciones resultantes. El S_{ref} se actualiza para incorporar tanto buenas soluciones como dispersas. Estos pasos se repiten hasta que se alcanza

algún criterio de parada. SS ha despertado gran interés en los últimos años, entre otros problemas, ha sido aplicado a problemas de asignación multiobjetivo [140] y a problemas de ordenamiento lineal [48].

Optimización basada en colonias de hormigas (ACO)

Los *algoritmos de optimización basados en colonias de hormigas o Ant Colony Optimization* (ACO) fue propuesta por Dorigo [71, 72, 74]. Esta metaheurística emplea estrategias inspiradas en el comportamiento de las colonias de hormigas para descubrir fuentes de alimentación. Mientras que una hormiga realiza su camino va depositando una sustancia química, denominada *feromona*. Esta sustancia ayudará encontrar la comida al resto de las hormigas. Como resultado de la exploración y cooperación entre los individuos de la colonia se establece, en general, el camino más corto entre dicha fuente de alimentación y el hormiguero. Este comportamiento es el que intenta simular este método para resolver problemas de optimización. La técnica se basa en dos pasos principales: construcción de una solución basada en el comportamiento de una hormiga y actualización de los rastros de feromonas artificiales. La metaheurística ACO ha sido exitosamente aplicada a problemas de planificación [55], de asignación cuadrática [90], del viajante de comercio [73], etc.

2.4. Metaheurísticas paralelas

Aunque el uso de metaheurísticas permite reducir en forma significativa la complejidad temporal del proceso de búsqueda, este tiempo sigue siendo muy elevado en problemas industriales. De esta manera, el paralelismo es necesario no solamente para disminuir el tiempo de resolución, sino también para introducir mejoras en la calidad de las soluciones provistas. Para cada una de las dos familias de metaheurísticas, descritas en la Sección 2.3, se han propuesto diferentes modelos paralelos en la literatura. Cada uno de ellos muestra una opción alternativa para manejar y usar el paralelismo. En las siguientes secciones mostraremos los distintos modelos paralelos atendiendo a la clasificación de metaheurísticas realizadas.

2.4.1. Modelos paralelos para métodos basados en trayectoria

Los modelos paralelos de las metaheurísticas basadas en trayectoria se pueden subdividir en tres modelos generales: *modelo de múltiples ejecuciones*; exploración y evaluación paralela del vecindario o *modelo de movimientos paralelos*; y evaluación paralela de una única solución o *modelo de aceleración del movimiento*. En los siguientes párrafos explicaremos cada uno de ellos.

Modelo de múltiples ejecuciones

Consiste en lanzar en forma simultánea varias metaheurísticas basadas en trayectoria para obtener mejores soluciones. Pueden ser heterogéneos u homogéneos, independientes o cooperativos, comenzar desde la misma solución o una diferente, configurarse con los mismos parámetros o diferentes.

Modelo de movimientos paralelos

Es un modelo maestro-esclavo de bajo nivel que no altera el comportamiento de la metaheurística. Una búsqueda secuencial obtiene los mismos resultados empleando mayor tiempo. Al comienzo de cada iteración, el maestro duplica la solución actual y la distribuye entre distintos nodos. Cada uno maneja algunas soluciones candidatas y retorna el resultado al maestro.

Modelo de aceleración del movimiento

La calidad de cada movimiento se evalúa de una manera central y en paralelo. Este modelo es particularmente interesante cuando la función de evaluación es en sí misma paralelizable por su alto consumo de tiempo de CPU y/o intensivo uso de entradas/salidas. En este caso, la función puede ser vista como una agregación de una cierta cantidad de funciones parciales.

2.4.2. Modelos paralelos para métodos basados en población

Manejar grandes conjuntos de individuos por muchas generaciones requiere cantidades significativas de recursos computacionales. Por lo tanto, hay dos razones para paralelizar una metaheurística basada en población. La primera razón es lograr reducciones en el tiempo al distribuir el esfuerzo computacional (*modelo maestro-esclavo*). La segunda causa es obtener un beneficio de las propiedades paralelas desde el punto de vista algorítmico, en analogía con la evolución paralela de poblaciones distribuidas espacialmente (*modelo con población estructurada*). A continuación explicaremos cada una de ellas.

Modelo maestro-esclavo

Uno de los métodos muy usados en los comienzos del paralelismo fue el método maestro-esclavo. En este método, un procesador central realiza las operaciones que manejan la población (el operador de selección en el caso de EAs) mientras que los procesadores esclavos realizan operaciones donde sólo están involucrados individuos de esa población (función de evaluación en el caso de EAs). Aunque esta técnica es interesante para lograr menores tiempos de cómputo, no implica ningún cambio de la metaheurística en sí misma y, por lo tanto, no ofrece ningún nuevo desafío para mejorar la calidad de las soluciones halladas por los procesos evolutivos.

Modelo con poblaciones estructuradas

Los modelos con población estructurada están basados en la idea que las poblaciones naturales tienden poseer una estructura espacial y organizarse en *demes*. Los demes son grupos semi-independientes de individuos o subpoblaciones, las cuales tienen una baja interacción con sus subpoblaciones vecinas. Varios modelos basados en esta idea han sido propuestos en la literatura.

Muchos investigadores usan varios procesadores para que la ejecución de los algoritmos secuenciales resulten más rápidas, simplemente porque las ejecuciones independientes se pueden realizar en menos tiempo usando varios procesadores que en el caso de realizarlas sobre un único procesador. Así no existe interacción entre las ejecuciones independientes.

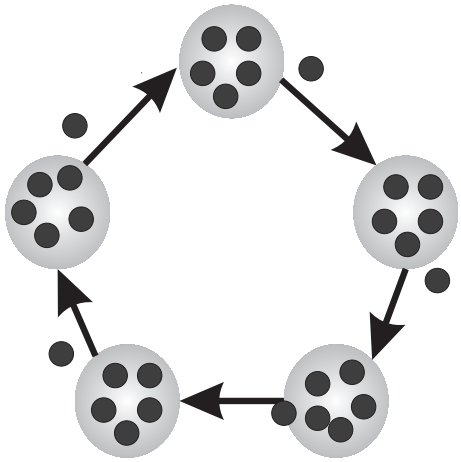


Figura 2.2: Modelo distribuido

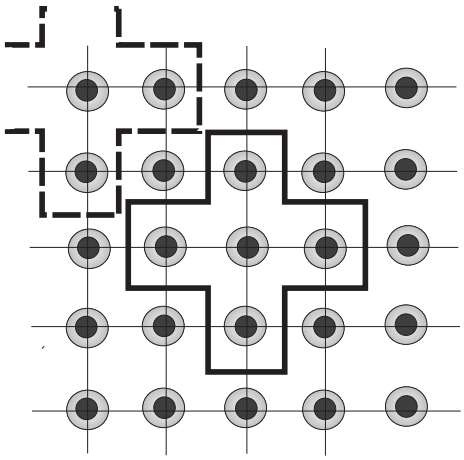


Figura 2.3: Modelo celular

Sin embargo, la mayoría de los EAs paralelos (PEAs) hallados en la literatura utilizan alguna clase de disposición espacial para los individuos y entonces se paralelizan las porciones resultantes, asignándolas en un conjunto de procesadores. Los tipos más ampliamente conocidos de EAs estructurados son los algoritmos *distribuidos* (dEA) (o de grano grueso) y los *celulares* (cEA) (o de grano fino) [18].

En el caso de los EAs distribuidos, la población se divide en un conjunto de islas, donde se ejecutan EAs aislados (ver Figura 2.2). Por lo general, se realizan intercambios esporádicos de individuos entre esas islas; el objetivo es introducir alguna diversidad en las subpoblaciones y evitar, de esta manera, que caigan en un óptimo local. El modelo distribuido requiere la identificación de una política de migración adecuada. Los principales parámetros de la política de migración son los siguientes: periodo de migración, que es el número de iteraciones entre dos intercambios consecutivos de información; tasa de migración, que es el número de individuos enviados a las subpoblaciones vecinas; criterio de selección de los individuos para migrar y criterio de reemplazo, el primero indica si se reemplazan algunos individuos de la población actual para introducir a los inmigrantes y el segundo determina qué individuos se reemplazarán; y topología, que indica dónde se envían los individuos de cada isla y de dónde se pueden recibir. Finalmente, se debe decidir si estos intercambios se realizan de forma síncrona o asíncrona.

En el caso de los EAs celulares, se introduce el concepto de vecindario, por lo que un individuo sólo puede interactuar con sus vecinos más próximos en el ciclo de reproducción (ver esquema presentado en la Figura 2.3). La superposición de los vecindarios de cEA contribuye a la exploración del espacio de búsqueda porque una lenta difusión de las soluciones a través de la población proporciona exploración, mientras que la explotación se lleva a cabo dentro de cada vecindario.

2.5. Metaheurísticas heterogéneas

Varias estrategias de paralelización se pueden aplicar a metaheurísticas, como hemos detallado en la sección anterior. Entre ellas, existen modelos paralelos donde múltiples hilos de búsqueda exploran en forma concurrente el espacio de búsqueda. Estos modelos son llamados estrategias paralelas de *tipo 3* en [62, 63] y opciones de *caminatas múltiples* en [64]. Si cada uno de los hilos de búsqueda realiza la misma clase de búsqueda, es decir, el mismo genotipo, operadores, etc., se está frente a *metaheurísticas paralelas homogéneas*. La mayoría de los dGAs propuestos en la literatura pertenecen a esta categoría [170, 211]. Por otra parte, si cada uno de los hilos de búsqueda utiliza distinto procedimiento de búsqueda, ya sea a nivel algorítmico o a nivel de configuración de parámetros, se obtienen *metaheurísticas paralelas heterogéneas o Parallel Heterogeneous Metaheuristics* (PHM). La utilización de múltiples hilos de búsqueda con distintos valores de parámetros y diferentes estrategias permite una mayor diversidad y mejor exploración del espacio de búsqueda, lo cual producirá soluciones más exactas.

Las metaheurísticas heterogéneas podrían ser consideradas como algoritmos híbridos [198], pero hay una razón conceptual básica para estudiar PHMs: mientras la hibridación consiste en agregar conocimiento particular del problema en el algoritmo para resolver problemas específicos, la heterogeneidad ayuda desarrollar metaheurísticas más robustas tratando de ofrecer un alto nivel de performance sobre una variedad amplia de configuraciones y características del problema. En definitiva, son formas diferentes de mejorar una metaheurística.

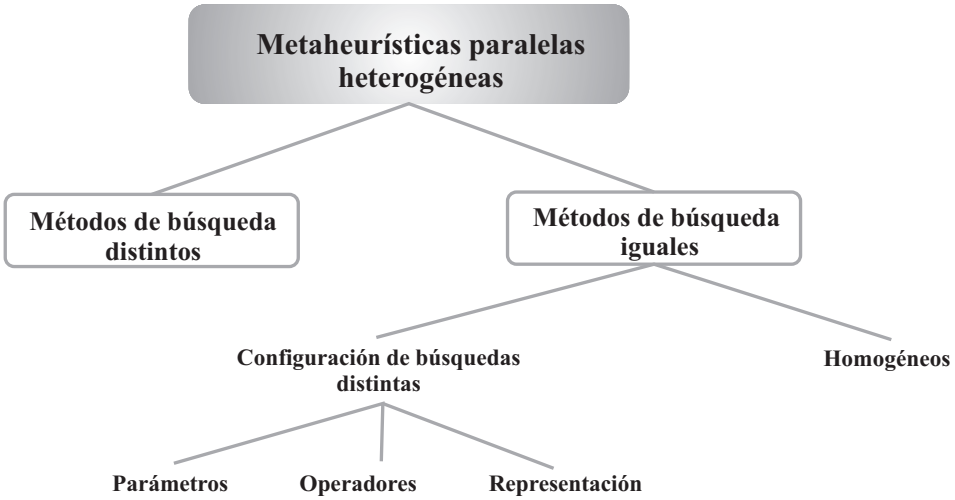


Figura 2.4: Taxonomía de las metaheurísticas paralelas heterogéneas

Luna et al. [155] realizan una extensa reseña de las distintas metaheurísticas heterogéneas que se pueden encontrar en la literatura, considerando las distintas propuestas desde un punto de vista histórico como desde una perspectiva orientada a la aplicación. Además en ese trabajo, los autores desarrollan una taxonomía de las PMHs desde dos perspectivas: jerárquica y plana, las cuales serán descritas en la siguiente sección. La Figura 2.4 muestra la estructura de la taxonomía.

2.5.1. Clasificación de las PHMs

En un primer nivel, los hilos de búsqueda de las PHMs pueden usar el mismo o diferentes métodos de búsqueda. En este último caso, cada hilo de búsqueda es guiado por una metaheurística diferente [38, 178]. Esta clase de heterogeneidad permite el diseño de métodos de búsqueda más robustos debido a la utilización de diferentes metaheurísticas con equilibrios específicos entre intensificación y diversificación.

En el caso de PHMs compuestos for varios hilos de búsqueda que utilizan la misma técnica de optimización pero con distintas configuraciones, se pueden obtener distintas subclasificaciones en función del lugar donde se han introducido los cambios en cada método de configuración. De esta manera se pueden distinguir entre:

Configuración de parámetros. Esta opción consiste en usar diferentes parámetros en cada hilo hilo de búsqueda. Estos parámetros pueden estar relacionados con la utilización de distintas probabilidades de recombinación y/o mutación en cada subpoblación de un GA [210]. Es la clase de PHM con más difusión, ya que representa una extensión directa de los modelos homogéneos canónicos.

Operadores. En general, las metaheurísticas utilizan operadores heurísticos para realizar la exploración del espacio de búsqueda. Ejemplos son los operadores de recombinación y mutación en un GA o el procedimiento de construcción de soluciones de las hormigas en ACO. En cada campo de investigación pueden aparecer varias variantes de cada operador. De esta forma, es posible usar distintos procedimientos para la misma operación en cada hilo de búsqueda de una metaheurística paralela [15, 82, 114].

Representación. Es la clase más sutil de heterogeneidad donde cada hilo de búsqueda usa una representación diferente de las soluciones para explorar el espacio de búsqueda [89, 148, 213, 218].

Todas las PHMs mencionadas anteriormente son *heterogeneidades a nivel de software*: la diferencia entre los métodos de exploración de los hilos de búsqueda está definida por el diseño de software. Sin embargo, la homogeneidad y heterogeneidad se podrían entender en términos de la plataforma de ejecución: *heterogeneidad a nivel de hardware*, donde cada isla ejecuta sobre distinto hardware y/o sistema operativo [9, 17, 61, 208].

En función de la comunicación entre los hilos de búsqueda, se pueden distinguir dos modelos de ejecución de las PHMs: *ejecuciones independientes* e *hilos de búsqueda colaborativos*. En el modelo de ejecuciones independientes no hay comunicación entre los hilos de búsqueda, es decir, no interactúan durante la exploración y de esta manera los hilos realizan búsquedas totalmente independientes. Por lo general, al final de las operaciones se presentan las mejores soluciones como resultado y las restantes son descartadas [24, 142].

Si los hilos comparten información producida durante la trayectoria que ellos investigan se obtiene un modelo de búsqueda colaborativa. En el campo de los algoritmos evolutivos, esta interacción, denominada migración, es ampliamente usada y consiste en intercambiar

uno o más individuos entre subpoblaciones del algoritmo [211]. Sin embargo, PHMs involucrando algoritmos basados en trayectoria, por lo general, utilizan un pool de soluciones como un punto central de comunicación entre los distintos hilos [39].

Otro nivel ortogonal de heterogeneidad se puede definir al observar la relación de comunicación que mantienen las metaheurísticas básicas del algoritmo entre sí. Si la cantidad de recursos que usa un hilo para realizar la búsqueda (cantidad de individuos, tamaño de la lista tabú, etc.) no es constante durante la evolución, es decir, depende del éxito previo de su estrategia de búsqueda, entonces se dice que los hilos están compitiendo. En otro caso, se dice que las subpoblaciones colaboran para hallar el óptimo. De esta manera, se está diferenciando entre *heterogeneidad basada en la competencia* [127, 174, 227] y *heterogeneidad basada en la colaboración* [60, 115].

2.6. Metaheurísticas usadas en este trabajo

En la Sección 2.3 hemos realizado una descripción muy general de las distintas metaheurísticas. Por lo tanto, en esta sección nos centraremos en detallar más exhaustivamente las metaheurísticas que hemos utilizado como mecanismo de búsqueda para aplicarlos al problema de empaquetado elegido. En este trabajo de tesis utilizamos metaheurísticas basadas en población: una clase de algoritmos evolutivos, algoritmos genéticos, y algoritmos de optimización basados en colonias de hormigas. Es importante mencionar aquí que, debido a la complejidad de los problemas abordados, tanto la representación como los operadores genéticos utilizados dependen del problema que resolver, por lo que sólo incluimos los esquemas generales de los dos algoritmos. Los detalles de implementación serán introducidos en los capítulos siguientes.

2.6.1. Algoritmos evolutivos

Alrededor de los años 60, varias corrientes de investigación comenzaron, en forma independiente, formando lo que ahora se conoce como *computación evolutiva* [28], tomando su inspiración en la evolución biológica. La idea era implementar algoritmos basados en el modelo de la evolución orgánica, denominados *algoritmos evolutivos*, como un intento de

resolver tareas complejas de optimización en computadoras. Hoy en día, debido a su robustez, a su amplia aplicabilidad y a la disponibilidad de cada vez mayor poder computacional, la computación evolutiva recibe una atención creciente por parte de los investigadores de un gran número de disciplinas dispares.

Los algoritmos evolutivos constituyen una técnica general de resolución de problemas de búsqueda y optimización inspirada en los procesos biológicos que se pueden apreciar en la naturaleza, como la selección natural [65] y la herencia genética [161]. Estos algoritmos permiten abordar problemas complejos que surgen en las ingenierías y los campos científicos: problemas de planificación de tareas, horarios, tráfico aéreo y ferroviario, búsqueda de caminos óptimos, optimización de funciones, etc.

Los algoritmos evolutivos se pueden caracterizar como modelos computacionales del proceso evolutivo. En cada iteración, se aplica un cierto conjunto de operadores a los individuos de la población actual para generar los individuos de la población de la próxima generación (iteración). Generalmente, los EAs usan operadores denominados recombinación o cruce para recombinar dos o más individuos para producir nuevos individuos. Además usan operadores de mutación , es decir, modificaciones que causan una auto-adaptación de los individuos. La fuerza motriz de los algoritmos evolutivos es la selección de los individuos sobre la base de su aptitud (que puede basarse en la función objetivo, el resultado de un experimento de simulación o algún otro tipo de medida de calidad). Los individuos con una mayor aptitud tienen mayor probabilidad de ser elegidos como miembros de la población de la siguiente iteración (o como padres para la generación de nuevos individuos). Esto corresponde al principio de la supervivencia del más apto en la evolución natural. Es la capacidad de la naturaleza para adaptarse a un entorno cambiante lo que dio la inspiración para los algoritmos evolutivos

Analizamos a continuación en forma detalla el funcionamiento de un EA, cuyo pseudocódigo se muestra en el Algoritmo 1 [11]. Los algoritmos evolutivos mantienen una población de individuos, $P(t) = x_1^t, x_2^t, \ldots, x_\mu^t$ en la iteración t, donde μ es el tamaño de la población. Cada individuo representa una solución potencial al problema y se implementa como una estructura de datos S. En cada iteración del algoritmo se selecciona un conjunto

Algoritmo 1 Pseudocódigo de un EA

```
t ← 0;
P(t) ←generarPoblaciónInicial();
evaluar(P(t));
while not(condición de terminación) do
   P'(t) ← seleccionarPadres(P(t));
   P'(t) ← alterar P'(t);
   evaluar(P'(t));
   P(t) ←seleccionarNuevaPoblación(P(t), P'(t))
   t ← t + 1;
end while
return la mejor solución encontrada
```

de individuos (paso seleccionarPadres) a los que se denomina padres. A estos miembros se les aplica ciertas transformaciones (paso alterar) por medio de operadores "genéticos" para formar λ nuevas soluciones (denominadas hijos). Hay transformaciones de orden superior c_j (tipo crossover), las cuales crean nuevos individuos al combinar varias partes de dos o más individuos ($m_i : S \times \cdots \times S \rightarrow S$) y transformaciones unarias m_i (tipo mutación), las cuales crean nuevos individuos al producir pequeños cambios en un único individuo ($m_i : S \rightarrow S$). En [81] se han investigado los méritos del uso de más de dos padres en el proceso de reproducción. Cada solución x_i^t se evalúa (paso evaluar) para dar alguna medida de su aptitud o "*fitness*". Entonces, se forma una nueva población (iteración $t+1$) al seleccionar los individuos más idóneos (paso seleccionarNuevaPoblación). Tras una cierta cantidad de generaciones, el algoritmo converge; es de esperar que el mejor individuo represente una solución cercana a la óptima.

Hay varias variantes de algoritmos evolutivos, las cuales incluyen:

- Programación evolutiva: se hace evolucionar una población de máquinas de estados finitos sometiéndolas a transformaciones unitarias.
- Estrategias evolutivas: se hace evolucionar una población de estructuras compuestas por un vector real y una variable aleatoria (parámetro) que codifican las posibles soluciones de un problema numérico y las dimensiones de los cambios o saltos. La selección es implícita.

- Programación genética: se hace evolucionar una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.
- Algoritmos genéticos: se hace evolucionar una población de enteros binarios sometiéndolos a transformaciones unitarias y binarias genéticas y a un proceso de selección.

A pesar de las similitudes entre los algoritmos evolutivos, hay también muchas diferencias entre ellos (a menudo ocultas a un bajo nivel de abstracción). Frecuentemente usan estructuras de datos S diferentes para la representación de los cromosomas, en consecuencia, los operadores genéticos también son diferentes. Pueden incorporar o no alguna otra información (para controlar el proceso de búsqueda) en sus genes.

Hay muchos métodos para seleccionar individuos para supervivencia y reproducción. Esos métodos incluyen:

- *selección proporcional*: la probabilidad de selección es proporcional al *fitness* del individuo.
- *métodos de ranking*: todos los individuos de la población se ordenan de mejor a peor y las probabilidades de selección son fijas durante todo el proceso de evolución.
- *selección por torneo*: algunos individuos (usualmente dos) compiten por ser seleccionados para la nueva generación; este paso de competencia (torneo) se repite μ veces.

La selección proporcional puede necesitar el uso de métodos de truncamiento o de escalamiento. Hay diferentes formas para asignar probabilidades en métodos de *ranking* (distribuciones lineales o no lineales). El tamaño de un torneo juega un rol significativo en el método de selección por torneo.

Es importante determinar las políticas generacionales, es decir, especificar la forma en la cual quedará conformada la próxima generación. Por ejemplo, es posible reemplazar la población entera de tamaño μ por la población de λ hijos, o seleccionar los mejores individuos de las dos poblaciones (padres e hijos), esta selección se puede hacer de una manera determinística o no determinística. Es también posible producir pocos hijos (por lo general

$\lambda = 1$ o $\lambda = 2$), con los cuales reemplazar algunos individuos, consecuentemente coexisten con sus padres. Sistemas basados en tales políticas son llamados de *estado estacionario*. En un modelo generacional, el conjunto de los padres se mantiene fijo mientras se generan todos los hijos que formarán parte de la próxima generación. También, se puede usar un modelo elitista, el cual pasa el mejor individuo de una generación a otra. Esto significa que si el mejor individuo de la generación actual corre el riesgo de perderse, ya sea por la selección o los operadores genéticos, el sistema fuerza a que quede en la próxima generación; tal modelo es muy útil para resolver muchas clases de problemas de optimización. Existe un modelo intermedio ($1 < \lambda < \mu$) en el cual se determina un porcentaje de la población para participar en la creación de la nueva generación. El modelo se denomina de *gap generacional*. El gap es el porcentaje de la población que participa en la aplicación de los operadores evolutivos. La propuesta constituye una generalización de los modelos de evolución, que incluye los dos enfoques opuestos ya presentados (modelo generacional y de estado estacionario).

Las representaciones de las soluciones potenciales para un problema particular, junto con el conjunto de operadores genéticos, constituyen los componentes esenciales de cualquier algoritmo evolutivo. Esos son los elementos claves que permiten distinguir varios paradigmas de métodos evolutivos.

Una de las principales dificultades de los algoritmos evolutivos (en particular cuando se aplica búsqueda local) es la convergencia prematura hacia soluciones sub-óptimas. El mecanismo más simple para diversificar el proceso de búsqueda es el uso de un operador de mutación aleatorio. Con el fin de evitar la convergencia prematura, existen otras formas de mantener la diversidad de la población. Probablemente, la más antigua es *crowding* [67]. Estrategias más recientes son *fitness sharing* [102] y *niching* [158] en el cual la aptitud para la reproducción asignada a un individuo en una población se reduce proporcionalmente al número de otros individuos que comparten la misma región del espacio de búsqueda.

Los algoritmos evolutivos basan parte de sus buenos resultados en el equilibrio entre una eficiente exploración y una eficiente explotación cuando se resuelve un problema difícil. La explotación consiste en utilizar el conocimiento adquirido por la exploración para llegar a mejores posiciones en el espacio de búsqueda. Por otra parte, la exploración es investigar las

zonas nuevas y desconocidas del espacio de búsqueda. Por consiguiente, tanto la explotación y exploración deben ser debidamente controladas durante el proceso de búsqueda genética a fin de lograr la convergencia a la solución óptima. La exploración es útil para evitar estancarse en óptimos locales mientras que la explotación se usa para obtener el óptimo global una vez que se ha aproximado a él lo suficiente.

Dos importantes filosofías para tratar con el problema de la intensificación/diversificación es la hibridación de un algoritmo [207] y los algoritmos estructurados y paralelos [18], temas abordados más tarde en este trabajo de tesis.

Según la Definición 2.2.1, la principal diferencia de los EAs con otros algoritmos metaheurísticos es la función Φ_{EA}, que se calcula a partir de otras tres funciones [51]:

$$\Phi_{EA}(x,t,y) = \sum_{z\in\mathcal{T}^\lambda}\sum_{w\in\mathcal{T}^\lambda}\Big(\omega_s(x,t,z)\times\omega_r(z,t,w)\times\prod_{j=1}^{\lambda}\omega_m(w_j,t,y_j)\Big), \tag{2.6.1}$$

donde:

- $\omega_s : \mathcal{T}^\mu \times \prod_{i=1}^{v} D_i \times \mathcal{T}^\lambda \to [0,1]$, representa el operador de selección de padres. Esta función debe cumplir para todo $x \in \mathcal{T}^\mu$ y para todo $t \in \prod_{i=1}^{v} D_i$

$$\sum_{y\in\mathcal{T}^\lambda}\omega_s(x,t,y) = 1, \tag{2.6.2}$$

$$\forall y \in \mathcal{T}^\lambda, \omega_s(x,t,y) = 0 \vee \omega_s(x,t,y) > 0 \wedge (\forall i \in 1,\ldots,\lambda, \exists j \in 1,\ldots,\mu, y_i = x_j). \tag{2.6.3}$$

- $\omega_r : \mathcal{T}^\lambda \times \prod_{i=1}^{v} D_i \times \mathcal{T}^\lambda \to [0,1]$, representa el operador de recombinación. Esta función debe cumplir para todo $x \in \mathcal{T}^\lambda$ y para todo $t \in \prod_{i=1}^{v} D_i$

$$\sum_{y\in\mathcal{T}^\lambda}\omega_r(x,t,y) = 1, \tag{2.6.4}$$

- $\omega_m : \mathcal{T} \times \prod_{i=1}^{v} D_i \times \mathcal{T} \to [0,1]$, representa el operador de mutación. Esta función debe cumplir para todo $x \in \mathcal{T}$ y para todo $t \in \prod_{i=1}^{v} D_i$

$$\sum_{y\in\mathcal{T}}\omega_m(x,t,y) = 1, \tag{2.6.5}$$

El siguiente resultado demuestra que la función Φ_{EA} cumple 2.2.2 y, por lo tanto, está bien definida.

Proposición 2.6.1. *La función Φ_{EA} definida en 2.6.1 cumple*

$$\sum_{y\in\mathcal{T}^\lambda}\Phi_{EA}(x,t,y)=1, \tag{2.6.6}$$

Demostración: La demostración consiste en una sencilla comprobación basada en las propiedades de ω_s, ω_r y ω_m que se muestran en 2.6.2, 2.6.4 y 2.6.5, respectivamente. En primer lugar tenemos:

$$\sum_{y\in\mathcal{T}^\lambda}\Phi_{EA}(x,t,y)=\sum_{y\in\mathcal{T}^\lambda}\sum_{z\in\mathcal{T}^\lambda}\sum_{w\in\mathcal{T}^\lambda}\Big(\omega_s(x,t,z)\times\omega_r(z,t,w)\times\prod_{j=1}^{\lambda}\omega_m(w_j,t,y_j)\Big). \tag{2.6.7}$$

En virtud de las propiedades distributiva y conmutativa, podemos introducir los sumarios dentro de la expresión entre paréntesis hasta colocarlos delante del término en el que aparece su índice, esto es,

$$\sum_{y\in\mathcal{T}^\lambda}\Phi_{EA}(x,t,y)=\sum_{t\in\mathcal{T}^\lambda}\omega_s(x,t,z)\times\left[\sum_{w\in\mathcal{T}^\lambda}\omega_r(z,t,w)\times\left(\sum_{y\in\mathcal{T}^\lambda}\prod_{j=1}^{\lambda}\omega_m(w_j,t,y_j)\right)\right]. \tag{2.6.8}$$

Podemos reescribir el término interior de la expresión anterior del siguiente modo

$$\begin{aligned}\sum_{y\in\mathcal{T}^\lambda}\prod_{j=1}^{\lambda}\omega_m(w_j,t,y_j)&=\sum_{y\in\mathcal{T}^\lambda}\omega_m(w_1,t,y_1)\omega_m(w_2,t,y_2)\ldots\omega_m(w_\lambda,t,y_\lambda)=\\&\sum_{y_1\in\mathcal{T}^\lambda}\sum_{y_2\in\mathcal{T}^\lambda}\cdots\sum_{y_\lambda\in\mathcal{T}^\lambda}\omega_m(w_1,t,y_1)\omega_m(w_2,t,y_2)\ldots\omega_m(w_\lambda,t,y_\lambda)=\\\left(\sum_{y_1\in\mathcal{T}^\lambda}\omega_m(w_1,t,y_1)\right)&\left(\sum_{y_2\in\mathcal{T}^\lambda}\omega_m(w_2,t,y_2)\right)\ldots\left(\sum_{y_\lambda\in\mathcal{T}^\lambda}\omega_m(w_\lambda,t,y_\lambda)\right)=\\&\prod_{j=1}^{\lambda}\left(\sum_{y_j\in\mathcal{T}^\lambda}\omega_m(w_j,t,y_j)\right)\end{aligned} \tag{2.6.9}$$

De esta forma, podemos escribir

$$\sum_{y\in\mathcal{T}^\lambda}\Phi_{EA}(x,t,y)=\sum_{z\in\mathcal{T}^\lambda}\omega_s(x,t,z)\times\left\{\sum_{w\in\mathcal{T}^\lambda}\omega_r(z,t,w)\times\left[\prod_{j=1}^{\lambda}\left(\sum_{y_j\in\mathcal{T}}\omega_m(w_j,t,y_j)\right)\right]\right\}. \tag{2.6.10}$$

En virtud de 2.6.5 tenemos $\sum_{y_j \in \mathcal{T}} \omega_m(w_j, t, y_j) = 1$ y llegamos a

$$\sum_{y \in \mathcal{T}^\lambda} \Phi_{EA}(x, t, y) = \sum_{z \in \mathcal{T}^\lambda} \omega_s(x, t, z) \times \left(\sum_{w \in \mathcal{T}^\lambda} \omega_r(z, t, w) \right). \tag{2.6.11}$$

Finalmente, usando las ecuaciones 2.6.4 y 2.6.2 demostramos el enunciado.

Algoritmos genéticos

Una de las ramas más desarrolladas dentro de algoritmos evolutivos son los algoritmos genéticos. Los GAs son un tipo de algoritmo evolutivo pensado inicialmente para trabajar con soluciones representadas mediante cadenas binarias, denominadas cromosomas. No obstante, a lo largo de los años, se han usado otras representaciones no binarias, como permutaciones [181], vectores de enteros [79], reales [116] e incluso estructuras de datos complejas [12] y muy particulares de problemas determinados [209].

La principal característica de los GAs es el uso de un operador de recombinación o cruce como mecanismo principal de búsqueda: construye descendientes que poseen características de los cromosomas que se cruzan. Su utilidad viene dada por la suposición de que diferentes partes de la solución óptima pueden ser descubiertas independientemente y luego ser combinadas para formar mejores soluciones. Adicionalmente, se emplea un operador de mutación cuyo uso se considera importante como responsable del mantenimiento de la diversidad. El principal problema de los GAs se debe a que no usan información sobre el problema que están intentando resolver.

En concreto, en esta tesis nos hemos centrado en los algoritmos genéticos de estado estacionario (ssGA) con ($\lambda = 1$), cuyo pseudocódigo se presenta en el Algoritmo 2. Esta decisión tiene que ver con una característica fundamental de este tipo de algoritmos que los hacen más adecuados que los GAs generacionales. Es bien conocido dentro del campo de los GAs que la selección por estado estacionario permite al algoritmo converger más rápidamente que el esquema generacional [194, 195]. De hecho, en [194] se demuestra que el ssGA evoluciona con la misma dinámica que un GA generacional, utilizando la mitad del costo computacional. Como estamos abordando problemas del mundo real que conllevan tiempos de evaluación muy altos para la función de *fitness*, este esquema de selección

Algoritmo 2 Pseudocódigo de un ssGA

```
t ← 0;
P(t) ←generarPoblaciónInicial();
evaluar(P(t));
while not(condición de terminación) do
   padres ← seleccionarPadres(P(t));
   hijo ← recombinar(padres);
   hijo ← mutar(hijo);
   evaluar(hijo);
   P(t) ←seleccionarNuevaPoblación(P(t), hijo)
   t ← t + 1;
end while
return la mejor solución encontrada
```

permitirá reducir considerablemente los tiempos de cómputo.

El criterio de reemplazo en un ssGA cobra una mayor importancia que en un algoritmo evolutivo generacional. El criterio de reemplazo puede ser aleatorio, proporcional al *fitness* o elitista. Normalmente se trabaja en hipótesis de competencia entre el descendiente generado y el individuo a reemplazar. Este mecanismo de reproducción paso a paso fue inicialmente propuesto como un modo de evitar inconvenientes en la resolución de problemas de entrenamiento de redes neuronales. Luego se extendió su uso para una variedad de problemas y áreas de aplicación.

El modelo de evolución de estado estacionario consiste en una interesante propuesta para mantener el equilibrio entre los mecanismos de exploración y de explotación del algoritmo evolutivo. El algoritmo evolutivo mantiene las características poblacionales (garantizando la exploración) y a la vez realiza una explotación local del tipo *hill-climbing* que presiona hacia los individuos mejor adaptados. Las técnicas de selección elitistas (elitismo propiamente dicho o selección por *ranking*) son usualmente utilizadas en los algoritmos evolutivos de estado estacionario. La estrategia de reemplazar al peor individuo usualmente conduce a convergencia prematura. La estrategia de reemplazo aleatorio funciona correctamente en general, aunque la convergencia es lenta. Para disminuir los efectos adversos puede utilizarse reemplazo probabilístico, inversamente proporcional a los valores de *fitness*.

La mayor diferencia entre el modelo de estado estacionario y el modelo generacional

consiste en que para cada x miembros de la población creados, el modelo de estado estacionario necesita realizar $2x$ selecciones. Como consecuencia, la presión de selección y la "deriva genética" (pérdida de información) será el doble en un algoritmo evolutivo de estado estacionario. El algoritmo de estado estacionario será como mínimo mucho más veloz para converger que el algoritmo generacional [194, 195]. En [194] se demuestra que el GA de estado estacionario evoluciona con la misma dinámica que un GA generacional utilizando la mitad del coste computacional.

2.6.2. Optimización basada en colonias de hormigas

Los algoritmos basados en colonias de hormigas son modelos inspirados en el comportamiento de colonias de hormigas reales. Estudios realizados explican cómo animales casi ciegos, como son las hormigas, son capaces de seguir la ruta más corta en su camino de ida y vuelta entre la colonia y una fuente de abastecimiento [104]. Esto es debido a que las hormigas pueden "transmitirse información" entre ellas gracias a que cada una de ellas, al desplazarse, va dejando un rastro de una sustancia llamada *feromona* a lo largo del camino seguido. Así, mientras una hormiga aislada se mueve de forma esencialmente aleatoria, los "agentes" de una colonia de hormigas detectan el rastro de feromona dejado por otras hormigas y tienden a seguir dicho rastro. Éstas a su vez, van dejando su propia feromona a lo largo del camino recorrido y, por tanto, lo hacen más atractivo, puesto que se ha reforzado el rastro de feromona. Sin embargo, la feromona también se va evaporando con el paso del tiempo provocando que el rastro de feromona sufra, por otro lado, cierto debilitamiento. En definitiva, puede decirse que el proceso se caracteriza por una retroalimentación positiva, en la que la probabilidad con la que una hormiga escoge un camino aumenta con el número de hormigas que previamente hayan elegido el mismo camino.

Un sistema de hormigas utiliza un grafo ponderado $G = (N, A)$, donde N es el conjunto de componentes (nodos) y A es el conjunto de aristas que conectan el conjunto de componentes N. Los estados δ (y por tanto las soluciones S) se corresponden con caminos en el grafo, esto es secuencias de nodos o aristas. Las aristas del grafo son conexiones/transiciones que definen la estructura del vecindario. Cada arista (r, s) tiene una medida de deseabilidad

$\tau(r, s)$, llamada *feromona*, la cual es actualizada en tiempo de ejecución por las hormigas artificiales (hormigas de ahora en más). Los rastros de feromonas representan un tipo de memoria indirecta y a largo plazo del proceso de búsqueda. Además cada componente o conexión debe tener asociado valores heurísticos η, que representan la información heurística disponible del problema que resolver.

El primer algoritmo basado en la optimización mediante colonias de hormigas fue aplicado al problema del viajante de comercio[73], obteniéndose resultados bastante alentadores. A partir de dicho algoritmo se han desarrollado diversos heurísticos que incluyen varias mejoras y han sido aplicados no sólo a TSP sino también a problemas como VRP y QAP, entre otros [75]. En la literatura se han propuesto diversos algoritmos que siguen la metaheurística ACO. Entre los algoritmos de ACO para problemas de optimización combinatoria $\mathcal{NP}$-duros, se encuentran el sistema de hormigas o *Ant System* (AS) [74], el sistema de colonia de hormigas o *Ant Colony System* (ACS) [73], el sistema de hormigas Max-Min o *Max-Min Ant System* $\mathcal{MM}$AS [204], el AS con ordenación o *Rank-Based Ant System* (RBAS)[41] y el sistema de la mejor-peor hormiga o *Best-Worst Ant System* (BWAS) [57, 58]. Por mayor información se puede consultar el libro de Dorigo y Stützle [76] para una descripción de todas estas variantes de ACO. AS presenta un interés histórico ya que fue el primer algoritmo ACO, los otros cuatro normalmente alcanzan resultados mucho mejores. En lo que sigue, presentaremos una descripción del algoritmo ACS, por ser la variante ACO usada en el presente trabajo de tesis.

El modo de operación básico de un algoritmo ACS es como el que se muestra en el Algoritmo 3. Las μ hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los estados adyacentes del problema, representado por el grafo G. Este movimiento se realiza siguiendo una *regla de transición* basada en la información local disponible en los nodos. Esta información local incluye la información heurística y memorística (rastros de feromona) para guiar la búsqueda. Al moverse por el grafo, las hormigas pueden depositar feromona cada vez que crucen un arco (conexión) mientras que construyen la solución (actualización en línea paso a paso de los rastros de feromonas - actualización local). Una vez que cada hormiga ha generado una solución, tras un proceso

Algoritmo 3 Pseudocódigo de un algoritmo ACS

```
InicializarValoresFeromonas(τ); {Inicializa los rastros de feromona}
ant^bs=generarSolución(τ, η){Se inicializa ant^bs con una solución inicial aleatoria}
while not(condición de terminación) do
  for j ← 1 to μ do
    ant^i=ConstruirSolución(τ, η);{La hormiga ant^i construye una solución}
    ActualizarFeromonaLocal(τ,ant^j); {Actualización local de los rastros de feromona (ACS)}
  end for
  evaporarFeromonas(τ) {evaporación}
  actualizarFeromonasGlobal(τ,ant,ant^bs) {intensificación, actividad del demonio}
  for j ← 1 to μ do
    if f(ant^i) < f(ant^bs) then
      ant^bs=ant^i {Actualizar la mejor solución, actividad del demonio}
    end if
  end for
end while
return la mejor solución encontrada
```

de evaluación de la solución generada, puede depositar una cantidad de feromona que es función de la calidad de su solución (actualización en línea de los rastros de feromonas). Esta información guiará la búsqueda de las otras hormigas de la colonia en el futuro.

Además, el modo de operación genérico de un algoritmo ACS incluye dos procedimientos adicionales: la evaporación de los rastros de feromonas y las acciones del proceso auxiliar (*daemon* en Inglés). La evaporación de feromona se usa como un mecanismo que evita un estancamiento en la búsqueda y permite que las hormigas busquen o exploren nuevas regiones del espacio. Las acciones del proceso auxiliar son acciones opcionales –que no tienen un contrapunto natural– para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen. Ejemplos son observar la calidad de todas las soluciones generadas y depositar una nueva cantidad de feromona adicional sólo en las transiciones/componentes asociadas a algunas soluciones. Otro ejemplo es aplicar un procedimiento de búsqueda local a las soluciones generadas por las hormigas antes de actualizar los rastros de feromona. En ambos casos, el demonio reemplaza la actualización en línea a posteriori de feromona y el proceso pasa a llamarse actualización fuera de línea de rastros de feromona.

Regla de Transición de estados

En ACS la regla de transición de estados es como sigue: una hormiga se posiciona en el nodo $r \in N_k$ y elige el nodo s como próximo movimiento al aplicar la regla dada por la Ecuación 2.6.12.

$$s = \begin{cases} \text{máx}_{u \in N_k(r)} \left\{ [\tau(r,u)]^\alpha \times [\eta(r,u]^\beta \right\} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{2.6.12}$$

donde q es un número aleatorio uniformemente distribuido en [0..1], q_0 es un parámetro del algoritmo ($0 \leq q_0 \leq 1$) y S es una variable aleatoria seleccionada de acuerdo a la distribución de probabilidades dada en la Ecuación 2.6.13.

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]^\alpha \times [\eta(r,s)]^\beta}{\sum_{u \in N_k(r)} [\tau(r,u)]^\alpha \times [\eta(r,u]^\beta} & \text{si } s \in N_k(r) \\ 0 & \text{en otro caso} \end{cases} \tag{2.6.13}$$

donde τ es la feromona, η es la información heurística, $N_k(r)$ es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo r (es decir, el conjunto de componentes que la hormiga k no ha incorporado aún a la solución) y α y $\beta \in \mathbf{R}$ son dos parámetros que ponderan la importancia relativa de los rastros de feromona y la información heurística. En particular en un ACS α toma el valor 1 y por consiguiente no se considera dicho parámetro. Cada hormiga k almacena la secuencia que ha seguido hasta el momento y su memoria L_k se utiliza para determinar $N_k(r)$ en cada paso de construcción.

Volviendo al parámetro β, su función es la que sigue. Si $\beta = 0$, sólo se tienen en cuenta los rastros de feromona para guiar el proceso constructivo, lo que puede causar un rápido estancamiento, esto es, una situación en la que los rastros de feromona asociados a una solución son ligeramente superiores que el resto, provocando por tanto que las hormigas siempre construyan las mismas soluciones, normalmente óptimos locales.

La regla de transición de estados resultante de la Ecuación 2.6.12 y 2.6.13 es llamada regla de transición proporcional pseudo-aleatoria. Como puede observarse, la regla tiene una doble intención: cuando $q \leq q_0$, explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y los rastros de feromona. Sin embargo, si

$q > q_0$ se aplica una exploración controlada. En resumen, la regla establece un compromiso entre la exploración de nuevas conexiones y la explotación de la información disponible en ese momento.

Regla de actualización global

En ACS se realiza una actualización fuera de línea de los rastros de feromona. Para llevarla a cabo, sólo se considera una hormiga concreta para depositar feromonas: la que generó la mejor solución global $S_{mejor-global}$. Esta elección, junto con el uso de la regla proporcional pseudo-aleatoria, pretende hacer la búsqueda más directa. La actualización global se realiza tras que todas las hormigas han completado su solución. El nivel de feromona se actualiza aplicando la regla de la Ecuación 2.6.14.

$$\tau(r,s) = (1-\rho) \times \tau(r,s) + \rho \times \Delta\tau_{rs}^{k}, \qquad \forall r,s \in S_{mejor-global} \tag{2.6.14}$$

donde $\rho \in (0,1]$ es la tasa de evaporación y $\Delta\tau_{rs}^{k} = f(S_{mejor-global})$, es decir, la cantidad de feromona que se deposita depende de la calidad $S_{mejor-global}$ de la mejor solución. Como puede verse, la regla de actualización incluye tanto la evaporación de feromona como el depósito de la misma.

Regla de actualización local

Mientras una hormiga construye una solución, ésta aplica una actualización en línea paso a paso de los rastros de feromonas, que favorece la generación de soluciones distintas a las ya encontradas. Cada vez que una hormiga viaja por una arista a_{rs}, aplica la regla descrita en la Ecuación 2.6.15.

$$\tau(r,s) = (1-\varphi) \times \tau(r,s) + \varphi \times \tau_0 \tag{2.6.15}$$

donde $\varphi \in (0,1]$ es un segundo parámetro de decremento de feromona. Ya que la cantidad de feromona depositada es muy pequeña. De hecho, τ_0 es el valor del rastro de feromona inicial y se escoge de tal manera que en la práctica se corresponda con el límite inferior del rastro de feromona, es decir, con la elección de las reglas de actualización de feromona del ACS

ningún rastro de feromona puede caer por debajo de τ_0. La aplicación de esta regla hace que los rastros de feromona entre las conexiones recorridas por las hormigas disminuyan. Así, esto lleva a una técnica de exploración adicional de ACS, ya que las conexiones atravesadas por un gran número de hormigas son cada vez menos atractivas para el resto de hormigas que las recorren en la iteración actual, lo que ayuda claramente a que no todas las hormigas sigan el mismo camino.

Lo más significativo de la formación de esta metaheurística siguiendo la Definición 2.2.1 es que $\mu_{ACO} = 0$ y existe una variable de estado de la metaheurística, $\mathcal{P} \in first(\Xi_{ACO})$ que representará la matriz de feromona y la heurística [51]. Llamaremos $D_{\mathcal{P}}$ al dominio de esta variable. Este dominio está formado por todos los posibles grafos de construcción con arcos y/o nodos ponderados con valores reales para representar los rastros de feromonas y, opcionalmente, heurística. En este caso, las función Φ_{ACO} tendrá la siguiente forma:

$$\Phi_{ACO} : D_{\mathcal{P}} \times \prod_{i=2}^{v} D_i \rightarrow \mathcal{T}^{\lambda} \tag{2.6.16}$$

donde hemos supuesto que la variable $\xi_1 = \mathcal{P}$. También tiene especial importancia en esta metaheurística la función $\mathcal{U}_{ACO}$, que realiza la actualización de los rastros de feromona.

2.6.3. Comparación de las metaheurísticas

En esta sección se presenta una comparación conceptual de algoritmos evolutivos y algoritmos basados en colonia de hormigas, lo cual puede ayudar en el entendimiento de los detalles acerca de los algoritmos usados en este trabajo, cuyos particularidades para resolver el problema de empaquetado se presentan en los capítulos siguientes.

La característica más relevante que comparten las metaheurísticas EAs y ACO es la siguiente: son enfoques basados en población, como ya se mencionó en este capítulo. Sin embargo, la "población" que cada una de ellas procesa y la forma en la cual es tratada es muy diferente.

Como enfoque basado en instancia, los EAs manejan una población de individuos "competitivos" que representan un conjunto de posibles soluciones completas al problema planteado. En consecuencia, los distintos operadores se deben definir para explorar el espacio

de búsqueda del problema para conseguir soluciones de buena calidad. Sin embargo, los algoritmos ACO son representativos de un enfoque basado en modelo [231]. Su población, en lugar de representaciones de soluciones, se compone de un conjunto de hormigas cooperativas (agentes podría ser un término alternativo) que iterativamente construyen una solución paso a paso, es decir, comienzan con una solución vacía y agregan componentes del problema, uno a la vez, para crear una solución completa al mismo.

Por ejemplo, el problema de empaquetado estudiado en esta tesis acepta, como se describe más tarde, una permutación de número naturales como una solución posible. Por lo tanto, con una instancia del problema de dimensión M, el tamaño del espacio de búsqueda para este problema es de $M!$ (suponiendo una permutación de representación de las soluciones).

Primero, analizamos la exploración del espacio de búsqueda desde la perspectiva del EAs. La siguiente expresión muestra un panorama general del proceso evolutivo iterativo seguido por un EA, cuando se explora el espacio de búsqueda:

$$P(0) \quad \stackrel{\mathcal{E}}{\rightarrow} \quad P(1) \quad \stackrel{\mathcal{E}}{\rightarrow} \quad \cdots \quad \stackrel{\mathcal{E}}{\rightarrow} \quad P(t_{max}) \tag{2.6.17}$$

donde $P(t) \subset S, t = 0, 1, \ldots, t_{max}$, representa la población en la generación t, t_{max} es la cantidad máxima de iteraciones y S espacio de búsqueda completo de todas las posibles permutaciones de tamaño M. Las sucesivas poblaciones se obtienen al aplicar una transformación $\mathcal{E}$. Esta transformación se puede concebir como una composición de varios operadores genéticos específicos orientados en la exploración del espacio de búsqueda, con $\mathcal{E} : 2^S \rightarrow 2^S$ donde $\mathcal{E}(P(t)) = P(t+1)$. La buena combinación de material genético de las diferentes soluciones determina la construcción de bloques constructivos del enfoque evolutivo para lograr soluciones de mayor calidad mediante la exploración del espacio de búsqueda.

La situación descrita en el párrafo anterior es diferente cuando se considera un algoritmo ACO general, puesto que consiste de un proceso iterativo que involucra una población de hormigas $\mathcal{A}$ que paso a paso construyen un conjunto de soluciones (digamos $P(t)$ con $t =$

$0, 1, \ldots, t_{max}$), de acuerdo a los valores de feromonas ($\tau(t)$) y la información heurística (η):

$$\begin{array}{ccccccccc} (\mathcal{A}, \tau(0), \eta) & \rightarrow & (\mathcal{A}, \tau(1), \eta) & \rightarrow & \cdots & \rightarrow & (\mathcal{A}, \tau(t_{max}), \eta) \\ \downarrow & \overset{\mathcal{U}}{\nearrow} & \downarrow & \overset{\mathcal{U}}{\nearrow} & & \overset{\mathcal{U}}{\nearrow} & \downarrow \\ P(0) & & P(1) & & \cdots & & P(t_{max}) \end{array} \tag{2.6.18}$$

En la secuencia anterior, $\mathcal{U}$ representa el proceso de actualización sobre la estructura τ la cual representa los bloques de construcción globalmente compartidos por la colonia. Es importante resaltar que los valores de η dependen, en algunos casos, tanto de la iteración en la cual se encuentra el proceso evolutivo como de los valores de feromona. En este caso, la expresión 2.6.18 se debería modificar cambiando η por $\eta(t)$ para $t = 0, 1, \ldots, t_{max}$. Además, se puede observar que el conjunto de hormigas $\mathcal{A}$ es la mismo durante todo el proceso de búsqueda. La actualización del rastro de feromona $\mathcal{U}$ se puede definir como $\mathcal{U} : 2^{\tau} \times 2^{S} \rightarrow 2^{\tau}$ tal que $\mathcal{U}(\tau(t), P(t)) = \tau(t+1)$.

Sin embargo, el proceso de generación de soluciones, tanto en un algoritmo evolutivo como en uno de hormigas, es diferente cuando se hibrida con, por ejemplo, un procedimiento de búsqueda local, como ocurre en nuestro trabajo. En general, el valor añadido del procedimiento de búsqueda local es aprovechar las buenas soluciones encontradas por el algoritmo de base al explorar el espacio de búsqueda, y al mismo tiempo de su aplicación, cuando sea posible, con una retroalimentación positiva sobre la exploración de regiones prometedoras del espacio de búsqueda. Por lo general, la búsqueda local (indicado aquí por $\mathcal{L}$) se aplica entre dos iteraciones consecutivas. Así, las secuencias en la Ecuación 2.6.17 y 2.6.18 se cambian, respectivamente, por:

$$P(0) \overset{\mathcal{E}}{\rightarrow} \hat{P}(1) \overset{\mathcal{L}}{\rightarrow} P(1) \overset{\mathcal{E}}{\rightarrow} \hat{P}(2) \overset{\mathcal{L}}{\rightarrow} \cdots \overset{\mathcal{E}}{\rightarrow} \hat{P}(t_{max}) \overset{\mathcal{L}}{\rightarrow} P(t_{max}) \tag{2.6.19}$$

y,

$$\begin{array}{ccccccccccc} (\mathcal{A}, \tau(0), \eta) & & \rightarrow & (\mathcal{A}, \tau(1), \eta) & & \rightarrow & (\mathcal{A}, \tau(1), \eta) & \cdots & \rightarrow & (\mathcal{A}, \tau(t_{max}), \eta) \\ \downarrow & & \overset{\mathcal{U}}{\nearrow} & \downarrow & & \overset{\mathcal{U}}{\nearrow} & \downarrow & & & \downarrow \\ \hat{P}(0) & \overset{\mathcal{L}}{\rightarrow} & P(0) & \hat{P}(1) & \overset{\mathcal{L}}{\rightarrow} & P(1) & \hat{P}(2) & \ldots & & P(t_{max}) \end{array} \tag{2.6.20}$$

donde $\hat{P}(i)$ con $i = 1, \ldots, t_{max}$ representa, tanto para las opciones EA como ACO, un conjunto intermedio de soluciones sobre el que se aplica el procedimiento de búsqueda local antes de la próxima iteración del algoritmo. Por lo tanto, el proceso $\mathcal{L}$ intensifica la búsqueda en torno a algunas regiones del espacio de búsqueda y proporciona una retroalimentación para el algoritmo con la ayuda adicional cuando se diversifica la búsqueda.

2.7. Conclusiones

En este capítulo hemos realizado una introducción al campo de las metaheurísticas. Inicialmente hemos presentado una definición formal de metaheurísticas propuesta por Francisco Chicano [51] en su tesis doctoral. A continuación, hemos ofrecido un repaso por las técnicas más importantes dentro de este campo, las cuales han sido clasificadas en dos categorías, metaheurísticas basadas en trayectoria y en población, en función del número de soluciones tentativas con las que trabajan en cada iteración. Como siguiente paso, hemos introducido las metaheurísticas paralelas y realizado una breve descripción de los distintos modelos paralelos desarrollados en la literatura para metaheurísticas basadas en trayectoria y en población. A continuación hemos presentado modelos paralelos heterogéneos y describimos una taxonomía de PHM. Finalmente, hemos detallado en forma genérica los algoritmos metaheurísticos utilizados en este trabajo de tesis. Detalles de su adecuación para tratar al problema en estudio serán presentados en los capítulos correspondientes.

Capítulo 3

Problemas de corte y empaquetado

Los problemas de corte y empaquetado (C&P) son un conjunto de problemas ampliamente estudiados, los cuales son definidos como problemas combinatorios geométricos. Partiendo de unos modelos básicos, se encuentra gran cantidad de variantes derivadas de la amplia gama de aplicaciones prácticas existentes y dependiendo de quién lo esté tratando. En este capítulo, en primer lugar (Sección 3.1) presentamos una descripción de los distintos problemas de corte y empaquetado existentes y sus similitudes y diferencias. En la Sección 3.2 realizamos una clasificación de los problemas de empaquetado teniendo en cuenta sus principales características. En la Sección 3.3 nos centramos en el problema de empaquetado en dos dimensiones, para pasar en la Sección 3.4 con la descripción formal del problema de *strip packing*, objeto de estudio en este trabajo de tesis. Finalmente, la Sección 3.5 presenta las distintas instancias del problema de *strip packing* utilizadas a lo largo de la experimentación que se describirá en los capítulos siguientes.

3.1. Descripción de los problemas de C&P

Los problemas de corte y empaquetado consisten en colocar un conjunto de *elementos*, por lo general pequeños, en uno o más *objetos*, por lo general de grandes dimensiones, sin que se superpongan, de modo de minimizar/maximizar una función objetivo dada. Es un problema de optimización combinatoria con muchas aplicaciones importantes en las industrias de la madera, del vidrio, del aluminio y del cuero, además en el diseño de circuitos

integrados (Large Scale Integration –LSI y Very Large Scale Integration –VLSI), en el paginado de periódicos y en la distribución de la carga de contenedores y camiones. Por muchas décadas, el corte y empaquetado ha atraído la atención de los investigadores en varias áreas. La investigación operativa es un ámbito en el cual se han investigado y/o desarrollado métodos para resolver C&P y por otro lado, el sector financiero es un ámbito en el cual (de manera más abstracta) se pueden encontrar problemas de C&P. También se puede incluir el área de ciencias de la computación, manufactura, etc.

Los problemas de C&P se pueden clasificar usando distintos criterios. La dimensionalidad del problema es uno de los criterios y la mayoría de los problemas se definen en una, dos o tres dimensiones. En este trabajo de tesis consideramos problemas en dos dimensiones. El próximo criterio para clasificar los problemas de C&P en dos dimensiones es la figura de los elementos pequeños. Nos enfocamos principalmente en problemas regulares. Este problema ha sido estudiado en las últimas décadas. Cuando las figuras de los elementos son polígonos o tienen figuras arbitrarias, el problema es denominado irregular.

La mayoría de los problemas de C&P en dos dimensiones son conocidos como $\mathcal{NP}$-duros (ver [69] para una revisión general) y por lo tanto es imposible resolverlos en tiempo polinomial, a menos que $\mathcal{NP} = \mathcal{P}$. Por lo tanto, las heurísticas y las metaheurísticas son importantes para diseñar algoritmos prácticos para este problema.

Los problemas de C&P describen patrones que consisten de combinaciones geométricas de grandes objetos y pequeños elementos. En el caso de los problemas de empaquetado, los objetos grandes (contenedores) necesitan ser llenados con pequeños elementos (por ejemplo, cajas). Por su parte, los problemas de corte están caracterizados por grandes objetos (por ejemplo, planchas o rollos) que deben ser cortados en pequeños elementos (por ejemplo, figuras de dos dimensiones). El objetivo de los procesos de corte y empaquetado es maximizar la utilización del material, es decir, asignar todos los elementos sin superposición en un mínimo número de contenedores o planchas.

Debido al papel dominante que juegan los patrones y su naturaleza como combinaciones geométricas, se puede decir que los problemas de C&P pertenecen al campo de combinatorias geométricas. Los problemas de C&P por un lado se denominan geométricos porque

dentro de cada objeto grande se deben ordenar uno o más objetos pequeños de tal manera de evitar solapamientos y de encajar en los límites del objeto geométrico, es decir, cómo cortar los pequeños objetos. También se denominan combinatorios debido a que se deben tomar decisiones de cuáles serán los elementos que producir y de qué objeto se obtendrán. En otras palabras, a cada uno de los grandes objetos se le asigna una serie de pequeños elementos y además cada elemento se asigna como máximo a un único objeto grande. Estos dos problemas se entrelazan y las soluciones deben ser viables tanto desde el punto de vista "cuantitativo" (es decir, del número mínimo o máximo de pequeños elementos que producir y la disponibilidad de los grandes objetos) y del punto de vista geométrico (es decir, no producir solapamiento entre los pequeños elementos y la contención de los pequeños elementos dentro de los límites de los grandes objetos).

3.2. Clasificación de los problemas de C&P

En las últimas décadas, se produjo una gran investigación en problemas de C&P y ha sido ampliamente descrita en la literatura. Esos problemas surgen en muchas industrias y no están restringidos al sector de la manufactura. Por ejemplo, se pueden encontrar problemas de empaquetado de una manera más abstracta en investigación operativa y en el sector financiero.

Dada esta diversidad de problemas y áreas de aplicación, problemas de C&P similares aparecen bajo diferentes nombres en la literatura. Dickhoff [69] presenta una caracterización integrando todas las clases de problemas de corte y empaquetado, la cual generaliza la clasificación presentada por Hinxman [119] a principios de los años 80. La taxonomía propuesta por Dickhoff permite identificar propiedades comunes de ciertos problemas, los cuales parecieran no tener relación a primera vista. Por otra parte, diferencias entre problemas que aparentan ser similares surgen al analizar sus principales características. Dickhoff distingue entre problemas de C&P involucrando dimensiones espaciales y aquellos involucrando dimensiones no espaciales. El primer grupo consiste de problemas de corte y empaquetado o carga definidos en un máximo de tres dimensiones en el espacio euclideo (por ejemplo,

problemas de *cutting stock*, carga de vehículos y *pallet loading*). El otro grupo abarca los problemas de corte y empaquetado abstractos definidos en un espacio de dimensiones no espaciales, tales como peso, tiempo o dinero (por ejemplo, asignación de memoria, presupuestación del capital, cambio de moneda y balance en línea).

Esta taxonomía se basa en la estructura lógica básica de los problemas de C&P, la cual se puede determinar como sigue:

1. Existen dos grupos de datos básicos cuyos elementos definen cuerpos geométricos de formas fijas (figuras) en un espacio de una o más dimensiones de números reales:
   - materia prima, también llamada objeto.
   - lista u orden de elementos.
2. El proceso de corte o empaquetado se realiza en base a la generación óptima de patrones, los cuales son combinaciones geométricas de pequeños elementos asignados a grandes objetos, que determinan la posición de cada elemento en los grandes objetos. Los espacios residuales, es decir, figuras que ocupan lugar en los patrones pero que no pertenecen al conjunto de pequeños elementos, se tratan por lo general como desperdicio.

La clasificación de Dickhoff describe cuatro características de las más importantes de los problemas de C&P [69, 70]:

1. La característica más importante es la *dimensión*, la cual define la cantidad mínima de dimensiones necesarias para describir la geometría de los patrones (una dimensión, dos dimensiones, tres dimensiones o más dimensiones). Problemas con más de tres dimensiones se obtienen cuando se expanden a dimensiones no espaciales, como por ejemplo tiempo o peso.
2. La *clase de asignación* describe si se deben asignar todos los objetos y elementos o sólo una parte de ellos.

3. La *variedad de los objetos* distingue entre problemas que tienen los objetos de idéntica forma o diferente.

4. La *variedad de los elementos* se refieren a la forma y cantidad de los elementos. Los problemas pueden consistir de pocos elementos, elementos congruentes, muchos elementos de muchas formas diferentes y numerosos elementos de una cantidad relativamente reducida de formas diferentes.

La tipología de Dyckoff ha presentado algunas deficiencias, las cuales ha creado problemas para tratar con recientes desarrollos e impide que sean aceptados más generalmente. Recientemente, Wäscher et al. [222] han presentado una nueva tipología para los problemas de C&P, basada parcialmente en las ideas originals de Dyckhoff, pero introduce nuevos criterios para la categorización, lo que permite definir categorías de problemas en forma diferente a lo que hace Dyckoff. Información adicional sobre corte y empaquetado es disponible en el sitio web [1] del ESICUP-EURO, el cual es un grupo de interés especial sobre corte y empaquetado.

Como ha hemos mencionado, el objetivo de los problemas de C&P es la eficiente ubicación de figuras en una región contenedora sin superposición. Por lo tanto, la complejidad de los problemas de C&P está relacionada con la forma geométrica de los elementos que deben ser empaquetados o cortados. Se pueden distinguir dos tipos de formas:

- *regulares*: que se pueden describir por pocos parámetros (por ejemplo, rectángulos, círculos, etc.)

- *irregulares*: incluyendo asimetrías y concavidades.

Los problemas de C&P regulares están relacionados con el empaquetado de un conjunto de rectángulos en un objeto también rectangular. El empaquetado irregular (ver ejemplo en Figura 3.1) también es conocido como de anidación, por ejemplo en la industria naval, y como problema de diseño de marcaciones en la industria textil. En este trabajo de tesis nos

[1] ESICUP-EURO http://www.apdio.pt/esicup

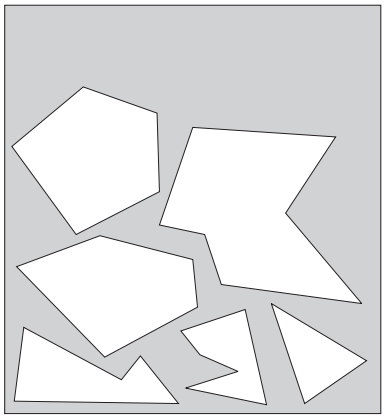


Figura 3.1: Ejemplos de patrones con piezas irregulares

enfocamos en una clase de problema regulares que describiremos en detalle más adelante en este capítulo.

En función a la geometría de los elementos a ser empaquetados se pueden distinguir dos tipos de distribuciones. En el caso de elementos regulares, los patrones pueden ser *ortogonales* (cortes que deben ser paralelos a los lados del objeto) y *no ortogonales*. En las Figuras 3.2 y 3.3 se muestran ejemplos de estos distintos tipos de cortes. A su vez, los cortes ortogonales pueden ser *guillotina* o *no guillotina*. En los primeros se deben realizar una serie de cortes paralelos a los ejes del gran objeto que atraviesen el mismo de de lado a lado; mientras que los cortes no guillotina no imponen esta restricción: un elemento se puede colocar en cualquier posición disponible, siempre que no resulte en un solapamiento de piezas. Algunos patrones deben respetar cortes guillotina pero en niveles en n-etapas, es decir por una sucesión de cortes horizontales o verticales considerando el ancho de la plancha. La restricción de corte guillotina se impone generalmente por las características tecnológicas de las máquinas de corte automatizadas, mientras que generalmente no está presente en aplicaciones de empaquetado. En la Figura 3.3(b) se muestra un ejemplo de corte guillotina. En un primer corte vertical se divide la plancha en dos partes: de una de ellas, realizando dos cortes horizontales, se obtienen las piezas 1 y 4 y el material de desperdicio. La otra parte necesita de más cortes, realizando un corte horizontal por el ancho se obtienen nuevamente dos partes: una con las piezas 3 y 7 (que se separan por un corte vertical) y con las piezas 2, 6

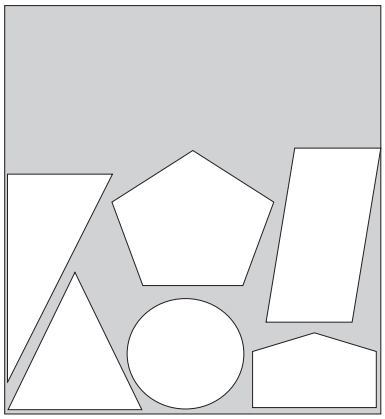
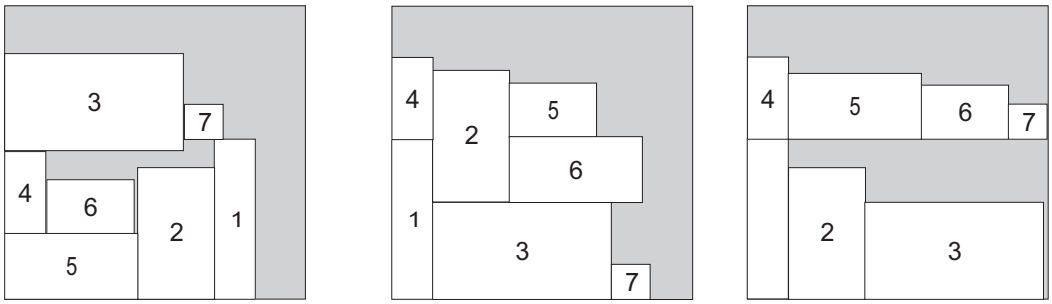


Figura 3.2: Ejemplos de patrones con cortes no ortogonales



(a) (b) (c)

Figura 3.3: Ejemplos de patrones con cortes ortogonales: (a) corte no guillotina; (b) corte guillotina; (c) corte guillotina en niveles

y 5 que para separarlas se realiza primero un corte vertical y luego uno horizontal (además son necesarios otros cortes para separar el desperdicio). En la Figura 3.3(c) se presenta un ejemplo también de cortes guillotina, pero en este caso las piezas están distribuidas formando niveles, estas piezas se obtienen realizando primero cortes horizontales de lado a lado de la plancha para obtener los distintos niveles, luego cortes verticales para obtener las distintas piezas y material de desperdicio.

En algunos problemas se asume que los elementos tienen orientación fija (es decir, no se pueden rotar) y no se impone restricción sobre el patrón de corte. En ciertos contextos reales, se permite la rotación de elementos, generalmente en 90^o, a fin de producir mejores asignaciones. Por ejemplo, la rotación no está permitida cuando los elementos son artículos

para acomodar en la página de un periódico o son piezas para cortar de planchas corrugadas o decoradas; mientras que sí se permite en el corte de materiales lisos y en la mayoría de los contextos de empaquetado.

3.3. Problemas de empaquetado rectangular

Consideremos el siguiente problema de empaquetado rectangular en dos dimensiones. Dados n elementos (rectángulos pequeños) $I = \{1, 2, \ldots, n\}$, cada uno caracterizado por su ancho w_i y su altura h_i, y uno o más objetos grandes (rectángulos). Los elementos se deben colocar en forma ortogonal sin solapamiento (los lados de cada elemento son paralelos a los lados del objeto) de modo de minimizar/maximizar una función objetivo determinada. El problema de empaquetado rectangular surge en muchas aplicaciones industriales, frecuentemente con pequeñas variaciones en las restricciones impuestas. Muchas variantes del problema se han considerado en la literatura. Las siguientes características son importantes para clasificar al problema: tipo de asignación, ordenamiento de los objetos y ordenamiento de los elementos.

A continuación, presentamos los seis tipos de problemas de empaquetado rectangular más estudiados. Por simplicidad, definimos los problemas asumiendo que cada elemento tiene una orientación fija y no se impone la restricción de corte guillotina. Es simple extender la definición para otros casos donde cada elemento puede ser rotado en 90° y/o se imponen cortes guillotina. Primero consideramos dos tipos de problemas de empaquetado típicos consistentes de un gran objeto rectangular grande y alto, el cual puede crecer en una o más dimensiones, donde se deben colocar los elementos sin superposición. Los problemas son llamados *strip packing* y *minimización de área*.

Problema de *strip packing*. Dados n elementos (pequeños rectángulos), cada uno caracterizado por su ancho w_i y su altura h_i, y un gran objeto (llamado tira o *strip*), cuyo ancho W es fijo, pero su altura H es variable. El objetivo es reducir al mínimo la altura H usada de la tira, de manera que todos los elementos puedan ser empaquetados.

Problema de minimización de área. Dados n elementos, cada uno definido por su ancho w_i y su altura h_i, y un gran objeto cuyo ancho W y altura H son variables. El objetivo es minimizar el área $W \times H$ del objeto de manera que todos los artículos pueden ser empaquetados en la tira.

En la sección 3.4 nos enfocamos principalmente en el problema de *strip packing* por ser el problema objeto de estudio en este trabajo de tesis. Este problema se puede expresar formalmente a través del siguiente programa matemático:

$$\text{minimizar } H \tag{3.3.1}$$

sujeto a:

$$0 \leq x_i \leq W - w_i \qquad \forall i \in \mathbf{N} \tag{3.3.2}$$

$$0 \leq y_i \leq H - h_i \qquad \forall i \in \mathbf{N} \tag{3.3.3}$$

y al menos se cumple una de las siguientes desigualdades para cada par i y j:

$$x_i + w_i \leq x_j \tag{3.3.4}$$

$$x_j + w_j \leq x_i \tag{3.3.5}$$

$$y_i + h_i \leq y_j \tag{3.3.6}$$

$$y_j + h_j \leq y_i \tag{3.3.7}$$

donde (x_i, y_i) son las coordenadas del borde inferior izquierdo del elemento i. Las restricciones 3.3.2 y 3.3.3 significan que todos los elementos se deben colocar en el objeto grande. Mientras que las restricciones 3.3.4 a 3.3.7 indican que los elementos no se debe superponer (es decir, cada desigualdad significa una de las cuatro ubicaciones relativas: a la izquierda de, a la derecha de, debajo de o encima de).

Otros dos problemas de empaquetado son el *bin packing en dos dimensiones* y *problema de la mochila*, en los cuales los objetos grandes tiene dimensiones fijas y se puede tener uno único o varios de ellos.

Problema de *bin packing* en dos dimensiones. Dado un conjunto de elementos, donde cada elemento i tiene un ancho w_i y una altura h_i, y una cantidad ilimitada de grandes objetos (cajas rectangulares) con idéntico ancho y alto. El objetivo es reducir al mínimo el número de cajas rectangulares utilizados para colocar todos los elementos.

Problema de la mochila en dos dimensiones. Dado un conjunto $\mathbf{I}$ de elementos, donde cada elemento $i \in \mathbf{I}$ tiene un ancho w_i, una altura h_i y un valor c_i. También se cuenta con una mochila rectangular con ancho W y alto H. El objetivo es hallar un subconjunto $\mathbf{I}' \subseteq \mathbf{I}$ con valor máximo total $\sum_{i \in \mathbf{I}'} c_i$ tal que todos los elementos $i \in \mathbf{I}'$ puedan ser empaquetados en la mochila.

Para el problema de *bin packing* en dos dimensiones, Lodi et al. [152] propone métodos heurísticos y algoritmos metaheurísticos y realizan experimentos sobre instancias de varios casos de prueba. Para el problema de la mochila en dos dimensiones, Wu et al. [223] propone algoritmos heurísticos que son efectivos para muchas instancias de prueba.

También es importante mencionar otros dos problemas: *cutting stock en dos dimensiones* y *pallet loading.*

Problema de *cutting stock* en dos dimensiones. Dado un conjunto de elementos, donde cada elemento i tiene un ancho w_i, una altura h_i y una demanda d_i, y una cantidad ilimitada de objetos de idéntico ancho y alto. El objetivo es reducir al mínimo la cantidad de objetos utilizados para colocar todos los elementos (es decir, para cada elemento i se colocan d_i copias en los distintos objetos).

Pallet loading. Dada una cantidad suficientemente grande de elementos de idéntico tamaño (w, h) y un gran objeto rectangular de tamaño (W, H), el objetivo es colocar la máxima cantidad de elementos en el objeto rectangular, en el que cada elemento se puede girar 90°.

Existen muchos estudios sobre el problema de *cutting stock* en dos dimensiones, pero el de Gilmore y Gomory [94] provee uno de los primeros métodos de solución. Los autores

proponen un esquema de generación de columnas en la cual los nuevos patrones de corte son producidos al resolver un problema de la mochila generalizado. Recientemente, se han propuesto algunos algoritmos para el problema de *cutting stock* en dos dimensiones [214, 215]. El problema de *pallet loading* con orientación fija de los elementos es un problema trivial de resolver. Morabito y Morales [168] propusieron un algoritmo simple pero efectivo para el problema de loading pallet.

3.4. Problema de empaquetado regular en dos dimensiones con restricciones: 2SPP

En esta sección explicaremos en detalle uno de los más populares problemas de empaquetado, por ser el estudiado en este trabajo de tesis. El *problema de strip packing en dos dimensiones* o *2-dimensional strip packing problem* (2SPP) está presente en muchas aplicaciones del mundo real tal como en la industria del vidrio, papel, textil, entre otras, con algunas variaciones en su formulación básica. Por ejemplo, se deben cortar varias piezas irregulares de un rollo de tela para confeccionar una prenda. Un mueble puede necesitar piezas rectangulares de vidrio, como también de madera. En todos los casos, es necesario minimizar tanto la cantidad usada como el desperdicio. Se han propuesto varias opciones en la literatura y, a nuestro entender, la revisión más completa ha sido presentada en [122]. Sin embargo, en los últimos años el interés en esta área ha crecido, también como el número de trabajos presentando nuevas opciones y mejoras a las estrategias existentes.

Generalmente, 2SPP consiste de:

- un conjunto de M piezas rectangulares $\pi_i, i \in (1, \ldots, M)$, caracterizadas por su ancho w_i y su alto h_i, y
- un gran rectángulo de ancho fijo W y altura ilimitado H, denominado *plancha*.

Se asume que $0 < w_i \leq W$ y que $h_i \geq 0$. El objetivo es hallar una distribución (*layout*) de todas las piezas en la plancha que minimice la altura H requerida de la plancha y, cuando fuese necesario, se consideren restricciones adicionales. Las piezas se deben empaquetar de manera tal que no se superpongan y que sus lados queden paralelos a los lados de la plancha

(cortes ortogonales). En función de la tipología mejorada propuesta por Wäscher et al. [222], este problema se categoriza como problema regular de dimensión abierta en dos dimensiones (2D regular ODP) con una única dimensión variable.

Adicionalmente, en el presente estudio se imponen otras restricciones en la formulación del problema: las piezas no se pueden rotar y se deben empaquetar formando patrones respetando niveles en tres etapas. En esos patrones, las piezas se empaquetan en *niveles* horizontales, paralelos a la base de la plancha. Cada nivel consiste de un conjunto de *pilas*, y cada pila consiste de un conjunto piezas que poseen el mismo ancho. Los patrones de empaquetado pueden ser siempre transformados a una forma normal al mover cada pieza a una posición inferior izquierda, de modo que el espacio libre sólo aparezca en la parte superior de la pila, a la derecha de la última pila en cada nivel y por encima del último nivel en la plancha. La Figura 3.4 muestra un ejemplo de un patrón de empaquetado factible para este problema. En este trabajo sólo consideraremos patrones de empaquetado que estén en su forma normal. Los patrones en tres niveles se usan en muchas aplicaciones reales, principalmente en las industrias metalúrgicas y de vidrio, y es por esta razón que se incorpora esta restricción en la formulación del problema

Asumimos que el orden de las piezas dentro de cada pila no afecta la factibilidad y el valor objetivo de una solución, de modo que las piezas pueden ser ordenadas de acuerdo a sus índices, de menor a mayor. Una solución puede contener un máximo de M pilas. Por convención, etiquetamos a cada pila con el índice del elemento más alto que contiene, es decir, con el índice de la pieza más alta. Similarmente, una solución puede tener un máximo de M niveles. El rótulo del nivel se corresponde con la etiqueta de la pila más alta. En el ejemplo mostrado en la Figura 3.4, las etiquetas de las pilas son 2, 1, 5, 6, 3, 8 y 7, mientras que los niveles son etiquetados en 2, 6 y 8.

En la formulación matemática presentada en la Ecuación 3.3.1, junto con sus restricciones, se contempla 2SPP desde un modo genérico. Por consiguiente a continuación presentamos el modelo matemático para 2SPP con restricciones que se aborda en este trabajo. Las formulaciones que en este trabajo se plantean están basadas en los conceptos para el problema de *bin packing* en dos dimensiones en tres etapas no restringido presentados por

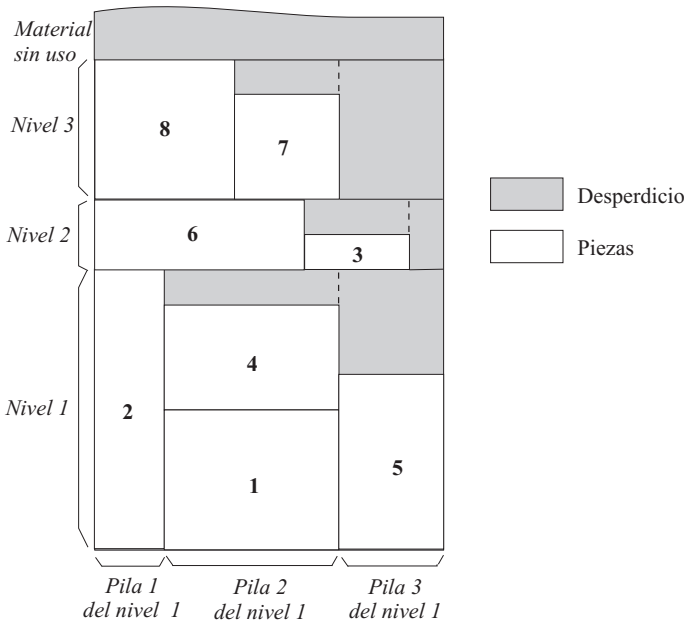


Figura 3.4: Patrón de empaquetado por niveles

Puchinger en [182]. En este caso la altura de la pila no necesariamente está dada por la pieza más alta y, además, la pila más alta no necesariamente contiene la pieza más alta del nivel.

El modelo usa las siguientes variables:

- $\alpha_{j,i} \in \{0,1\}, j = 1,\ldots,M, i = j,\ldots,M$: igual a 1 si y sólo si (sssi) la pieza i está ubicada en la pila j.
- $\beta_{k,j} \in \{0,1\}, j < k, j = 1,\ldots,M, k = 1,\ldots,M$: igual a 1 si y sólo si (sssi) la pila j está ubicada en el nivel k.
- $\gamma_k \in \{0,1\}, k = 1,\ldots,M$: igual a 1 si y sólo si (sssi) $\beta_{k,k} = 1$.
- $\delta_{i,j} \in \{0,1\}, j = 2,\ldots,M, i = j,\ldots,M$: igual a 1 si y sólo si (sssi) la pieza i contribuye a la altura total del nivel y está contenida en la pila j; es decir, la pieza i está contenida en la pila j, la pila j está contenida en el nivel j (y por lo tanto define su altura). Simplemente se puede decir:

$$\delta_{i,j} = 1 \leftrightarrow \alpha_{j,i} = 1 \wedge \gamma_j = 1 \tag{3.4.1}$$

Habiendo definido las variables involucradas, la formulación matemática propuesta para 2SPP en tres etapas, que se aborda en este trabajo de tesis, es la siguiente:

$$\text{minimizar } \sum_{i=1}^{M} h_i \gamma_i + \sum_{i=1}^{M} h_i \sum_{j=1}^{i-1} \delta_{i,j} \tag{3.4.2}$$

sujeto a:

$$\sum_{j=1}^{i} \alpha_{j,i} = 1, \qquad \forall i = 1, \ldots, M \tag{3.4.3}$$

$$\sum_{i=j+1}^{M} \alpha_{j,i} \leq (M-j)\alpha_{j,j}, \qquad \forall j = 1, \ldots, M-1 \tag{3.4.4}$$

$$\alpha_{j,i} = 0, \qquad \forall j = 1, \ldots, M-1, \forall i > j | w_i \neq w_j \tag{3.4.5}$$

$$\sum_{k=1}^{M} \beta_{k,j} = \alpha_{j,j}, \qquad \forall j = 1, \ldots, M \tag{3.4.6}$$

$$\sum_{i=j}^{M} h_i \alpha_{j,i} < \sum_{i=k}^{M} h_i \alpha_{k,i}, \qquad \forall k = 2, \ldots, M; \forall j = 1, \ldots, k-1 \tag{3.4.7}$$

$$\sum_{i=j}^{M} h_i \alpha_{j,i} \leq \sum_{i=k}^{M} h_i \alpha_{k,i}, \qquad \forall k = 1, \ldots, M-1; \forall j = k+1, \ldots, M \tag{3.4.8}$$

$$\sum_{j=1}^{M} w_j \beta_{k,j} \leq W \beta_{k,k}, \qquad \forall k = 2, \ldots, M, \forall k = 1, \ldots, M \tag{3.4.9}$$

$$\alpha_{j,i} + \gamma_j - 1 \leq \delta_{i,j} \leq (\alpha_{j,i} + \gamma_j)/2, \qquad \forall i = 2, \ldots, M; \forall j = 1, \ldots, i-1 \tag{3.4.10}$$

$$\alpha_{j,i} \in \{0,1\}, \qquad j = 1, \ldots, M; i = j, \ldots, M \tag{3.4.11}$$

$$\beta_{k,j} \in \{0,1\}, \qquad k = 1, \ldots, M; j = 1, \ldots, M \tag{3.4.12}$$

$$\gamma_k \in \{0,1\}, \qquad k = 1, \ldots, M \tag{3.4.13}$$

$$\delta_{i,j} \in \{0,1\}, \qquad i = 2, \ldots, M; j = 1, \ldots, i-1 \tag{3.4.14}$$

La función objetivo (Ecuación 3.4.2) minimiza la altura de la tira de material. La Ecuación 3.4.3 asegura que cada elemento i ha sido empaquetado una sola vez. La Ecuación 3.4.4

se usa para garantizar que los elementos se asignen a una pila j que esté en uso. Como las piezas empaquetadas en una misma pila deben tener anchos idénticos se introduce 3.4.5. En la implementación actual es sólo necesario considerar las variables $\alpha_{i,j}$ para las cuales $w_i = w_j$. Sin embargo, aquí mantenemos todas las variables por cuestiones de claridad. Cada pila usada j, es decir, la pila j conteniendo a la pieza i así $\alpha_{j,i} = 1$, es empaquetada una sola vez en el nivel k de acuerdo a la Ecuación 3.4.6. Las Ecuaciones 3.4.7 y 3.4.8 aseguran que la altura de cada pila j, es decir, la altura total de todos las piezas contenidas en la pila j, nunca exceda la altura del nivel k asociado. Dividimos esas restricciones en restricciones con "menor estricto" y con "menor o igual que" a fin de evitar ambigüedades cuando las pilas tienen alturas idénticas: la pila más alta con el índice más pequeño determina el índice k del nivel. Las restricciones 3.4.6, 3.4.7 y 3.4.8 en forma conjunta aseguran que las piezas no se empaqueten en pilas sin uso. La restricción 3.4.9 garantiza que el ancho W de la tira no es excedido por algún nivel k y que ninguna pila está empaquetada en un nivel sin uso ($\beta_{k,k} = 0$). Las desigualdades 3.4.10 fuerza a las variables $\gamma - i, j$ que se establezcan los valores destinados de acuerdo con la definición en 3.4.1.

3.5. Casos de estudio - instancias abordadas

Las instancias abordadas en este trabajo se obtuvieron de una implementación propia de un generador de datos, siguiendo las ideas propuestas por [219]. Este generador es una rutina recursiva para construir un conjunto de datos constituido por rectángulos, los cuales pueden ser empaquetados en una región rectangular sin generar áreas desperdiciadas. Este procedimiento permite a los usuarios especificar un rango de variación tanto en las dimensiones y áreas de los rectángulos generados, es decir, permite controlar la relación entre la altura y el ancho de los rectángulos resultantes.

Como primer paso en el proceso de generación de los rectángulos, se solicita al usuario el ancho W y el alto H de la plancha de material, la cantidad de rectángulos deseada M y la relación de aspecto de los rectángulos. La instancia es entonces generada al subdividir el rectángulo de dimensiones $W \times H$ en pequeños rectángulos satisfaciendo ciertas relaciones

de forma especificadas. El generar instancias de esta forma presenta una ventaja esencial: conocer la altura H de la plancha implica conocer el óptimo *a priori*. Por otro lado, las instancias se corresponden con una clase especial de 2SPP ya que respeta empaquetados guillotina, pero es importante resaltar que no son patrones por niveles (restricción impuesta a lo largo del presente trabajo). Los conjuntos de datos generados contiene rectángulos cuyas alturas y anchos son número enteros.

Los aspectos que analizaremos de las instancias generadas se describen a continuación. Como primer paso debemos introducir una notación para una instancia del problema dada. Sea M la cantidad de piezas rectangulares. Consideramos para cada rectángulo i ($i = 1, \ldots, M$) las siguientes medidas: $mind_i$ es la dimensión del lado más pequeño, $maxd_i$ es la dimensión del lado más grande y finalmente a_i es el área del rectángulo. Indicamos como *tipos* la cantidad de todos los tipos de elementos, donde dos elementos pertenecen al mismo tipo si las dimensiones de sus lados son iguales. Finalmente, *promd* denota el valor medio de las dimensiones de todos los lados. La *relación de aspecto* de un rectángulo i se define como $\rho_i = maxd_i/mind_i$ para $i = 1, \ldots, M$. La *relación de área* de un par ordenado de rectángulos i, j está dada por $\gamma_{i,j} = a_i/a_j$ para $i, j = 1, \ldots, M; i \neq j$. Los factores considerados son [35]:

- la *máxima relación de aspecto* de todos los rectángulos de una instancia es $\rho = max\{\rho_i | i = 1, \ldots M\}$.
- la *máxima relación de área* de todos los pares de rectángulos de una instancia es $\gamma = max\{\gamma_{i,j} | i, j = 1, \ldots, M; i \neq j\}$.
- la *relación de heterogeneidad* es $v = tipos/M$.
- la *relación entre anchos* es $\delta = W/promd$.

En las instancias generadas el algoritmo produce un conjunto de rectángulos cuyas relaciones de aspecto caen dentro del rango $[1/\rho, \rho]$ donde $\rho = 3$. Consideramos cinco instancias generadas del problema con M igual a 100, 150, 200, 250 y 300 piezas. Las dimensiones de la plancha desde donde se obtienen las distintas piezas por cortes guillotina sucesivos son: W=100 y H=200. Por consiguiente como se conoce la altura de la plancha también

Tabla 3.1: Estadísticas de las dimensiones de los rectángulos

Inst.	**ancho**			**altura**		
	max	***min***	***prom***	***max***	***min***	***prom***
100	100	1	11,14	48	1	9,91
150	33	1	9,90	46	1	9,73
200	43	1	8,24	41	1	7,63
250	47	1	6,78	45	1	6,80
300	50	1	5,95	68	1	6,10

Tabla 3.2: Factores de las instancias

Inst.	ρ	γ	v	δ
100	3,00	3500,00	0,67	9,50
150	3,00	575,00	0,56	21,77
200	3,00	1240,00	0,41	26,25
250	3,00	1890,00	0,34	28,64
300	3,00	3400,00	0,28	16,60

se conoce el valor óptimo de las instancias, el cual en este caso se corresponde con el valor 200, es decir, la mínima altura de la plancha necesaria para ubicar todas las piezas sin desperdicio. Las instancias se encuentran públicamente disponibles [2].

Las estadísticas de las medidas de los lados de los rectángulos generados se muestran el la Tabla 3.1. Se detallan valores máximos, mínimos y promedios tanto para la altura como para el ancho de los rectángulos. Se observa una alta variación tanto en el ancho como en la altura de las rectángulos generados en cada instancia. La Figura 3.5 muestra los rectángulos para cada una de las instancias, la mayoría de ellas presenta rectángulos con dimensiones entre 14 y 20 unidades, principalmente la instancia con M=300. Con esto se arriba a que existen muchos rectángulos que pertenecen al mismo tipo. Esto permitirá el apilamiento de rectángulos durante el proceso de empaquetado.

La Tabla 3.2 muestra las características de las instancias abordadas en cuanto a las medidas descritas anteriormente. El generador utilizado tiene en cuenta el primer factor ρ. Como se puede ver, las instancias muestran distintos valores de heterogeneidad, la cual disminuye con el tamaño de la instancia, esto sugiere que la cantidad de piezas que pertenecen a un mismo tipo disminuye con la dimensión de la instancia. Por consiguiente, no sólo la instancia se torna difícil por la cantidad de piezas sino también porque la variedad de los

[2] http://mdk.ing.unlpam.edu.ar/~lisi/2spp.htm

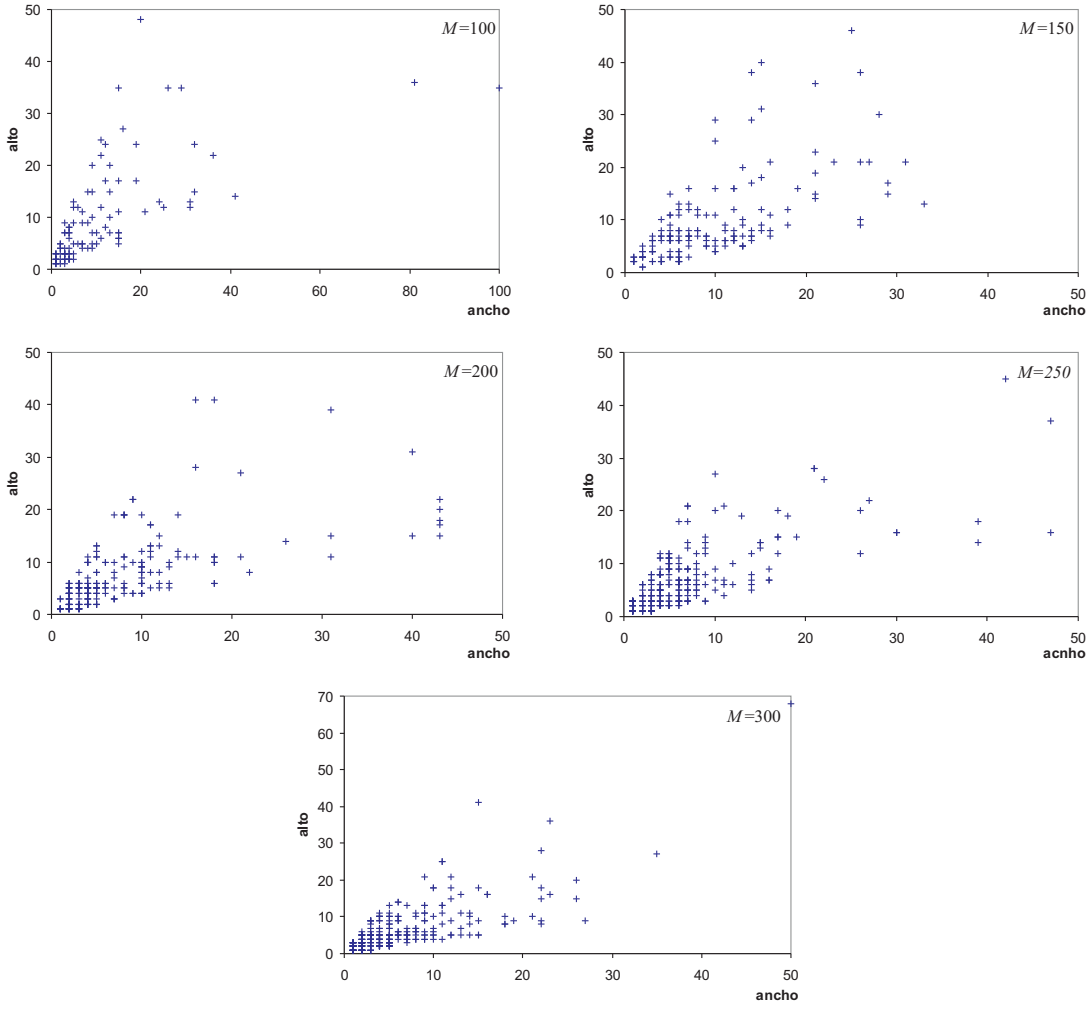


Figura 3.5: Distribución de altura y anchura de los rectángulos de las instancias generadas

rectángulos se incrementa. Por otra parte, la relación de ancho que presentan las distintas instancias también es variable y no está relacionada con la dimensión de la instancia. En cuanto a la máxima relación de área, se observa que las instancias con M=100 y M=300 presentan valores muy similares, γ=3500 y γ=3400, respectivamente. Esto indica que poseen algunos rectángulos con dimensiones muy disímiles.

3.6. Conclusiones

En este capítulo hemos revisado los problemas de corte y empaquetado, dando sus características. El tratamiento se ha centrado en los problemas de empaquetado regular, donde hemos caracterizado a aquellos problemas que tienen mayor difusión y tratamiento en la literatura. El objetivo ha sido introducir el problema de *strip packing* abordado en esta tesis, así como dar una breve revisión del mismo. También hemos presentado las instancias del problema con las cuales se realizará la experimentación a fin de obtener algoritmos que resuelvan el *strip packing* de una manera eficiente.

Capítulo 4

Métodos de solución para 2SPP

El problema 2SPP es $\mathcal{NP}$-duro y por lo tanto resolverlo en tiempo polinomial podría ser una tarea imposible. De esta manera, las heurísticas y metaheurísticas son muy importantes para diseñar algoritmos para esta clase de problemas. Muchas opciones se han propuesto en la literatura y, a nuestro entender, la revisión más completa ha sido presentada por Hopper en su trabajo doctoral del año 2000 [122]. Sin embargo, en los últimos años el interés en esta clase de problemas se ha incrementado, tanto como la cantidad de trabajos presentando nuevas opciones y mejoras a las estrategias existentes. Más recientemente Riff et al. [193] reven algunos resultados. La Tabla 4.1 provee una enumeración de las revisiones y estudios en el área del problema de C&P. En este capítulo ponemos nuestra atención sobre distintos métodos que se encuentran en la literatura para resolver algún tipo de problema de *strip packing* en dos dimensiones, tomando como base para realizar la reseña el trabajo de Hopper [122]. La organización del capítulo es la siguiente. En la Sección 4.1 presentamos los distintos métodos heurísticos propuestos para resolver este problema. En la Sección 4.2 revemos los métodos metaheurísticos que se han encontrado en la literatura para resolver algún tipo de problema de *strip packing*. Esta sección está organizada en subsecciones, donde en cada una se trata una metaheurística en particular.

Tabla 4.1: Revisiones sobre trabajos de C&P en la literatura

Autores	**Años**	**Problemas**
Haessler y Sweeney [108]	1991	problemas de corte de 1D y 2D
Dyckhoff y Finke [70]	1992	análisis de una gran variedad de problemas. Clasificación de Dyckhoff
Sweeney y Paternoster [205]	1992	revisión de más de 400 problemas
Dowsland y Dowsland [78]	1992	problemas de empaquetado en 2D, principalmente regulares
Hopper y Turton [123]	1997	problemas de empaquetado de 2 y 3 dimensiones y algoritmos genéticos
Hopper y Turton [122, 126]	2000-2001	problemas de *strip packing* y metaheurísticas
Lodi et al. [151]	2002	problemas de packing de 2D y metaheurísticas
Wäscher et al. [222]	2007	problemas de C&P y nueva tipología
Riff et al. [193]	2009	problemas de *strip packing*, heurísticas y metaheurísticas

4.1. Procedimientos heurísticos para resolver 2SPP

En esta sección repasamos los algoritmos heurísticos más populares a la hora de resolver 2SPP. Las heurísticas las clasificamos según se apliquen a problemas no guillotina como a los que sí incorporan esta restricción en sus patrones de corte.

4.1.1. Heurísticas para problemas no guillotina

Una de las heurísticas clásicas más documentadas es la heurística *bottom-left* (BL) propuesta por Baker et al. [26] en 1980. En las últimas décadas se han propuesto algunas variantes de este método. La heurística BL coloca sucesivamente los rectángulos en la posición de la plancha más profunda posible; luego los desplazada totalmente hacia la izquierda. Por lo tanto, las soluciones están determinadas por el orden en que se introducen los rectángulos y su posible rotación. Este método fue mejorado por Chazelle [50] y llamado *bottom-left fill* (BLF). Este nuevo método puede llenar los huecos producidos en el patrón de empaquetado (ver la Figura 4.1(a)).

Jakobs [130] utiliza otro método BL. En este caso, cada pieza se posiciona inicialmente en la esquina superior derecha de la estructura. A continuación, se desplaza hasta la posición más profunda posible. Una vez allí, la pieza es movida hacia la izquierda tanto como sea

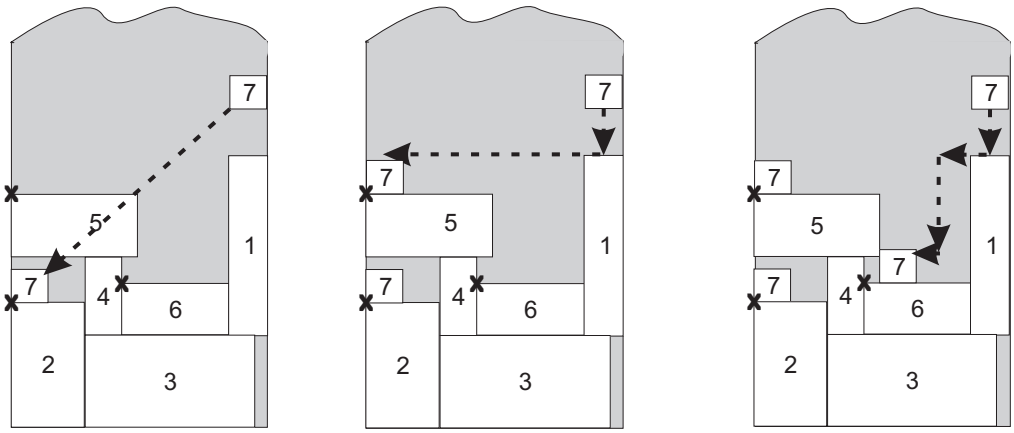


Figura 4.1: Ejemplos de patrones de empaquetado obtenidos por la heurística BL para el problema de *strip packing*: (a) Chazele [50]; (b) Jakobs [130]; (c) Liu and Teng [150]

posible, repitiéndose esta rutina hasta que la pieza alcance una posición inamovible. Esta estrategia corre en $O(n^2)$. La principal desventaja de esta rutina consiste en la creación de áreas vacías en el patrón. Liu y Teng [150] desarrollaron otra heurística BL similar al algoritmo de Jakobs. En su estrategia, el movimiento hacia abajo tiene prioridad de modo tal que el elemento se mueve hacia la izquierda sólo si el corrimiento hacia abajo no es posible. Este algoritmo necesita un tiempo de $O(n^2)$ (ver el ejemplo de la Figura 4.1(b)).

Burke et al. [47] han propuesto la heurística *best fit* (BF) que usa un ordenamiento dinámico de los rectángulos, en contraposición a una permutación de elementos como las heurísticas mencionadas anteriormente. El algoritmo analiza el conjunto de espacios disponibles comenzando por la posición más a la izquierda e inferior posible, posición a la que denominan (LAG), y selecciona por cada lugar el rectángulo que mejor se adapte, si es que existe alguno. Esto permite al algoritmo tomar decisiones informadas acerca de cuál será el siguiente elemento y dónde se deberá asignar. Una implementación natural de esta estrategia se ejecuta en $O(n^2)$. La Figura 4.2 muestra un ejemplo del procedimiento de esta heurística *ad hoc* y muestra en cada caso la posición LAG. Esta heurística fue hibridada con opciones metaheurísticas tales como TS, SA y GA. BF+SA ha obtenido los mejores resultados.

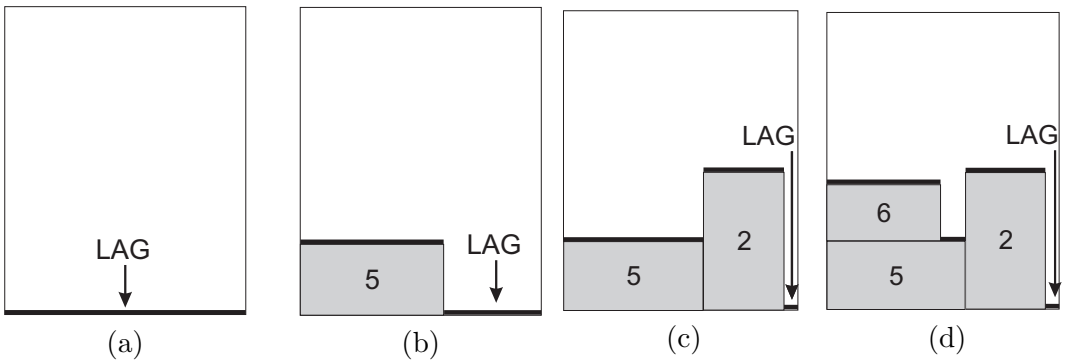


Figura 4.2: Ejemplos de patrones de empaquetado obtenidos por la heurística BL de Burke et al. para 2SPP

Hopper y Turton [125] presentan BLD la cual es una estrategia mejorada de BL, donde los objetos son ordenados de acuerdo a algunos criterios (alto, ancho, perímetro y área) y el algoritmo selecciona aquella pieza que de el mejor resultado. Lesh et al. [143] and Lesh and Mitzenmacher [144] enfocan su investigación en la mejora de la heurística BLD. Ellos nombran su nueva heurística como BLD*. Al comienzo, BLD* construye una lista de objetos de acuerdo a cierto criterio (alto, ancho u otro) en forma decreciente. La estrategia BLD* repite distintas ubicaciones con ordenamientos aleatorios específicos hasta que se alcanza un tiempo límite. Su algoritmo mejora otros algoritmos heurísticos y metaheurísticos basados en la estrategia BL reportados en la literatura.

4.1.2. Heurísticas para problemas guillotina

En el caso de los patrones de empaquetado en niveles, se han desarrollado principalmente tres heurísticas basándose en algoritmos populares para problemas de una dimensión [54, 171]. Como primer paso, las piezas se ordenan en forma decreciente teniendo en cuenta la altura. Luego se construyen los patrones de empaquetado como una serie de *niveles*. Cada pieza se coloca de modo que su base está apoyada sobre uno de esos niveles, lo más a la izquierda posible. El primer nivel está ubicado en la base de la plancha de material: su ancho está determinado por el ancho de la plancha de material y su altura está dada por la altura del primer objeto asignado a ese nivel. Los siguientes niveles se definen trazando una línea horizontal que toca la parte superior de la pieza más alta del nivel inferior.

A continuación describiremos las tres heurísticas que caen dentro de esta categoría. Denotemos como i a la pieza a ubicar y l el último nivel creado. Las estrategias son las siguientes:

- *Next-Fit Decreasing Height* (NFDH) [54]: la pieza i se coloca en el nivel l en la posición más a la izquierda posible, si hay lugar. De otra manera, se crea un nuevo nivel ($l = l + 1$) sobre el nivel l, y la pieza i se asigna en el borde izquierdo. Es decir, la heurística *ad hoc* acomoda las piezas en un nivel hasta que la próxima pieza no entre (debido a que el ancho de la pieza excede el espacio remanente en el nivel); en este caso la pieza se utiliza para iniciar un nuevo nivel por encima del anterior, sobre el que el proceso de empaquetado continua. El empaquetado progresa de izquierda a derecha

- *First-Fit Decreasing Height* (FFDH) [54]: la pieza i se asigna en el primer nivel que tenga suficiente lugar para acomodarla. Si ninguno de los niveles tiene espacio suficiente, entonces crea un nuevo nivel como en NFDH.

- *Best-Fit Decreasing Height* (BFDH) [171]: la pieza i se empaqueta, lo más a la izquierda posible, en aquel nivel que tenga suficiente espacio para acomodarla, en el que se genere el menor desperdicio. Si la pieza no cabe en ningún nivel, se crea uno nuevo como en NFDH. Esto significa que para empaquetar una pieza, se debe buscar en todos los niveles por un espacio lo suficientemente grande para acomodar la pieza y calcular el área desperdiciada que se generaría si se acomodase el rectángulo en cada nivel.

La principal diferencia entre estas heurísticas recae en la forma de seleccionar un nivel para ubicar la pieza i. NFDH sólo considera el nivel actual y los niveles son considerados de una manera voraz (una vez que un nivel se completa no se reconsidera), mientras que tanto FFDH como BFDH analizan todos los niveles ya construidos para acomodar la pieza. La complejidad de FFDH y BFDH es de $O(n \lg n)$, donde n es la cantidad de piezas consideradas, mientras que la complejidad de NFDH es lineal $O(n)$, si están apropiadamente

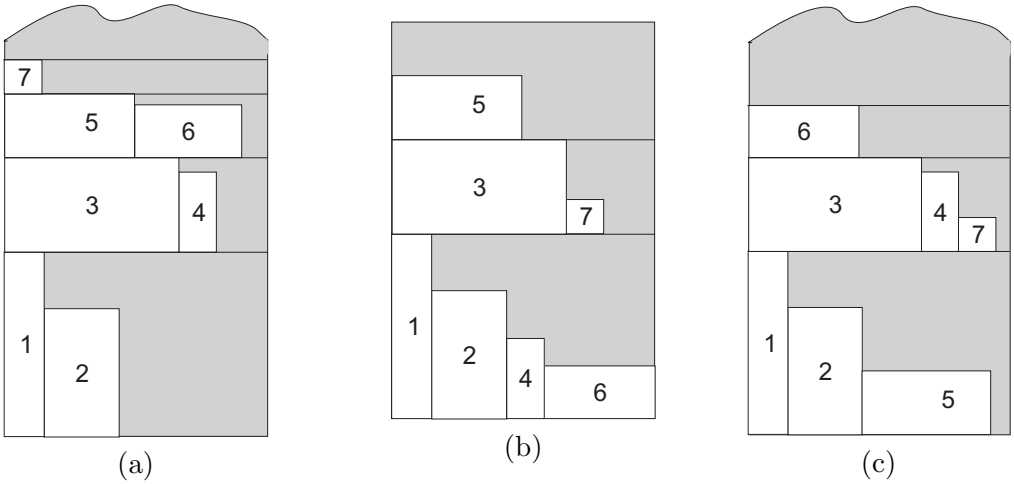


Figura 4.3: Ejemplos de patrones de empaquetado por niveles para el problema de *strip packing*: (a) next fit; (b) first fit; (c) best fit

implementadas. En la Figura 4.3 se presentan ejemplos de patrones de empaquetado obtenidos tras la aplicación de las distintas estrategias por niveles. En esta figura, las piezas están ordenadas en forma decreciente por altura y numeradas en función de ese ordenamiento. El patrón de empaquetado resultante siempre satisface la restricción de corte guillotina. Más precisamente, ellos son denominados patrones guillotina en dos etapas debido a que deben ser cortados en dos etapas: la primer etapa se corresponde con cortes horizontales y la segunda con cortes verticales.

Varios estudios han evaluado el desempeño de estas heurísticas. Tal es el caso del trabajo de Valenzuela y Wang [171] quienes concluyen que FFDH obtiene mejores resultados sobre NFDH. Tanto FFDH y NFDH se vuelven más efectivas a medida que la dimensión de las instancias analizadas se incrementa (dimensión medida en cantidad de piezas). Otro trabajo es el realizado por Ntene and van Vuuren [172], quienes además proponen mejoras a esos procedimientos. Los autores concluyen que no se observan diferencias siginificativas entre los resultados reportados por las distintas heurísticas analizadas.

Zhang et al. [228] introducen una heurística recursiva, a la que denominan HR, para problemas con restricciones guillotina. Cuando el primer objeto se coloca en la plancha en el borde izquierdo inferior, se generan dos áreas restantes. La heurística recursivamente

continúa colocando los objetos restantes. Para mejorar la performance de la heurística, los autores presenta un algoritmo determinista que da prioridad a los objetos con grandes áreas. Zhang et al. concluyen que su algoritmo obtiene buenos resultados en forma rápida utilizando los casos de prueba de Hopper.

4.2. Métodos metaheurísticos para resolver 2SPP

Aunque este tipo de problemas se ha resuelto mediante técnicas heurísticas, el mayor número de procedimientos desarrollados para encontrar soluciones adecuadas a 2SPP se basan en metaheurísticas muy variadas, las cuales ofrecen la habilidad de buscar en espacios de soluciones complejos y grandes de una manera sistemática y eficiente. Hopper Turton [122, 126] revieron los enfoques desarrollados hasta el año 2000 para resolver 2SPP usando GAs, SA y TS; llegan a la conclusión de que los GAs son las metaheurísticas más ampliamente investigadas en esta área. A nuestro entender, esta es la revisión más detallada que existe. En la Tabla 4.2 se presenta un resumen de tales enfoques, indicando: referencia del trabajo (columna Ref.) y año de publicación (columna Año). A continuación se indican las características del problema abordado: regularidad de las piezas (columna Reg), si se permite la rotación de las piezas (columna Rot) y si los patrones son guillotina (G). Se resalta la metaheurística utilizada (columna Metah), las heurísticas utilizadas como decodificadores (en el caso que corresponda) (columna Decodif) y se da una breve descripción de las características más importantes del algoritmo (columna Descripción). En esta sección nos centramos en rever soluciones a 2SPP utilizando distintas metaheurísticas (SA, TS, GRASP, GAs y ACO), partiendo del trabajo de Hopper [122], es decir, reviendo trabajos a partir del 2000 hasta la fecha. En cada apartado se presenta una tabla a modo de resumen, poniéndose énfasis en las características del problema abordado: regularidad de las piezas (columna Reg), si se permite la rotación de las piezas (columna Rot) y si los patrones son guillotina (G). Además se resaltan las heurísticas utilizadas como decodificadores (en el caso que corresponda) (columna Decodif) y se da una breve descripción de las características más importantes del algoritmo (columna Descripción).

Tabla 4.2: Soluciones previas al año 2000 para 2SPP usando metaheurísticas

Ref.	Año	Reg	Rot	G	Metah	Decodif	Descripción
[200]	1985	✓	✓	×	GA	Slide alg.	Rep. indirecta (permutación)
[130]	1996	✓	✓	×	GA	BL	Rep. indirecta (permutación)
[149]	1999	✓	✓	×	GA	improved-BL	Rep. indirecta (permutación)
[124, 125]	1999-2001	✓	✓	×	GA	BLF	Rep. indirecta. OX
[145]	1999	✓	×	×	GA	Difference Process	Rep. indirecta (permutación)
[225]	1997	✓	✓	×	GA	*sliding* algorithm	Rep. indirecta (permutación)
[135, 136, 137]	1991-1993	✓	✓	×	GA	BL	Rep. directa (árbol binario)
[113]	1996	✓	✓	×	GA	–	Rep. directa (cadena y matriz binaria)
[128]	1994	✓	✓	✓	GA	–	Rep. directa (árbol binario)
[134]	1995	✓	✓	✓	GA	–	Estructura de árbol representando cadenas
[183]	1995	✓	×	✓	GA	–	Rep. directa (árbol)
[21]	1996	✓	×	✓	GA	–	Rep. directa (árbol)
[128]	1994	✓	✓	✓	GA	FF y BF	Rep. indirecta (permutación)
[59]	1997	✓	✓	✓	GA	heurística *ad hoc*[1]	Rep. indirecta (permutación)
[77]	1993	✓	✓	×	SA	–	Rep. directa (posiciones en el *layout*)
[83]	1999	✓	×	✓/×	SA	BL con restricciones guillotina	Rep. indirecta (permutación)
[145]	1999	✓	×	×	SA	Difference Process	Rep. indirecta (permutaciones)

✓ - si × - no

[1] Heurística que considera restricciones tecnológicas

4.2.1. Recocido simulado

En esta sección se realiza una revisión de las distintas propuestas algorítmicas para resolver 2SPP utilizando SA. La Tabla 4.3 presenta un resumen de las mismas.

Hopper y Turton [125] utilizan un SA para 2SPP donde se permite la rotación de piezas y los empaquetados son no guillotina. La representación de las soluciones adoptada es por permutaciones, la cual indica el orden que serán empaquetadas las piezas. Utilizan las heurísticas BL y BLF para determinar la calidad de los patrones de empaquetado. Utilizan dos estructuras de vecindario: una simple que consiste en intercambiar dos piezas en la permutación y la otra que selecciona una pieza al azar y le cambia su orientación. SA+BLF produce mejores distribuciones de piezas que SA+BL. Los autores son los primeros en realizar una comparación entre metaheurísticas utilizando este problema.

Tabla 4.3: SA en la literatura para resolver 2SPP

Ref.	Reg	Rot	G	Decodif	Descripción
[125]	✓	✓	×	BL - BLF	Rep. indirecta. Dos estructuras de vecindario
[229]	✓	✓	×	HR	Mov.: intercambio de piezas
[46]	✓	✓	×	BF + BLF	Solución generada en dos pasos. Decremento proporcional de temperatura.
[103]	×	–	–		Modelos de PL para optimizar patrones. Movimiento: intercambio de piezas
[201]	✓	×	×	BLF	intercambio de piezas y desplazamiento de piezas. Decremento proporcional de temperatura.
[68]	✓	✓	Rec	no	Decremento proporcional de temperatura. Mov: cambio en la orientación de piezas.

✓ - si × - no

Zhang et al. [229] proponen un SA con una heurística recursiva (HR) [228]. Una nueva solución se obtiene al intercambiar las posiciones de dos rectángulos en la secuencia. La heurística se utiliza para evaluar las soluciones factibles de SA y así obtener su valor objetivo. Los autores comparan SA+HR con GA+BLF y SA+BLF [125] y reportan que SA+HR muestra resultados más satisfactorios en menos tiempo.

Burke et al. [46] proponen un SA donde la solución se genera en dos etapas: una cantidad inicial de piezas se asigna utilizando BF, las restantes piezas se asignan utilizando BLF. Estas últimas piezas están sujetas a cambios que puede producir la metaheurística, mientras que las posiciones de las piezas asignadas con BF son inamovibles. La temperatura se modifica usando la regla geométrica de reducción. Los movimientos consisten en el intercambio de piezas y el cambio en la orientación de la pieza seleccionada. Los autores comparan su metaheurística híbrida con los resultados de [23] y de [125], indicando que su algoritmo mejora significativamente las soluciones dadas en estos dos últimos trabajos.

Gomes y Oliviera [103] desarrollaron un SA híbrido combinado con técnicas de programación lineal para optimizar los patrones durante el proceso de búsqueda. La estructura de vecindario está basada en el intercambio de piezas. El problema que resolver consta de piezas irregulares. Los modelos de programación lineal, que se utilizan localmente para optimizar los diseños, se derivan de la aplicación de algoritmos de separación y compactación. Los autores realizaron una comparación con varios casos de pruebas típicos de los problemas de empaquetado irregular usados en la literatura. Los autores muestran cómo su SA híbrido

Tabla 4.4: TS en la literatura para resolver 2SPP

Ref.	Reg	Rot	G	Decodif	Descripción
[46]	✓	✓	×	BF + BLF	Rep. indirecta. Dos fases para la construcción de una solución
[109]	✓	✓	×	–	Rep. directa. Mecanismos de diversificación. Lista tabú incluye información del problema.

mejora todos los resultados hasta el momento publicados de todas las instancias analizadas.

Soke y Bingul [201] combinan un SA con un algoritmo BLF mejorado para resolver 2SPP no guillotina. En el trabajo los autores estudian la influencia de varios parámetros en la solución al problema de empaquetado. La representación de la solución es por permutaciones. En el trabajo estudian dos tipos de movimientos posibles: el intercambio y el desplazamiento de piezas, concluyendo que el primero produce mejores resultados. Utilizan dos estrategias para decrementar la temperatura: el método proporcional y el propuesto por Lundy y Teng [156], siendo con este último con el que obtienen las mejores soluciones.

Dereli y Sena Daş [68] proponen un SA con un nuevo procedimiento de asignación de piezas recursivo (Rec). El vecindario es producido al habilitar o no la rotación de una pieza. Tras la aplicación de la perturbación las piezas son ordenadas en forma decreciente por la altura. Usan un mecanismo de decremento de la temperatura proporcional. La opción híbrida reporta mejores resultados que un SA propuesto por Hopper y Turton [125], pero con tiempos de cómputos mucho mayores.

4.2.2. Búsqueda tabu

En esta sección se analizan los distintos algoritmos TS propuestos en la literatura para resolver 2SPP. La Tabla 4.4 muestra un resumen de las distintas alternativas.

Burke et al. [46] proponen un TS para 2SPP con una extensa lista tabú. La primer parte de la solución es inalterable y es generada por BF. El resto de las posiciones (segunda parte) de la solución pueden ser alteradas por la metaheurística y son evaluadas con BLF. Una solución vecina se crea entonces al intercambiar dos piezas elegidas aleatoriamente de posiciones pertenecientes a la segunda parte, además se rota la primer pieza seleccionada con una probabilidad del 50 %. Los resultados obtenidos por TS para instancias conocidas de la literatura indican que presenta soluciones de calidad intermedia entre un SA y un GA.

Tabla 4.5: GRASP en la literatura para resolver 2SPP

Ref.	Reg	Rot	G	Decodif	Descripción
[30]	✓	✓	×	BF	Uso de VNS para mejorar los empaquetados. Uso de permutaciones.
[20]	✓	×	×	basado en BF	uso de varias estrategias para la construcción y mejora.

Hamiez et al.[109] trabajan con TS usando una representación directa, es decir con la ubicación de los objetos en el *strip*. TS trata al problema de optimización, el cual consiste en minimizar la altura del patrón de empaquetado, como sucesivos problemas de satisfacción: comenzando desde un empaquetado $s0$ (obtenido con un método voraz) de altura $H(s0)$, TS trata de resolver 2SPP decrementando el valor $H(s0)$. El algoritmo incluye dos vecindarios aplicados con probabilidades complementarias, un mecanismo de diversificación (basado en una estrategia de perturbación aleatoria) y una estructura tabú particular que incluye conocimiento del problema para evitar visitar patrones similares. Los autores reportan que TS se comporta ligeramente mejor que un algoritmo GRASP [20].

4.2.3. GRASP

En esta sección hacemos una breve reseña de las distintas opciones que se encuentran en la literatura relacionadas con resoluciones de un problema de *strip packing* usando GRASP. La Tabla 4.5 muestra un resumen de tales trabajos haciendo hincapié en las restricciones del problema consideradas.

Beltrán et al. [30] proponen una metaheurística híbrida que combina GRASP con VNS. En la formulación de 2SPP utilizado se permite la rotación de piezas y los patrones de empaquetado no son guillotina. Esta opción repite las siguientes dos fases: las piezas se asignan utilizando una fase constructiva basada en BF randomizada. Luego la solución se pasada a un VNS, y la solución obtenida es mantenida sólo si es mejor que la previa. Específicamente, VNS intenta mejorar el empaquetado de los últimos rectángulos introducidos a la solución. Los autores comparan la opción propuesta con un SA usando BLF [125], concluyendo que la eficacia es comparable o mejor que la de SA.

Alvarez-Valdez et al. [20] proponen un algoritmo GRASP reactivo para 2SPP no guillotina y con orientación fija. Los autores investigan distintas estrategias para las fases de

Tabla 4.6: GAs en la literatura para resolver 2SPP

Ref.	Reg	Rot	G	Decodif	Descripción
[117]	✓/×	✓	×	heurística constructiva	Representación indirecta.
[171]	✓	✓	✓	FFDH	Rep. directa (postorden). Empaquetados en niveles.
[46]	✓	✓	×	BF+BLF	Rep. indirecta. Dos fases para la construcción de una solución
[34]	✓	✓	✓/×	–	Representación directa. Se usa BFDH para generar la población inicial.
[201]	✓	✓	×	BL	Representación por permutaciones.
[230]	✓	✓	×	heurística propia	

construcción y mejora. Cuando seleccionan las piezas, consideran el conjunto de piezas del mismo tipo para tratar de empaquetarlas en forma conjunta en lugar de considerar sólo las piezas en forma individual. Los autores comparan sus resultados con métodos exactos[159], varias heurísticas [47, 129, 143] y metaheurísticas [34, 46] presentes en la literatura, y arriban a que para la mayoría de las instancias el algoritmo GRASP propuesto mejora los resultados existentes.

4.2.4. Algoritmos evolutivos

A continuación revemos las distintas propuestas de solución a 2SPP utilizando algoritmos genéticos, cuyo resumen se presenta en la Tabla 4.6.

Yeung et al. [226] proponen una combinación de GA con la heurística *Lowest-Fit Left-Aligned* (LFLA) (asigna las piezas una a una en la posición más baja posible alineada en el borde izquierdo). Utilizan una representación por permutaciones. El operador de recombinación es *order crossover*, además utilizan un operador de inversión y una mutación por intercambio (ambos con muy baja probabilidad de aplicación) para introducir diversidad genética. Proponen un GA paralelo que trabaja con 5 subpoblaciones de tamaño 20.

Valenzuela y Wang [171] proponen un GA con representación en postorden para resolver 2SPP guillotina en niveles en dos etapas. Realizan una comparación entre las heurísticas en niveles (descriptas en 4.1.2) y concluyen que FFDH es la que mejor patrones de empaquetado obtiene. Luego comparan FFDH con el GA y concluyen que para instancias grandes la heurística FFDH supera al GA, mientras que la situación contraria se presenta para las instancias más chicas.

Hifi y Hallah [117] plantean una solución a 2SPP tanto regular como irregular. Proponen una heurística *ad hoc* basada sobre una opción constructiva diseñada específicamente para piezas irregulares, pero que se adapta a regulares. Esta heurística se usa como decodificador en un GA. El algoritmo presenta buenos resultados en poco tiempo computacional.

Bortfeld [34] propone un GA que realiza la búsqueda con los patrones de empaquetado completamente definidos exhibiendo una estructura en niveles, en lugar de usar un esquema de codificación. Utiliza operadores genéticos específicos para manejar los patrones. El algoritmo genera una población inicial usando una heurística BFDH*, la cual es una versión mejorada de BFDH [171]. Está permitida la rotación de piezas. Para problemas sin la restricción de cortes guillotina, un procedimiento de post-optimización rompe la estructura en niveles. Compara su algoritmo con 11 métodos que fueron propuestos entre 1993 y 2004, reportando que su GA mejora todos los resultados de sus competidores. El mismo GA es usado en Bortfeldt y Gehring [35] con distintos parámetros para grandes instancias (1000 piezas). El procedimiento es sólo aplicado a problemas con restricciones de corte guillotina.

Burke et al. [46] proponen un GA hibrizado con BF y BLF para 2SPP en que las piezas se pueden rotar 90° y las soluciones no están restringidas a patrones guillotina. La solución se genera en dos pasos: una cantidad inicial de piezas se asigna utilizando BF, las restantes piezas se asignan utilizando BLF. El GA trata de mejorar la parte de la solución generada con BLF. Los experimentos realizados demuestran que la opción es superada por un SA y TS en la mayoría de las instancias analizadas.

Soke y Bingul [201] presentan un GA para resolver 2SPP no guillotina. Adoptan una representación por permutaciones con un decodificador BL mejorado. Prueban varios parámetros del GA (tamaño de población, probabilidades) así como también el comportamiento de seis operadores de recombinación tradicionales para representaciones basadas en permutaciones. La mutación usada consiste en el intercambio de piezas. Comparan la performance del GA híbrido con un SA híbrido con BL mejorada, concluyendo que el primero obtiene mejores resultados en un número menor de evaluaciones.

Zhang et al. [230] introducen una heurística recursiva combinada con GA. Se crean inicialmente combinaciones de piezas en niveles sin espacios libres. Luego se usa la heurística

para calcular la altura de las órdenes restantes y usan la capacidad de los algoritmos genéticos para reducir esa altura. Aunque el algoritmo puede obtener mejores soluciones que el GA+BLF y SA+BLF de [125], necesita mucho tiempo, en especial para problemas grandes.

4.2.5. Optimización por colonia de hormigas

En esta sección se presentan las soluciones al problema de *strip packing* usando algoritmos ACO (ver la Tabla 4.7 para un resumen de las distintas opciones encontradas).

Burke y Kendall [43] proponen uno de los primeros algoritmos ACO para un problema de empaquetado de polígonos irregulares. Utilizan la heurística NFG, las visibilidades son actualizadas a medida que se van asignando las piezas. Los resultados son comparados con SA [45], GA [44] y TS . El ACO obtiene patrones de empaquetado de menor altura que el GA, pero de mayor altura que los de TS. Los autores proponen seguir trabajando con parámetros y mejorando el algoritmo al agregar algún método de búsqueda local.

Thiruvady et al. [212] proponen una solución a 2SPP usando un sistema de colonia de hormigas (ACS), donde solamente la mejor hormiga realiza la actualización del rastro. Las soluciones se representan por medio de permutaciones y utilizan la heurística BLF como decodificador, a la cual introducen modificaciones a fin de determinar la mejor orientación para una pieza en el momento de su asignación. Los autores proponen distintas formas de significar los rastros de feromonas: una matriz donde se almacena información sobre el ordenamiento de piezas y otra donde se considera la orientación de las piezas. Los autores experimentan con las cuatro combinaciones posibles y concluyen que aprender el ordenamiento y aplicar elecciones locales respecto de la selección de la rotación de los elementos produce el mejor resultado.

Chunyu y Xinbao [52] usan un $\mathcal{MM}$AS que genera la secuencia de entrada (permutación) para la estrategia de ubicación de las piezas, en particular consideran una combinación de estrategias para empaquetados en niveles y un método de *sliding* para reducir el espacio libre. Los autores agregan un conjunto de reglas para seleccionar la próxima pieza de la lista candidata. Antes de aplicar la regla probabilística de selección, se busca en la lista candidata piezas con la misma altura de la pieza ya asignada o con ancho igual al espacio remanente

Tabla 4.7: ACOs en la literatura para resolver 2SPP

Ref.	Reg	Rot	G	Decodif	Descripción
[43]	×	✓	×	NFG	Actualización dinámica de visibilidades. Uso de permutaciones.
[212]	✓	✓	×	BLF	Uso de permutaciones. Distintos significados para el rastro de feromona.
[52]	✓	✓	×	BD	Uso de permutaciones. Heurísticas por niveles y método para reducir el espacio libre.

en el nivel. La información heurística da preferencia a piezas que tengan alturas similares a la pieza ya asignada. La actualización del rastro la hace la mejor hormiga de la iteración. Los resultados son comparados a un GA y SA. Para las instancias de Hopper y Turton [125], ACO reduce la distancia al óptimo resultando un algoritmo rápido. En cambio para las instancias de Burke [47] los resultados de ACO son bastante lejanos a los que obtienen SA y GA.

4.3. Conclusiones

En este capítulo, hemos resumido las propuestas algorítmicas más recientes para el problema de *strip packing* en dos dimensiones, el cual tiene muchas aplicaciones industriales. Hemos explicado varios algoritmos heurísticos y metaheurísticos encontrados en la literatura partiendo de los trabajos presentados por Hopper y Turton [126].

Las heurísticas más documentadas son BL y BLF. Aunque son métodos veloces, la calidad de las soluciones no es buena. La mayoría de las heurísticas se usan como decodificadores en metaheurísticas que utilizan representaciones indirectas de las soluciones, para construir el patrón de empaquetado correspondiente a una solución para 2SPP. Aunque este tipo de problemas se ha resuelto mediante técnicas heurísticas, el mayor número de procedimientos desarrollados para encontrar soluciones adecuadas a los diferentes problemas de corte y empaquetado se basan en metaheurísticas muy variadas.

Los algoritmos evolutivos son las técnicas más ampliamente investigadas en el área de corte y empaquetado. El trabajo hecho hasta el momento usa en su mayoría representaciones basadas en permutaciones; también hay trabajos usando representaciones basadas en árboles. La mayoría de las propuestas que se reportan en la literatura resuelven esta clase

de problema con procedimientos en dos etapas, denominados algoritmos genéticos híbridos. El algoritmo genético manipula las soluciones codificadas, que luego son evaluadas por un algoritmo de decodificación al transformar la secuencia de empaquetado en la correspondiente disposición física. Como el conocimiento del dominio se construye en el proceso de decodificación, el tamaño del espacio de búsqueda se puede reducir. La estrategia de empaquetado, por ejemplo, sólo puede generar configuraciones sin superposición, lo cual restringe el espacio de búsqueda a soluciones sólo válidas. La necesidad de una heurística *ad hoc* de decodificación excluye cierta información acerca de la disposición de las piezas de las estructuras de datos que los algoritmos genéticos manipulan. Por lo tanto, no toda la información relativa al fenotipo está a disposición de los operadores genéticos y puede, por tanto, que no se transfieran a la próxima generación.

También existen trabajos donde los GAs incorporan alguna información de la distribución de las piezas en las técnicas de codificación. Una de las alternativas es usar estructuras de árbol binario usando alguna regla adicional para determinar la posición en el patrón. Otra alternativa es que el GA trabaje sin codificación y resuelva el problema en el espacio 2D. Aunque esta última alternativa ha sido ampliamente aplicado en SA y TS, siendo la aplicación de un proceso de optimización indirecto por medio del uso de codificación es una idea más reciente para estas metaheurísticas. Este último tiempo han surgido soluciones la problema de *strip packing* usando las metaheurísticas más recientes como lo son GRASP y ACO, reportando buena calidad de soluciones.

Además es de remarcar el surgimiento de investicaciones en este campo donde se presentan nuevas propuestas y además éstas se comparan tanto contra métodos heurísticos eficientes como con otras metaheurísticas presentes en la literatura.

A pesar de existir algunas comparaciones con procesos de búsqueda específicos y métodos de optimización local, sólo unos pocos intentos se han hecho para comparar el desempeño de varias metaheurísticas en este área. Burke y Kendall [45] y Leung et al. [146] han realizado trabajos de investigación en este área. El primer trabajo indica que TS y SA presentan mejor rendimiento que los GAs. Leung et al. [145] comparan rendimiento y eficiencia de GAs y SA concluyendo que los primeros superan a SA en término de área no usada.

Como se puede observar en las distintas tablas resumen que se presentaron en este capítulo, pocos autores incluyen en la formulación del problema a los empaquetados guillotina, salvo [34, 171]. En particular, empaquetados en niveles con tres etapas, como se trata en esta tesis, aparecen en trabajos de Puchinger et al. [179, 180]. Estos trabajos tratan con el problema de *bin packing* cuyo objetivo es minimizar la cantidad de hojas de material (de dimensiones fijas) usada para empaquetar todas las piezas, un objetivo diferente, pero relacionado, al considerado en el presente trabajo. Es por esta razón que hemos desarrollado nuestro propio conjunto de datos de prueba generado de una manera estructurada para poder realizar la experimentación de forma coherente, exhaustiva en cuanto a los valores de los atributos de las piezas. Además para evitar disparidades en la complejidad de los datos de prueba existentes. Los datos de prueba, como ya hemos mencionado anteriormente, está disponible públicamente para evitar situaciones de este estilo en el futuro. Por otra parte es de destacar la ausencia, en los papers existentes en la literatura, de los datos que medimos en este estudio y de las características del problema consideradas.

El estudio realizado en este capítulo no intenta cubrir todo el espectro de publicaciones en el área, pero esperamos que pueda brindar información de utilidad a investigadores interesados en algoritmos heurísticos y metaheurísticos para el problema de *strip packing* en dos dimensiones. Por consiguiente la exclusión de algún trabajo en particular no es intencional ni tampoco puede ser considerado como un juicio hacia ese trabajo.

Parte II

Resolución del problema de corte y empaquetado

Capítulo 5

Resolución de 2SPP usando algoritmos genéticos

En este capítulo abordamos la solución de 2SPP con un GA, con el objetivo de encontrar un patrón de empaquetado de altura mínima. Tomando como punto de partida un GA básico, en este trabajo investigamos las ventajas de utilizar operadores de recombinación y mutación que incorporan conocimiento específico del problema, tal como información respecto de la asignación de las piezas, contra otros operadores genéticos clásicos. En particular, analizamos el comportamiento de esta metaheurística al variar distintos operadores de recombinación y mutación y estudiamos sus ventajas relativas para resolver el problema, con el objetivo de encontrar la mejor configuración. A fin de reducir el desperdicio en cada nivel, se aplica un operador adicional, llamado operador de relocalización, a cada hijo generado. También investigamos las ventajas de agregar semillas en la población inicial, las cuales se generan utilizando un conjunto de reglas de construcción; la característica esencial es incluir información del problema en cuestión, tal como ancho de piezas, área de las piezas, etc. La intención de este estudio es que la población inicial resulte más especializada para el problema bajo análisis. Por último, discutimos la inclusión de conocimiento del problema en el método de búsqueda de un GA para presentar técnicas híbridas más eficientes para resolver 2SPP. El principal objetivo de este capítulo es hallar un GA mejorado para resolver grandes problemas, y cuantificar los efectos de incluir operadores específicos del problema,

semillas en el algoritmo e hibridación, buscando la mejor relación entre exploración y explotación del proceso de búsqueda. El capítulo se organiza como sigue. La Sección 5.1 presenta la descripción de un GA básico utilizado en este capítulo, para que a continuación en las Secciones 5.2 y 5.3 se detalle la representación y función de *fitness* utilizada para resolver 2SPP con restricciones de empaquetado, respectivamente. La Sección 5.4 muestra los operadores genéticos propuestos, tanto de recombinación como de mutación; mientras que la Sección 5.5 describe el nuevo operador de relocalización desarrollado en este trabajo de tesis para resolver 2SPP. En la Sección 5.6 se presentan los detalles de la generación de la población inicial. En la Sección 5.7 se dan los detalles de las técnicas híbridas propuestas. La Sección 5.8 comienza con un detalle de los parámetros del algoritmo genético utilizados en los experimentos para luego analizar los resultados obtenidos tras la aplicación de GA a las instancias del problema. Por último, la Sección 5.10 presenta las conclusiones finales sobre los resultados obtenidos.

5.1. Descripción del algoritmo

En esta sección presentamos un ssGA que utilizamos en este trabajo de tesis para resolver 2SPP. La elección de un GA estacionario frente a un GA generacional se debe a que existen trabajos que muestran una mayor eficacia del primero [216]. La estructura genérica del algoritmo se presenta en el Algoritmo 4. Este algoritmo crea una población inicial $P(0)$ de μ soluciones de una manera aleatoria (uniforme). Luego evalúa esas soluciones; para lo cual se usa un algoritmo de asignación que acomoda las piezas en la plancha de material (*strip*) para construir un patrón de empaquetado factible. A continuación, la población pasa por un ciclo, denominado evolución. Este ciclo consiste en la selección de dos padres por torneo binario y la aplicación de algunos operadores genéticos para crear la nueva solución ($hijo$). El nuevo individuo generado reemplaza al peor de la población sólo si es mejor. El criterio de parada para el ciclo es alcanzar un número máximo de evaluaciones ($max\_evaluaciones$). La mejor solución es identificada como el mejor individuo hallado, el cual presenta la mínima altura de material necesaria para empaquetar todas las piezas.

Algoritmo 4 Pseudocódigo del algoritmo genético utilizado

```
t = 0; {generación actual}
inicializar(P(t));
evaluar(P(t));
while (t < max_evaluaciones) do
  padres = seleccionarPadres(P(t));
  hijo = aplicarRecombinación(padres);
  hijo = aplicarMutación(hijo);
  evaluar(hijo);
  P(t + 1) = seleccionar(P(t), {hijo});
  t = t + 1;
end while
retornar la mejor solución de P(t)
```

En este trabajo de tesis, se usa una opción híbrida (procedimiento en dos etapas) para resolver 2SPP: GA se usa para determinar el orden para empaquetar las piezas y una rutina de asignación para determinar la distribución de las piezas, satisfaciendo las restricciones de empaquetado por niveles en tres etapas.

5.2. Representación

En la mayoría de los casos, aplicar un GA significa desarrollar una representación especializada para codificar una solución candidata para un problema dado. Una búsqueda efectiva deberá dificultarse si se busca por las coordenadas x e y de cada elemento, ya que la cantidad de soluciones es inmensurable y no es sencillo eliminar la superposición de piezas. Para sobrellevar esta dificultad, se han propuesto varios esquemas de codificación y muchos algoritmos para problemas de empaquetado regular están basados en esos esquemas de codificación. Uno de los esquemas más populares es representar la solución por una permutación de M elementos, donde una solución codificada (es decir, una permutación de M elementos) especifica el orden en el que se asignarán las piezas. La cantidad de posibles soluciones es $O(M!)$, la cual es más pequeña que para otros esquemas de codificación en la literatura. Cada permutación corresponde a una distribución sin solapamiento de elementos. Un algoritmo de decodificación calcula la distribución de una solución codificada al especificar las ubicaciones de los elementos uno a uno, el cual define la traslación entre la solución codificada a patrones de empaquetado (distribución de los elementos).

Un cromosoma es una permutación $\pi = (\pi_1, \pi_2, ..., \pi_M)$ de M números naturales, donde cada número representa los identificadores de las piezas. Como ejemplo consideremos el siguiente problema con 20 piezas ($M = 20$) y una plancha de material de ancho W y altura ilimitado. Un posible cromosoma es π = (6,0,8,4,5,9,12,15,1,3,10,11,13,17,18,2,19,14,7,16), donde 6 indica la pieza 6, 0 la pieza 0, 8 la pieza 8, y así sucesivamente. El cromosoma da un orden para considerar las piezas, es cual es luego usado por el algoritmo de decodificación (también llamado de asignación), representado por la función de evaluación (ver explicación en la Sección 5.3). Este algoritmo dispondrá las piezas en la tira de material (también denominada plancha o *strip*) para construir el patrón de empaquetado. GA genera la mejor permutación posible para que el algoritmo de asignación halle un patrón de empaquetado óptimo, el cual minimiza la altura de la plancha de material requerida.

La Figura 5.1 muestra el patrón de empaquetado correspondiente a la permutación π que se obtiene al utilizar una versión modificada de NFMH que se describirá en las próximas secciones. El patrón respeta las restricciones guillotina y de empaquetado en tres etapas, descritas en la Sección 3.4. En la primer etapa se realizan cortes horizontales para obtener los distintos niveles. En la siguiente etapa los cortes son verticales para obtenerse las pilas y finalmente en la tercer etapa se efectúan cortes horizontales para obtener las piezas y material de desperdicio.

5.3. Función de evaluación

Los GAs son guiados por los valores calculados por una función objetivo para cada solución tentativa, hasta que se halla un óptimo o una solución aceptable. En nuestro problema, el objetivo es minimizar la altura de la plancha, (*strip.alto*), necesaria para construir el patrón de empaquetado correspondiente a una solución π dada.

Para determinar la calidad de una solución, primero es necesario obtener la distribución de las piezas. A fin de generar un patrón por niveles en tres etapas adoptamos una heurística NFDH modificada (explicada en la Sección 4.1.2), de aquí en más referenciada como *next-fit modificada* o *NFMH*, la cual ha probado ser muy eficiente en [179, 197].

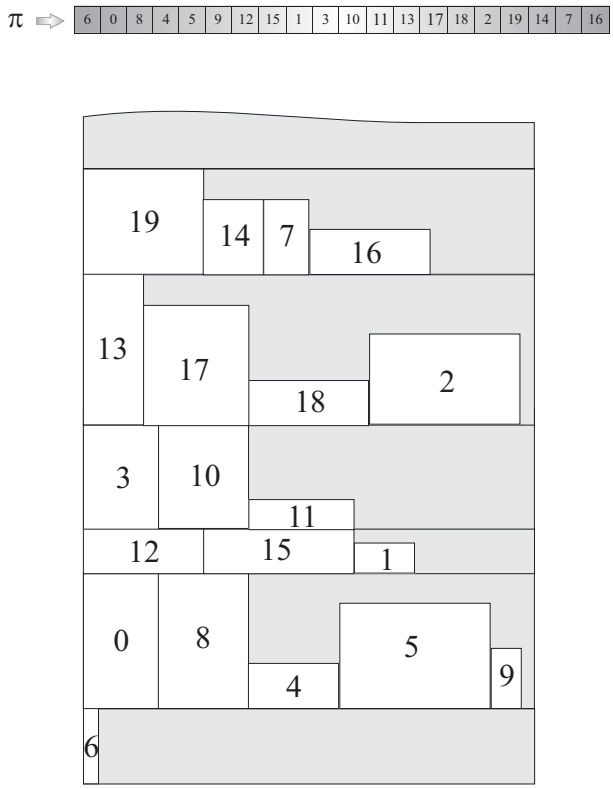


Figura 5.1: Patrón de empaquetado para la permutación π

Esta heurística, dada una secuencia de piezas ordenadas como entrada (una permutación de piezas), construye el patrón de empaquetado al colocar piezas en pilas (piezas vecinas con el mismo ancho se apilan una sobre la otra) y luego pilas en niveles paralelos a la base del *strip*. Las pilas se justifican hacia el borde izquierdo. Una vez que se crea una nueva pila o nivel, los anteriores no se reconsideran. El Algoritmo 5 incluye una descripción de alto nivel de este algoritmo.

El algoritmo comienza con la primer pieza π_1 del vector $\pi = (\pi_1, \pi_2, ..., \pi_M)$. Las variables que utiliza son:

- *strip*: representa el *strip*,
- l: referencia al nivel actual, y
- s: hace referencia a la pila actual

Algoritmo 5 Proceso de asignación de la heurística NFMH

```
NFMH(π: vector, M:integer, W:integer)
iniciar contadores:
  i = 1; {pieza actual}
iniciar strip:
  strip.alto = 0;
while (i ≤M) do
   iniciar_nivel(π_i, l);
   strip.alto = strip.alto + l.alto;
   while (i ≤ M and factible π_i en l) do
      iniciar_pila(π_i, s, l);
      while (i ≤M and factible π_i en s) do
         push_pila(π_i, s, l);
         s.alto = s.alto + altura(π_i);
         l.desperdicio = l.desperdicio - área(π_i);
         i = i + 1;
      end while
   end while
end while
setear l para ser el último nivel;
return strip.alto, l;

iniciar_nivel(π_i:pieza, l: nivel)
l.alto = altura(π_i);
l.anchoRest = W;
l.desperdicio = strip.area;

iniciar_pila(π_i:pieza s: pila, l:level)
s.ancho = ancho(π_i);
s.alto = 0;
l.anchoRest = l.anchoRest − ancho(π_i);

push_pila(π_i :pieza, s: pila, l: nivel)
if l.alto < s.alto + altura(π_i) then
   desperdicio = l.desperdicio + (s.alto + alto(π_i) − l.alto) × W;
   if (desperdicio−área(π_i) ≤ l.desperdicio) then
      l.desperdicio = desperdicio;
      l.alto = s.alto + altura(π_i);
      strip.alto = strip.alto + (s.alto + altura(π_i) − l.alto);
   end if
end if
```

Tanto el *strip* como el primer nivel se inicializan con la altura de π_1 ($l.alto = altura(\pi_1)$), y la primera pila de ese nivel con el ancho de π_1 ($s.ancho = ancho(\pi_1)$). Una vez creado un nivel, la próxima pieza, π_i, de π comienza la segunda pila del nivel actual si se cumplen las siguientes condiciones: (a) la $altura(\pi_i)$ no excede la altura del nivel, $l.alto$, y (b) el $ancho(\pi_i)$ no excede el ancho restante del nivel, $l.anchoRest$. Si la siguiente pieza j tiene ancho igual a $ancho(\pi_i)$, entonces son apiladas una sobre la otra si la altura de la pila resultante, $s.alto$, no excede $l.alto$. En el caso de que la próxima pieza π_j difiere de $ancho(\pi_i)$ o $s.alto$ excede

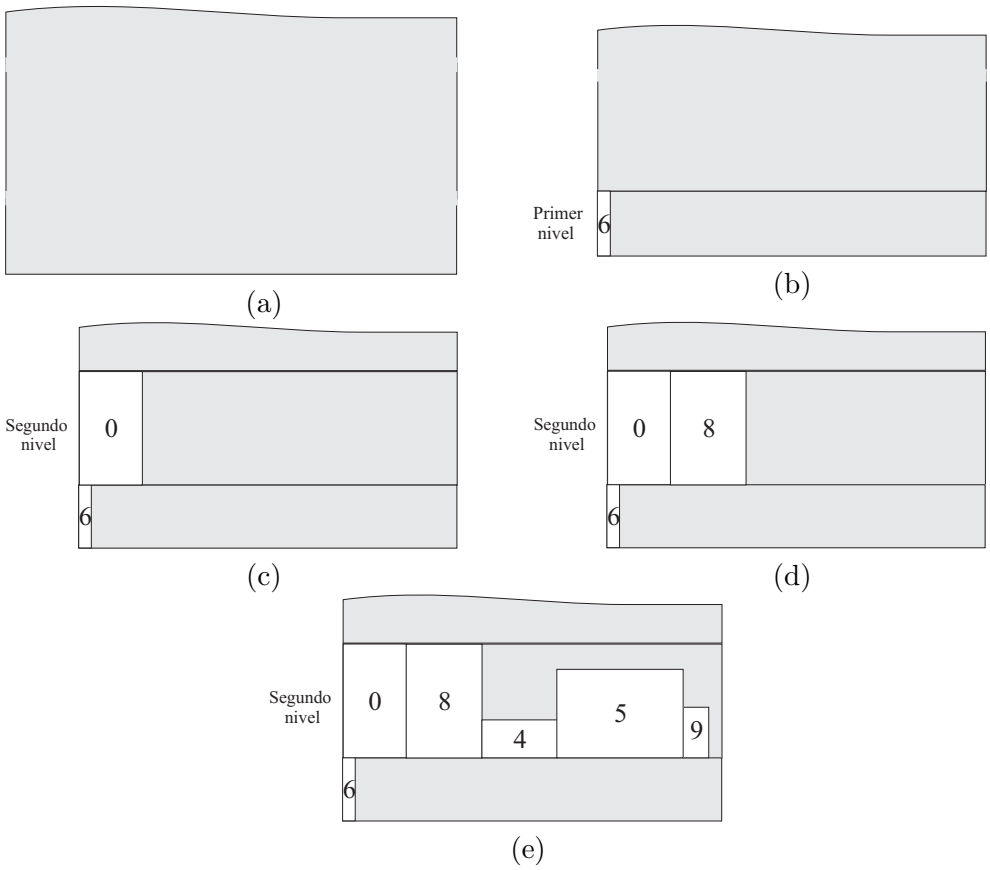


Figura 5.2: Pasos para construir un patrón de empaquetado por niveles en tres etapas

$l.alto$, entonces se comienza una nueva pila con π_j si $ancho(\pi_j) \leq l.anchoRest$; en caso contrario, se comienza un nuevo nivel con π_j. Este proceso se repite hasta empaquetar todas las piezas del vector π.

La Figura 5.2 muestra paso a paso cómo se realizaría la asignación de cada una de las piezas, cuyo orden está dado por la permutación π, para construir el patrón de empaquetado que se muestra en la Figura 5.1. Inicialmente la plancha de material o *strip* está vacío (Figura 5.2(a)). La primer pieza de la permutación $\pi_1 = 6$ se acomoda en el borde izquierdo inferior de la plancha y su altura $altura(\pi_1)$ define la altura del primer nivel $l.alto = altura(\pi_1)$ (Figura 5.2(b)), se inicia el primer nivel. Se continúa con la siguiente pieza π_2, en este caso la pieza con identificador 0. Como se puede observar la pieza 0 tiene una altura mayor a la del primer nivel, por lo tanto no se puede acomodar en el mismo. En consecuencia

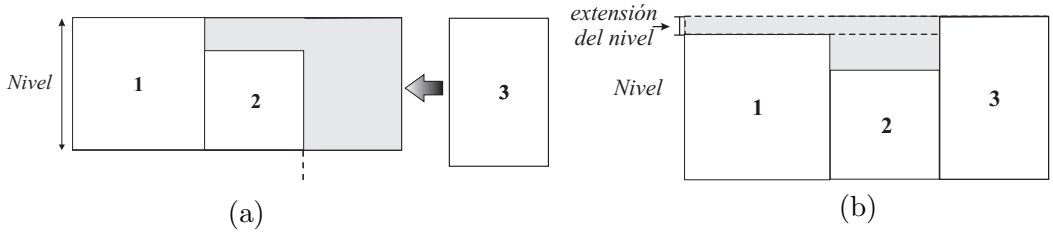


Figura 5.3: Ejemplo de extensión de nivel: (a) nivel antes de la asignación de la pieza 3 y (b) nivel extendido al asignar la pieza 3

se inicia un nuevo nivel con esta pieza (con altura igual a la correspondiente a la pieza 0) y el primer nivel se cierra (el área en gris denota el espacio desperdiciado que se generado), es decir, no se puede seleccionar para ubicar próximas piezas. La base del segundo nivel se apoya sobre la parte superior de la pieza 6 y la altura del nivel $l.altura = altura(\pi_2)$ (Figura 5.2(c)). La siguiente pieza π_3=8 tiene una altura igual o menor que $l.altura$ y además $ancho(\pi_3)$ es menor al ancho restante del nivel, por lo tanto se puede asignar en el segundo actual (Figura 5.2(d)). Lo mismo ocurre con las piezas con identificadores 4, 5, y 9 (Figura 5.2(e)). Estos pasos continúan hasta que se agoten las piezas para finalmente obtenerse el patrón de empaquetado de la Figura 5.1.

Nuestra versión de NFMH permite extender la altura del nivel si se produce un decremento en el desperdicio total del nivel en las siguientes situaciones: en el caso de apilar piezas y que la altura de la pila $s.alto$ resultante exceda la altura del nivel o cuando la altura de una pieza en una nueva pila sea más alta que el nivel $l.alto$. Consideremos el ejemplo de la Figura 5.3(a) con un nivel (el área gris representa el desperdicio) y la próxima pieza que asignar (pieza con identificador 3). La nueva pieza puede ser ubicada en el nivel si el área de la nueva pieza es mayor o igual al área del desperdicio producido al alargar el nivel (representado por un rectángulo con líneas punteadas en la Figura 5.3(b).

Como ya hemos mencionado, el *fitness* de una solución π se define como la altura necesaria del *strip* para construir el correspondiente patrón de empaquetado, definido como la suma de las alturas de los niveles generados. Es importante considerar que dos patrones de empaquetado podrían tener el mismo alto —por lo tanto sus *fitness* resultarían iguales— aunque, desde el punto de vista de reuso del desperdicio, uno de ellos puede ser mejor porque

el desperdicio en el último nivel (el cual se conecta con el resto del *strip*) es mayor que el presente en el último nivel del otro patrón de empaquetado. Los patrones de empaquetado que dejan la parte más grande sin uso en el último nivel son más prometedores. Tomando esto en cuenta, usamos la siguiente función de *fitness*:

$$F(\pi) = strip.alto - \frac{l.desperdicio}{l.alto \times W} \ , \tag{5.3.1}$$

donde *strip.alto* es la altura del patrón de empaquetado correspondiente a la permutación π, $l.desperdicio$ es el área del desperdicio reusable en el último nivel l, $l.alto$ es la altura del último nivel y W es el ancho de la plancha. El segundo operando de la ecuación indica la proporción de área desperdiciada en el último nivel respecto del área del último nivel ($strip.alto \times W$).

5.4. Operadores genéticos utilizados

En esta sección se describen los operadores de recombinación y mutación aplicados en nuestros algoritmos. Por un lado, la recombinación de dos soluciones tentativas deberá permitir intensificar la búsqueda en las regiones del espacio de búsqueda definida por los dos padres (operador con aridad 2). Por el otro lado, la mutación deberá permitir la diversificación de una solución (operador unario) para poder escapar de un óptimo local y también introducir diversidad genética en la población. Esta última se puede perder gradualmente a medida que el algoritmo converge durante la búsqueda.

En la sección 5.4.1 se presentan varios operadores de recombinación clásicos para permutaciones, incluyendo la descripción completa de un nuevo operador de recombinación desarrollado para 2SPP que incluye información del problema en su procedimiento. En la sección 5.4.2 se describen los operadores de mutación aplicados para realizar la búsqueda.

5.4.1. Operadores de recombinación propuestos

En esta sección describimos los cinco operadores de recombinación estudiados en este trabajo. Cuatro de estos operadores fueron propuestos para una representación con permutaciones para resolver el problema del viajante de comercio. Estos operadores se centran

en combinar información de orden o adyacencia presente en los padres. Operadores como Partial-Mapped Crossover (PMX), Order Crossover (OX) y Cycle Crossover (CX) consideran la posición y orden de las piezas como opuesto ejes, es decir, enlaces entre las piezas. Mientras que la idea general del Edge Recombination (ER) es preservar la conexión de una pieza con otras. El principal problema de estos operadores es que no utilizan información sobre el problema que se está intentando resolver. Por consiguiente el quinto operador, denominado *Best Inherited Level recombination* (BILX), incorpora conocimiento específico del problema en su mecanismo a fin de reducir el área desperdiciada. A continuación explicaremos el funcionamiento de cada uno de ellos.

Considerando que empaquetados parciales residentes en el interior de una solución parecen ser los bloques de construcción naturales en este problema, nuestra intención es que la recombinación transmita estos bloques de construcción de padres a hijos.

El operador PMX fue propuesto por Goldberg y Lingle [101]. Está diseñado para preservar las posiciones absolutas de ambos padres. Para construir un hijo, elige una subsecuencia de un patrón de empaquetado del primer padre y la copia al hijo. Esta transferencia también define un conjunto de asignaciones entre las piezas que han sido copiadas y las piezas en las posiciones correspondientes en el segundo padre. A continuación, se copian los elementos restantes en las posiciones que ocupan en el segundo padre. Si una posición está ocupada por un elemento ya copiado desde el primer padre, se considera la pieza proporcionada por el conjunto de asignaciones. PMX es esencialmente una clase de crossover de dos puntos para representaciones con permutaciones, junto con un procedimiento de reparación especial para resolver la ilegitimidad causada por el crossover de dos puntos.

OX fue propuesto por Davis [66]. Para ello, se seleccionan dos puntos al azar de un padre, definiendo de esta manera una región de cruce. Esta región se transmite directamente a los hijos. Mientras que las posiciones restantes se llenan con los elementos que no pertenecen a esa región, en el orden de aparición en el segundo padre. Este operador se puede ver como una especie de variación del PMX utilizando un procedimiento diferente de reparación.

El operador CX, propuesto por Oliver et al. [175], preserva la información contenida en ambos padres. En CX el hijo hereda todos las piezas que se encuentran en la misma posición

en ambos padres. Luego, a partir de una posición sin asignar en el hijo elegida al azar, se selecciona un elemento de uno de los padres en forma aleatoria. Después de eso, se realizan asignaciones adicionales para garantizar que no se produce mutación implícita. Entonces, la posición no asignada del lado derecho se procesa de la misma forma hasta que todos los elementos han sido considerados.

EX fue desarrollado por Whitley et al. [220] y más tarde mejorado en [202]. Este operador transfiere relaciones entre piezas presentes en ambos padres. Esto se hace con la ayuda de una lista creada a partir de las relaciones en ambos padres. La lista establece, para cada pieza i, las piezas que están cerca de la pieza i en al menos uno de los padres.

El quinto y último operador, llamado *Best Inherited Level Recombination* (BILX), es uno nuevo (introducido en [197] como BIL) propuesto especialmente para adaptarse a este problema, incorporando en su procedimiento información sobre la distribución de las piezas. Este operador transmite los mejores niveles (grupos de piezas que definen un nivel en el patrón de empaquetado) de uno de los padres a la descendencia; es decir, trasmite aquellos niveles con el menor desperdicio. De esta manera, los niveles heredados podrían mantenerse o incluso capturar algunas piezas de sus niveles vecinos, dependiendo de cuán compactos son los niveles.

El operador BILX funciona de la siguiente manera. Sea nl la cantidad de niveles en uno de los padres $Padre_1$. En una primera etapa, se calcula el desperdicio resultante para cada uno de los niveles nl de $Padre_1$. Luego, se asigna a cada nivel un valor de probabilidad de selección, inversamente proporcional a sus valores de desperdicio. Con estas probabilidades se seleccionan $nl/2$ niveles de $Padre_1$ empleando selección proporcional al desperdicio. Las piezas π_i pertenecientes a los niveles seleccionados se colocan en las primeras posiciones del hijo. Mientras que las posiciones restantes se llenan con las piezas que no pertenecen a esos niveles, en el orden en que aparecen en el otro padre $Padre_2$. La Figura 5.4 presenta un ejemplo de la transferencia de niveles en el curso de la operación de recombinación y también el proceso de llenado de las posiciones restantes.

En realidad, el operador BILX difiere del propuesto por Puchinger y Raidl en [179]. Nuestro operador transmite los mejores niveles de uno de los padres y el resto de las piezas

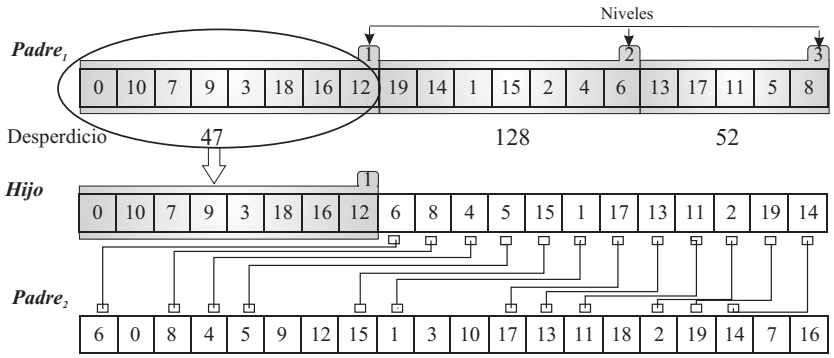


Figura 5.4: Transferencia de niveles y piezas en el funcionamiento del operador BILX

se obtienen del otro padre, por lo tanto no es necesario considerar estrategias de reparación. Mientras que el operador de recombinación de Puchinger empaqueta todos los niveles de los dos padres, ordenados en forma decreciente de acuerdo a ciertos valores calculados como el producto de la suma de todos los valores de piezas contenidas en los niveles por un valor aleatorio. Tras copiar todos los niveles, suele ser necesario un procedimiento de reparación a fin de garantizar la factibilidad del hijo generado (en este caso, para eliminar duplicados y considerar restricciones de ramificación), convirtiéndose en un mecanismo más complejo y costoso que nuestro BILX.

5.4.2. Operadores de mutación propuestos

A continuación describiremos los cuatro operadores de mutación usados en nuestra propuesta, los cuales fueron introducidos en [197], y que son específicos para resolver 2SPP. La mutación permite preservar la diversidad genética en sistemas biológicos, pero desde el punto de vista de optimización, no permite que la población tenga una convergencia prematura hacia un óptimo local.

El más simple de los operadores es *intercambio de piezas (Piece Exchange - PE)*, el cual selecciona dos piezas en forma aleatoria en un cromosoma e intercambia sus posiciones. Otro operador similar a PE es *intercambio de niveles (Level Exchange* – SE), pero en este caso considerando niveles en su procedimiento. Para ello elige dos niveles del patrón de empaquetado en forma aleatoria y los intercambia en el cromosoma.

Otra de las mutaciones propuestas es *intercambio del mejor y peor nivel (Best and Worst Level Exchange – BW_SE)*, las piezas del mejor nivel, aquel con el menor desperdicio, son reubicadas en las primeras posiciones del cromosoma, mientras que los piezas del peor nivel se mueven al final.

Finalmente, la mutación *reubicar las piezas del último nivel (Last Level Rearrange - LLR)* toma las piezas pertenecientes al último nivel del patrón de empaquetado y las trata de reacomodar en otros niveles. Para ello, toma la primer pieza π_i del último nivel y analiza todos los niveles del patrón de empaquetado (siguiendo una heurística NFMH) tratando de hallar un lugar para esa pieza. Si la búsqueda no es exitosa, la pieza π_i no se mueve. Este proceso se repite por cada una de las piezas pertenecientes al último nivel. Durante este proceso, las posiciones de las piezas dentro del cromosoma son consistentemente reacomodadas. La Figura 5.5(b) presenta un ejemplo de cómo se transformaría la disposición de las piezas al aplicar LLR, considerando como base el patrón de empaquetado anteriormente mostrado en la Figura 5.1 (Figura 5.5(a)). En el patrón de empaquetado modificado, las piezas sombreadas pertenecen al último nivel del patrón original, el cual ha desaparecido tras la mutación produciendo una reducción de la altura total de la plancha.

En los dos últimos operadores de recombinación propuestos, los movimientos pueden ayudar a que los niveles involucrados, o sus vecinos, puedan tomar piezas de los niveles próximos, reduciendo así su desperdicio.

5.5. Operadores de relocalización

Además de los operadores de recombinación y mutación utilizados para resolver 2SPP, proponemos un nuevo operador para el problema particular atendiendo a sus restricciones, denominado *operador de relocalización*. En concreto, su función es la de mejorar la solución obtenida tras la recombinación y la mutación. Hemos desarrollado dos versiones de este operador: una (*MFF_Adj*) consiste en la aplicación de una versión modificada de la heurística FFDH, y la otra (*MBF_Adj*) se basa en la aplicación de una versión adaptada de la heurística BFDH. Ambas heurísticas fueron descritas previamente en la Sección 4.2.

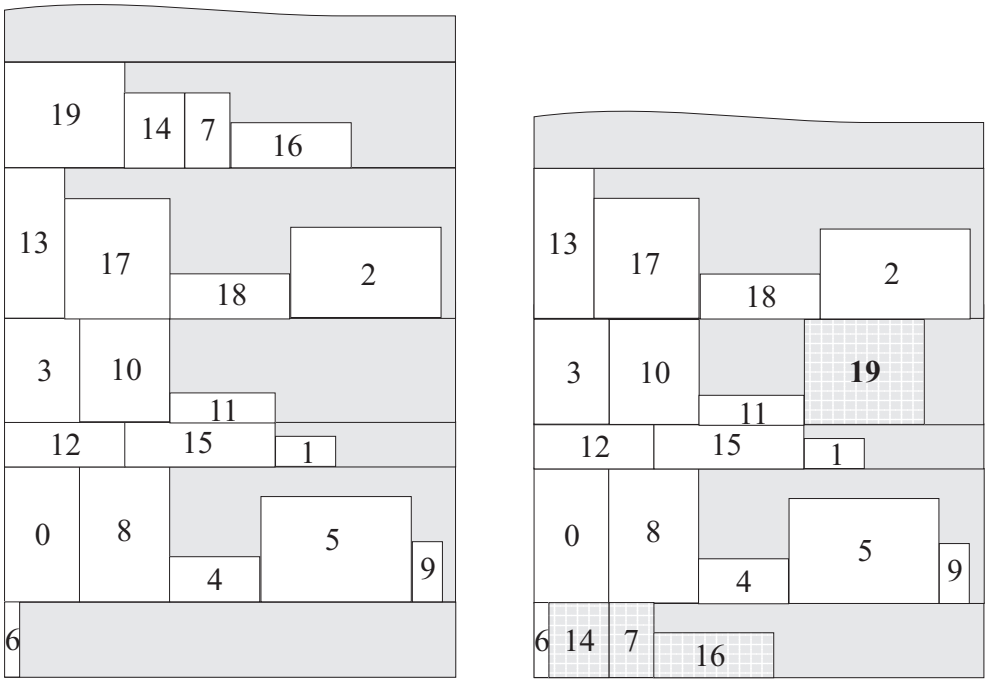


Figura 5.5: Posible mejora en la distribución de las piezas al aplicar la mutación LLR. (a) Patrón de empaquetado correspondiente a la permutación π, (b) modificación tras la aplicación de la mutación LLR

La versión MFF_Adj del operador de relocalización trabaja de la siguiente manera. Este operador considera las piezas en el orden dado por la permutación π. La pieza π_i es asignada en el primer nivel donde exista suficiente espacio en una pila existente o en una nueva, siempre respetando las restricciones del problema. Si no se halla espacio, se crea una nueva pila para contener a π_i en un nuevo nivel en la parte restante del *strip*. El proceso anterior se repite hasta asignar todas las piezas de π.

Por su parte, MBF_Adj procede de una manera similar, pero con la diferencia de que una pieza π_i se asigna en el nivel (entre los que puedan acomodarla siguiendo las restricciones del problema) con el menor desperdicio, ya sea en una pila existente o en una nueva. Estos pasos se repiten hasta ubicar todas las piezas de la permutación π.

Tanto en MFF_Adj como en MBF_Adj, como paso final se debe producir una reorganización adecuada del cromosoma para que refleje el nuevo patrón de empaquetado, de forma tal que se pueda llegar el mismo patrón tras aplicar la heurística NFMH (utilizada

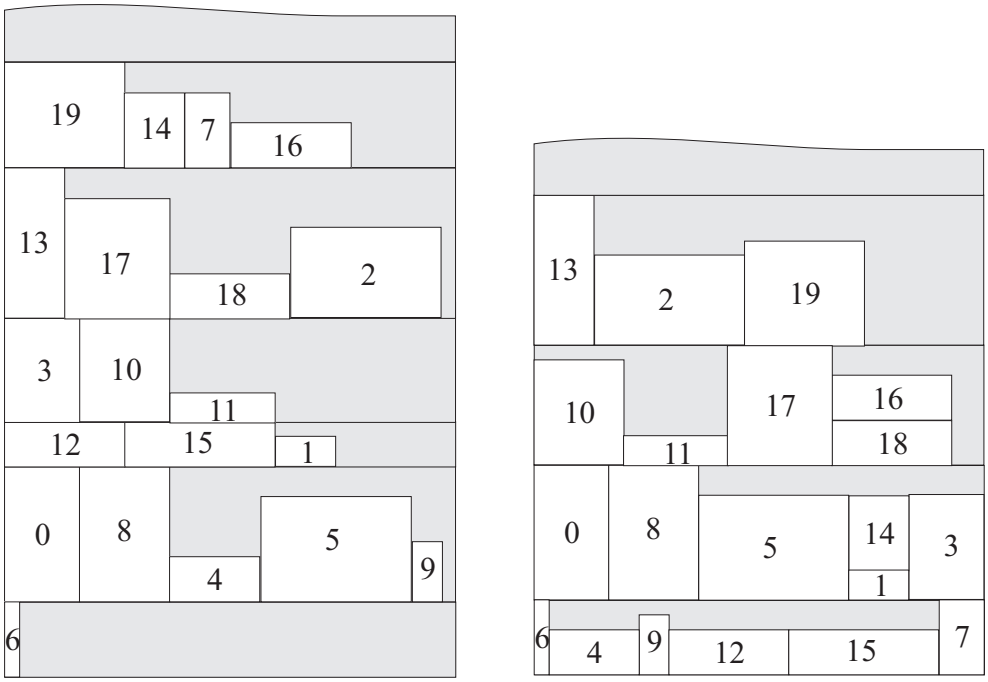


Figura 5.6: Posible mejora en la distribución de las piezas al aplicar el operador MFF_Adj: (a) Patrón de empaquetado correspondiente a la permutación π, (b) modificación tras la aplicación del operador MFF_Adj

en la función de evaluación) al cromosoma. La Figura 5.6 muestra un ejemplo de cómo se modificaría el patrón de empaquetado de la Figura 5.1 tras aplicar el operador MFF_Adj. En ese caso, la permutación π se debe modificar para reflejar la nueva secuencia de piezas y quedaría como π=(6,4,9,12,15,7,0,8,5,1,14,3,10,11,17,18,16,13,2,19).

5.6. Inicialización de la población

La inicialización de la población determina el proceso de creación de los individuos para el primer ciclo del algoritmo. Normalmente, la población inicial se forma de individuos creados aleatoriamente. El rendimiento de un GA es a menudo relacionado con la calidad de esa población inicial. La calidad está dada en dos medidas importantes: la aptitud promedio de los individuos y la diversidad en la población. Al tener una población inicial con mejores

Tabla 5.1: Reglas de construcción para generar la población inicial

#	**Descripción de las Reglas**
1	Ordena en forma decreciente las piezas considerando el ancho.
2	Ordena en forma creciente las piezas considerando el ancho.
3	Ordena en forma decreciente las piezas considerando la altura.
4	Ordena en forma creciente las piezas considerando la altura.
5	Ordena en forma decreciente las piezas considerando el área.
6	Ordena en forma creciente las piezas considerando el área.
7	Ordena las piezas en forma decreciente considerando en forma alternada anchos y alturas.
8	Ordena las piezas alternando entre anchos decrecientes y alturas crecientes.
9	Ordena las piezas en forma creciente considerando en forma alternada anchos y alturas.
10	Ordena las piezas alternando entre anchos crecientes y alturas decrecientes.
11	Reorganiza las piezas siguiendo una heurística BFDH modificada.
12	Reorganiza las piezas siguiendo una heurística FFDH modificada.

valores de *fitness*, se pueden encontrar con mayor rapidez mejores individuos finales [5, 6, 190]. Además, la alta diversidad de la población impide la convergencia prematura a un óptimo local.

Hay muchas formas de manejar esta diversidad inicial. La propuesta que se sigue en este trabajo es comenzar con una población sembrada con cromosomas buenos, denominados *semillas*, creados siguiendo algunas reglas simples de construcción, propuestas para el problema concreto que se está abordando. Es de esperar que esta incorporación de semillas a la población inicial nos permita obtener buenos patrones de empaquetado en las primeras etapas de la búsqueda, es decir, se logre una evolución más rápida. Estas reglas incluyen conocimiento del problema, como dimensiones de las piezas, y también incorporan ideas de las heurísticas BFDH y FFDH.

Con esta propuesta, los individuos de la población inicial se generan en dos pasos. En el primero, los cromosomas se producen por un muestreo aleatorio del espacio de búsqueda siguiendo una distribución uniforme. Luego, cada cromosoma se somete a un paso de modificación al aplicar alguna de las reglas de construcción que describiremos a continuación, con el objetivo de mejorar la ubicación de las piezas en el patrón de empaquetado correspondiente.

La Tabla 5.1 lista las reglas de construcción utilizadas para generar la población inicial. El objetivo de estas reglas propuestas es producir individuos con buenos valores de *fitness* y también introducir diversidad en la población inicial. De esta manera, ordenar las piezas

considerando su ancho (Reglas 1 y 2) deberá incrementar la probabilidad de apilar piezas y, en consecuencia, producir niveles más densos. Por otra parte, ordenar las piezas por altura (Reglas 3 y 4) generará niveles con menor desperdicio de espacio en cada una de las pilas de esos niveles, especialmente cuando las alturas de las piezas son muy similares. Las reglas 11 y 12 reubicarán las piezas con el objetivo de reducir el espacio desperdiciado en cada uno de los niveles. Finalmente, las reglas comprendidas entre la 7 y la 10 han sido introducidas para incrementar al diversidad inicial. De esta manera, las reglas propuestas no sólo son útiles para generar la población inicial, sino que varias de ellas se pueden usar como simples algoritmos voraces para realizar una búsqueda local durante el proceso de optimización. El operador MBF_Adj es un ejemplo, ya que está basado en la regla 11. El operador MFF_Adj por su parte es otro ejemplo basado en la regla 12.

5.7. Hibridación con SA

Recientemente, se han producido diferentes intentos para combinar ideas y métodos tanto de opciones de búsqueda local como de metaheurísticas, resultando en algoritmos híbridos. Una de las formas más comunes de metaheurísticas híbridas es incorporar optimización local como un paso extra al ciclo de la metaheurística. Con la opción híbrida, la optimización local se aplica a cada solución nueva generada para moverla a un óptimo local. Por lo tanto, GA se usa para realizar la exploración global, mientras que el algoritmo adicional de búsqueda local se usa para realizar la exploración local alrededor de una solución particular. Debido a las propiedades de las metaheurísticas consideradas en este trabajo y los algoritmos de búsqueda local, la opción híbrida frecuentemente mejora a ambas opciones trabajando por separado.

Uno de los procedimientos de búsqueda local usados en este trabajo de tesis para hibridar GA consiste en aplicar las dos versiones del operador de relocalización, introducido y explicado en la Sección 5.5. Además, se considera otra técnica para hibridar GA como lo es recocido simulado (SA). GA utiliza en cada generación a SA como un operador evolutivo, combinación que ha probado ser exitosa en [53]. La razón de esta selección de algoritmos es

que, mientras GA ubica buenas regiones del espacio de búsqueda (exploración), SA permite la explotación de las mejores regiones halladas por GA.

5.8. Experimentos

En esta sección presentamos la experimentación realizada para analizar el comportamiento de los operadores genéticos y de relocalización propuestos, utilizando las instancias del problema de empaquetado presentadas en la Sección 3.5. Se analiza, además, el comportamiento de GA utilizando los distintos métodos planteados para generar la población inicial, así como también al hibridar GA con métodos de búsqueda local. Todas estas propuestas se han comparado en términos de la calidad de sus resultados. Además, se orientó el análisis del funcionamiento del algoritmo a fin de conocer la relación entre valores de *fitness* y diversidad. Comenzamos detallando la configuración utilizada por GA en la Sección 5.8.1. Los resultados se han dividido en cuatro apartados. En primer lugar, hemos comparado todas las posibles combinaciones de operadores de recombinación (PMX, OX, CX, EX y BILX) y mutación (PE, SE, BW_SE y LLR) propuestos en este trabajo, a fin de poder determinar la mejor combinación de operadores para el problema en estudio (Sección 5.8.2). El siguiente paso (Sección 5.8.3) es evaluar cómo se ve afectado el comportamiento de GA desde un punto de vista de calidad de soluciones al incorporar un operador de relocalización. En la Sección 5.8.4 se analiza el comportamiento de un GA utilizando los distintos procedimientos para generar la población inicial para determinar si poblaciones iniciales mejor adaptadas permiten mejorar el rendimiento del algoritmo. Para terminar, la Sección 5.9 concluye con los resultados obtenidos al hibridar GA con SA a fin de comprobar su influencia en el proceso de búsqueda.

5.8.1. Configuración del algoritmo

Para resolver 2SPP usamos un GA de estado estacionario (ver Algoritmo 4) con una población de 512 individuos. Por defecto, la población inicial se genera en forma aleatoria. La selección de padres se realiza por torneo binario. El hijo generado en cada paso reemplaza al peor de la población, en caso de que fuese mejor (reemplazo elitista). El criterio de parada

es alcanzar un total de 2^{16} evaluaciones. Los operadores de recombinación se aplican con una probabilidad de 0,8, mientras que la probabilidad de mutación es 0,1. Los operadores de relocalización se aplican a todas las nuevas soluciones generadas.

El algoritmo fue implementado usando el paquete MALLBA [16], una librería de software implementada en C++ que permite un rápido prototipado de algoritmos híbridos y paralelos. Las máquinas utilizadas para los experimentos son Pentium 4 a 2.4 GHz con 1 GB de RAM y sistema operativo Linux (versión del kernel 2.4.19-4GB).

Para cada variante del algoritmo y por cada una de las instancias, realizamos 30 ejecuciones independientes. A fin de obtener conclusiones significativas, realizamos un análisis estadístico de los resultados. Si los resultados siguen una distribución normal, usamos la prueba de t para probar diferencias entre dos medias, sin embargo si pretendemos evaluar si existen diferencias entre dos o más muestras deberemos recurrir al análisis de varianza (ANOVA). Consideramos $\alpha = 0{,}05$, a fin de indicar un nivel de confianza del 95 % en los resultados. Si los resultados no siguen una distribución normal, usamos la prueba no paramétrica de Kruskal Wallis, para distinguir diferencias significativas entre las medianas de los resultados de cada algoritmo. Este tipo de análisis la realizaremos a lo largo del trabajo de tesis.

Por otra parte, la evaluación considera dos medidas importantes para cualquier proceso de búsqueda: la capacidad para generar nuevas soluciones prometedoras y la habilidad para mantener diversificada una población. Para realizar este análisis, consideramos el avance del valor de *fitness* medio y de la medida de entropía propuesta en [105], esta última se calcula como sigue:

$$entropia(P) = \frac{\sum_{i=1}^{M} \sum_{j=1}^{M} (\frac{n_{ij}}{\mu}) \ln(\frac{n_{ij}}{\mu})}{M \ln M} \ , \tag{5.8.1}$$

donde n_{ij} representa las veces que la pieza i está en la posición j en la población P de tamaño μ. Esta función toma valores en el intervalo [0,1]. Un valor de entropía igual a cero indica que todos los individuos en la población son idénticos.

Las Tablas 5.2 a 5.7 muestran el mejor valor de *fitness* obtenido (columna *mejor*) y el *fitness* promedio de las mejores soluciones halladas en cada ejecución con su desviación

típica (columna $media_{\pm\sigma}$). A los valores mínimos de la columna *mejor* los hemos resaltado en negritas. Además las Tablas 5.4 y 5.7 incluyen un promedio de las evaluaciones para alcanzar el mejor *fitness* (column $eval_m$), lo cual representa el esfuerzo numérico del algoritmo. La Tabla 5.3 también muestra los tiempos medios (en segundos) incurrido en la búsqueda de la mejor solución (columna T_m) y en la búsqueda total (columna T_t).

5.8.2. Comparación de los operadores genéticos

Los primeros resultados que incluimos en este capítulo miden la calidad de las soluciones obtenidas por GA empleando los distintos operadores de recombinación y de mutación. Los distintos experimentos están listados en las Tablas 5.2 y 5.3. El objetivo del presente estudio es determinar la mejor combinación de operadores para resolver el problema bajo estudio.

El primer análisis con el que empezaremos esta discusión va a estar centrado en comparar nuestro operador de recombinación y los tradicionales de permutaciones. Como se puede observar en la Tabla 5.2 los resultados claramente muestran que el algoritmo genético con el operador BILX propuesto mejora de forma notable los GAs usando cualquier otra recombinación, para todas las instancias. Es de destacar que el mejor valor que alcanza GA aplicando BILX es, en todas las ejecuciones, mejor que cualquiera de los mejores valores obtenidos con el resto de los operadores, además de ser más robusto. Este análisis se desprende al comparar los valores de la columna $media_{\pm\sigma}$ para BILX y las respectivas columnas de cada uno de los otros operadores. Estas diferencias son más evidentes a medida que se incrementan las dimensiones de las instancias. Usando la prueba estadística Kruskal Wallis, hemos verificado que las diferencias entre los resultados son estadísticamente significativas. El ordenamiento parcial que se origina con esta métrica es: BILX, OX, PMX, EX y CX.

La mejora experimentada al usar el operador BILX se debe al hecho que, de alguna manera, explota la idea detrás de bloques de construcción; en este caso, definidos para el problema particular en estudio, como un nivel del patrón de empaquetado, es decir, como un grupo de piezas. También se atribuye a que tiende en la conservación de buenos niveles en el hijo durante la recombinación. Como resultado, BILX es el encargado de mezclar bloques buenos que se encuentren en los progenitores y que serán los que den a los mismos buenos

Tabla 5.2: Resultados experimentales de GA con los distintos operadores genéticos

Inst	Mut.	PMX		OX		BILX		EX		CX	
		mejor	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**
100	PE	255,44	270,48 $\pm 6,44$	244,76	261,67 $\pm 8,77$	**232,73**	243,63 $\pm 5,85$	247,74	273,01 $\pm 9,73$	263,54	279,16 $\pm 7,69$
	SE	254,64	263,86 $\pm 5,90$	241,70	260,00 $\pm 8,98$	**234,66**	240,21 $\pm 4,81$	266,50	283,94 $\pm 7,87$	271,49	291,68 $\pm 10,81$
	LLR	261,71	281,14 $\pm 8,16$	248,68	265,45 $\pm 7,69$	**237,71**	249,56 $\pm 5,51$	288,57	304,68 $\pm 9,59$	315,54	332,78 $\pm 8,63$
	BW_SE	255,65	272,38 $\pm 9,52$	245,59	260,28 $\pm 7,53$	**234,71**	247,44 $\pm 6,02$	271,03	294,02 $\pm 12,07$	307,55	321,51 $\pm 8,07$
150	PE	286,52	301,46 $\pm 7,31$	280,51	292,82 $\pm 8,62$	**241,80**	251,90 $\pm 5,51$	284,56	312,80 $\pm 12,00$	298,04	320,38 $\pm 11,58$
	SE	271,73	291,00 $\pm 10,88$	279,53	294,99 $\pm 8,47$	**238,43**	247,94 $\pm 5,30$	300,47	331,23 $\pm 17,08$	322,38	340,61 $\pm 10,25$
	LLR	288,54	317,63 $\pm 13,31$	274,55	289,99 $\pm 7,18$	**241,82**	253,96 $\pm 5,40$	336,59	366,07 $\pm 16,30$	393,37	411,69 $\pm 10,38$
	BW_SE	291,54	306,74 $\pm 11,50$	270,70	290,45 $\pm 8,50$	**243,73**	253,45 $\pm 5,82$	319,64	361,27 $\pm 17,13$	371,67	394,68 $\pm 11,15$
200	PE	283,79	300,10 $\pm 10,60$	276,44	291,38 $\pm 7,12$	**245,79**	254,00 $\pm 4,64$	289,34	310,98 $\pm 12,56$	290,68	317,73 $\pm 11,05$
	SE	266,52	290,28 $\pm 10,48$	274,27	287,63 $\pm 7,38$	**243,52**	250,94 $\pm 4,82$	296,49	319,52 $\pm 11,65$	303,53	327,98 $\pm 12,81$
	LLR	277,79	310,87 $\pm 12,72$	280,49	291,65 $\pm 7,64$	**246,67**	257,10 $\pm 4,87$	322,03	358,33 $\pm 14,50$	374,46	398,99 $\pm 7,74$
	BW_SE	282,65	302,62 $\pm 11,10$	262,17	283,16 $\pm 8,46$	**248,77**	259,62 $\pm 4,88$	293,56	340,11 $\pm 17,87$	341,51	380,50 $\pm 14,55$
250	PE	292,71	316,25 $\pm 10,06$	292,44	310,11 $\pm 8,28$	**243,86**	253,95 $\pm 5,91$	318,62	332,68 $\pm 10,58$	319,63	334,42 $\pm 8,34$
	SE	288,32	304,95 $\pm 10,47$	289,40	302,68 $\pm 5,94$	**241,68**	249,32 $\pm 4,19$	312,29	332,68 $\pm 10,83$	319,29	336,40 $\pm 9,71$
	LLR	294,27	324,28 $\pm 11,90$	288,53	308,31 $\pm 10,67$	**241,81**	257,72 $\pm 7,06$	348,53	378,55 $\pm 13,81$	377,60	411,80 $\pm 9,58$
	BW_SE	303,81	324,53 $\pm 10,56$	279,51	300,69 $\pm 10,17$	**242,87**	257,22 $\pm 6,96$	320,61	366,73 $\pm 18,65$	346,57	386,20 $\pm 13,56$
300	PE	325,12	339,06 $\pm 8,95$	297,51	324,80 $\pm 11,10$	**253,74**	263,27 $\pm 6,46$	325,71	349,15 $\pm 11,88$	339,41	355,93 $\pm 10,69$
	SE	294,16	322,09 $\pm 11,81$	294,45	322,50 $\pm 12,13$	**241,04**	258,10 $\pm 6,10$	330,23	353,56 $\pm 11,72$	337,41	364,88 $\pm 12,81$
	LLR	317,71	347,10 $\pm 13,29$	301,72	323,64 $\pm 10,43$	**251,78**	264,34 $\pm 7,46$	379,75	411,89 $\pm 14,52$	411,54	445,72 $\pm 9,70$
	BW_SE	309,55	335,86 $\pm 12,28$	294,50	313,86 $\pm 9,88$	**257,37**	263,51 $\pm 4,91$	344,25	385,13 $\pm 19,70$	360,64	423,50 $\pm 16,34$

valores de *fitness*. Por consiguiente, BILX puede descubrir y favorecer versiones compactas de bloques de construcción útiles.

Analizando el número medio de evaluaciones para alcanzar el mejor valor (columna $eval_m$ en la Tabla 5.3), BILX presenta diferencias significativas en media cuando se lo compara con OX, siendo este último su sucesor más cercano en cuanto a mejores valores de *fitness* (ver Tabla 5.2). No obstante, los valores medios de BILX no presentan diferencias significativas cuando se lo compara con EX y CX. Estos análisis los hemos confirmados estadísticamente, para lo cual usamos las pruebas de comparaciones múltiples. Ordenando los operadores de más eficiente a menos eficiente, la lista resultante ha sido: BILX, CX, EX, PMX, y finalmente OX.

Tabla 5.3: Resultados experimentales de GA con los distintos operadores genéticos (cont.)

Inst	Mut.	**PMX**			**OX**			**BILX**			**EX**			**CX**		
		$eval_m$	T_m	T_t	$eval_m$	T_m	T_t	$eval_m$	T_m	T_t	$eval_m$	T_m	T_t	$eval_m$	T_m	T_t
100	PE	61.141	39	42	61.825	40	42	31.731	24	50	63.263	41	43	62.852	37	38
	SE	41.232	28	45	53.755	36	44	13.604	11	51	26.132	18	45	10.256	6	40
	LLR	15.496	11	45	55.181	39	47	6.804	5	53	7.367	5	47	1.077	0	43
	BW_SE	23.270	17	47	49.034	35	47	7.217	6	56	10.270	8	48	1.875	1	44
150	PE	62.792	67	70	61.658	67	71	33.775	42	82	63.602	68	70	64.162	63	64
	SE	54.339	63	76	57.396	66	75	19.784	26	85	21.307	24	74	15.657	16	68
	LLR	22.463	27	80	58.532	72	81	8.028	10	87	9.869	12	79	1.076	1	77
	BW_SE	29.006	37	84	55.586	71	83	7.162	11	105	9.839	12	81	1.791	2	77
200	PE	60.996	96	103	62.825	100	104	42.648	81	124	63.409	98	101	62.294	88	92
	SE	50.312	86	111	58.343	100	112	27.593	55	128	20.901	35	108	12.942	19	99
	LLR	25.198	45	115	58.190	105	119	10.609	21	131	9.417	17	116	932	1	113
	BW_SE	28.761	54	124	59.000	109	122	9.101	21	153	11.679	20	120	2.084	3	113
250	PE	62.056	136	144	61.874	139	147	44.282	119	176	63.122	135	140	63.326	125	129
	SE	50.732	123	158	59.214	143	158	29.226	84	185	25.063	58	151	17.537	37	139
	LLR	28.083	72	167	55.743	144	170	11.150	32	187	8.203	20	162	804	1	161
	BW_SE	26.316	70	174	57.419	152	174	9.368	31	220	11.123	27	166	2.462	5	160
300	PE	63.365	185	192	61276	183	196	31.716	115	236	63.512	178	184	63.694	166	170
	SE	53.980	174	211	58.020	186	211	31.656	122	248	26.184	80	198	16.933	47	184
	LLR	25.826	88	221	57.446	196	225	8.447	33	252	7.132	22	214	851	2	209
	BW_SE	30.331	108	236	56.889	202	233	9.582	42	286	14.390	46	218	2418	6	212

Por otra parte, la opción de búsqueda más rápida en cuanto a los tiempos de cómputo consiste en aplicar CX (ver la Tabla 5.3), debido a que su mecanismo para generar los hijos es simple. La opción algorítmica más lenta es la resultante de aplicar BILX; esto se debe al tiempo incurrido en la búsqueda de los mejores niveles para trasmitir, además de la búsqueda necesaria para completar el cromosoma con las piezas que no pertenecen a los niveles trasmitidos. Usando las pruebas de comparaciones múltiples, verificamos que PMX y OX presenta tiempos de ejecución similares, mientras que las diferencias entre los otros operadores de recombinación es significativa. Clasificando los operadores de más rápidos a más lentos, se obtiene: CX, OX/PMX, EX y BILX.

Aunque BILX presenta el tiempo de ejecución más alto, es importante resaltar que este operador es el que encuentra las mejores soluciones en un menor número de evaluaciones, traduciéndose en un menor tiempo computacional (ver columna T_m de la Tabla 5.3). Por otra parte, cabe destacar que es el operador que encuentra los mejores patrones de empaquetado (menores valores de *fitness*). Teniendo en cuenta las dos características resaltadas, podemos decir que BILX presenta un buen equilibrio entre tiempo y calidad de resultados. En conclusión, BILX es el operador de recombinación más adecuado para resolver 2SPP.

En cierta medida, una de las desventajas del uso de decodificadores en un algoritmo genético, en el sentido de que limitan al algoritmo, es reducida con el uso del operador de recombinación propuesto en este trabajo de tesis. Debido a que este operador toma en cuenta la distribución de piezas para transmitir información entre padres e hijos, utiliza características claves del patrón de empaquetado y permite mantener ciertas subestructuras presentes en la distribución de piezas y soporta la herencia de esas características importantes, las cuales son significativas y efectivas para el objetivo de empaquetado. De esta manera, logramos una mezcla entre usar una representación basada en permutaciones, con lo cual se esconden ciertas características del dominio de búsqueda en la heurística decodificadora, y un operador de recombinación que de cierta manera rescata esas características.

Pongamos ahora la atención en analizar los resultados obtenidos por los diferentes operadores de mutación con GA usando BILX. Para ello nos centraremos en la columna correspondiente a BILX de las Tablas 5.2 y 5.3. En promedio, SE exhibe algunas ventajas respecto a los mejores valores alcanzados (menores valores de *fitness*). Aquí nuevamente podemos inferir que el uso de niveles como bloques de construcción influye en la mejora de los resultados obtenidos. Las pruebas de ANOVA muestran que esas diferencias son significativas para todas las instancias del problema. Tanto BW_SE como LLR necesitan cantidades similares de evaluaciones para obtener sus mejores resultados. Comparando esa cantidad con las evaluaciones que necesita SE vemos que es aproximadamente la mitad, y en el caso de PE, la tercera parte. A pesar de esta rápida convergencia, BW_SE y LLR obtienen un rendimiento muy bajo si de calidad de soluciones se habla. La razón podría estar fundamentada en que esas mutaciones no introducen alteraciones a las distribuciones de las piezas una vez que se han alcanzado "buenas" soluciones, ya que estas últimas se corresponden con patrones de empaquetado donde la densidad de las piezas es muy alta. Por otro lado, SE y PE siempre modifican el patrón de empaquetado debido a que sus procesos incluyen componentes de selección aleatorias. Usando la pruebas estadística de comparaciones múltiples de medias, verificamos que BW_SE y LLR usan un número de evaluaciones promedio similares para alcanzar sus mejores valores mientras que las diferencias entre los otros dos operadores es significativa. El ordenamiento resultante es BW_SE/LLR, SE y PE.

Con respecto al tiempo incurrido en la búsqueda, podemos observar que los valores medios de los operadores SE y BW_SE no tienen diferencias significativas, debido al hecho que sus procedimientos difieren básicamente en el paso de selección de niveles. Por otro lado, LLR es el operador más lento para todas las instancias, ya que debe realizar una búsqueda exhaustiva en todos los niveles, tratando de hallar un lugar para reacomodar cada una de las piezas pertenecientes al último nivel. Como era de esperar, PE es el operador más rápido debido a lo simplista de su procedimiento. Todas estas observaciones han sido corroboradas estadísticamente en nuestro estudio.

Finalmente, en la tercer fase analizamos la relación entre exploración y explotación del espacio de búsqueda. Para lo cual se muestra el comportamiento de GA con los distintos operadores de mutación para la instancia M=200, pero similares resultados se obtienen con el resto de las instancias. La Figura 5.7 y la Figura 5.8 presentan la evolución de la entropía poblacional y el *fitness* medio poblacional (en el eje y), respectivamente, con respecto a la cantidad de evaluaciones (eje x). Desde estas figuras podemos ver que SE presenta valores de entropía más altos que el resto de los operadores, implicando mayor diversidad a nivel genético. Por otra parte la curva representando el *fitness* medio poblacional de SE está siempre por debajo de las restantes. De esta manera, se puede concluir que la mejor relación entre exploración y explotación la logra el operador SE.

Como resumen de este apartado, podemos concluir que la combinación de BILX y SE es la mejor de todas las analizadas, teniendo en cuenta calidad de soluciones junto con una baja cantidad de evaluaciones para alcanzar sus mejores valores y también con un tiempo de ejecución razonable. Tanto BILX y SE están basados en el concepto de bloques de construcción. Aquí un bloque de construcción es un grupo de piezas las cuales definen un nivel en el genotipo. Esto marca la diferencia con alguno de los operadores tradicionales, los cuales seleccionan en forma aleatoria el conjunto de piezas para intercambiar. Para 2SPP las posiciones absolutas de las piezas dentro del cromosoma no tiene importancia relativa. Los buenos valores de los patrones de empaquetado obtenidos, junto con la reducción en la cantidad de evaluaciones para encontrar las mejores soluciones, ponen de manifiesto que hemos diseñado operadores genéticos muy apropiados para resolver 2SPP.

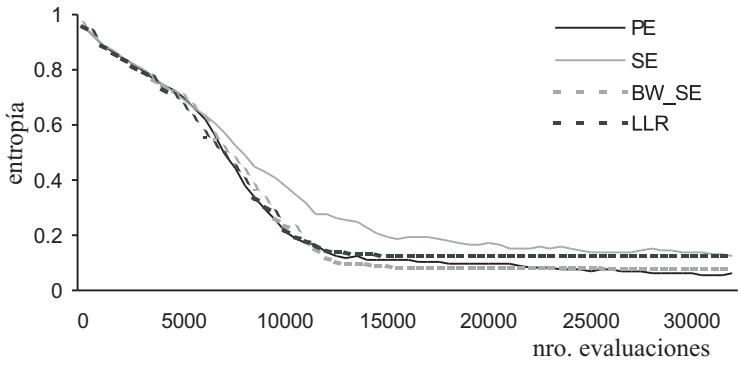


Figura 5.7: Entropía poblacional para BILX y cada operador de mutación (instancia con M=200)

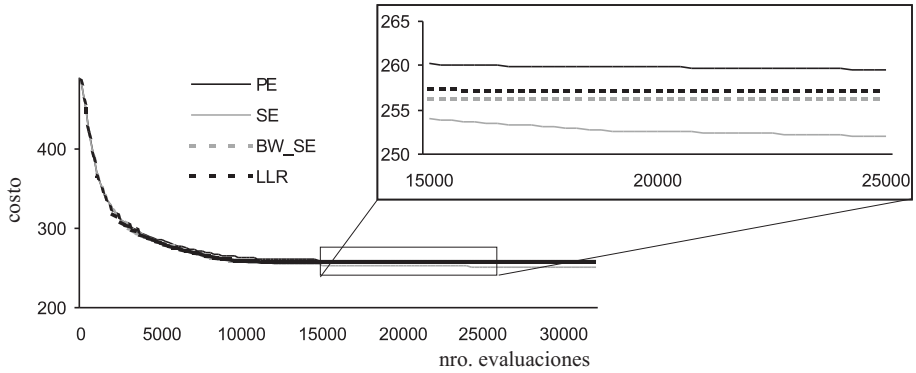


Figura 5.8: *Fitness* medio para BILX y cada operador de mutación (instancia con M=200)

5.8.3. Comparación de los operadores de relocalización

A fin de justificar el uso de los operadores de relocalización introducidos en la Sección 5.5, en esta sección presentamos los resultados obtenidos por tres GAs usando los operadores BILX y SE:

i) un GA puro, sin operadores de relocalización, referenciado como GA,

ii) un GA usando el operador MFF_Adj, al que denominaremos GA+MFF_Adj,

iii) un GA aplicando el operador MBF_Adj, denominado GA+MBF_Adj.

El objetivo del presente estudio es determinar si el uso del operador de relocalización favorece al proceso de búsqueda y permite mantener diversidad genética a lo largo del proceso. Los resultados de los algoritmos se presentan en la Tabla 5.4.

Tabla 5.4: Resultados experimentales para GA y GAs con operadores de relocalización

Inst	**GA**			**GA+MFF_Adj**			**GA+MBF_Adj**		
	mejor	***media*$\pm\sigma$**	$eval_m$	***mejor***	***media*$\pm\sigma$**	$eval_m$	***mejor***	***media*$\pm\sigma$**	$eval_m$
100	234,66	240,21 $\pm 4,81$	13.603,70	**212,78**	216,10 $\pm 1,45$	21.949,80	213,70	216,88 $\pm 2,15$	21.601,87
150	238,43	247,94 $\pm 5,30$	19.784,47	**210,89**	216,02 $\pm 1,77$	38.386,70	212,64	216,11 $\pm 1,64$	33.481,23
200	243,52	250,94 $\pm 4,82$	27.593,10	**207,69**	211,00 $\pm 1,81$	42.446,00	207,75	211,35 $\pm 1,69$	39.837,30
250	241,68	249,32 $\pm 4,19$	29.226,27	211,48	213,19 $\pm 1,16$	37.566,00	**210,83**	213,54 $\pm 1,37$	40.941,67
300	241,04	258,10 $\pm 6,10$	31.656,37	**207,24**	211,25 $\pm 1,84$	38.092,50	208,65	211,60 $\pm 1,44$	41.586,00

En todas las instancias analizadas del problema, se puede detectar una clara tendencia en la que se observa que las variantes genéticas incluyendo alguno de los operadores de relocalización mejoran notablemente la calidad de las soluciones obtenidas, al ser comparadas con GA. Hemos realizado pruebas estadísticas para estos resultados, que muestran que realmente la incorporación del operador de relocalización permite computar mejores soluciones que las obtenidas por GA.

Estos datos nos indican que la capacidad de búsqueda de GA+MFF_Adj y GA+MBF_Adj supera a la de GA para este problema. Esto es un resultado muy prometedor; la explicación es la mejora en la distribución de las piezas obtenida tras la aplicación de esos operadores, los cuales reacomodan las piezas con el objetivo de reducir el desperdicio en cada uno de los niveles.

Comparando entre sí los resultados de los GAs aplicando los operadores de relocalización, observamos en general que la calidad de las mejores soluciones en promedio no difieren. No obstante ello, las pruebas t-student (con un nivel de confianza del 95 %) revelan que esas diferencias en término de calidad de soluciones son significativas, con una leve diferencia a favor de GA+MFF_Adj.

Finalmente queremos probar nuestra hipótesis respecto a que este operador mantiene diversidad genética. La Figura 5.9 muestra que ambos algoritmos usando operadores de relocalización decrementan en forma similar los valores de entropía hasta la evaluación 17000. A partir de ese punto, GA+MFF_Adj mantiene valores de entropía más altos que GA+MBF_Adj. Un punto interesante para resaltar es cómo la incorporación de los operadores de relocalización permiten mantener buenos niveles de diversidad genética a través

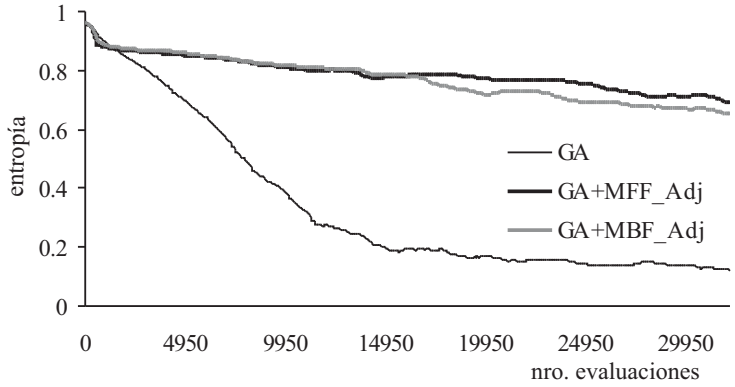


Figura 5.9: Entropía poblacional de todos los algoritmos (instancia con M=200)

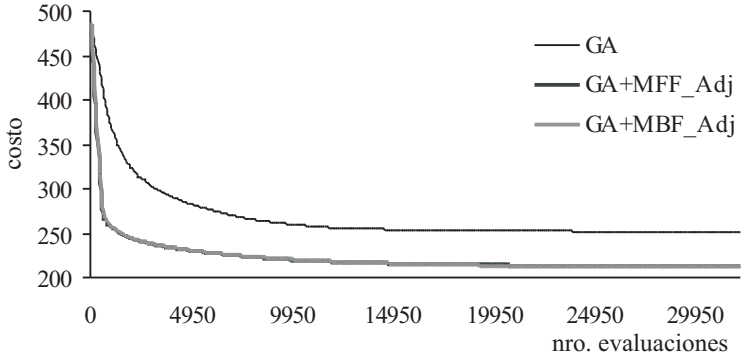


Figura 5.10: *Fitness* medio poblacional de todos los algoritmos (instancia con M=200)

del proceso del búsqueda. La Figura 5.10 muestra que las curvas correspondientes a los GAs usando operadores de relocalización son cercanas, mientras que GA puro muestra el peor comportamiento (pérdida de diversidad genética). En conclusión, GA+MFF_Adj provee la mejor relación entre exploración y explotación del espacio de búsqueda.

Resumiendo, podemos concluir que la aplicación del operador de relocalización durante el proceso de búsqueda permite mejorar el rendimiento de GA al hallar buenos patrones de empaquetado. En particular, el operador MFF_Adj debido a que resulta más rápido que MBF_Adj.

5.8.4. Resultados con inicialización de la población

Hasta este punto del análisis, los GAs estudiados comenzaban con una población de patrones de empaquetado generados en forma aleatoria. En esta sección analizaremos los resultados obtenidos cuando la población inicial se siembra con soluciones generadas usando ciertas reglas descritas en la Sección 5.6, las cuales en este caso incorporan conocimiento específico del problema. El objetivo del estudio es determinar si se observa una mejora en el rendimiento de GA al usar una población inicial mejorada. Para tal fin, estudiaremos GAs usando los operadores BILX y SE (debido a los buenos resultados mostrados en secciones previas) combinado con tres métodos diferentes de generar la población inicial:

I) por medio de la generación aleatoria de los μ patrones de empaquetado (GA),

II) por medio de generar los μ cromosomas utilizando una de las reglas de construcción de la Tabla 5.1 (GA_i donde i identifica un número de regla)

III) por medio de aplicar una regla de construcción (ver la Tabla 5.1), elegida aleatoriamente, para cada uno de los μ cromosomas que conforman la población inicial (GA_{Rseed}). La generación aleatoria de un patrón de empaquetado también se considera otra regla aplicable en este caso.

La Tabla 5.5 muestra los resultados obtenidos por las diferentes estrategias de generación de la población inicial para todas las instancias. Para marcar las diferencias en las poblaciones iniciales pertenecientes a las distintas propuestas, incluimos en esta tabla información relacionada con el valor de *fitness* medio de la población inicial. Los resultados claramente indican que cualquier GA con semillas en la población inicial comienza el proceso de búsqueda con mejores patrones de empaquetado, determinado por los valores de *fitness* más bajos, que un GA con población inicial generada en forma aleatoria. La Tabla 5.6 muestra los resultados para los distintos GAs teniendo en cuenta la calidad de los resultados finales a los que han arribado.

En promedio, las mejores poblaciones iniciales (en cuanto a calidad de soluciones) las obtiene el algoritmo GA_4, pero tales poblaciones presentan una pobre diversidad genética

Tabla 5.5: Valores promedio de *fitness* de las poblaciones iniciales de los distintos GAs usando diferentes estrategias de generación de la población inicial

Alg	$M = 100$	$M = 150$	$M = 200$	$M = 250$	$M = 300$
GA	417,26	504,83	485,36	488,54	531,41
GA_1	353,30	408,04	381,72	354,59	409,64
GA_2	298,17	368,69	352,22	331,22	345,29
GA_3	325,88	274,10	294,71	284,16	273,27
GA_4	282,92	**240,37**	**244,53**	**239,70**	**239,11**
GA_5	381,35	427,78	415,78	377,27	401,09
GA_6	371,87	377,75	359,10	357,14	375,08
GA_7	396,65	486,68	452,28	412,23	454,39
GA_8	299,20	274,69	275,21	262,78	259,63
GA_9	384,48	499,78	460,07	450,02	486,95
GA_{10}	353,97	353,53	369,00	321,23	336,15
GA_{11}	**277,93**	290,17	278,52	273,96	283,35
GA_{12}	281,64	294,10	282,66	277,04	288,46
GA_{Rseed}	340,35	368,81	358,60	339,49	360,13

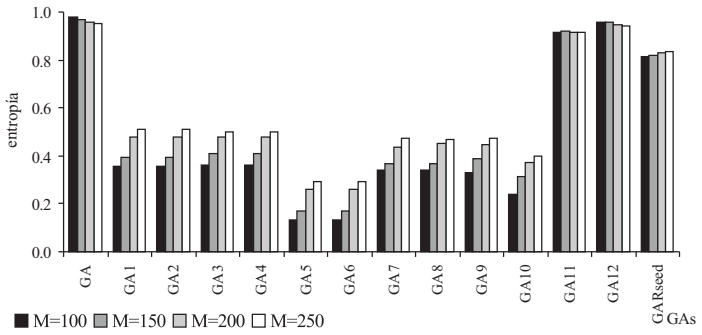


Figura 5.11: Valores de entropía de las poblaciones iniciales

(ver los valores de entropía media para la población inicial en la Figura 5.11). La regla de construcción 4 ordena las piezas por su altura, por lo tanto las piezas en cada nivel tienen alturas similares y, por consiguiente, el espacio desperdiciado en cada nivel tiende a reducirse. La baja diversidad genética inicial se debe a que la regla tiende reestructurar todas las soluciones de la misma manera. Por otro lado, se observa un pobre rendimiento tanto con GA_7 como con GA_9, ya que presentan las poblaciones iniciales con peores medias poblacionales además de tener un bajo grado de diversidad genética. Ambas reglas en su segunda fase reordenan las piezas considerando en forma alternada las dimensiones (anchos y alturas) de las piezas, por ende los patrones de empaquetado resultantes están formados por niveles donde su espacio interior está mal optimizado.

Tabla 5.6: Resultados experimentales de los GAs usando diferentes estrategias de generación de la población inicial

Alg	$M = 100$		$M = 150$		$M = 200$		$M = 250$		$M = 300$	
	mejor	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$
GA	234,66	240,21 ± 4,81	238,43	247,94 ± 5,30	243,52	250,94 ± 4,82	241,68	249,32 ± 4,19	241,04	258,10 ± 6,10
GA_1	229,71	241,84 ± 9,37	231,73	243,70 ± 5,97	223,71	240,08 ± 6,15	230,79	237,54 ± 3,45	234,70	248,01 ± 5,78
GA_2	229,70	235,63 ± 5,51	236,73	245,88 ± 4,10	229,57	240,29 ± 4,59	227,74	237,90 ± 4,55	241,24	246,00 ± 3,35
GA_3	229,63	233,85 ± 4,98	226,39	229,17 ± 1,88	222,54	228,01 ± 2,48	222,41	228,08 ± 2,43	227,79	236,89 ± 3,45
GA_4	226,70	228,80 ± 0,83	226,18	227,82 ± 0,90	221,36	225,32 ± 2,04	221,59	224,80 ± 1,51	224,79	230,62 ± 2,65
GA_5	240,60	251,90 ± 5,37	257,62	271,73 ± 6,99	249,63	261,49 ± 5,28	242,56	255,23 ± 5,37	255,64	268,16 ± 5,92
GA_6	249,54	260,05 ± 5,40	257,61	267,92 ± 4,14	241,37	253,18 ± 5,91	248,56	258,88 ± 4,65	272,59	286,64 ± 5,30
GA_7	236,68	244,61 ± 6,47	239,61	258,42 ± 7,75	236,61	249,62 ± 6,61	243,27	251,62 ± 4,74	260,31	273,58 ± 5,63
GA_8	229,72	238,15 ± 3,58	241,49	246,04 ± 1,70	233,68	238,32 ± 1,62	231,21	234,61 ± 1,62	241,63	244,70 ± 1,48
GA_9	233,74	246,34 ± 6,22	245,62	258,00 ± 6,09	237,29	250,37 ± 5,15	246,60	255,76 ± 5,91	259,57	272,17 ± 5,01
GA_{10}	237,33	246,60 ± 6,74	255,68	272,59 ± 7,13	241,35	251,28 ± 5,45	240,66	253,39 ± 5,85	269,65	275,64 ± 3,74
GA_{11}	229,32	235,17 ± 2,97	232,77	241,57 ± 4,02	227,66	233,69 ± 3,01	226,68	232,63 ± 2,79	229,37	234,40 ± 3,11
GA_{12}	226,26	238,15 ± 4,29	237,69	246,23 ± 4,32	229,65	236,37 ± 3,56	229,65	235,21 ± 2,99	232,60	241,03 ± 3,76
GA_{Rseed}	222,75	230,18 ± 2,28	226,10	228,56 ± 1,66	220,74	224,85 ± 2,67	218,68	224,23 ± 1,81	222,63	230,71 ± 2,94

La mayoría de los GAs que introducen semillas en la población inicial no presentan mejoras desde el punto de vista genético, exhibiendo un grado de diversidad menor al 0,5. No obstante ello, se puede exceptuar a GA_{11}, GA_{12} y GA_{Rseed}, quienes presentan un grado de diversidad inicial cercano a 1,0, similar al presentado por GA (sin proceso de mejora inicial). Las opciones GA_{11} y GA_{12} aplican como reglas de construcción de semillas versiones modificadas de las heurísticas BFDH y FFDH, respectivamente, a soluciones generadas en forma aleatoria. En particular estas heurísticas no sólo consideran las dimensiones de las piezas para reacomodar las piezas en el patrón de empaquetado, sino que también intentan aprovechar de la mejor manera los espacios libres dentro de cada nivel, por lo que se logra una disminución del desperdicio dentro de cada nivel. Por lo tanto, esa aleatoriedad inicial en la distribución de las piezas en un patrón hace que los patrones mejorados resulten diferentes para cada solución tras aplicar esas reglas. Por otra parte, GA_{Rseed} combina todas las observaciones previas con la generación aleatoria de patrones de empaquetado, por lo que eran de esperar altos grados de diversidad genética (valores cercanos a 0,8). Si

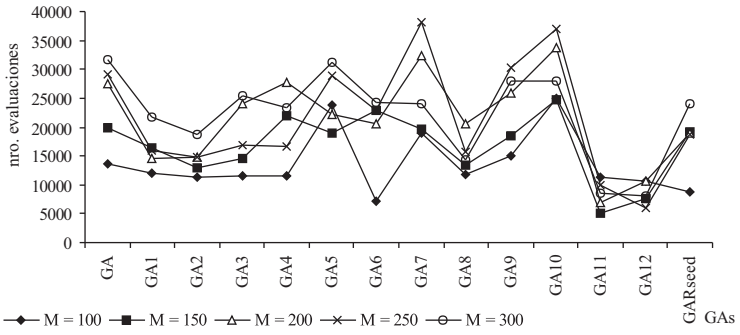


Figura 5.12: Número de evaluaciones promedio para alcanzar el mejor valor por cada instancia

ordenamos las distintas propuestas considerando los valores medios de *fitness* exhibidos por las poblaciones iniciales, la primer posición la ocupa GA_{11} mientras que en el último puesto se encuentra el GA. GA_{Rseed} se encuentra en la posición 7, siendo una posición intermedia.

Como era de esperar, GA_{Rseed} reduce la cantidad de evaluaciones para alcanzar sus buenas soluciones cuando se lo compara con GA (ver Figura 5.12). Para confirmar estas observaciones hemos usado pruebas estadísticas, las cuales indican que las diferencias entre GA_{Rseed} y GA son significativas (valores de p[1] cercanos a 0).

Llegado a este punto podemos extraer como conclusión, atendiendo a los resultados de los experimentos realizados que: GA_{Rseed} mejora en forma significativa a un GA con un método tradicional de inicialización de la población, en todas las métricas bajo estudio. Resultados que fueron corroborados estadísticamente, obteniéndose en todos los casos valores p cercanos a cero. Esto sugiere que la eficiencia de un GA se puede mejorar simplemente incrementado la bondad de su población inicial, determinada no sólo por variedad de patrones de empaquetado sino también por la calidad de los mismos.

Con estos resultados hemos mostrado que en caso de no iniciar la población inicial en forma aleatoria, hemos garantizado que dentro de la población inicial se tenga la diversidad estructural de soluciones para tener una representación de la mayor parte del espacio de soluciones posible o al menos evitar la convergencia prematura.

[1] conocidos como *p-values* en la literatura

Tabla 5.7: Resultados experimentales de GA con semillas y operador de relocalización

Inst	GA_{Rseed}			$GA_{Rseed}FF$			$GA_{Rseed}BF$		
	mejor	***media***$\pm\sigma$	$eval_m$	***mejor***	***media***$\pm\sigma$	$eval_m$	***mejor***	***media***$\pm\sigma$	$eval_m$
100	222,75	230,18 ± 2,28	8.787,53	**214,48**	217,09 ± 1,11	11.426,30	214,79	220,00 ± 1,76	13.803,21
150	226,10	228,56 ± 1,66	19.125,93	**213,54**	214,77 ± 0,69	8.821,35	213,82	216,11 ± 0,76	22.327,71
200	220,74	224,85 ± 2,67	19.018,50	**207,79**	211,11 ± 1,25	15.696,52	207,91	210,96 ± 1,52	18.951,38
250	218,68	224,23 ± 1,81	18.811,87	211,34	212,51 ± 0,65	15.266,35	**210,80**	212,92 ± 1,13	17.441,08
300	222,63	230,71 ± 2,94	24.025,80	209,93	211,91 ± 0,90	17.975,77	**206,41**	212,60 ± 2,21	35.747,48

Debido los buenos resultados obtenidos por el motor de búsqueda de los GAs que incorporan los operadores de relocalización, procedemos a continuación con la comparación del funcionamiento del GA_{Rseed} con otros dos algoritmos: $GA_{Rseed}FF$ y $GA_{Rseed}BF$, los cuales incluyen en su mecanismo los operadores MFF_Adj y MBF_Adj, respectivamente.

La Tabla 5.7 incluye los resultados de las tres configuraciones del GA_{Rseed} para todas las instancias del problema. Como era previsible, lo primero para observar es que el algoritmo que halla los mejores patrones de empaquetado es siempre uno usando un operador de relocalización, para todas las instancias. Nuevamente, hay diferencias estadísticas entre los resultados de cada grupo (GA_{Rseed} versus GA_{Rseed} incorporando algún operador de relocalización), pero no se observan diferencias dentro de cada grupo.

Ahora nos centramos en el análisis del valor de *fitness* medio de las mejores soluciones. Los resultados de $GA_{Rseed}FF$ son muy prometedores y además presentan una desviación pequeña, indicando una estabilidad en el funcionamiento del algoritmo. En términos de la cantidad de evaluaciones, la Tabla 5.7 (columna $eval_m$) muestra que, en general, $GA_{Rseed}FF$ realiza menor número de evaluaciones para encontrar el mejor valor que el resto de los algoritmos.

Para finalizar esta sección comparamos en forma conjunta los mejores valores alcanzados por los algoritmos aplicando operadores de relocalización, tanto con generación aleatoria de la población inicial (resultados de la Tabla 5.4) como con reglas de construcción para armar la población inicial (resultados de la Tabla 5.7). Observamos que los patrones de empaquetado de $GA_{Rseed}FF$ son de inferior calidad a aquellos obtenidos por un GA sin semillas, pero aplicando el operador MFF_Adj. Las diferencias son significativas en una

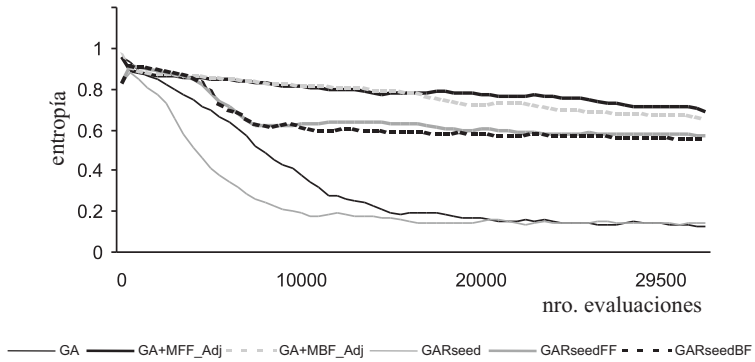


Figura 5.13: Entropía poblacional de todos los algoritmos (instancia con M=200)

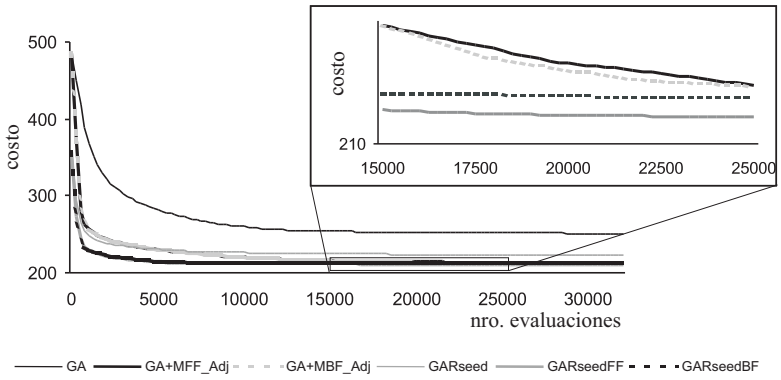


Figura 5.14: *Fitness* medio poblacional de todos los algoritmos (instancia con M=200)

prueba t, con valores p cercanos a 0, excepto para las instancias con M=200 y M=300. Esto sugiere que los operadores son efectivos en el proceso de búsqueda cuando se aplican a una población con una alta diversidad genética como la mostrada por GA puro; es decir, el operador tiene más chances de realizar una buena redistribuión de las piezas. Por otro lado, los valores medios de $GA_{Rseed}FF$ y $GA + MFF\_Adj$ no presentan diferencias significativas en una prueba estadística t-student.

Para estos algoritmos también hemos estudiado la evolución de la diversidad genética como así también la del *fitness* medio poblacional. Los resultados se muestran en la Figura 5.13 y Figura 5.14. Los algoritmos GA_{Rseed} aplicando el operador de relocalización disminuyen en forma similar el *fitness* medio poblacional, pero por otro lado mantienen grados más elevados de diversidad poblacional que GA_{Rseed}. A pesar de estas observaciones,

los primeros fueron capaces de hallar las mejores soluciones finales (aquellas con menores valores de *fitness*), obteniéndose así una rápida convergencia. De hecho, podemos ver que las poblaciones iniciales con semillas presentan valores de entropía más altos (cercanos a 0,85), es decir, alta diversidad, pero GA_{Rseed} usando operadores de relocalización produce mejores individuos en forma más rápida (cerca de la evaluación 2000).

5.9. GAs híbridos

En esta sección nos centramos en el análisis de los distintos GAs híbridos propuestos en este trabajo descritos en la Sección 5.7. A fin de evaluar la eficacia de los diferentes algoritmos híbridos propuestos, analizaremos el desempeño de un GA puro con los siguientes híbridos:

i) GA aplicando el operador de relocalización MFF_Adj (GA+FF)

ii) un GA hibridado con SA (GA+SA),

iii) un GA usando tanto SA como MFF_Adj (GA+SA+FF).

Como primer paso, explicamos brevemente los principales parámetros del algoritmos de enfriamiento simulado (SA) que utilizamos en este trabajo. La cantidad de iteraciones entre dos cambios consecutivos de la temperatura están dados por el parámetro *largo_Cadena_Markov*, cuyo nombre alude al hecho que la secuencia de soluciones aceptadas es una cadena de Markov (una secuencia de estados en el cual cada estado sólo depende del previo). Este parámetro es fijado al valor 30. La cantidad máxima de evaluaciones realizadas por el algoritmo es de 100. La actualización de la temperatura T la realizamos con un esquema que sigue la ley geométrica utilizando el parámetro $\alpha \in (0, 1)$ (proporción de enfriamiento), fijado en 0,9. De esta manera, tras *largo_Cadena_Markov* iteraciones, la temperatura es $T \leftarrow \alpha^N T_0$, con $T_0 = 1000$.

Los valores de la Tabla 5.8 claramente muestran que los GAs híbridos mejoran al GA puro, en términos de calidad de soluciones, para todas las instancias. En todas las corridas, el mejor patrón de empaquetado obtenido usando hibridación tiene un valor de *fitness*

Tabla 5.8: Resultados experimentales de los GAs híbridos

Inst	**GA**			**GA+FF**			**GA+SA**			**GA+SA+FF**		
	mejor	***media*$\pm\sigma$**		***mejor***	***media*$\pm\sigma$**		***mejor***	***media*$\pm\sigma$**		***mejor***	***media*$\pm\sigma$**	
100	234,66	240,21 4,81	±	214,48	217,00 1,16	±	215,56	218,88 2,18	±	**211,34**	215,78 1,54	±
150	238,43	247,94 5,30	±	213,54	214,75 1,16	±	219,89	224,08 2,71	±	**212,74**	214,32 0,62	±
200	243,52	250,94 4,82	±	**207,79**	211,02 1,16	±	215,66	217,04 0,83	±	208,87	212,02 1,01	±
250	241,68	249,32 4,19	±	**211,34**	212,47 1,16	±	218,72	221,04 0,94	±	211,71	212,68 0,56	±
300	241,04	258,10 6,10	±	209,93	211,99 1,16	±	227,44	230,40 1,47	±	**209,83**	213,05 1,11	±

asociado más bajo. Por otro lado, son más robustos que los mejores obtenidos con GA (ver la columna *media*$\pm\sigma$). Estas diferencias son más evidentes a medida que la dimensión de la instancia se incrementa. Usando la prueba de comparaciones múltiples, verificamos que las diferencias entre los resultados de GA+FF y GA+SA+FF no son significativas. La razón de este desempeño es la mejora, en la distribución de las piezas, obtenida tras la aplicación del operador FF, el cual reasigna las piezas en orden de reducir el desperdicio dentro de cada nivel. Sin embargo, GA+SA+FF presenta los mejores patrones de empaquetado en media, pero no hay diferencias significativas con GA y GA+SA.

Analizando la cantidad de evaluaciones necesarias para encontrar el mejor valor que presenta cada algoritmo, la Figura 5.15 muestra que los algoritmos híbridos encuentran sus mejores soluciones en forma más rápida que un GA puro, ya que exhiben menores valores de evaluaciones. A medida que se incrementa la dimensión de las instancias, la diferencia entre GA y GA+SA se torna más notable. Usando las pruebas estadísticas, corroboramos que estas diferencias entre los algoritmos son significativas bajo esta métrica.

En particular, GA+SA+FF explota la capacidad de un GA para encontrar en forma rápida y explotar regiones prometedoras del espacio de búsqueda y la efectiva combinación de SA y FF para encontrar los óptimos locales de una región pequeña del espacio de búsqueda.

La Figura 5.17 muestra que los GAs, excepto la opción híbrida GA+FF, tienen una disminución similar de los valores de diversidad genética (entropía). Un punto interesante es cómo GA+FF mantiene la diversidad genética en niveles bastantes altos durante el

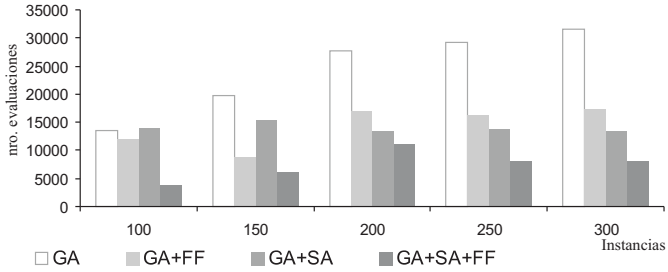


Figura 5.15: Número de evaluaciones promedio para alcanzar el mejor valor de GA y sus hibridaciones para cada instancia

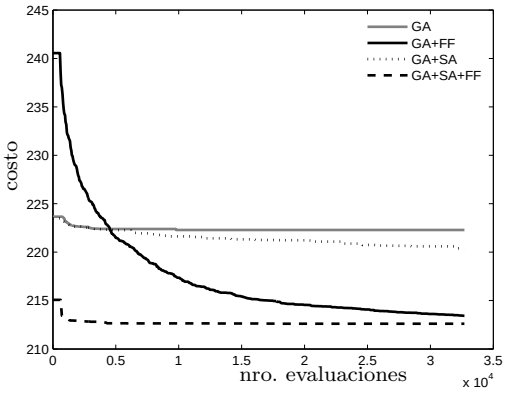


Figura 5.16: Evolución del mejor *fitness* para GA y sus hibridaciones (instancia con M=250)

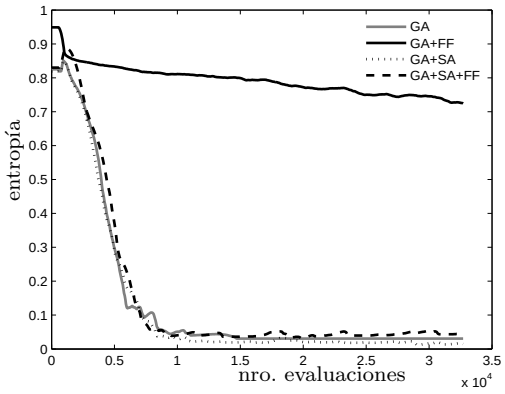


Figura 5.17: Evolución de la entropía para GA y sus hibridaciones (instancia con M=250)

proceso de búsqueda. De la Figura 5.16, observamos que las curvas que representan la evolución de los mejores valores de GA y GA+SA son muy cercanas durante todo el proceso de búsqueda, mostrando el peor comportamiento final. GA+FF presenta una disminución rápida del *fitness* medio poblacional. De hecho, podemos inferir que GA+FF provee la mejor relación entre exploración y explotación, en este caso MFF provee una retroalimentación al algoritmo con ayuda extra al mantener un conjunto diverso de soluciones.

Por lo tanto, la incorporación de procedimientos de búsqueda local en el proceso evolutivo contribuye de manera significativa a la mejora de los resultados de los algoritmos propuestos para 2SPP, proporcionando un muestreo rápido del espacio de búsqueda en todas las instancias estudiadas y un equilibrio entre exploración y explotación satisfactorio.

5.10. Conclusiones

En este capítulo hemos analizado el comportamiento de GAs mejorados para resolver un problema de empaquetado con restricciones. Hemos propuesto nuevos operadores específicos del problema y los hemos comparado con operadores tradicionales. Además hemos analizado distintos métodos de generación de la población inicial y de hibridar un GA con métodos de búsqueda local. El estudio, validado desde un punto de vista estadístico, analiza la capacidad de nuevos operadores y la incorporación de semilla a fin de generar individuos potencialmente prometedores y la capacidad de mantener una población diversificada.

Nuestros resultados muestran que GA usando BILX, el operador de recombinación propuesto en este trabajo, mejora los resultados de forma notable comparado con GAs usando operadores tradicionales de recombinación (OX, CX, EX y PMX), además en un número menor de evaluaciones. La combinación de BILX y la mutación SE, también propuesta en este trabajo, obtienen los mejores resultados. Por otra parte, la incorporación de un nuevo operador de relocalización en el proceso evolutivo produce mejoras significativas en los resultados de los algoritmos propuestos para 2SPP, proporcionando un muestreo más rápido del espacio de búsqueda en todas las instancias analizadas y exhibiendo una relación entre exploración y explotación satisfactoria. Se observa una diferencia a favor de GA+MFF_Adj en término de calidad de soluciones y también en cuanto a tiempos de ejecución. Los GAs usando BILX, SE y MFF_Adj se convierten en los algoritmos bases para los siguientes estudios.

Además, se observa una mejora en el rendimiento de GA al usar una población inicial mejorada, con respecto a la eficiencia (esfuerzo) y la calidad de las soluciones encontradas. GA_{Rseed} funciona correctamente: proporciona una buena diversidad genética de las soluciones iniciales y también una convergencia más rápida sin caer en un óptimo local. Esto demuestra que la ejecución de un GA es sensible a la calidad de sus poblaciones iniciales.

Analizando las versiones híbridas propuestas, GA+SA+FF y GA+FF obtienen patrones de empaquetado de calidad similar, por lo que se desprende que al hibridizar GA con un mecanismo específico para el problema, el rendimiento mejora de manera considerable,

Tabla 5.9: Resumen de los mejores valores reportados por algún GA para cada instancia

Inst	**Mejor solución**	**Algoritmo**
100	211,34	GA+SA+FF
150	210,89	GA+MFF_Adj
200	207,69	GA+MFF_Adj
250	210,80	$GA_{Rseed}BF$
300	206,41	$GA_{Rseed}BF$

permitiendo encontrar soluciones adecuadas. GA+FF mantiene la diversidad genética en niveles altos por más tiempo que en el caso de las restantes opciones híbridas y de GA.

Finalmente, en la Tabla 5.9 se presenta los mejores valores hallados para cada instancia, indicando en cada caso el algoritmo que lo ha obtenido. Cualquiera de los algoritmos utilizan uno de los operadores de relocalización, lo cual marca su influencia para obtener buenos patrones de empaquetado, promoviendo una intensificación en ciertas regiones prometedoras del espacio de búsqueda, que será aprovechada en generaciones posteriores por el algoritmo. En cuanto al número de evaluaciones promedio para alcanzar los mejores resultados, GA+SA+FF es el algoritmo que necesita menor esfuerzo (Figura 5.15), seguido por GA$_{Rseed}BF$, resultando GA+MFF_Adj la opción más lenta.

Capítulo 6

Resolución de 2SPP usando colonias de hormigas

En este capítulo presentamos las aportaciones metodológicas respecto de ACS realizadas en esta tesis doctoral para resolver 2SSP. La primer sección describe la estructura específica del algoritmo para tratar 2SPP. Las Secciones 6.2 a 6.6 describen los aspectos importantes de ACS que deben ser adaptados al problema: la información heurística considerada, la definición del rastro de feromonas, la regla de transición de estados y la función objetivo. A continuación se detallan los formas híbridas de ACS consideradas en este trabajo (Sección 6.7) y cómo se incorpora memoria externa para ayudar a las hormigas en el proceso de construcción de soluciones (Sección 6.8). La Sección 6.9 analiza los resultados obtenidos tras la aplicación de ACS a las instancias del problema. Por último, la Sección 6.10 presenta las conclusiones finales sobre los resultados obtenidos.

Vale la pena destacar que varios algoritmos de hormigas han sido aplicados con éxito a una amplia gama de problemas de optimización (ver [75]). Sin embargo, pocos artículos se refieren al problema de empaquetado, en particular en dos dimensiones. Además de los mencionados en la Tabla 4.7 para resolver el *strip packing*, Boschetti y Maniezzo [37] presentan un ACS básico para el problema de *bin packing* en dos dimensiones, que fue capaz de obtener soluciones de calidad comparables a los reportados por Boschetti et al. [36]. Levine y Ducatelle [147] utilizan un algoritmo ACO puro, así como un algoritmo ACO mejorado con búsqueda local, para un problema de *bin packing* y otro de *cutting stock*.

Los resultados son semejantes a los de la literatura. Además, el método híbrido supera los resultados existentes para la los problemas en cuestión. Nuestra propuesta pretende adaptar la forma que Levine y Ducatelle [147] usan para representar el rastro de feromonas en la resolución del problema de *bin packing* en una dimensión a una adecuada para 2SPP. Tanto de los trabajos de Boschetti et al. [36] y de Levine y Ducatelle [147] se extrae el uso de permutaciones para representar las soluciones que van generando las hormigas.

6.1. Descripción de ACS utilizado

En esta sección presentamos las adaptaciones introducidas al ACS para resolver 2SPP. El esquema particular de ACO utilizado se conoce como $\mathcal{MM}$AS en el *hyper-cube framework* (HCF) [32], cuyo funcionamiento se incluyó en la Sección 2.6.2. No obstante, dejamos para este capítulo la presentación de los aspectos particulares de la forma de generar una solución y de representar el rastro de feromonas, puesto que eran dependientes del problema. La estructura general del algoritmo se muestra en el Algoritmo 6.

Informalmente, ACS trabaja de la siguiente manera: a hormigas se posicionan sobre una pieza elegida al azar. A partir de allí, cada hormiga construye un patrón de empaquetado al aplicar repetidamente una regla voraz estocástica: la regla de transición de estados. Esta última está basada en la información local disponible en las componentes, incluyendo la información heurística y memorística (rastros de feromonas) para guiar la búsqueda. Mientras construye un patrón de empaquetado, una hormiga modifica la cantidad de feromona depositada sobre las piezas seleccionadas al aplicar una regla de actualización local. Una vez que cada hormiga ha generado su patrón de empaquetado, se evalúa y se observa la calidad de todos los patrones de empaquetado. En este último paso se deposita una nueva cantidad de feromona adicional sólo en las piezas asociadas al mejor patrón de empaquetado, que identifica a la mejor hormiga de la iteración (actualización global).

Algoritmo 6 Algoritmo ACS

```
inicializarValoresFeromonas(τ); {Inicializa los rastros de feromona}
ant^bs=generarSolución(τ, η){Se inicializa ant^bs con una solución inicial aleatoria}
while not(condición de terminación) do
  for j ← 1 to μ do
    ant^i=construirSolución(τ, η);{La hormiga ant^i construye una solución}
    actualizarFeromonaLocal(τ,ant^j); {Actualización local de los rastros de feromona (ACS)}
  end for
  evaporarFeromonas(τ) {Evaporación}
  actualizarFeromonasGlobal(τ,ant,ant^bs) {intensificación, actividad del demonio}
  for j ← 1 to μ do
    if f(ant^i) < f(ant^bs) then
      ant^bs=ant^i {Actualizar la mejor solución, actividad del demonio}
    end if
  end for
end while
return la mejor solución encontrada
```

6.2. Representación del rastro de feromonas

En el proceso de construcción, las hormigas artificiales usarán otra información: los rastros de feromona dejados por las hormigas precedentes. Esto será almacenado en una nueva matriz τ_{ij}. La definición del significado del rastro de feromonas es importante para la calidad de un ACS. La información heurística deberá reflejar la información más relevante en el proceso de construcción de soluciones. Siguiendo las ideas de Boschetti y Maniezzo [37], nosotros mantenemos soluciones en la forma de permutaciones de un conjunto de piezas, la cual será traslada directamente en el correspondiente patrón de empaquetado por un algoritmo de asignación (*layout*). En este trabajo, hemos considerado dos alternativas para definir el significado del rastro de feromonas, las cuales describiremos a continuación.

La idea de una de las alternativas, denominada $fero_{i-nivel}$, es favorecer el par pieza/nivel, porque creemos que hay una correspondencia entre la pieza y el nivel donde ésta será asignada. De esta manera, el rastro τ_{ij} codifica la preferencia de que la pieza i y la j estén en el mismo nivel [147]. Debemos considerar por consiguiente la información de las piezas ya asignadas en el nivel actual. La matriz de feromonas tiene de esta manera una fila por cada pieza y una columna por cada nivel del patrón de empaquetado, inicialmente se generan tantos niveles como piezas posea la instancia. Al hacer esto enfatizamos la

importancia de combinar piezas que llenan niveles tan bien como sea posible. Los bloques de construcción, los cuales consisten de patrones de empaquetado parciales en una solución (subconjunto de varias piezas asignadas en un mismo nivel) pueden emerger al reforzar posiciones contiguas en el subconjunto de piezas.

En este trabajo se evalúa también otra forma de definir el significado del rastro de feromona, denominada $fero_{i-j}$, el cual está basado en la propuesta seguida para TSP [73]. El problema TSP es un problema de ordenamiento, así el objetivo es poner las distintas ciudades en un determinado orden. Esto se traslada en el significado de los rastros de feromona, los cuales codifican la preferencia de visitar una cierta ciudad j tras cierta ciudad i. Siguiendo las ideas propuestas para TSP, en este caso la idea es favorecer pares de piezas, es decir poner las distintas piezas en un determinado orden. Esto se traslada al rastro codificando la preferencia de asignar la pieza j tras la i. De esta manera la matriz resultante tendrá M filas y M columnas y será simétrica.

6.3. Construcción de soluciones

En este proceso de construcción, las hormigas utilizan información heurística obtenida a partir de los datos del problema (dimensiones de las piezas, áreas, etc.) y también la información que las hormigas precedentes aportan sobre el grado de eficiencia de los patrones de empaquetado (soluciones al problema) previamente obtenidos.

Cada posible solución del problema puede entenderse como una permutación del número de piezas, y como tal, representada como un vector de M componentes enteras, tal como se describió en la Sección 5.2.

En el algoritmo, k hormigas artificiales construirán sendos patrones de empaquetado realizando asignaciones parciales de piezas a niveles. Para comenzar, cada hormiga elige de forma aleatoria la primer pieza de manera independiente unas de otras, que inicializa el primer nivel y por ende la primera pila de ese nivel. A continuación se usa una regla probabilística para decidir cuáles serán las siguientes piezas que asignar.

La regla de transición de estados indica la probabilidad con la cual cada hormiga k

deberá elegir una pieza j como la próxima pieza que asignar al patrón de empaquetado parcial. Dependiendo de la codificación del rastro (detallados en la Sección 6.9.3) se deben considerar distintas reglas de transición de estados.

Utilizando una codificación del rastro de feromonas $fero_{i-nivel}$, la regla de transición de estados indica la probabilidad (Ecuación 6.3.3) con la cual cada hormiga k deberá elegir una pieza j como la próxima pieza del nivel actual l en la solución parcial $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_{M'})$ con $0 \leq M' < M$. Esta idea fue propuesta por Levine et al. [147] para problemas de empaquetado en una dimensión. Definamos a $J_k(\pi', l)$ como el vecindario alcanzable de la hormiga k; es decir, $J_k(\pi', l)$ es el conjunto de piezas que califican para su inclusión en el nivel actual por la hormiga k. El conjunto incluye aquellas piezas que no se han incluido aún en la solución parcial π' y, además, sus dimensiones son adecuadas para caber en el nivel actual l considerando las restricciones del problema (se siguen las condiciones descritas en la Sección 3.4):(a) la $altura(j)$ no excede la altura del nivel, $l.alto$, y (b) el $ancho(j)$ no excede el ancho restante del nivel, $l.anchoRest$. Por lo tanto, el valor de feromona $\tau_l(j)$, para la pieza j en un nivel l, está dado por:

$$\tau_l(j) = \begin{cases} \frac{\sum_{i \in A_l} \tau_{ij}}{|A_l|} & \text{if } A_l \neq \emptyset \\ 1 & \text{otherwise} \end{cases} \tag{6.3.1}$$

donde A_l es el conjunto de piezas ya asignadas en el nivel l. En otras palabras, $\tau_l(j)$ es la suma de todos los valores de feromonas de las piezas ya asignadas al nivel l, dividida por la cantidad de piezas ya ubicadas en el nivel. Esta opción es similar a la seguida por Levine y Ducatelle [147].

La regla de decisión se puede escribir formalmente como [147]:

$$j = \begin{cases} \text{máx}_{j \in J_k(\pi', l)} \left[\tau_l(j)\right] \times \left[\eta_j\right]^{\beta} & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{6.3.2}$$

donde $J_k(\pi', l)$ es el vecindario factible, $\tau_l(j)$ es el valor de feromonas para la pieza j en el nivel l, η_j es la información heurística que guía a la hormiga y β es el parámetro que define la importancia relativa de la información de las feromonas versus la información heurística.

Otro parámetro utilizado es q el cual es un número aleatorio uniformemente distribuido en [0..1] y q_0 es un parámetro constante ($0 < q_0 < 1$) que determina la importancia relativa entre la explotación y la exploración. S es una variable aleatoria seleccionada acorde a la distribución de probabilidad dada en Ecuación 6.3.3

$$p_k(\pi', l, j) = \begin{cases} \frac{[\tau_l(j)] \times [\eta_j]^\beta}{\sum_{g \in J_k(\pi', l)} [\tau_l(g)] \times [\eta_g]^\beta} & \text{if } j \in J_k(s, l) \\ 0 & \text{otherwise} \end{cases} \tag{6.3.3}$$

En cambio, codificando el rastro de feromonas favoreciendo pares de piezas, la regla de transición de estados indica la probabilidad con la cual cada hormiga k deberá elegir una pieza j como la siguiente pieza a i, recientemente asignada en la solución parcial $\pi' = (\pi'_1, \pi'_2, \ldots, \pi'_{M'})$ con $0 \leq M' < M$. En este caso la regla de decisión se puede escribir formalmente como:

$$j = \begin{cases} \text{máx}_{j \in J_k(\pi', l)} [\tau_{ij}] \times [\eta_j]^\beta & \text{if } q \leq q_0 \\ S & \text{otherwise} \end{cases} \tag{6.3.4}$$

donde $J_k(\pi', l)$ como el vecindario factible, τ_{ij} es el valor de feromonas para el par de piezas i, j y η_j es la información heurística. La variable S se selecciona acorde a la distribución de probabilidad dada en la Ecuación 6.3.5.

$$p_k(\pi', l, j) = \begin{cases} \frac{[\tau_{ij}] \times [\eta_j]^\beta}{\sum_{g \in J_k(\pi', l)} [\tau(ig)] \times [\eta_g]^\beta} & \text{if } j \in J_k(s, l) \\ 0 & \text{otherwise} \end{cases} \tag{6.3.5}$$

6.4. Función objetivo

El proceso de construcción seguido por cada hormiga en ACS da como resultado una solución (patrón de empaquetado). Esta última representa una permutación $\pi = (\pi_1, \pi_2, \ldots, \pi_M)$ de M número naturales, es decir la misma estructura del cromosoma utilizado en nuestro GA (ver sección 5.2). Por esta razón, definimos a la función objetivo F de ACS como la planteada como función de *fitness* de GA (ver Sección 5.3) en la Ecuación 5.3.1.

6.5. Actualización del rastro de feromonas

Una vez que todas las hormigas han completado sus patrones de empaquetado, se debe actualizar el rastro de feromonas para que aporten información nueva a las hormigas de la siguiente iteración. En este caso se aplica una regla de actualización global. Tras cada iteración, sólo la mejor hormiga (cuya solución se identifica como π^{best}) puede depositar feromona. Esto se hace de acuerdo a la siguiente expresión:

$$\tau_{ij} = \rho \times \tau_{ij} + (1/F(\pi^{best})) \tag{6.5.1}$$

donde $0 < \rho < 1$ es el parámetro de decaimiento del valor de feromona, y $F(\pi^{best})$ es el valor objetivo (descrito en la Sección 6.4) de π^{best} en la iteración actual. La actualización de los valores de feromonas usando la mejor hormiga hace a la búsqueda mucho más agresiva. El objetivo que se persigue con la actualización global es proveer mayor cantidad de feromona a buenos patrones de empaquetado.

Sin embargo, mientras las hormigas construyen una solución, también aplicamos una regla de actualización local. El efecto de este tipo de actualización es lograr que la preferencia de las piezas cambie en forma dinámica. La actualización local se realiza en función de la siguiente expresión:

$$\tau_{ij} = (1 - \xi) \times \tau_{ij} + \xi \times \Delta\tau_{ij} \tag{6.5.2}$$

donde $0 < \xi < 1$ es el parámetro y $\Delta\tau_{ij}$ es fijado al valor de τ_0. Dorigo y Gambardella [73] usan esta expresión en sus experimentos, alcanzados resultados satisfactorios.

Otra forma de promover la exploración es definiendo un límite inferior (τ_{min}) para el valor de la feromona. La siguiente fórmula fija el valor de τ_{min} [147] como:

$$\tau_{min} = \frac{(1/(1-\rho))(1 - \sqrt[M]{pbest})}{(prom - 1)\sqrt[M]{pbest}} \tag{6.5.3}$$

donde *pbest* es la aproximación de la probabilidad real para construir la mejor solución, *prom* es la cantidad promedio de piezas elegibles en cada punto de decisión cuando se

construye una solución, definida como $M/2$. También se produce una fase de evaporación en cada iteración al actualizar el rastro de feromonas, definida por:

$$\tau_{ij} = \gamma \times \tau_{ij} \tag{6.5.4}$$

donde $0 < \gamma < 1$ es un parámetro. Esta evaporación permite al algoritmo "olvidar" aquellas decisiones previamente tomadas y que condujeron a soluciones con valores objetivos asociados altos (patrones de empaquetado malos). Si las hormigas no hacen uso de una asignación particular, el correspondiente valor en la matriz decrecerá de forma exponencial con el número de iteraciones.

6.6. Importancia de la información heurística

Es bien sabido que el proceso de búsqueda de los algoritmos ACS está sesgado y guiado por dos componentes principales: la heurística y los valores de feromonas. En muchos problemas de optimización se puede usar información heurística dependiente del problema para dar sugerencias adicionales a las hormigas en sus decisiones.

Una de las tareas de diseño que tener en cuenta es, entonces, definir de manera apropiada la preferencia heurística de cada decisión que debe tomar una hormiga mientras construye una solución: definir la información heurística η_j asociada a cada pieza j. Es importante resaltar que la información heurística es crucial para un buen rendimiento especialmente si no existen o no pueden ser aplicados algoritmos de búsqueda local.

Para el problema en estudio, el valor heurístico utilizado por una hormiga es definido tomando información de las dimensiones de las piezas. En este trabajo consideramos tres formas distintas de definir la información heurística:

I) $\eta_j = h_j$, la altura de la pieza j.

II) $\eta_j = w_j$, el ancho de la pieza j.

III) $\eta_j = area(j)$, el área de la pieza j.

Con la primer regla las hormigas prefieren piezas con alturas similares a la del nivel, por consiguiente habrá una tendencia para conformar niveles con piezas de altura similar, dando lugar a un probable mejor aprovechamiento del espacio interior del mismo. Con la segunda regla, la preferencia estará puesta a piezas que tengan anchos parecidos al ancho sin usar del nivel. Finalmente, considerar el área de la pieza es una manera de valorar en forma conjunta las dos dimensiones de una pieza, tratando de sopesar las dos justificaciones vertidas anteriormente. Esta información heurística no se modifica durante la ejecución del algoritmo por la naturaleza del problema.

6.7. ACS y búsqueda local

El método de búsqueda local propuesto por ACS para este problema es justamente la búsqueda local detallada en la Sección 5.7. Los procedimientos de búsqueda local, usados en este trabajo de tesis para hibridar un ACS, consisten de la aplicación de versiones modificadas de la heurística FFDH descrita en la Sección 4.1.2 y aquí llamada MFF, la cual es similar al operador de relocalización, introducido y explicado en la Sección 5.5. Tal como se hizo con GA también se considera la técnica recocido simulado (SA) para hibridar ACS (Sección 5.7).

Con la versión híbrida de ACS, MFF o SA se aplican después que las hormigas hayan generado a soluciones (tras el **for** más externo en el Algoritmo 6). De esta forma, la búsqueda local comienza desde una solución creada por ACS hacia el óptimo local más cercano de esa solución. Tras esta fase de mejora, se actualiza el rastro de feromonas, de hecho esto es una forma de búsqueda Lamarkiana. Más específicamente, MFF y SA se aplican a las mejores $a/2$ soluciones generadas por las hormigas con la intención de mejorar el desperdicio en todos los niveles. La nueva posible distribución de piezas obtenida de esta forma debe ser transmitida a la colonia de hormigas. Esto se realiza por medio de la actualización el rastro de feromonas, de tal forma que se incluya la información para que más tarde las hormigas construyan soluciones sesgadas a patrones de empaquetado mejorados. Un ejemplo de patrones de empaquetado mejorados se muestra en la Figura 5.6 el cual refuerza un rastro

de feromona distinto al patrón de empaquetado de la Figura 5.1, el primero se obtiene al aplicar MFF al segundo patrón.

6.8. ACS con memoria

Recientemente, la idea de incorporar conocimiento de iteraciones previas se ha vuelto atractiva en ACO, principalmente inspirada en los estudios de algoritmos genéticos. Esta incorporación se puede lograr usando una memoria interna o externa [80, 184]. En la metaheurística ACO, el trabajo de Montgomery y Randall [167] fue la primer implementación de una opción basada en el uso de una memoria interna, mientras que el trabajo de Guntsch et al.[107] es el primer ejemplo de una implementación usando memoria externa. Recientemente, Acan [2] propuso otra estrategia basada en memoria externa para ACO. La memoria está compuesta por segmentos de tamaño variable extraídos de soluciones construidas por hormigas elite en iteraciones previas. Un segmento se extrae de una solución al seleccionar la posición inicial y final en forma aleatoria. Además, el valor objetivo de la solución se asocia al segmento como valor de calidad. El proceso de construcción de una solución seguido por una hormiga se ve, en consecuencia, modificado, con el fin de considerar la información de la memoria externa. Sobre la base de la calidad del valor asociado a cada segmento, una hormiga extrae un segmento de la memoria. La hormiga considera el componente final del segmento como punto de partida, a partir de allí la solución se completa de la manera tradicional. Acan en un trabajo posterior [3] propuso otro ACO con memoria externa, pero en esta oportunidad compuesta por secuencias parciales de la permutación, las cuales se obtienen seleccionando en forma aleatoria un número arbitrario de componentes de la solución. Los enfoques propuestos han logrado mejoras significativas en las soluciones comparados con los algoritmos ACO convencionales.

En este trabajo, introducimos memoria externa a un ACS para resolver 2SPP que almacena permutaciones parciales extraídas de buenas soluciones. Pero en lugar de realizar una selección aleatoria de los posiciones de los segmentos de tamaño variable, consideramos algún conocimiento específico del problema, tal como información de la distribución de las

piezas, a fin de seleccionar grupos de piezas para copiar en la memoria externa que parecen dar lugar a mejores soluciones. En cierto modo se sigue la idea de bloques de construcción de los algoritmos genéticos. La motivación para introducir memoria externa es realizar un seguimiento de la buena combinación de piezas que lleva a diseñar niveles de corte con un mínimo desperdicio. Por lo tanto, el método que proponemos es copiar estos buenos niveles a una nueva solución (produciendo la explotación de buena información previa) y dejar que la hormiga construya el resto de la solución usando el rastro de feromona y la información heurística para promover una fase de exploración.

La memoria externa almacena L secuencias de soluciones, es decir, permutaciones parciales, de tamaño variable extraídas de buenas soluciones, patrones de empaquetado con mínima altura de plancha utilizada, halladas en iteraciones previas. En este caso, las secuencias están formadas por $k/2$ niveles de una solución seleccionados considerando mínimos desperdicios. Dado que no sabemos de antemano el número de piezas en los niveles seleccionados para ser almacenados en la memoria, cada fila de la memoria debe tener la longitud de una solución completa (M). Por lo tanto, las dimensiones de la memoria externa es $L \times M$, donde L es la cantidad de elementos o filas en la memoria y M es el número de piezas. Un ejemplo se muestra en la Figura 6.1, donde inicialmente la memoria externa está vacía. El patrón de empaquetado correspondiente a la mejor solución encontrada en la iteración está formada por cinco niveles ($k = 5$), por cada uno de ellos se detalla el desperdicio que se genera. En este caso, se seleccionan dos niveles ($k/2 = 2,5$ redondeando al entero más chico) con menor desperdicio (nivel 1 y 3) para transferir a la memoria principal. Las piezas correspondientes a estos niveles se copian en la primer entrada de la memoria, el resto de las posiciones de esa entrada se dejan en blanco.

A cada permutación parcial π se le asociada un valor de tiempo de vida, $TpoVida$, y el valor objetivo, *costo*, de la solución de la cual fue extraída. Inicialmente, $TpoVida$ toma el valor uno y se incrementa en una unidad iteración a iteración de ACS. Cada permutación tiene un tiempo de vida máximo de permanencia en la memoria externa definido en diez iteraciones realizadas por ACS. Esos valores permiten asignar una puntuación a cada permutación parcial π, calculada como $S(\pi) = costo(\pi) + TpoVida^2(\pi)$ [3]. Este valor lo usa

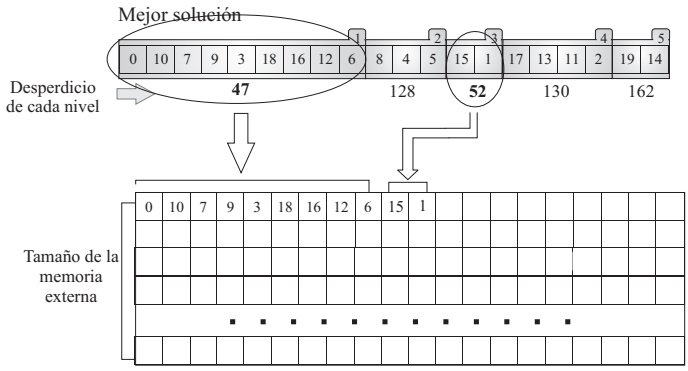


Figura 6.1: Ejemplo de transferencia de niveles a la memoria externa en el ACO

la hormiga para seleccionar una permutación parcial durante el proceso de construcción de una nueva solución.

ACS comienza el proceso evolutivo con una memoria externa vacía, la cual se llena tras algunas iteraciones del algoritmo. En cada iteración se consideran las k mejores soluciones, de las cuales se selecciona una cierta cantidad de niveles y se almacenan en la memoria. Mientras este proceso de llenado toma lugar, las hormigas construyen sus soluciones siguiendo sólo el procedimiento probabilístico clásico de un algoritmo ACS (como descrito en la Sección 6.3). Una vez que la memoria está llena, la fase de construcción de soluciones seguida por una hormiga se modifica. Bajo esta implementación, algunas hormigas construyen la solución extrayendo segmentos de la memoria. En este caso y para el problema bajo estudio, estas partes son un conjunto de piezas que constituyen niveles en el patrón de empaquetado. La selección de una secuencia de memoria se hace usando selección por torneo entre T permutaciones seleccionadas aleatoriamente, basadas en la puntuación asignada a cada secuencia. La ganadora es aquella secuencia con el valor de puntuación más bajo. Esta secuencia se copia en las primeras posiciones de la solución, mientras tanto tiene lugar el proceso de actualización de feromona exactamente de la misma forma que se hace en condiciones normales del algoritmo ACS. Una vez finalizado este proceso, se debe actualizar el vecindario eliminando las piezas actualmente incorporadas en la solución. Hasta este punto, hay piezas que faltan en la solución parcial. Por lo tanto, el resto de la solución se construye

siguiendo el procedimiento usual, es decir, sobre la base de los rastros de feromona y de la información heurística. En nuestra implementación, un porcentaje de hormigas continúa la construcción de sus soluciones con el procedimiento tradicional, porcentaje fijado en 50 %. El motivo es permitir que las hormigas construyan una nueva y completa solución desde la retroalimentación del rastro de feromonas y de la información heurística, lo que contribuye a la exploración de nuevas combinaciones de piezas para elaborar nuevos niveles.

Al final de cada iteración, ACS actualiza la memoria considerando las k mejores soluciones generadas por las hormigas. De cada una de esas soluciones, si l es la cantidad de niveles en la solución, entonces los $l/2$ niveles con el menor desperdicio son insertados en alguna entrada de la memoria externa. En el caso de que la memoria esté completa se aplica una estrategia de reemplazo, la cual consiste en seleccionar el elemento con el peor valor objetivo o bien, el más viejo.

6.9. Experimentos

En esta sección presentamos la experimentación realizada para analizar el comportamiento de un ACS, considerando: cambios en la información heurística que utilizar, codificación del rastro de feromonas e incorporación de búsqueda local al procedimiento típico de un ACS. Además, se analiza el comportamiento de ACS incorporando memoria externa con soluciones parciales que utilizan las hormigas en la construcción de nuevas soluciones. Todas estas propuestas se han comparado en términos de la calidad de sus resultados. Además, se orientó el análisis del funcionamiento del algoritmo a fin de conocer la relación entre valores objetivos y diversidad. Comenzamos detallando la configuración utilizada por ACS (Sección 6.9.1). Hemos dividido a los resultados en cuatro apartados. En primer lugar, en la Sección 6.9.2 comparamos las distintas variantes de información heurística para guiar a las hormigas en sus decisiones en este problema en particular. Un punto importante de un ACS es definir el rastro de feromonas, por lo cual en la Sección 6.9.3 analizamos tres alternativas de codificación del rastro (según lo planteado en la Sección 6.2). El siguiente paso incluido en la Sección 6.9.4 ha sido evaluar cómo se modifica el comportamiento de

ACS desde un punto de vista de calidad de soluciones al incorporar una técnica búsqueda local. El cuarto y último apartado de experimentación (Sección 6.9.5), presenta un ACS al que le incorporamos una memoria externa con permutaciones parciales de soluciones de alta calidad obtenidas en iteraciones previas.

6.9.1. Configuración del algoritmo

Para resolver el problema 2SPP usamos un ACS $\mathcal{M}$in-$\mathcal{M}$ax. El número de hormigas es de 50. Cada hormiga inicia la construcción de una solución con una pieza seleccionada en forma aleatoria. Los valores de los parámetros son los siguientes: $q_0 = 0.9$, $\rho = 0.8$, γ=0.96, ξ=0.1, y $\beta = 1, 2, 5, 10$. Los valores iniciales de feromonas, $\tau(0)$ es fijado a τ_{min}. Como criterio de parada se ha elegido finalizar el algoritmo cuando se alcance un máximo de evaluaciones fijada en 2^{16}. Estos parámetros no se han elegido de forma arbitraria; son el resultado de un estudio previo dirigido para encontrar la mejor configuración para abordar nuestro problema de empaquetado. En cada apartado, por cada variante del algoritmo hemos realizados 30 ejecuciones independientes para cada una de las instancias.

El algoritmo fue implementado usando el paquete MALLBA [16]. Las máquinas utilizadas para los experimentos son Pentium 4 a 2.4 GHz con 1 GB de RAM y sistema operativo Linux (versión del kernel 2.4.19-4GB).

6.9.2. Ajuste de la información heurística

El primer conjunto de experimentos tiene que ver con el estudio del comportamiento de ACS al variar la preferencia heurística que utilizada por cada hormiga (detallada en la Sección 6.6), a fin de determinar su importancia a la hora de obtener soluciones de alta calidad. Analizamos los siguientes ACSs utilizando los siguientes valores heurísticos:

I) la altura de las piezas ($\eta_j = h_j$),

II) el ancho de la pieza j ($\eta_j = w_j$),

III) el área de la pieza j ($\eta_j = area(j)$).

La Tabla 6.1 contiene los resultados obtenidos por cada una de los valores heurísticos propuestos para cada una de las instancias de 2SPP. Como se puede observar, existe una clara

Tabla 6.1: Resultados experimentales de ACS comparando información heurística

Inst	$\eta_j = h_j$		$\eta_j = w_j$		$\eta_j = a_j$	
	mejor	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$
100	**252,47**	257,54 $\pm$ 1,36	299,10	307,72 $\pm$ 3,97	287,21	297,84 $\pm$ 4,04
150	**258,72**	265,68 $\pm$ 3,31	362,13	376,00 $\pm$ 7,07	309,64	326,62 $\pm$ 6,86
200	**251,35**	260,35 $\pm$ 3,13	326,39	343,94 $\pm$ 5,23	303,52	327,54 $\pm$ 7,51
250	**238,53**	248,72 $\pm$ 2,95	323,47	331,53 $\pm$ 4,86	306,64	316,13 $\pm$ 4,43
300	**253,31**	260,04 $\pm$ 2,34	353,44	368,86 $\pm$ 6,04	331,51	337,27 $\pm$ 3,12

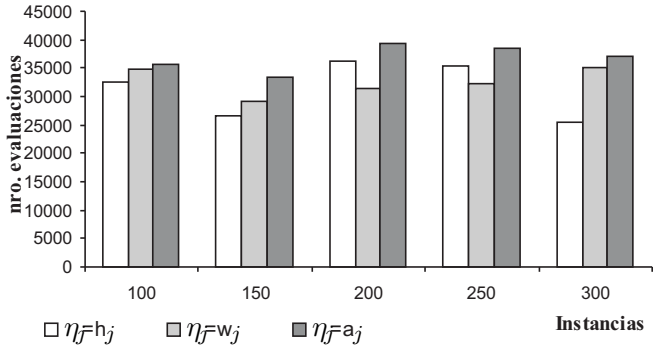


Figura 6.2: Número de evaluaciones promedio para alcanzar los mejores valores de los distintos ACSs utilizando distinta información heurística

ventaja al considerar la altura de las piezas como valor heurístico ($\eta_j = h_j$) en todas las instancias. La razón de este comportamiento es que cuando se calcula la probabilidad de seleccionar a la siguiente pieza (ver Ecuación 6.3.3), aquellas piezas con una preferencia heurística más alta (altura de la pieza s parecida a la del nivel) tienen mayor probabilidad de ser seleccionadas. De este modo los niveles generados son más compactos, es decir, con menor desperdicio. Por otro lado, considerar como valor heurístico el ancho de las piezas produce la peor configuración. En este caso, se da preferencia a las piezas más anchas que aún entran en el espacio sin ocupar del nivel actual, pero sin considerar el espacio desperdiciado sobre la pieza. Un nivel intermedio en la calidad de los resultados se logra considerando el área de la pieza, donde entran en juego tanto el ancho como la altura de la pieza. Estas afirmaciones fueron verificadas estadísticamente por medio de ANOVA, resultando en todos los casos valores de p muy cercanos a 0, lo cual indica que existen diferencias estadísticas significativas en los valores medios de los distintos algoritmos propuestos.

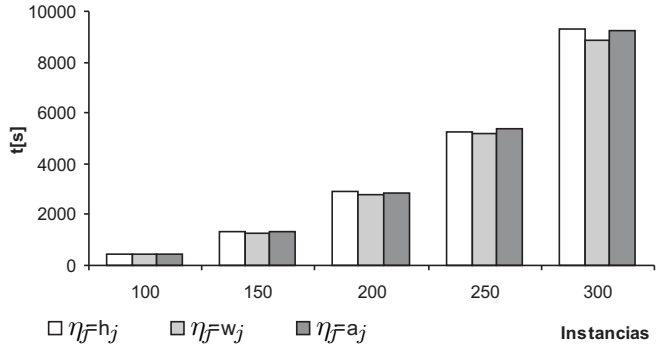


Figura 6.3: Tiempos promedio para los distintos ACSs utilizando distintos distinta información heurística

La Figura 6.2 muestra el número de evaluaciones promedio necesarias para alcanzar los mejores valores de cada uno de los algoritmos. Observamos que ACS con $\eta_j = h_j$ tiende a que el número de evaluaciones necesarias sea menor que el de los restantes para obtener la mejor solución al problema, en general. Pero es de resaltar que no existen diferencias estadísticas significativas entre las propuestas respecto del esfuerzo necesario para alcanzar los mejores valores. Respecto del tiempo de ejecución (ver Figura 6.3), la tres opciones presentan tiempos similares, lo cual es de esperar debido a que los cálculos de los valores de η son semejantes en los tres casos, corroborado por estudios estadísticos.

Como resumen de esta sección podemos indicar que la adecuación de la información heurística para resolver 2SPP mejora la performance del algoritmo ya que disminuye el error medio de los resultados logrados.

6.9.3. Definición del rastro

En esta sección comparamos el rendimiento de ACS al considerar diferentes formas de definir el significado del rastro de feromonas, tal como se describió en la Sección 6.9.3, para proporcionar un cuerpo de conocimiento en este ámbito que ha sido en general poco estudiado en la literatura relacionada a 2SPP. Para tal fin, estudiaremos ACSs con $\eta_j = h_j$ (debido a los buenos resultados mostrados en la sección previa) y los tres métodos diferentes de definir el rastro de feromonas:

Tabla 6.2: Resultados experimentales de ACS usando diferentes codificaciones del rastro

Inst	$ACS_{i-nivel}$		ACS_{i-j}		ACS_{2Mat}	
	mejor	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$
100	**252,47**	257,54 ± 1,36	256,59	260,56 ± 1,91	253,65	258,84 ± 1,97
150	258,72	265,68 ± 3,31	258,53	266,39 ± 3,98	**257,10**	266,86 ± 4,01
200	**251,35**	260,35 ± 3,13	258,09	262,60 ± 2,52	258,30	262,85 ± 1,95
250	238,53	248,72 ± 2,95	241,26	249,49 ± 3,78	**237,26**	247,89 ± 3,45
300	**253,31**	260,04 ± 2,34	257,26	261,77 ± 1,80	255,47	260,88 ± 1,91

I) $ACS_{i-nivel}$ utiliza la alternativa de codificación del rastro de feromonas $fero_{i-nivel}$,

II) ACS_{i-j} usa la codificación $fero_{i-j}$,

III) ACS_{2Mat} usa dos matrices de feromonas: una con $fero_{i-nivel}$ y la otra con $fero_{i-j}$. La hormiga para su decisión utiliza ambas informaciones con igual peso.

En la Tabla 6.2 resumimos los resultados de estas pruebas. Podemos observar que cuando se favorece la preferencia de que la pieza i y la j estén en el mismo nivel ($ACS_{i-nivel}$) se obtienen los mejores patrones de empaquetado. Por otra parte, cuando la decisión de la hormiga está sesgada considerando con el mismo peso las dos opciones para definir el rastro (ACS_{2Mat}), vemos que los resultados obtenidos toman valores intermedios a los de $ACS_{i-nivel}$ y ACS_{i-j}, excepto para la instancia con M=150 y M=250 para las cuales los mejores patrones de empaquetado obtenidos por ACS_{2Mat} presentan valores de *fitness* levemente inferiores que la opción $ACS_{i-nivel}$.

Analizando estos resultados con herramientas estadísticas, se observa que para las instancias con M=150 y con M=250 no presentan diferencias significativas en los valores medios de los algoritmos, mientras que en las instancias restantes las diferencias son significativas. Detallando este análisis, en la instancia con M=100 las diferencias entre los tres algoritmos es significativa, obteniéndose un ordenamiento parcial de $ACS_{i-nivel}$, ACS_{2Mat} y ACS_{i-j}. Para el caso de M=200, $ACS_{i-nivel}$ presenta diferencias significativas con las medias de los otros dos algoritmos, mientras que no existen diferencias entre ACS_{i-j} y ACS_{2Mat}. Finalmente con M=300 no se observan diferencias significativas entre ACS_{2Mat} y el grupo formado por los dos ACS restantes, pero dentro de este último grupo sus valores medios presentan diferencias significativas, se obtiene de esta manera un ordenamiento parcial consistente de $ACS_{i-nivel}$, ACS_{2Mat} y ACS_{i-j}.

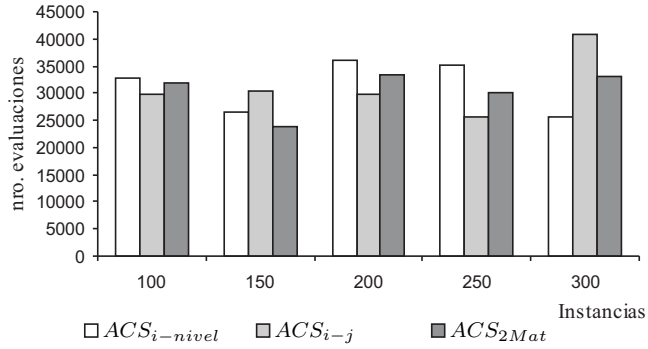


Figura 6.4: Número de evaluaciones promedio para alcanzar los mejores valores de los distintos ACSs utilizando distintos rastros de feromonas

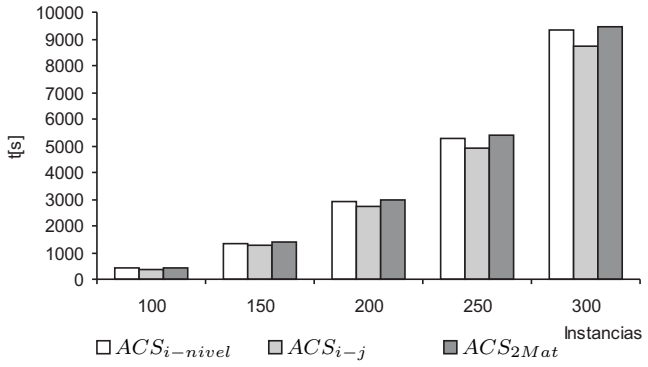


Figura 6.5: Tiempos promedio para los distintos ACSs utilizando distintos rastros de feromonas

Con respecto al esfuerzo, observamos un número de evaluaciones ligeramente más elevado en el caso de $ACS_{i-nivel}$, resultando ACS_{2Mat} el algoritmo que en promedio necesita la menor cantidad de evaluaciones para encontrar sus mejores soluciones. Pero analizando estos resultados a través de un estudio estadístico ANOVA, tales diferencias no son significativas (valores de p superiores al valor de confianza 0.05) salvo para la instancia de dimensión 300 donde las diferencias entre $ACS_{i-nivel}$ y ACS_{i-j} son significativas.

En cuanto a los tiempos de ejecución obtuvimos el resultado esperado. La ejecución de ACS_{i-j} es más rápida que el resto de los algoritmos. Esta ganancia en tiempo se debe a que cada hormiga k al calcular la probabilidad para escoger la siguiente pieza j de la

Tabla 6.3: Resultados experimentales de ACS híbridos

Inst	ACS		ACS+MFF		ACS+SA		ACS+SA+MFF	
	mejor	*media*$\pm\sigma$	*mejor*	*media*$\pm\sigma$	*mejor*	*media*$\pm\sigma$	*mejor*	*media*$\pm\sigma$
100	252,47	257,54 ± 1,36	**215,65**	218,76 ± 0,92	256,24	259,41 ± 1,87	217,72	219,59 ± 1,02
150	258,72	265,68 ± 3,31	**216,75**	218,44 ± 0,82	265,13	271,17 ± 3,34	217,71	219,13 ± 0,74
200	251,35	260,35 ± 3,13	**210,15**	214,83 ± 1,23	254,65	263,24 ± 2,95	211,60	215,92 ± 1,35
250	238,53	248,72 ± 2,95	208,63	209,80 ± 0,54	246,52	252,80 ± 2,32	**208,52**	210,91 ± 0,94
300	253,31	260,04 ± 2,34	**213,46**	215,28 ± 0,62	259,64	262,67 ± 1,68	214,65	215,83 ± 0,56

ya asignada i sólo debe contemplar el valor $\tau(i, j)$. En otras palabras, la hormiga k en ACS_{i-j} debe realizar menos cantidad de cálculos que en el caso de $ACS_{i-nivel}$ donde debe contemplar todas las piezas ya asignadas al nivel (ver Ecuación 6.3.1).

Por lo recién expuesto y analizado, queda claro que la mejor manera de representar el rastro de feromonas para 2SPP es considerar la naturaleza del mismo. Era de esperar que al representar el rastro de feromonas tratando a 2SPP como un problema de ordenamiento (como es el caso de TSP, cuyo objetivo es poner las distintas ciudades en un cierto orden), los patrones de empaquetado resultantes no serían de buena calidad. Esto lleva a que es crucial elegir una definición del rastro que resulte acorde con la naturaleza del problema. 2SPP es un problema de agrupamiento, por lo tanto la idea es favorecer el agrupar piezas en niveles.

6.9.4. ACO híbrido con búsqueda local

El siguiente estudio concierne a la incorporación de búsqueda local al procedimiento de un ACS, como hemos descrito en la Sección 6.9.4. El objetivo de este estudio es comprobar si la inclusión de técnicas de búsqueda local mejora la precisión de un ACS para el problema bajo estudio. A fin de evaluar la eficacia de los diferentes algoritmos híbridos descritos, analizaremos los siguientes ACSs:

i) un ACS hibridado con SA (ACS+SA),
ii) un ACS aumentado con la heurística MFF (ACS+MFF), y
iii) un ACS usando tanto SA como MFF (ACS+SA+MFF).

A continuación se explican brevemente los principales parámetros del algoritmos de enfriamiento simulado (SA) que utilizamos en este trabajo. La cantidad de iteraciones entre dos cambios consecutivos de la temperatura están dados por el parámetro *largo_Cadena_Markov*, cuyo nombre alude al hecho que la secuencia de soluciones aceptadas es una cadena de Markov (una secuencia de estados en el cual cada estado sólo depende del previo). Este parámetro es fijado al valor 30. La cantidad máxima de evaluaciones realizadas por el algoritmo la fijamos en 100. La actualización de la temperatura T la realizamos con un esquema que sigue la ley geométrica utilizando el parámetro $\alpha \in (0, 1)$ (proporción de enfriamiento), fijado en 0,9. De esta manera, tras *largo_Cadena_Markov* iteraciones, la temperatura es $T \leftarrow \alpha^N T_0$, con $T_0 = 1000$.

En la Tabla 6.3 mostramos los resultados obtenidos cuando se compara ACS puro con las tres versiones híbridas. A partir de estos resultados podemos concluir que la incorporación de la heurística MFF después de cada paso del algoritmo aumenta de forma drástica la capacidad de exploración del mismo. En la mayoría de las instancias, la opción ACS+MFF halla los mejores valores objetivos, salvo en la instancia con M =250 en la cual ACS+SA+MFF obtuvo un patrón de empaquetado ocupando menor altura de la plancha de material. ACS+MFF obtiene en media los mejores patrones de empaquetado en todas las instancias, también presentando pequeñas desviaciones. El peor rendimiento de un ACS híbrido para 2SPP se observa cuando ACS se aumenta con SA. En consecuencia, los buenos patrones de empaquetado reportados por ACS+SA+MFF se deben solamente a la mejora que ofrece MFF. Realizando pruebas estadísticas, verificamos que tanto ACS como ACS+SA tienen valores medios significativamente diferentes de ACS+MFF, mientras que las diferencias entre las dos opciones no son significativas para las instancias con M=100, 150, 250 y 300. Para la instancia con M=200, ACS+MFF y ACS+SA+MFF presentan patrones de empaquetados similares. Lo mismo se obtuvo para ACS y ACS+SA, pero en este caso las diferencias dentro del grupo son significativas.

Con respecto al esfuerzo computacional, la Figura 6.6 muestra que, en general, todas las opciones necesitan similar número de evaluaciones para encontrar sus mejores soluciones (corroborado estadísticamente). En particular, para la instancia de tamaño M=250, ACS

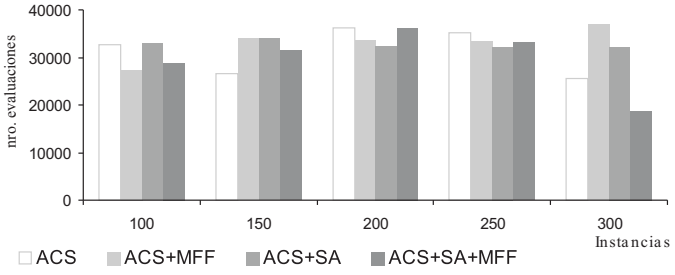


Figura 6.6: Número de evaluaciones promedio para alcanzar el mejor valor para cada instancia

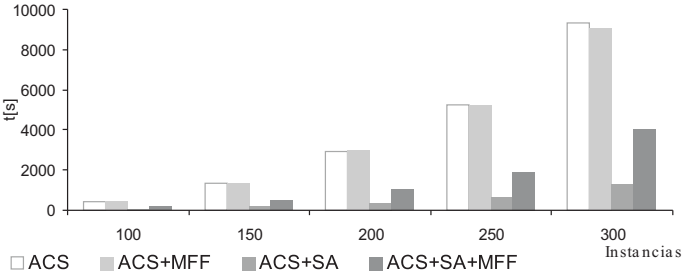


Figura 6.7: Tiempos promedios de cada algoritmo para cada instancia

y ACS+MFF necesitan en promedio una cantidad similar de evaluaciones para lograr su mejor solución, mientras que la diferencia entre las otras dos opciones es significativa.

Un punto interesante surge al analizar el tiempo de ejecución de las diferentes versiones de ACS en estudio (ver Figura 6.7). Los dos ACSs híbridos que incluyen SA son más rápidos que ACS+MFF y ACS puro, resultando este último la opción más lenta. La principal razón de este resultado es la siguiente: en ACS puro, todas las soluciones son generadas por las hormigas mientras que en ACS+SA con o sin MFF, algunas son generadas por las hormigas mientras que el resto lo hace la heurística de búsqueda local. El tiempo que SA invierte en mejorar una solución existente es mínima comparada con el tiempo consumido por la hormiga para generar una nueva solución.

Finalmente queremos probar nuestras conclusiones respecto a que la incorporación de búsqueda local al proceso de un ACS permite mantener diversidad genética. Las Figuras 6.8 a 6.11 ilustran el comportamiento de los diferentes algoritmos ACS para las instancias con M=100 y con M=250, pero situaciones similares se observan con el resto de las instancias.

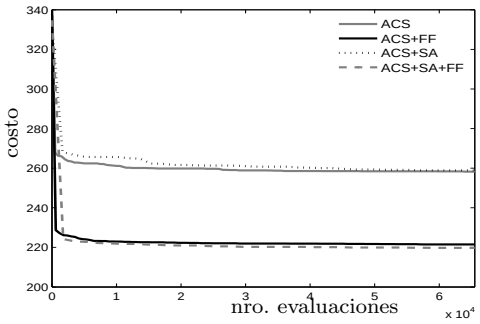


Figura 6.8: Evolución del mejor *fitness* para la instancia con M=100

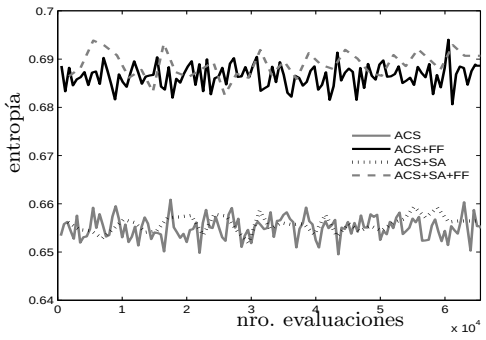


Figura 6.9: Variación de la entropía de la colonia para la instancia con M=100

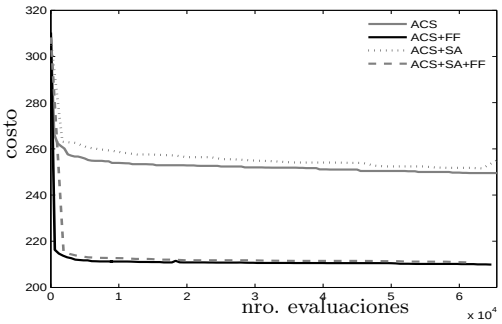


Figura 6.10: Evolución del mejor *fitness* para la instancia con M=250

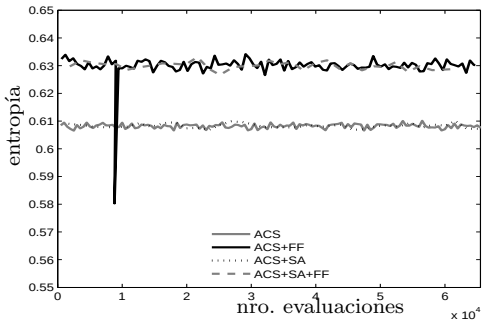


Figura 6.11: Variación de la entropía de la colonia para la instancia con M=250

En estas gráficas se presenta la evolución de la entropía de la colonia y de las mejores soluciones, respectivamente (en el eje y), con respecto al número de evaluaciones (eje x). Por entropía de la colonia se entiende la diversidad del conjunto de soluciones generadas por las hormigas de la colonia. Se puede observar que la entropía es más alta para las opciones híbridas con MFF que para las restantes, mientras que los valores objetivo son siempre más bajos para los ACSs con MFF. Por lo tanto, la mejor relación entre exploración y explotación se obtiene al agregar MFF como búsqueda local a un ACS para el problema en estudio (algoritmo ACS+MFF).

Tabla 6.4: Resultados experimentales de ACS con memoria externa

Inst	**ACS+MFF**		**ACS+MFF+Mem**		***t*-student**
	mejor	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**	
100	215,78	218,29 ± 0,88	**215,00**	218,09 ± 0,91	-
150	**216,38**	217,82 ± 0,79	216,84	218,50 ± 0,97	+
200	**211,61**	214,37 ± 1,12	211,78	214,86 ± 1,18	-
250	**207,68**	209,20 ± 0,76	207,80	210,14 ± 0,70	+
300	213,66	214,74 ± 0,57	**212,45**	214,62 ± 0,94	-

6.9.5. ACO híbrido con memoria de búsqueda

Finalmente, en esta sección nos centraremos en analizar la influencia, en términos de calidad de soluciones y de la velocidad de convergencia del algoritmo, que aporta la incorporación de memoria externa, la cual guarda buenos niveles de patrones de empaquetado (aquellos con bajo desperdicio de material) que luego serán utilizados por las hormigas para construir el patrón de empaquetado completo.

La Tabla 6.4 muestra los resultados de ACS+MFF y de ACS+MFF+Mem, un algoritmo ACS que agrega memoria externa para ayudar a las hormigas en el proceso de construcción de las soluciones. Ambos algoritmos utilizan un método de búsqueda local, en particular MFF, explicado en la sección anterior. De esta tabla podemos observar que ambos algoritmos ACS presentan comportamiento similar en todas las instancias, por lo que no se puede concluir sobre la superioridad de ninguna de las versiones: ambas son igualmente adecuadas y eficaces para las instancias consideradas. Esta afirmación es corroborada estadísticamente en más de la mitad de los casos, ya que no hay diferencias estadísticas significativas entre estas dos opciones, puesto que los respectivos valores p de la prueba t-student son mayores que el nivel de significación 0,05 (véanse los símbolos "+" indicando diferencias significativas en las pruebas t-student).

Considerando la cantidad de evaluaciones, es decir, el esfuerzo numérico para resolver el problema, la Figura 6.12 muestra los resultados de ACS+MFF y ACS+MFF+Mem. En general, los dos algoritmos necesitan esfuerzo similar para resolver todos los casos de 2SPP (estadísticamente corroborado por pruebas t-student).

Un aspecto para resaltar de los algoritmos basados en memoria puede ser observado en la Figura 6.13: ACS+MFF+Mem presenta una ejecución más rápida que ACS+MFF. De

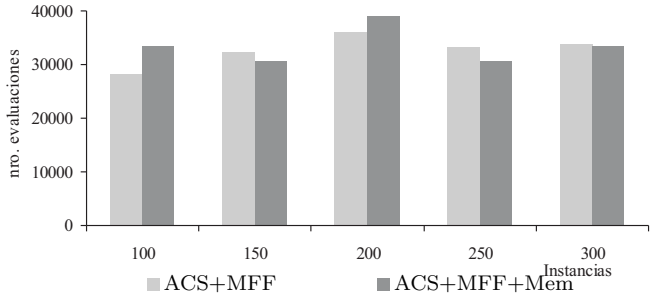


Figura 6.12: Número de evaluaciones promedio para alcanzar los mejores valores por instancia

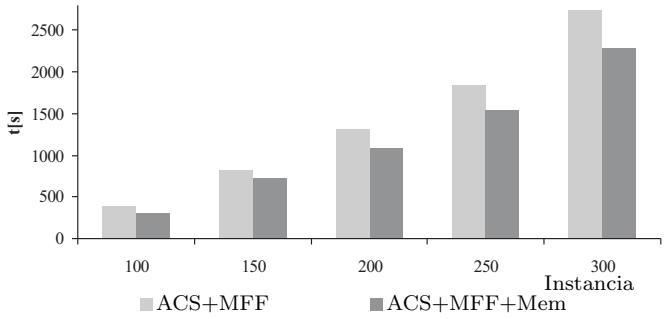


Figura 6.13: Tiempos de ejecución promedio por instancia

hecho, podemos indicar que hay diferencias estadísticas significativas para esta afirmación. Los resultados tienen su razón de ser, puesto que en ACS+MFF+Mem una hormiga comienza el proceso de construcción desde una secuencia parcial de piezas copiada desde la memoria externa. El proceso de transferencia de una permutación parcial de la memoria a la solución de la hormiga es más rápido en tiempo que las decisiones probabilísticas necesarias para construir esa permutación parcial.

Por consiguiente, los bajos tiempos de ejecución y los buenos resultados presentados por ACS+MFF+Mem, convierten a nuestra propuesta en una buena opción para resolver 2SPP.

6.10. Conclusiones

En este capítulo hemos analizado el comportamiento de ACS mejorados para resolver 2SPP con restricciones. Hemos propuesto distintas formas de codificar el rastro de feromonas, atendiendo a las particularidades del problema, como así también el tipo de información heurística considerada durante el proceso de construcción de soluciones. Presentamos diferentes alternativas para hibridar un ACS con el objetivo de lograr algoritmos más avanzados para el tratamiento de 2SPP. Finalmente incorporamos a un ACS memoria externa, la cual almacena soluciones parciales en la forma de grupo de piezas, pertenecientes a niveles con desperdicio mínimo, obtenidas de soluciones elites generadas por las hormigas en iteraciones previas. Estas soluciones parciales se seleccionan en función de cierta información obtenida de la distribución de las piezas. Durante la construcción de las soluciones, las hormigas copian parte de la solución desde la memoria externa y luego toman sus decisiones en base a los rastros de feromonas y la información heurística. El estudio, validado desde el punto de vista estadístico, analiza la capacidad de las distintas opciones para generar nuevas soluciones potencialmente prometedores y la habilidad para mantener poblaciones diversificadas.

El primer estudio que se realiza aquí trata de ajustar la información heurística que utilizará cada hormiga. Los resultados muestran una clara ventaja al usar $\eta_j = h_j$ en todas las instancias, tanto en calidad de soluciones como en esfuerzo numérico, debido a la forma en la que se construyen los patrones de empaquetado, la información heurística más adecuada para usar en la resolución de este problema es la altura de las piezas. El peor rendimiento se observa al utilizar $\eta_j = w_j$ y $\eta_j = a_j$ muestra un comportamiento intermedio ya que incorpora en la información la altura de la pieza.

Otro aspecto clave analizado fue definir la codificación del rastro de feromona que resulte más adecuado para guiar a las hormigas en su proceso de construcción de soluciones a 2SPP. Los resultados muestran que $ACS_{i-nivel}$ obtiene los mejores patrones de empaquetado, seguido por ACS_{2Mat}. En cuanto al esfuerzo numérico no se observan diferencias significativas entre ellos. Por lo tanto, la mejor alternativa para codificar cada entrada de la matriz de feromonas es expresar la preferencia de tener dos piezas en el mismo nivel.

Tabla 6.5: Resumen de los mejores valores reportados por algún GA para cada instancia

Inst	**Mejor solución**	**Algoritmo**
100	215,00	ACS+MFF+Mem
150	216,76	ACS+MFF
200	210,15	ACS+MFF
250	207,68	ACS+MFF
300	212,46	ACS+MFF+Mem

De las opciones híbridas de ACS propuestas, los resultados de ACS+MFF son mejores que los obtenidos con ACS y con ACS hibridado con SA. Es de destacar, la marcada reducción obtenida en los tiempos de ejecución con los enfoques híbridos ACS+SA y ACS+SA+MFF. Esta última observación sumada a los buenos resultados obtenidos, hacen de ACS+MFF una buena alternativa para resolver 2SPP en cuestión.

Los resultados de un ACS+MFF+Mem son similares a los obtenidos con un ACS+MFF. Sin embargo, la incorporación de memoria externa ayuda en la reducción de los tiempos de ejecución, ya que el tiempo que insume la transferencia de las piezas pertenecientes a las soluciones parciales disponibles en la memoria es menor que el que llevaría construir esa solución parcial usando decisiones probabilísticas con en un ACS+MFF.

Finalmente, la Tabla 6.5 presenta los mejores valores hallados para cada instancia y el algoritmo que lo ha obtenido. Los mejores valores para cada instancia son obtenidos por dos algoritmos: ACS+MFF+Mem y ACS+MFF, sin embargo las diferencias entre ellos no son significativas; lo mismo ocurre con el esfuerzo computacional. Los algoritmos utilizan la heurística MFF lo cual marca su influencia para obtener buenos patrones de empaquetado, situación semejante a lo observado con GAs.Esto es muy interesante ya que permite reducir la discusión a otro criterio: el tiempo de ejecución. Hemos encontrado que ACS+MFF+Mem es más rápido que ACS+MFF. Para el problema en estudio, ACS+MFF+Mem combina la explotación de buena información previa y la exploración que realiza la hormiga mientras construye el resto de la solución, lo cual produce un algoritmo altamente eficiente.

Para finalizar las conclusiones de este capítulo, introduciremos una breve comparativa entre las mejores versiones secuenciales de las dos metaheurísticas utilizadas en este trabajo. Para ello en la Tabla 6.6 mostramos los resultados de GA+MFF_Adj (extraídos de la Tabla 5.4 presentada en el Capítulo 5) y de ACS+MFF (extraídos de la Tabla 6.3). En esta

Tabla 6.6: Resumen comparativo entre ACS+MFF y GA+MFF+SA

Inst	**ACS+MFF**			**GA+MFF+SA**			**A**
	mejor	***media*$\pm\sigma$**	***eval***	***mejor***	***media*$\pm\sigma$**	***eval***	
100	215.65	218.76 $\pm$ 0.92	27287.63	**211.34**	215.78 $\pm$ 1.54	20624.00	+
150	216.75	218.44 $\pm$ 0.82	33853.73	**212.74**	214.32 $\pm$ 0.62	20314.00	+
200	210.15	214.83 $\pm$ 1.23	33725.43	**208.87**	212.02 $\pm$ 1.01	20719.00	+
250	**208.63**	209.80 $\pm$ 0.52	33437.50	211.71	212.68 $\pm$ 0.56	20521.00	+
300	214.66	215.46 $\pm$ 0.57	28671.60	**209.83**	213.05 $\pm$ 1.11	20725.00	+

tabla se muestran los mejores valores alcanzados (columna *mejor*), el valor promedio de las mejores soluciones halladas junto con su desviación típica (columna *promedio*), la cantidad de evaluaciones para alcanzar el óptimo (columna *eval*) y el resultado de las pruebas estadísticas ANOVA realizadas. Podemos observar que las mejores soluciones obtenidas por ACS+FF son de inferior calidad que GA+SA+FF, excepto para la instancia con M=250, Las diferencias son estadísticamente significantes bajo una prueba ANOVA (ver símbolo "+" en la columna A) para todas las instancias. Por consiguiente, MFF trabaja adecuadamente tanto para ACS como para GA, intensificando la búsqueda alrededor de algunas regiones del espacio de búsqueda.

Además la comparación se extiende a la evolución de la entropía (Figuras 6.15 y 6.17) como así también a la evolución de la media poblacional (Figuras 6.14 y 6.16). Esta comparación es ilustrada usando las instancias con M=150 y M=250, como ejemplos ilustrativos de las restantes. De la Figura 6.14, podemos observar que el GA+MFF tiene un decremento rápido en la media poblacional (resultando en una convergencia rápida) mientras que la situación opuesta (un decremento rápido de la media poblacional de ACS+MFF) puede también verse en la Figura 6.16. Por otra parte, ACS+MFF presenta valores de entropía aceptables durante todo el proceso de búsqueda, una cualidad remarcable de ACS+MFF.

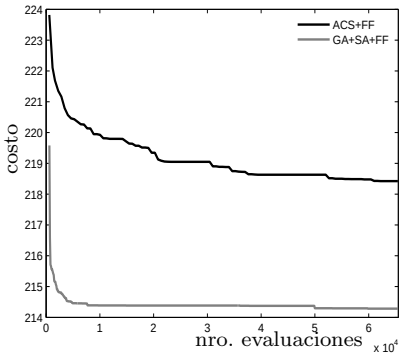


Figura 6.14: Evolución de la media poblacional para ACS+MFF y GA+MFF y la instancia con M=150

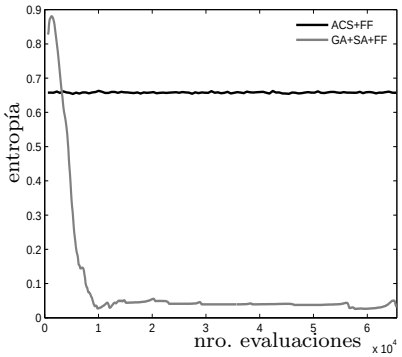


Figura 6.15: Evolución de la entropía para ACS+MFF y GA+MFF (instancia con M=150)

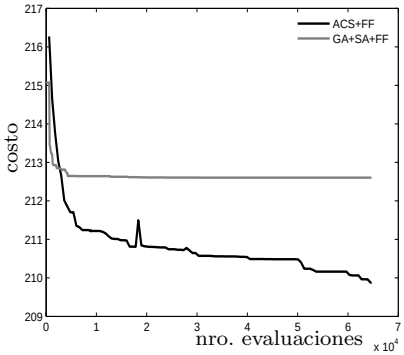


Figura 6.16: Evolución de la media poblacional para ACS+MFF y GA+MFF y la instancia con M=250

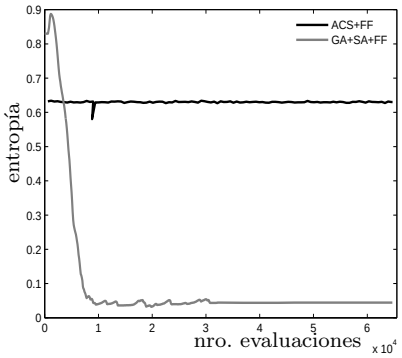


Figura 6.17: Evolución de la entropía para ACS+MFF y GA+MFF (instancia con M=250)

Capítulo 7

Metaheurísticas paralelas y 2SPP

En el Capítulo 2 abordamos las metaheurísticas paralelas pero desde una perspectiva muy genérica. Por consiguiente, en este capítulo daremos los detalles particulares de implementación de nuestras propuestas paralelas, a saber GAs distribuidos y ACSs distribuidos para resolver 2SPP con restricciones abordado en este trabajo de tesis. Como primer paso presentamos en la Sección 7.1 un breve resumen de las métricas paralelas, tal como *speedup* y métricas de rendimiento relacionadas, que utilizaremos a lo largo de este capítulo para evaluar nuestros algoritmos. A continuación, en la Sección 7.2 describimos el GA distribuido utilizado siguiendo un modelo isla mientras que en la Sección 7.3 hacemos lo mismo con el ACS distribuido propuesto. Ambas secciones presentan una estructura similar, divididas en tres partes: en la primera se presentan las características de los algoritmos distribuidos, en la segunda se definen los parámetros utilizados en los experimentos, para finalmente, en la tercera parte dedicarnos a los experimentos realizados y al análisis de resultados. El capítulo acaba en la Sección 7.4 con las principales conclusiones alcanzadas.

7.1. Medidas de rendimiento

Hay varias métricas para medir el rendimiento de los algoritmos paralelos. Una de las más importantes es el *speedup*, que compara el tiempo de ejecución secuencial contra el tiempo correspondiente para el caso paralelo a la hora de resolver un problema particular. Si indicamos por T_m el tiempo de ejecución de un algoritmo usando m procesadores, el *speedup*,

Tabla 7.1: Taxonomía de las medidas de *speedup* propuestos por Alba

- I. *Speedup* fuerte
- II. *Speedup* débil
  - A. *Speedup* con parada por calidad de soluciones
    1. Versus panmixia
    2. Ortodoxa
  - B. *Speedup* con esfuerzo predefinido

para algoritmos no determinísticos, deberá comparar el valor promedio de los tiempos de ejecución secuencial contra el tiempo medio paralelo:

$$s_m = \frac{E[T_1]}{E[T_m]} \tag{7.1.1}$$

Con esta definición se puede distinguir entre: *speedup* sublineal ($s_m < m$), *speedup* lineal ($s_m = m$) y *speedup* superlineal ($s_m > m$). La principal dificultad con esta medida es que los investigadores no acuerdan el significado de T_1 y T_m. Alba [10] en su trabajo distingue entre diferentes definiciones de *speedup* dependiendo del significado de esos valores. La Tabla 7.1 presenta la taxonomía generada.

El *speedup* fuerte (tipo I) compara el tiempo de ejecución paralelo respecto al mejor algoritmo secuencial. Esta es la definición más exacta de *speedup*, pero la mayoría de los diseñadores de algoritmos paralelos no la usan debido a lo complicado que es conseguir el algoritmo más eficiente actual. El *speedup* débil (tipo II) compara el algoritmo paralelo desarrollado por el investigador con su propia versión secuencial. En este caso, se pueden usar dos criterios de parada: por calidad de soluciones y por máximo esfuerzo. El autor de la taxonomía descarta esta última definición debido a que compara algoritmos que no producen soluciones de similar calidad, lo que va en contra del espíritu de esta métrica. Para el *speedup* débil con criterio de parada basado en la calidad de soluciones se proponen dos variantes: comparar el algoritmo paralelo con la versión secuencial canónica (tipo II.A.1) o comparar el tiempo de ejecución del algoritmo paralelo en un procesador con el tiempo que tarda el mismo algoritmo pero en m procesadores (tipo II.A.2). En la primer variante es claro que se están comparando dos algoritmos diferentes. En este trabajo adoptamos la forma ortodoxa de medir el *speedup*.

Aunque el *speedup* es la medida más utilizada, existen otras medidas para comparar el comportamiento de las metaheurísticas paralelas. A continuación presentamos las medidas utilizadas en este trabajo de tesis, además del *speedup*.

La *eficiencia* (ver Ecuación 7.1.2) es una normalización del *speedup* y permite comparar diferentes algoritmos (e_m=1 significa *speedup* lineal):

$$e_m = \frac{s_m}{m} \tag{7.1.2}$$

Karp y Flatt [131] desarrollaron una interesante métrica para medir el rendimiento de cualquier algoritmo paralelo, que puede ayudarnos con la identificación de factores más sutiles que los proporcionados por el *speedup* por sí solo. Esta métrica se denomina *fracción serie* de un algoritmo (7.1.3).

$$f_m = \frac{1/s_m - 1/m}{1 - 1/m} \tag{7.1.3}$$

Idealmente, la fracción serie debería permanecer constante para un algoritmo, aunque se cambie la potencia de cálculo de la plataforma donde se ejecuta. Si el *speedup* es pequeño pero el valor de la fracción serie se mantiene constante para diferentes números de procesadores (m), entonces podemos concluir que la pérdida de eficiencia se debe al limitado paralelismo del programa. Por otro lado, un incremento ligero de f_m puede indicar que la granularidad de las tareas es demasiado fina. Se puede dar un tercer escenario en el que se produce una reducción significativa en el valor de f_m, lo que indica alguna clase de *speedup* superlineal.

7.2. Algoritmos genéticos centralizados vs. descentralizados

En esta sección se presenta en detalle el algoritmo genético distribuido, dGA, utilizado en este trabajo para resolver 2SPP. Este algoritmo es un modelo multi población (modelo isla), en el cual se realizan intercambios esporádicos de individuos, denominado proceso de migración, entre las distintas subpoblaciones P_i (donde i es el identificador de la isla). La política de migración define: *i*)la topología de intercomunicación, *ii*)cuándo se realizan

las migraciones, *iii*) cuáles son los individuos para intercambiar, *iv*)la sincronización entre las subpoblaciones y *v*) la forma de integrar los individuos recibidos en la subpoblación receptora.

En el Algoritmo 7 presentamos la estructura de un subalgoritmo genético (dGA_i), en el cual explicaremos los pasos seguidos en la resolución de nuestras tareas de empaquetado. Este algoritmo es muy similar al algoritmo presentado en el Capítulo 5, la única diferencia es el agregado de la fase de comunicación con las islas vecinas. En particular, se utiliza un GA con selección por estado estacionario. Cada dGA_i crea una población inicial P_i, formada por μ soluciones a 2SPP, siguiendo las reglas de construcción propuestas en la Sección 5.6. A continuación se realiza la evaluación de esas soluciones, para lo cual se utiliza un algoritmo de ubicación (*ad hoc* o heurístico) que acomoda las piezas en la plancha de material para construir un patrón de empaquetado factible. Después de esto, la población entra en un ciclo en el que se produce la evolución: se seleccionan por torneo binario las soluciones padres, luego se aplican los operadores genéticos para crear $\lambda = 1$ hijos. Este ciclo también incluye una fase de intercambio de individuos con un conjunto de subalgoritmos vecinos, denominados dGA_j. Por último, cada iteración termina seleccionando μ individuos para crear la nueva población del conjunto de $\mu + \lambda$ existentes, en particular el hijo generado en cada paso reemplaza al peor individuo de la población actual, sólo en el caso de ser mejor que el peor individuo de la población. La mejor solución es identificada como el mejor individuo que se ha encontrado que reduce al mínimo la altura de la plancha de material necesaria para empaquetar todas las piezas.

Tanto la representación de las soluciones como los operadores genéticos que usa dGA son los presentados en el Capítulo 5. Además, el modelo propuesto utiliza el operador de relocalización, en particular MFF_Adj, como paso de optimización de las soluciones generadas por recombinación y mutación.

7.2.1. Configuración de los algoritmos

Partiendo de esta base y, después de realizar experimentación preliminar, la configuración detallada de los algoritmos distribuidos es la siguiente. El GA específico que hemos

Algoritmo 7 Pseudocódigo de un dGA$_i$

```
t = 0; {evaluación actual}
generarPoblaciónInicial(P_i(t));
evaluar(P_i(t));
while (no max_evaluaciones) do
  padres = seleccionarPadres(P_i(t))
  hijo = aplicarRecombinación(padres);
  hijo = aplicarMutación(hijo);
  hijo = aplicarBL(hijo);
  evaluar (hijo);
  P_i(t+1) = seleccionarNuevaPoblación(P_i(t) ∪ hijo);
  P_i(t) = intercambioIndividuos(dGA_j); {interacción con vecindario}
  t = t + 1;
end while
return la mejor solución encontrada
```

implementado es uno de estado estacionario, o $(\mu + 1)$-GA, donde sólo una nueva solución se construye en cada paso, con una selección por torneo binario de cada padre. El nuevo individuo generado en cada paso reemplaza al peor individuo de la población, sólo en el caso que tenga mejor *fitness*. Los algoritmos de optimización utilizados y comparados son un GA panmíctico (seqGA) y GAs distribuidos o dGAn, donde n indica la cantidad de islas del modelo.

Nuestras propuestas utilizan una población compuesta de 512 individuos y cada isla tiene una población de $512/n$ individuos, donde n representa la cantidad de islas. El operador de recombinación utilizado es BILX (ver Sección 5.4.1) con probabilidad de aplicación del 0,8 mientras que SE (ver Sección 5.4.2) es el operador de mutación con probabilidad de 0,1. El operador de relocalización, en su versión MFF_Adj (Sección 5.5), se aplica siempre (probabilidad igual a 1,0). La estrategia seguida para generar la población inicial consiste en la selección aleatoria de una regla, de un conjunto de reglas que incluyen información del problema (descritas en la Sección 5.6, utilizando una distribución uniforme de probabilidad. Tanto los operadores como sus probabilidades se eligen de forma consecuente con las conclusiones obtenidas en el Capítulo 5. En dGAs, se selecciona un migrante en forma aleatoria para ser enviado a la subpoblación vecina, mientras que la isla receptora selecciona su peor individuo para ser reemplazado con el inmigrante (sólo si es mejor). Entre dos

Tabla 7.2: Resultados experimentales para seqGA y dGA_8

Inst	seqGA		1 procesador		8 procesadores	
	mejor	*media*$\pm\sigma$	*mejor*	*media*$\pm\sigma$	*mejor*	*media*$\pm\sigma$
100	**214,48**	217,28 ± 1,11	214,87	217,49 ± 0,79	216,65	217,73 ± 0,56
150	213,54	214,83 ± 0,65	**212,76**	214,98 ± 0,59	213,82	215,11 ± 0,49
200	**207,79**	211,11 ± 1,11	208,79	211,39 ± 1,14	208,78	211,68 ± 1,00
250	211,34	212,43 ± 0,70	**209,86**	212,23 ± 0,92	210,74	212,27 ± 0,75
300	**209,93**	211,92 ± 0,94	210,42	212,37 ± 0,88	210,48	211,89 ± 0,84

envíos consecutivos deben transcurrir al menos 512 evaluaciones. Los subalgoritmos están dispuestos en un anillo unidireccional y la comunicación es asíncrona, es decir, los individuos son ingresados a la población receptora apenas arriban, por cuestiones de eficiencia.

Antes de comenzar con el análisis de resultados, justificaremos varias configuraciones de los parámetros del GA. Parámetros como tamaño de población, criterio de parada y frecuencia de migración fueron elegidos no en forma aleatoria, sino examinando valores previamente usados con éxito (ser [19]como ejemplo).

Las máquinas utilizadas para los experimentos son Pentium 4 a 2.4 GHz con 1 GB de RAM y sistema operativo Linux (versión del kernel 2.4.19-4GB).

7.2.2. Resultados experimentos

En esta sección se presenta la experimentación realizada para abordar las instancias de 2SPP por medio de algoritmos genéticos distribuidos. Esta sección está formada por dos partes bien diferenciadas. En la primer parte se realiza un estudio de los algoritmos imponiendo el mismo esfuerzo numérico. En la segunda se analizan los efectos del paralelismo en nuestros algoritmos.

Resultados con criterio de parada por evaluaciones

En esta sección se incluye la parametrización de los algoritmos distribuidos utilizados para resolver este problema. Para hacer una comparación apropiada, todas las propuestas utilizan la misma condición de parada: 65.536 evaluaciones (2^{32}).

La Tabla 7.2 muestra los resultados obtenidos por seqGA y dGA_8, este último ejecutado en secuencia (todas las islas comparten un único procesador) y en paralelo (una isla en un procesador dedicado), para cada una de las instancias, esto sugiere que se cuenta con un

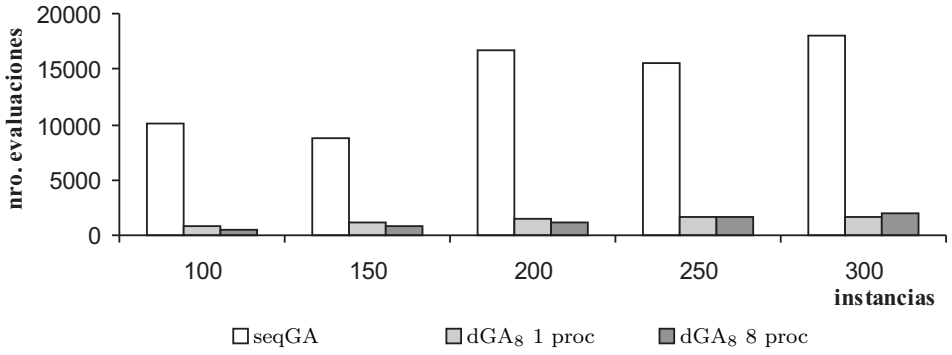


Figura 7.1: Número de evaluaciones promedio para alcanzar el mejor valor de cada algoritmo dGAn

cluster formado por al menos 8 máquinas. En esta tabla se muestra los valores de *fitness* de las mejores soluciones halladas (columna *mejor*) y el promedio de esos valores sobre las 30 ejecuciones realizadas junto con su desviación típica (columna $media_{\pm\sigma}$). En este caso, dGA8 tanto en su versión secuencial y como paralela fueron capaces de hallar soluciones de la misma calidad (valores p cercanos a 0). Este hallazgo confirma que los dos algoritmos realizan básicamente la misma búsqueda, siendo sólo el tiempo el que marca la diferencia entre ellos (valores p mayores a 0,05), ya que se ejecutan en uno y ocho procesadores, respectivamente. El algoritmo seqGA mejora los valores de *fitness* promedio, pero la mayoría de ellos no tienen diferencias significantes al realizar los estudios estadísticos que aquellos hallados por el distribuido.

Otra diferencia importante está relacionada con la elevada cantidad de evaluaciones que seqGA necesita para encontrar las mejores soluciones en contraposición de los dGAs. La prueba estadística realizada marca una diferencia significativa para esta métrica. Ahora, analizando las propuestas distribuidas entre sí, vemos que ejecutar dGA_8 en uno u ocho procesadores no altera la cantidad de evaluaciones necesarias para alcanzar el mejor valor, aseveración también confirmada por la prueba estadística realizada.

Tabla 7.3: Calidad de soluciones objetivo para detener el algoritmo

M	100	150	200	250	300
calidad	217,99	216,23	213,12	213,59	213,90

Efectividad y eficiencia de los algoritmos

Esta sección está dedicada a evaluar las propuestas distribuidas desde el punto de vista del paralelismo. En lugar de utilizar como condición de parada la computación de 2^{16} evaluaciones, se ha considerado detener cada algoritmo cuando obtenga una misma calidad de soluciones finales predefinida para cada instancia o cuando la cantidad máxima de evaluaciones fuese alcanzada. De esta manera se mide el tiempo incurrido para hallar soluciones de calidad equivalente entre las distintas versiones de los algoritmos propuestos. El valor del *fitness* objetivo se corresponde con el valor de *fitness* medio hallado en pruebas previas para un máximo de 2^{16} evaluaciones. Los valores se muestran en la Tabla 7.3 para cada una de las instancias. Por cada algoritmo hemos realizado 50 ejecuciones independientes por instancia.

Se han probado tres configuraciones distintas del algoritmo dGA propuesto: dGA_2, dGA_4 y dGA_8, es decir, dGAs con dos, cuatro y ocho islas. Los algoritmos dGAs se ejecutaron en un sistema con único procesador, en este caso las n islas se asignaron a un único procesador dedicado, y en un cluster de estaciones de trabajo, donde cada isla fue asignada a un procesador diferente dedicado.

La Tabla 7.4 presenta un resumen de los resultados para seqGA y cada dGAn ejecutándose tanto en secuencial (un procesador) como en paralelo (n procesadores), sobre un total de 50 ejecuciones independientes. Los aspectos más relevantes medidos en esta comparación son los siguientes: la cantidad de veces que cada algoritmo alcanzó el valor de *fitness* por cada instancia (columna *éxitos*), el valor promedio de las mejores soluciones halladas junto con su desviación típica (columna *media*), la cantidad promedio de evaluaciones (columna *eval*) y el tiempo medio de ejecución expresado en segundos (columna *t[s]*).

De los valores presentados en la tabla, podemos inferir que hay diferencias importantes entre dGAn y seqGA teniendo en cuenta el esfuerzo numérico para resolver el problema.

Tabla 7.4: Resultados experimentales para seqGA y dGAn

Inst	**Alg**	**1 procesador**				n **procesadores**			
		éxitos	***media*$\pm\sigma$**	***eval***	***t[s]***	***éxitos***	***media*$\pm\sigma$**	***eval***	***t[s]***
100	seqGA	56	218,03 ± 0,19	1325,96					
	dGA_2	60	**217,99** ± 0,00	588,88	441,85	54	**217,99** ± 0,00	470,56	225,01
	dGA_4	36	217,99 ± 0,00	270,00	419,61	46	**217,99** ± 0,00	277,44	109,22
	dGA_8	44	218,01 ± 0,25	138,04	398,86	62	218,01 ± 0,14	267,08	52,92
150	seqGA	69	216,18 ± 0,68	1333,30					
	dGA_2	72	**216,09** ± 0,68	753,70	476,74	68	216,11 ± 0,72	696,02	274,05
	dGA_4	50	216,37 ± 0,70	326,74	833,41	72	**216,09** ± 0,68	325,94	120,83
	dGA_8	62	**216,25** ± 0,72	191,60	631,60	64	216,27 ± 0,61	201,98	100,73
200	seqGA	77	213,22 ± 0,43	1091,23					
	dGA_2	80	**213,19** ± 0,40	525,26	709,60	70	213,27 ± 0,49	570,54	559,55
	dGA_4	76	213,23 ± 0,43	291,68	872,25	76	**213,19** ± 0,49	267,14	222,69
	dGA_8	58	**213,20** ± 0,42	139,22	800,72	62	213,37 ± 0,49	143,30	189,42
250	seqGA	33	213,68 ± 0,55	2288,15					
	dGA_2	28	**213,72** ± 0,49	869,86	4433,81	34	213,73 ± 0,60	856,44	2049,38
	dGA_4	18	214,03 ± 0,64	403,54	5242,06	26	**213,83** ± 0,59	510,02	1274,57
	dGA_8	24	213,83 ± 0,55	203,88	4851,44	36	**213,63** ± 0,49	263,20	539,87
300	seqGA	4	213,94 ± 0,19	1171,35					
	dGA_2	4	**213,90** ± 0,34	426,68	11256,02	0	214,01 ± 0,20	424,89	5958,72
	dGA_4	0	213,98 ± 0,01	285,40	11640,00	4	**213,93** ± 0,20	236,71	2836,35
	dGA_8	0	213,97 ± 0,01	107,50	11591,67	4	**213,91** ± 0,31	278,86	1420,12

Las pruebas estadísticas realizadas confirman esta afirmación, debido a que los valores p obtenidos son inferiores a α=0,05, el valor de confianza, por cada instancia. Se corrobora que estos dos algoritmos realizan un proceso de búsqueda totalmente diferente, es decir, la capacidad de búsqueda de dGAn supera a la de seqGA para este problema. Por ejemplo, seqGA realiza un muestreo de más de tres veces la cantidad de puntos del espacio de búsqueda que dGA_2, antes de encontrar buenas soluciones. Por otra parte, se observa un decremento en la cantidad de evaluaciones a medida que se incrementa la cantidad n de islas, independientemente de ejecutar dGAn en uno o n procesadores. En términos generales, dGA_n ejecutado en uno o n procesadores necesita esfuerzos similares para resolver todas las instancias. Las pruebas t-student realizadas para estas columnas dan valores p mayores que α. Por consiguiente, no se puede concluir nada respecto de la superioridad de ejecutar dGAn en secuencia o en paralelo, ambas modalidades resultan apropiadas para resolver el problema en cuestión.

Otro aspecto que nos resta analizar es la cantidad de éxitos reportados (ver Figura 7.2). Tanto dGA_4 como dGA_8 ejecutándose en paralelo han alcanzado una cantidad de éxitos más alta que sus contrapartes ejecutándose en secuencia. Una excepción la marca

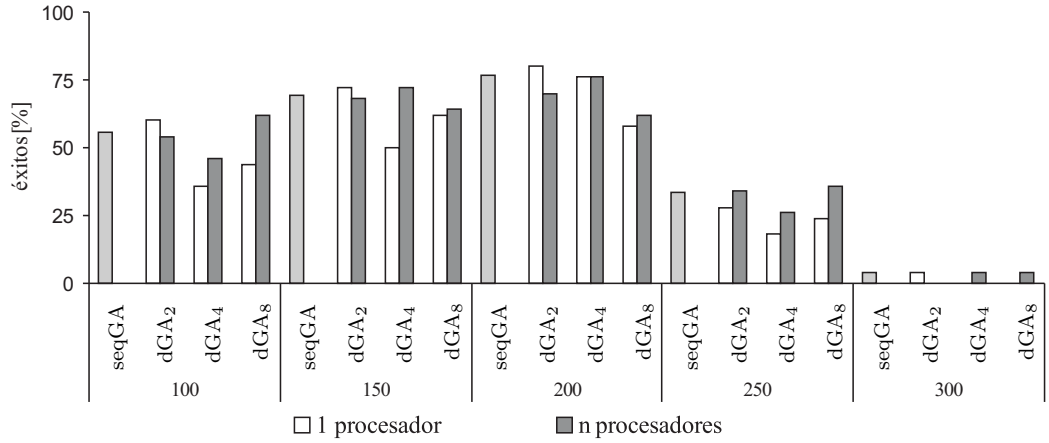


Figura 7.2: Porcentaje de éxitos de cada algoritmo dGAn por instancia

Tabla 7.5: Resultados experimentales del rendimiento paralelo de los dGAn

Inst	**Alg**	**sp**	**e**	**fs**
100	dGA_2	1,96	0,98 %	0,02
	dGA_4	3,84	0,96 %	0,01
	dGA_8	7,54	0,94 %	0,01
150	dGA_2	1,74	0,86 %	0,15
	dGA_4	6,90	1,72 %	-0,14
	dGA_8	6,27	0,78 %	0,04
200	dGA_2	1,27	0,63 %	0,58
	dGA_4	3,92	0,97 %	0,01
	dGA_8	4,23	0,52 %	0,13
250	dGA_2	2,16	1,08 %	-0,08
	dGA_4	4,11	1,02 %	-0,01
	dGA_8	8,99	1,12 %	-0,02
300	dGA_2	1,89	0,94 %	0,06
	dGA_4	4,10	1,02 %	-0,01
	dGA_8	8,16	1,02 %	0,00

dGA_2, puesto que ejecutándose en un procesador logra una mayor cantidad de éxitos que su correspondiente versión en paralelo (dos procesadores), exceptuando la instancia con M=250. Por otra parte, dGA_4 es el algoritmo con el menor número de éxitos en cada instancia.

Los algoritmos dGAn tanto en secuencial como en paralelo no presentan diferencias significativas en lo que respecta al promedio de los mejores valores. Ésta, es una situación esperable debido a que las dos versiones de dGAs utilizan el mismo algoritmo subyacente y sólo el tiempo de ejecución deberá ser afectado por las distintas cantidades de procesadores utilizados. Esas conclusiones están apoyadas por pruebas t-student realizadas.

Continuando con el análisis de los algoritmos, en esta parte nos centramos en el comportamiento paralelo de los algoritmos en las distintas instancias. La Tabla 7.5 incluye los valores de *speedup* (columna *sp*), usando la definición ortodoxa de *speedup*, eficiencia paralela (columna *e*) y fracción serie (columna *fs*), que se obtienen según lo explicado en la primer sección de este capítulo. Estas medidas se pueden obtener si se comparan los mismos algoritmos, dGAn, tanto en secuencia como en paralelo. Lo primero que se advierte es que los valores de *speedup* son bastante altos. Para las instancias con M=250 y M=300, con valores de n=4 y n=8 se observa un *speedup* ligeramente superior al lineal (superlineal). Estos resultados indican que estamos usando buenas implementaciones paralelas de los algoritmos.

Por otra parte lo que se observa es que, efectivamente, se consigue reducir el tiempo de cómputo a medida que aumentamos el número de procesadores. De hecho, la mayoría de los algoritmos ha obtenido un valor casi óptimo de eficiencia paralela. Hay una reducción de la eficiencia para la instancia con M=200, pero de todos modos es un valor aceptable (0,63 % para dGA_2 y más bajo 0,52 % para dGA_8). Como es esperado en un algoritmo bien paralelizado, la fracción serie (*fs*) permanece bastante estable, aunque se observa una reducción de este valor a medida que se incrementa la cantidad de procesadores (excepto en la instancia con M=200).

7.3. ACS paralelos

En esta sección presentamos una descripción del algoritmo ACO paralelo que hemos utilizado en este trabajo para dar solución a 2SPP planteado.

Es muy común en todas las opciones de ACO paralelo estudiadas en la literatura que la construcción de una solución (incluyendo la evaluación de la calidad de sus soluciones) por una hormiga no se divida entre varios procesadores, sino que siempre se realice en un mismo procesador [91, 164]. Una de las razones es que el proceso de construcción de las soluciones en ACO es típicamente un proceso secuencial y es difícil dividirlo en varias partes que pueden ser hechas de forma más o menos independiente. Esto está más relacionado con el

problema específico que se está resolviendo que con ACO en general. Por consiguiente, el tamaño de grano mínimo de un ACO paralelo es la construcción de una única solución. Esto implica que el correspondiente procesador debe poder conocer la información de feromonas y posiblemente la información heurística. La mayoría de los algoritmos ACO paralelos asignan más de una hormiga en cada procesador [13, 163]. Cuando varias hormigas son colocadas en un mismo procesador y esas hormigas trabajan en colaboración más estrecha que con hormigas en otros procesadores, a ese grupo de hormigas se les denomina con frecuencia una *colonia*.

Los algoritmos ACO que tienen varias colonias de hormigas con su propia matriz de feromonas y donde las matrices de feromonas de diferentes colonias no son necesariamente iguales son denominados algoritmos *ACO multicolonia*. Este es el tipo de algoritmo ACO utilizado en este trabajo de tesis. Originalmente, los algoritmos ACO multicolonia fueron diseñados para mejorar el comportamiento de un algoritmo ACO o para ser usado para optimización multiobjetivo [91, 163]. Sin embargo, los algoritmos ACO multicolonia son también muy adecuados para la paralelización, puesto que un procesador puede albergar una colonia de hormigas y en general hay menos intercambio de información entre las colonias como hubiera sido entre los grupos de hormigas en un ACO estándar. Los algoritmos ACO paralelos multicolonias presentan muchas similitudes con GAs distribuidos usando el modelo isla.

Randall y Lewis [185] propusieron una clasificación interesante de las estrategias de paralelismo para la metaheurística ACO: colonias de hormigas paralelas independientes, colonias de hormigas paralelas con interacción, hormigas paralelas, evaluación paralela de los elementos de las soluciones y una combinación de dos de las estrategias mencionadas. En este trabajo consideramos una versión de la primera y segunda estrategia, que describiremos en los párrafos siguientes. La razón para elegirlas es que han mostrado ser buenas alternativas para resolver otros problemas [13], y en general no han mostrado diferencias entre ellas en cuanto a la calidad de resultados reportada.

En el caso de un ACO con colonia de hormigas paralelas independientes, una cierta cantidad de ACOs secuenciales se asignan a los distintos procesadores disponibles. Este

método tiene la particularidad de que no se realizan intercambios de información entre las distintas colonias. El algoritmo de cada isla es el que se muestra en el Algoritmo 6. Esto tiene un efecto positivo sobre el comportamiento debido a que las colonias asignadas en cada procesador se pueden especializar en diferentes regiones del espacio de búsqueda.

La estrategia de colonias de hormigas paralelas interactuantes es similar a la anterior, excepto por el hecho del intercambio de información entre las subcolonias cada ciertas iteraciones preestablecidas. La información intercambiada implica enviar la estructura de los rastros de feromonas de la mejor colonia, aquella con el mejor rendimiento, a todas las restantes, de esta manera las distintas colonias comparten la misma información sobre los niveles de feromona.

El algoritmo ACS distribuido que utilizaremos en este trabajo es el que aparece en el Algoritmo 8. La diferencia con el Algoritmo 6 es el agregado de la fase de comunicación con las colonias vecinas. Tanto la forma de generar una solución como la de representar el rastro de feromonas utilizados en dACS es el descrito en el Capítulo 6. La estructura de comunicación utilizada en este caso se corresponde con una topología anillo unidireccional, donde la información se intercambia cada una cierta cantidad de evaluaciones que realice el algoritmo.

Uno de los aspectos para tener en cuenta en el diseño de un dACO es el tipo de información que se intercambia entre las colonias. Una alternativa es enviar soluciones que se han hallado en una colonia a sus colonias vecinas. Hay diferentes posibilidades para definir cuáles soluciones son las intercambiadas: *i*) migrantes, por lo general se envía la mejor solución de la hormiga de la iteración actual; *ii*) la mejor solución global se envía a todas las colonias; *iii*) la mejor solución del vecindario se envía a todo el vecindario; y *iv*) la mejor solución de una colonia. Otra alternativa es el envío de información de la matriz de feromonas. Estudios realizados por Krüger et al. [138] han demostrado que es más efectivo el intercambio de mejores soluciones que el intercambio de matrices de feromonas completas y añadir las matrices recibidas (multiplicadas por un factor pequeño) a la matriz de feromonas locales. Como los resultados de [138] indican que intercambio de matrices de feromonas no es conveniente, en este trabajo hemos optado por el envío de la mejor solución hallada por

Algoritmo 8 Algoritmo dACS$_i$

```
inicializarValoresFeromonas(τ); {Inicializa los rastros de feromona}
ant^bs=generarSolución(τ, η){Se inicializa ant^bs con una solución inicial aleatoria}
while not(condición de terminación) do
  for j ← 1 to μ do
    ant^i=construirSolución(τ, η);{La hormiga ant^i construye una solución}
    actualizarFeromonaLocal(τ,ant^j); {Actualización local de los rastros de feromona (ACS)}
  end for
  if iteración de intercambio then
    ant_i=seleccionarPeorHormiga(ant)
    ant_i=intercambiarSoluciones(dACS_j) {interacción con el vecindario}
  end if
  evaporarFeromonas(τ) {evaporación}
  actualizarFeromonasGlobal(τ,ant,ant^bs) {intensificación}
  for j ← 1 to μ do
    if f(ant^i) < f(ant^bs) then
      ant^bs=ant^i {Actualizar la mejor solución, actividad del demonio}
    end if
  end for
end while
return la mejor solución encontrada
```

la colonia.

De acuerdo a la topología anillo unidireccional adoptada en este trabajo para comunicar las distintas subcolonias, la subcolonia i trata de influir en los respectivos niveles de rastros de feromonas de la subcolonia $(i{+}1)$ mod n (n es la cantidad de de subcolonias de la red) enviando su mejor solución. La solución recibida se añade a las soluciones que han encontrado las hormigas en la iteración actual. Entonces todo este conjunto de soluciones se utiliza para la actualización de feromona. Por lo tanto, cuando sólo a la mejor solución de una colonia en una iteración se le permite actualizar los valores de feromona, la solución recibida puede influir sólo cuando es mejor que todas las soluciones que han encontrado las hormigas en la misma iteración. Esta actualización genera un uso muy indirecto de la información recibida.

Un complemento a este tipo de uso de la información recibida de la colonia vecina es usar la información de la disposición de las piezas de esta solución de una manera más directa. Por consiguiente, se opta por otra alternativa que consiste en extraer los buenos niveles de la solución recibida, es decir, aquellos niveles que generan el menor desperdicio y

pasarlos a una pseudo-solución. Esta pseudo-solución estará incompleta (le faltarán algunas piezas). La solución recibida se agrega al conjunto de soluciones como se explicó en el párrafo anterior y todo continúa de la forma tradicional. En la siguiente iteración del algoritmo, las hormigas en su proceso de construcción de soluciones tomarán de esa pseudo-solución los niveles seleccionados, terminará de construir la solución asignando las piezas restantes usando las reglas probabilísticas descritas en el Capítulo 6. Esta combinación permite una mezcla de explotación (por la solución recibida) y exploración (por la experiencia de la colonia receptora) a través de la respectiva matriz de feromona.

7.3.1. Configuración de los algoritmos

Para resolver 2SPP usamos un ACS $\mathcal{M}$in-$\mathcal{M}$ax. La cantidad total de hormigas es de 64, cada colonia tiene $64/n$ hormigas, donde n representa la cantidad de colonias. Cada hormiga inicia la construcción de una solución con una pieza seleccionada en forma aleatoria. Los valores de los parámetros son los siguientes: $q_0 = 0.9$, $\rho = 0.8$, γ=0.96, ξ=0.1, y $\beta = 2$. Los valores iniciales de feromonas, $\tau(0)$ son fijados a τ_{min}. Se aplica búsqueda local a todas las soluciones generadas por las hormigas. Estos parámetros no se han elegido de forma arbitraria; son el resultado de un estudio previo dirigido para encontrar la mejor configuración para abordar nuestro problema de empaquetado. El algoritmo ha sido implementado usando el paquete MALLBA [16]. Las máquinas utilizadas para los experimentos son Pentium 4 a 2.4 GHz con 1 GB de RAM y sistema operativo Linux (versión del kernel 2.4.19-4GB). En cada apartado, por cada variante del algoritmo hemos realizados 30 ejecuciones independientes para cada una de las instancias.

7.3.2. Resultados experimentales

En esta sección se presenta la experimentación realizada con un ACO multicolonia para abordar las instancias de 2SPP propuestas. En primer lugar se presenta una comparativa de los algoritmos multicolonia estableciendo un mismo esfuerzo, usando distintas estrategias para incorporar la información de la solución recibida detallada en la sección anterior, para luego estudiarlos desde un punto de vista del paralelismo.

Tabla 7.6: Resultados experimentales para $ACSseq$ y los dACS propuestos

Inst	$ACSseq$		$dACS_{ni}$		$dACS$		$dACS_{mem}$	
	mejor	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$
100	215,78	218,29 $\pm$ 0,88	215,64	218,29 $\pm$ 0,86	**214,73**	217,93 $\pm$ 1,08	215,64	218,18 $\pm$ 0,98
150	216,38	217,82 $\pm$ 0,79	215,64	218,29 $\pm$ 0,86	215,69	217,82 $\pm$ 0,70	**214,79**	217,69 $\pm$ 0,88
200	211,61	214,37 $\pm$ 1,12	215,64	218,29 $\pm$ 0,86	210,77	213,29 $\pm$ 1,28	**210,68**	213,68 $\pm$ 1,18
250	207,68	209,20 $\pm$ 0,76	215,64	218,29 $\pm$ 0,86	207,70	209,28 $\pm$ 0,74	**207,54**	209,20 $\pm$ 0,62
300	213,66	214,74 $\pm$ 0,57	215,64	218,29 $\pm$ 0,86	**211,27**	213,98 $\pm$ 0,96	211,79	213,92 $\pm$ 0,68

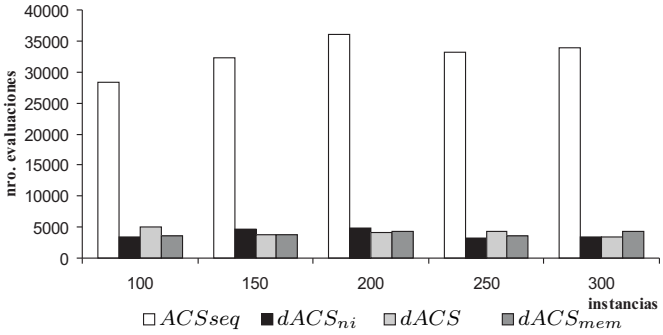


Figura 7.3: Número de evaluaciones promedio para alcanzar el mejor valor de cada algoritmo dACS y seqACS

Resultados con criterio de parada por evaluaciones

En esta sección se comparan diferentes estrategias paralelas de ACS para resolver 2SPP. Se usa un dACS con colonias independientes que no interactúan ($dACS_{ni}$) y dos dACS que intercambian información: $dACS$, que incorpora la solución recibida al conjunto de soluciones de la colonia y $dACS_{mem}$, que además de incorporar la mejor solución selecciona los mejores niveles de la solución recibida para que luego esta información sea utilizada por las hormigas de la siguiente iteración. Además de estos algoritmos multicolonia, se incluyen resultados de un ACO secuencial ($ACSseq$) para mostrar que los ACO multicolonia presentan no sólo tiempos de procesamiento inferiores sino también aportan mejores soluciones al problema. Para hacer una comparación apropiada, todas las propuestas utilizan la misma condición de parada: 65.536 evaluaciones (2^{32}). En la Tabla 7.6 resumimos los resultados obtenidos por ACSseq, $dACS_{ni}$, $dACS$ y $dACS_{mem}$, para cada una de las instancias.

Como se observa en la tabla, las versiones multicolonia son los algoritmos que alcanzan

los mejores patrones de empaquetado para las cinco instancias analizadas; pero sin diferencias significativas en las pruebas estadísticas realizadas para las instancias con M=100, M=150 y M=250, es decir, la prueba estadística indica que todos los algoritmos presentan los mismos valores medios. La diferencia más importante entre las versiones multicolonia y ACS secuencial es el tiempo insumido en la ejecución, como era de esperar (ver Figura 7.3).

Las versiones multicolonia no presentan diferencias entre sí en cuanto a la calidad de las soluciones encontradas, aunque hay una pequeña diferencia a favor de $dACS_{mem}$ si se tiene en cuenta que resuelve más eficazmente tres de las cinco instancias. En cuanto a los tiempos de ejecución hay diferencias significativas entre $dACS$ y $dACS_{mem}$ sólo en las instancias con M=100 y M=250, pero observando los valores medios de ejecución para esas instancias las diferencias son despreciables, por ejemplo, para la instancia con M=100 $dACS$ tarda 58,94 seg. mientras que $dACS_{mem}$, 59.76 seg., lo cual significa que la diferencia es de 0.82 seg., situación semejante se presenta con la instancia con M=250. Esto indica que el procesamiento adicional incurrido en salvar la solución recibida y extraer sus buenos niveles no afecta al tiempo de procesamiento en forma sustancial. Lo cual transforma a esta opción en una alternativa viable para obtener buenas soluciones a 2SPP.

Efectividad y eficiencia de dACS

Esta sección está dedicada para el análisis de las distintas propuestas dACS que intercambian información entre las subcolonias para mostrar sus características paralelas. En lugar de utilizar como condición de parada la computación de una cierta cantidad de iteraciones como en la sección previa, se ha considerado detener el algoritmo cuando alcance una cierta calidad en sus soluciones (ver Tabla 7.3). Para esta experimentación las 8 subcolonias de cada dACS se ejecutaron todas en un procesador y luego cada subcolonia se ejecuta en un procesador dedicado, por lo que hemos utilizado en este caso una plataforma paralela compuesta por 8 máquinas. Cada máquina es un Pentium 4 a 2.4 GHz con 1 GB de RAM y sistema operativo Linux (versión del kernel 2.4.19-4GB), interconectadas mediante una Fast-Ethernet a 100 Mbps. Por cada algoritmo realizamos 30 ejecuciones en forma independiente.

Tabla 7.7: Resultados experimentales para los dACS propuestos

Inst	**Alg**	**1 procesador**			**8 procesadores**		
		***media*$\pm\sigma$**	***eval***	***t[s]***	***media*$\pm\sigma$**	***eval***	***t[s]***
100	$dACS$	218,10 ± 1,10	65,75	156,88	**218,02** ± 0,78	391,92	17,84
	$dACS_{mem}$	218,00 ± 1,05	38,91	91,25	**217,81** ± 1,09	56,27	17,66
150	$dACS$	**217,62** ± 0,64	37,00	276,88	217,64 ± 0,70	4175,00	154,03
	$dACS_{mem}$	217,81 ± 0,80	7196,00	1113,10	**217,70** ± 0,68	103,00	102,64
200	$dACS$	213,94 ± 1,06	89,71	1305,46	**213,46** ± 1,25	496,08	166,26
	$dACS_{mem}$	**213,63** ± 1,10	56,00	910,65	214,02 ± 0,78	1073,25	136,03
250	$dACS$	212,35 ± 0,84	3,53	87,68	212,46 ± 1,01	2,27	8,28
	$dACS_{mem}$	**212,18** ± 1,14	3,40	86,27	212,22 ± 0,99	2,57	9,30
300	$dACS$	**214,04** ± 0,82	61,55	3382,20	214,19 ± 0,70	883,80	608,68
	$dACS_{mem}$	214,20 ± 0,55	76,20	3671,91	**214,01** ± 0,63	56,67	382,07

Comencemos estudiando el comportamiento numérico de los algoritmos. La Tabla 7.7 incluye los promedios de las mejores soluciones alcanzadas por los distintos algoritmos (columna *media*$_{\pm\sigma}$), junto con la cantidad de evaluaciones para alcanzar la calidad de soluciones propuestas (columna *eval*) y los tiempos medios de ejecución promediados teniendo en cuenta las ejecuciones donde se alcanzaron la calidad de soluciones propuestas como objetivo (columna *t[s]*). Lo primero que se observa es la reducción en los tiempos de ejecución entre las opciones ejecutadas en 1 procesador versus las ejecuciones en 8 procesadores, situación esperable. También es de resaltar la menor cantidad de evaluaciones que los algoritmos dACS con subcolonias ejecutándose en un único procesador necesitan para alcanzar el valor objetivo propuesto. Los algoritmos dACS multicolonia no presentan diferencias significativas en lo que respecta al promedio de los mejores valores al ser ejecutados en uno como en ocho procesadores, como era de esperar, debido a que en ambos casos se utiliza el mismo algoritmo de búsqueda. Estas afirmaciones están además validadas desde un punto de vista estadístico.

La Figura 7.4 muestra otro aspecto para analizar de estos algoritmos como es la tasa de éxito de cada algoritmo para resolver las instancias de 2SPP propuestas. Es de destacar que tanto $dACS$ como $dACS_{mem}$ alcanzaron en todas las ejecuciones el valor objetivo propuesto en la instancia con M=250, independiente de la cantidad de procesadores utilizados. Tanto la instancia con M=150 como con M=300, las subcolonias ACS ejecutadas en secuencia obtienen una cantidad semejante de éxitos respecto de sus contrapartes paralelas. La instancia con M=150 ha resultado difícil para cualquiera de los algoritmos dACS propuestos

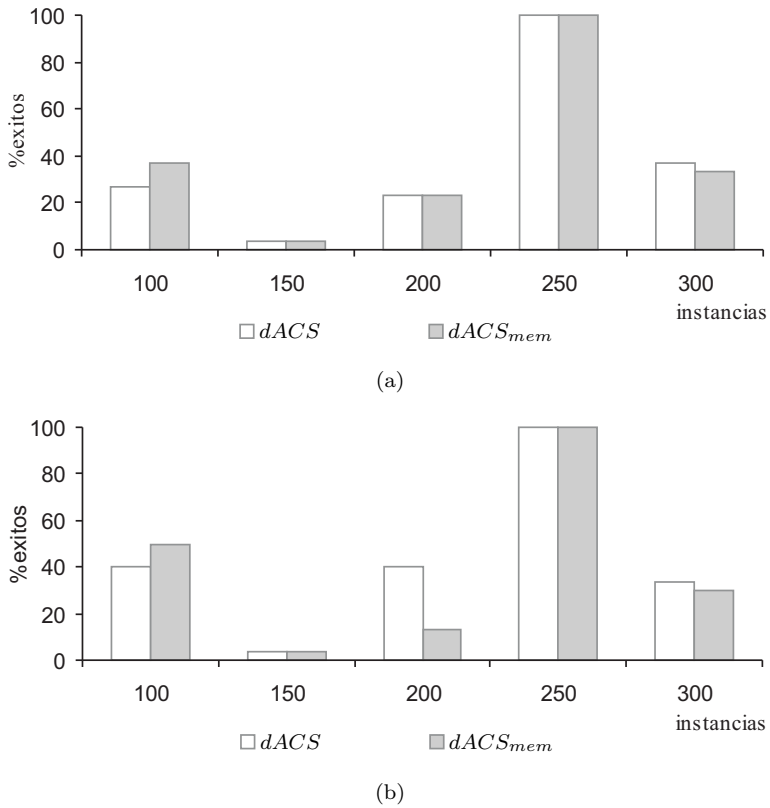


Figura 7.4: Porcentaje de éxitos de cada algoritmo dACS: (a) 1 procesador y (b) 8 procesadores

ya que la tasa de éxitos es igual al 3,3 %, lo que implica que en una única ejecución de las 30 se alcanzó un valor inferior al propuesto.

Si consideramos ahora el rendimiento paralelo de los algoritmos, en la Figura 7.5 se muestran los valores de *speedup* de los dACS, calculados siguiendo la definición ortodoxa de *speedup* (explicado en la Sección 7.1 de este capítulo). Podemos ver que los valores de *speedup* son altos en la mayoría de los casos, salvo el presentado por $dACS$ en la instancia con M=150 el cual no supera el valor dos. En particular, se observan valores de *speedup* superlineales para la instancia con M=250 para ambos algoritmos. Estos resultados indican que estamos usando buenas implementaciones paralelas de los algoritmos.

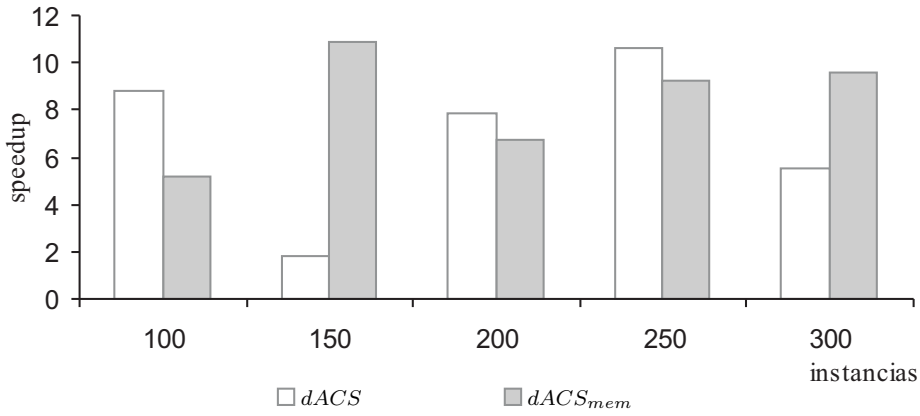


Figura 7.5: Valores de *speedup* de los dACS

7.4. Conclusiones

En este capítulo hemos mostrado variantes paralelas de los algoritmos poblacionales (algoritmos genéticos y algoritmos basados en colonia de hormigas) propuestos en este trabajo para resolver 2SPP con restricciones. En ambos casos se utilizó una estrategia de paralelización que consistió en un modelo multipoblacional (modelo isla), en el cual se realizan intercambios esporádicos de individuos entre las distintas subpoblaciones.

Las características de los modelos distribuidos propuestos han mostrado ser técnicas rápidas que producen buenos patrones de empaquetado, lo que representa un avance prometedor en este ámbito. Hemos mostrado que los dGA_n tanto en secuencial como en paralelo no presentan diferencias en las medias de las mejores soluciones alcanzadas, indicando que ambos algoritmos realizan el mismo tipo de búsqueda; la diferencia entre ellos está dada en los tiempos de ejecución. Los resultados muestran valores de *speedup* altos para dGAs, con valores casi óptimos de eficiencia paralela y en el caso de fracción serie, los valores son bastante estables a medida que se aumenta la cantidad de procesadores, lo cual indica que se consiguió un algoritmo paralelo eficiente para resolver 2SPP.

El siguiente estudio estuvo dedicado a comparar los modelos más difundidos para ACS. Los resultados muestran que los algoritmos multicolonia tienen un comportamiento similar al secuencial, pero los mejores patrones de empaquetado los obtiene $dACS_{mem}$ y $dACS$,

siendo marcada la diferencia en los tiempos de ejecución, como era de esperar. Por otro lado, los resultados indican que para los dACS propuestos el intercambio de información no favorece la búsqueda, conclusiones similares a las obtenidas en [13].

Finalmente, realizamos una comparación más detallada de todos los enfoques distribuidos propuestos en este capítulo. En cuanto a los tiempos de cómputo, cualquiera de los ACS multicolonia resultan en algoritmos más rápidos para resolver cada una de las instancias de 2SPP. Desde el punto de vista de la cantidad de éxitos para alcanzar la misma calidad de soluciones, se observa una tendencia a que dACS es más efectiva que dGA para resolver instancias más grandes del problema, empleando tiempos de cómputo considerablemente menores.

Capítulo 8

Metaheurísticas paralelas heterogéneas propuestas para 2SPP

En este capítulo resolvemos 2SPP por medio del uso de metaheurísticas heterogéneas paralelas (PHM). En el Capítulo 2 ya definimos las metaheurísticas heterogéneas y dimos una clasificación de las mismas. En este capítulo daremos las particularidades de nuestra propuesta. La principal característica de nuestros algoritmos heterogéneos es la utilización de múltiples hilos de búsqueda usando diferentes configuraciones algorítmicas. Proponemos varias clases de PHMs: (a) PHM basadas en operadores aplicando diferentes operadores de recombinación; (b) PHM basadas en la variación de parámetros donde cada hilo de búsqueda aplica diferentes valores de parámetros y (c) PHM basadas en distintas configuraciones donde cada hilo de búsqueda es una metaheurística distinta. Los modelos que nos ocupan en este trabajo presentan diferentes niveles de heterogeneidad. Por un lado, ellos son *modelos heterogéneos basados en la configuración de parámetros* (ver por ejemplo [4, 118]), ya que las subpoblaciones usan diferentes valores de probabilidades del operador MFF_Adj. Pero las subpoblaciones también utilizan distintos operadores de recombinación, por lo que pueden considerarse como *heterogeneidad basada en operadores* [114]. Por otro lado, los modelos muestran *heterogeneidad basada en colaboración*, ya que sus respectivas subpoblaciones cooperan, pero no compiten, a fin de realizar la búsqueda. La Sección 8.1.1 presenta las características de PHMs basadas en operadores y basadas en la variación de

parámetros y los resultados a los que hemos arribado con estos algoritmos. Por su parte la Sección 8.2 describe PHMs usando distintas metaheurísticas en cada hilo de búsqueda junto con el análisis de los resultados obtenidos. En ambos casos el desempeño de nuestras propuestas heterogéneas se compara con algoritmos homogéneos tanto desde un punto de vista numérico como desde el esfuerzo computacional.

8.1. Propuestas heterogéneas usando el mismo procedimiento de búsqueda

Esta sección está dedicada a las opciones heterogéneas que utilizan en cada hilo de búsqueda una misma metaheurística: algoritmos genéticos. Esta sección se divide en dos apartados: en el primero detallamos las variantes heterogéneas basadas en operadores y en el segundo, las basadas en variación de parámetros.

8.1.1. Heterogeneidad basada en operadores

La gran disponibilidad de operadores de recombinación para representaciones basadas en permutaciones posibilita diferentes grados de exploración y explotación en el mismo algoritmo. En este caso, cada dGA_i usa un operador de recombinación distinto (descritos en Sección 5.4.1) con la misma probabilidad. En este caso el operador de mutación es el mismo en cada dGA_i y la probabilidad de MFF_Adj se fija en 1,0.

Los algoritmos heterogéneos propuestos son un método multi-resolución usando varios operadores de recombinación. De esta forma los algoritmos heterogéneos pueden realizar en forma simultánea una búsqueda diversificada y una refinación local efectiva. Las migraciones entre subpoblaciones en el anillo pueden inducir el refinamiento y la expansión de las mejores zonas emergentes. El tiempo entre dos intercambios consecutivos está determinado en 100 evaluaciones.

Los algoritmos heterogéneos propuestos son:

- Het_{Trad}: las subpoblaciones de este primer algoritmo usan diferentes operadores de recombinación tradicionales para representaciones basadas en permutaciones (mencionados en la Sección 5.4). La Figura 8.1 presenta una gráfica de la disposición de

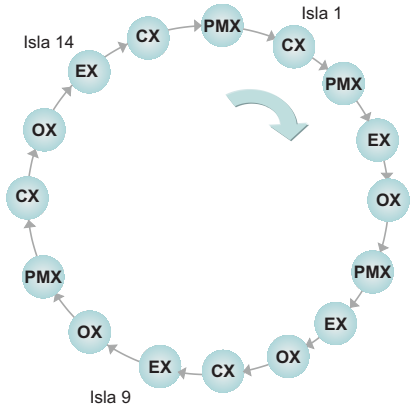


Figura 8.1: Configuración del algoritmo Het_{Trad}

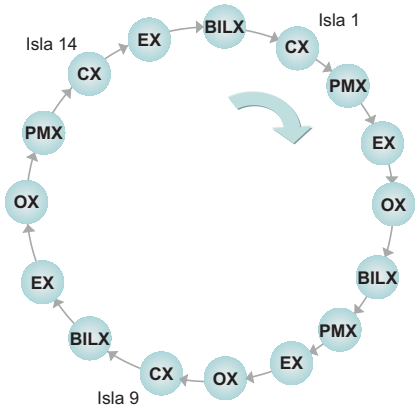


Figura 8.2: Configuración del algoritmo $\text{Het}_{BILX+Trad-1}$

las subpoblaciones en el anillo, indicando en cada caso el operador de recombinación utilizado.

- $\text{Het}_{BILX+Trad-1}$: esta PHM tiene subpoblaciones usando BILX junto con operadores tradicionales de recombinación. La Figura 8.2 muestra la secuencia en la cual se han entremezclado los diferentes operadores de recombinación.

- $\text{Het}_{BILX+Trad-2}$: este algoritmo tiene la mitad de las islas usando BILX (islas con numeración par) propagando buenos patrones de empaquetado a sus vecinas. Clasificamos las islas en función a su posición en el anillo: todas las islas con numeración par usan el operador BILX mientras que las islas impares usan PMX, OX, CX o EX (ver la Figura 8.3 para una explicación gráfica). La propuesta es que el envío de buenas soluciones por parte de las islas pares pueda beneficiar a las impares para mejorar sus patrones de empaquetado. Por otra parte, las islas con numeración impar pueden introducir diversidad genética a las islas con numeración par.

8.1.2. Heterogeneidad basada en la variación de parámetros

Varios estudios han mostrado que la combinación de varios hilos que implementan diferentes configuraciones de parámetros incrementan la robustez de la búsqueda global

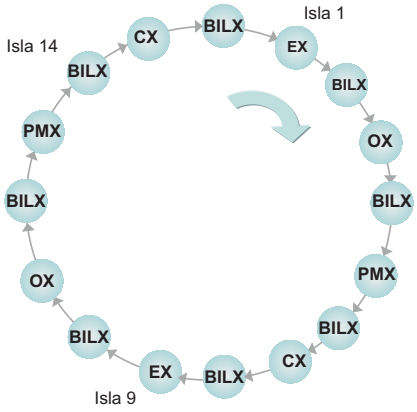


Figura 8.3: Configuration del algoritmo $\text{Het}_{BILX+Trad-2}$

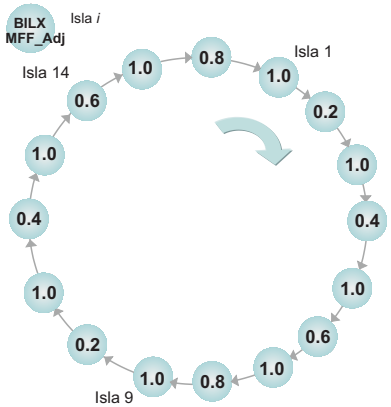


Figura 8.4: Asignación de probabilidades del operador MFF_Adj para cada isla del algoritmo $\text{Het}_{BILX}\text{Var}$

[14, 15, 115, 210]. Esta clase de heterogeneidad representa una extensión directa del modelo homogéneo canónico; consecuentemente, es la clase más difundida de PHMs.

La inclusión del operador MFF_Adj en el proceso evolutivo permite mantener la diversidad genética en niveles aceptables durante el proceso de búsqueda y provee un muestreo rápido del espacio de búsqueda, como se mostró en capítulos anteriores. Por lo tanto, los hilos de búsqueda con altas probabilidades de aplicar MFF_Adj deberán tener poblaciones con diversidad genética junto con buenos patrones de empaquetado, pero con tiempos de ejecución bastante elevados. A medida que se disminuye la probabilidad de aplicar MFF_Adj, GA pierde diversidad y calidad de soluciones. En consecuencia, es difícil establecer el valor correcto de probabilidad con el fin de lograr un equilibrio entre calidad de soluciones y tiempo de ejecución.

Teniendo en cuenta este equilibrio en mente, $\text{Het}_{BILX}\text{Var}$ y $\text{Het}_{BILX+Trad-1}\text{Var}$ aplican el operador MFF_Adj con diferentes probabilidades en cada subpoblación. Este parámetro varía desde 0,2 hasta 1,0. $\text{Het}_{BILX}\text{Var}$ usa BILX y SE como operadores genéticos para el GA que guía cada hilo de búsqueda, mientras que $\text{Het}_{BILX+Trad-1}\text{Var}$ usa la misma configuración en cuanto a los operadores de recombinación que el algoritmo $\text{Het}_{BILX+Trad-1}$ descrito en la sección anterior. La Figura 8.4 presenta una explicación gráfica del algoritmo, resaltando los valores de probabilidades de MFF_Adj usados en cada una de las islas.

8.1.3. Caso homogéneo base

La calidad de las soluciones y la eficiencia de los algoritmos heterogéneos propuestos son comparadas con un algoritmo homogéneo, denominado PHoGA, a fin de mostrar la robustez de los algoritmos heterogéneos como método de búsqueda alternativo. Este algoritmo homogéneo realiza la misma clase de búsqueda sobre diferente conjunto de individuos que se generan inicialmente en forma aleatoria. Los operadores genéticos considerados son BILX y SE (Sección 5.4), aplicados con las probabilidades usadas en los capítulos anteriores. Además PHoGA usa MFF_Adj (introducido en Sección 5.5) con una probabilidad del 100 % en cada isla.

8.1.4. Configuración de los algoritmos

En esta sección detallamos los parámetros utilizados por los algoritmos paralelos propuestos para resolver las instancias de 2SPP, cuyos resultados se muestran en la siguiente sección. La población de todos los modelos evaluados está compuesta por 512 individuos. Por lo tanto las distintas variantes dGA usan 32 individuos por subpoblación. Por defecto, la población inicial se genera en forma aleatoria. El máximo número de evaluaciones que realiza cada algoritmo es de 2^{16}. Los operadores de recombinación se aplican con una probabilidad de 0,8 mientras que la probabilidad de mutación es de 0,1. El operador de relocalización se aplica a todas las nuevas soluciones generadas. El esquema de distribución de los algoritmos distribuidos está basado en una topología anillo y el intercambio de individuos entre las subpoblaciones ocurre cada 100 evaluaciones. Estos parámetros (tamaño de población, criterio de parada, probabilidades, etc.) se eligen tras un examen de algunos valores usados previamente con éxito [196].

8.1.5. Resultados experimentales

En esta sección se presenta la experimentación realizada con los algoritmos heterogéneos planteados en la Sección 8.1 y PHoGA para abordar las instancias planteadas del problema de empaquetado en estudio. Los resultados se han dividido en tres apartados. En primer

lugar, se analizan las propuestas algorítmicas heterogéneas basadas en operadores y basadas a nivel de parámetros en un GA. En esta comparativa hemos incluido un algoritmo homogéneo, PHoGA (descrito en Sección 8.2.1) para medir la calidad de las soluciones y eficiencia de los algoritmos heterogéneos antes mencionados, a fin de mostrar la robustez del algoritmo heterogéneo como un método de búsqueda alternativo. El siguiente paso ha sido evaluar las propuestas heterogéneas no sólo desde un punto de vista de calidad de soluciones, sino también desde el punto de vista del paralelismo. Nótese que todos los algoritmos han sido ejecutados con dos criterios de parada distintos: uno basado en alcanzar un número máximo de evaluaciones (esfuerzo predefinido), mientras que el otro consiste en ejecutar un dGA hasta que se encuentre una solución con una calidad predefinida o se alcance el número máximo de evaluaciones estipulado (calidad de soluciones predefinida). Debemos medir el *speedup* entre el tiempo serial (dGAs usando un procesador) contra el tiempo paralelo (dGAs usando 16 procesadores), porque tanto las opciones heterogéneas como la homogénea usan el mismo algoritmo subyacente. Asumimos la forma ortodoxa para reportar nuestros resultados a la comunidad. Para terminar, esta sección concluye con un análisis del comportamiento de dos algoritmos heterogéneos para mostrar el comportamiento interno del proceso de búsqueda, a fin de adquirir un mejor entendimiento del funcionamiento de un PHM.

Resultados con esfuerzo predefinido

Los primeros resultados que incluimos en este capítulo miden la calidad de las soluciones obtenidas por los algoritmos heterogéneos basados en distintos operadores y basados en la variación de parámetros. Todos los resultados provistos representan un promedio sobre 30 ejecuciones independientes.

El primer análisis con el que empezaremos esta discusión está centrado en comparar las PHMs propuestas basadas en operador (explicados en la Sección 8.1.1). La Tabla 8.1 muestra los resultados de PHoGA y las tres opciones heterogéneas: Het_{Trad}, $\text{Het}_{BILX+Trad-1}$ y $\text{Het}_{BILX+Trad-2}$. Como se puede observar desde esa tabla, las mejores soluciones son siempre halladas por alguna opción heterogénea, excepto para las instancias con M=150

Tabla 8.1: Resultados experimentales obtenidos por PHoGA y PHMs basadas en operadores

Inst	**PHoGA**		**Het**$_{Trad}$		**Het**$_{BILX+Trad-1}$		**Het**$_{BILX+Trad-2}$	
	mejor	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$
100	215,75	217,58 $\pm 0,58$	**213,77**	217,10 $\pm 1,04$	213,84	217,01 $\pm 1,10$	214,46	217,20 $\pm 0,94$
150	**211,90**	214,83 $\pm 0,76$	212,86	214,44 $\pm 0,63$	213,72	214,69 $\pm 0,56$	213,65	214,68 $\pm 0,48$
200	207,87	211,51 $\pm 1,12$	208,38	210,32 $\pm 0,90$	**207,61**	210,66 $\pm 1,13$	209,69	210,64 $\pm 0,86$
250	**210,63**	212,36 $\pm 0,83$	210,69	212,22 $\pm 0,63$	210,77	212,27 $\pm 0,55$	210,90	212,22 $\pm 0,53$
300	210,73	212,31 $\pm 0,80$	**208,85**	211,62 $\pm 0,70$	209,73	211,54 $\pm 0,89$	210,68	211,97 $\pm 0,84$

Tabla 8.2: Resultados experimentales obtenidos por PHoGA y PHMs basadas en variación de parámetros

Inst	**PHoGA**		**Het**$_{BILX+Trad-1}Var$		**Het**$_{BILX}Var$	
	mejor	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$	***mejor***	***media***$\pm\sigma$
100	215,75	217,58 $\pm 0,58$	214,74	216,94 $\pm 0,95$	**212,56**	217,58 $\pm 0,58$
150	**211,90**	214,83 $\pm 0,76$	213,78	214,62 $\pm 0,35$	212,49	214,83 $\pm 0,76$
200	207,87	211,51 $\pm 1,12$	208,77	210,53 $\pm 0,94$	**207,70**	211,51 $\pm 1,12$
250	**210,63**	212,36 $\pm 0,83$	211,58	212,29 $\pm 0,43$	211,91	212,36 $\pm 0,83$
300	210,73	212,31 $\pm 0,80$	209,81	211,49 $\pm 0,79$	**208,92**	212,31 $\pm 0,80$

y M=250, pero podemos indicar que no hay diferencias significativas entre esas opciones, puesto que los respectivos valores p obtenidos con la prueba ANOVA son mayores que el nivel de confianza (establecido en 0,05). Sin embargo, la opción homogénea alcanza las mejores soluciones para las instancias con M=150 y M=250, siendo la diferencia con el algoritmo heterogéneo más notable en la primera de ellas.

Introduzcamos ahora en el análisis los resultados de PHMs basadas en la variación de parámetros presentadas en la Sección 8.1.2. Como se puede observar en la Tabla 8.2, Het$_{BILX}Var$ es el algoritmo que obtiene los patrones de empaquetado con menor altura para las instancias con M=100, M=200 y M=300, también es importante remarcar que las peores soluciones las presenta la opción homogénea. PHoGA, por su parte, muestra el mejor comportamiento para las instancias con M=150 y M=250, como fuese observado en el análisis previo. Los resultados de la prueba de comparaciones múltiples indican, además, que las diferencias no son significativas entre todas las opciones algorítmicas, utilizando un nivel de 95 % de confianza, en todas las instancias.

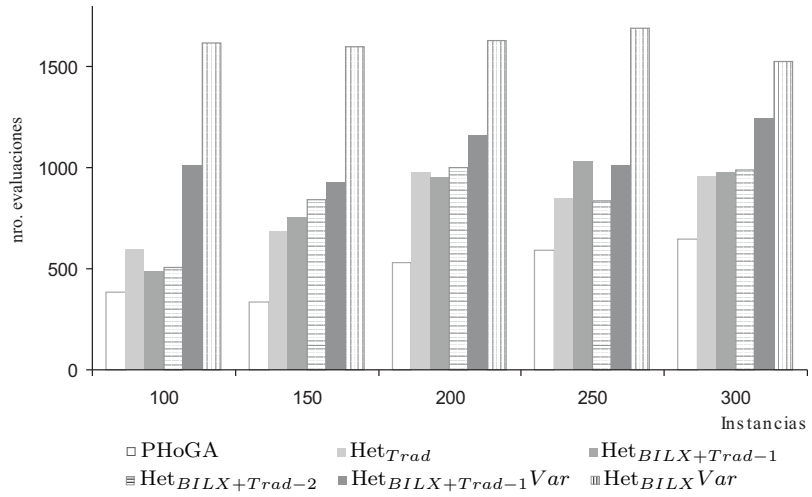


Figura 8.5: Número de evaluaciones promedio para alcanzar el mejor valor de las propuestas heterogéneas y PHoGA

La Figura 8.5 muestra la cantidad promedio de evaluaciones necesarias para que cada una de las opciones heterogéneas y PHoGA alcancen sus mejores valores. De esta figura se puede observar que PHMs basadas en operadores presentan un esfuerzo numérico menor para encontrar el mejor valor que PHMs basadas en la variación de parámetros. Vale destacar que las opciones heterogéneas propuestas presentan valores promedios bastante diferentes, en este caso superiores, a los valores de PHoGA. Entre las opciones heterogéneas basadas en operadores se observan diferencias, pero el análisis estadístico revela una similaridad en las evaluaciones medias de cada opción (los valores p son mayores que α), excepto para la instancia con M=100. En cuanto a PMHs basados en parámetros, la diferencia en el número de evaluaciones es importante, resultando $\text{Het}_{BILX+Trad-1}Var$ la mejor opción. Finalmente, podemos indicar que PHoGA presenta un mejor comportamiento que todos las opciones heterogéneas para esta métrica, situación corroborada por las pruebas estadísticas realizadas a un nivel de confianza del 95 %.

Por último, hacemos referencia al tiempo incurrido en la búsqueda por los distintos algoritmos, situación que se muestra en la Figura 8.6. Het_{Trad} resulta la opción con menor tiempo de procesamiento en todas las instancias. Esta situación es esperable debido a las diferencias en tiempos reportadas por los distintos operadores tradicionales y BILX (ver

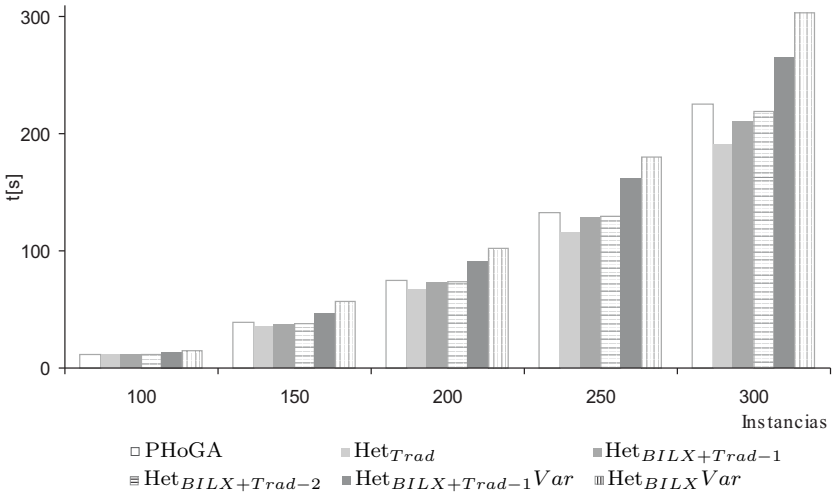


Figura 8.6: Tiempo de ejecución promedio de las opciones heterogéneas y PHoGA

Tabla 8.3: Valores numéricos para cada instancia

Instancias	100	150	200	250	300
Valor Objetivo	216,10	216,02	211,00	213,19	211,25

Sección 5.8.2). Por otro lado, PHMs basadas en valores de parámetros y PHoGA no presentan diferencias significativas. Estas afirmaciones son corroboradas por una prueba ANOVA (los valores p son cercanos a 0).

Llegado este punto, podemos extraer conclusiones atendiendo a los resultados de los experimentos realizados. Podemos indicar que las opciones heterogéneas basadas en operadores presentan buenos patrones de empaquetado con menor altura usada de la plancha de material junto con tiempos de ejecución aceptables comparada con PHoGA. En particular, como la calidad de soluciones ha resultado ser semejante entre las opciones, nos inclinamos a analizar tiempos de ejecución resultando Het_{Trad} la propuesta más adecuada. Por consiguiente, las opciones heterogéneas aquí planteadas podrían considerarse una técnica alternativa y potente para hallar soluciones más exactas para 2SPP bajo estudio.

Efectividad y eficiencia de las opciones heterogéneas

A continuación cambiamos la clase de análisis que estamos realizando para estudiar los algoritmos propuestos desde un punto de vista distinto. Hasta ahora hemos tratado

de dar un esfuerzo predefinido y hallar el mejor algoritmo, pero ahora nos enfrentaremos a un escenario en el que mediremos el esfuerzo computacional (número de evaluaciones) de los algoritmos para encontrar una solución preestablecida. Por lo tanto, los algoritmos se detienen cuando encuentran una determinada calidad de soluciones para una instancia dada. La solución objetivo depende claramente de la instancia y seleccionaremos el valor promedio obtenido por un GA panmíctico usando BILX, SE y MFF_Adj, extraído de la Tabla 5.4 y reproducido en Tabla 8.3 para cada instancia, como valor de parada para los algoritmos sobre una instancia, con el fin de ofrecer un escenario que no esté sesgado por un valor *ad hoc*.

Hasta ahora, hemos presentado resultados que se corresponden con valores medios de más de 30 ejecuciones independientes. A continuación, se muestra la tasa de éxito de los algoritmos homogéneos y heterogéneos. Esta medida es la relación entre la cantidad de ejecuciones en las que se alcanzó el valor de *fitness* objetivo (reportado en Tabla 8.3) y el número total de pruebas realizadas. Comenzamos con el análisis de los resultados cuando los algoritmos se ejecutan en 1 CPU, es decir los 16 procesos comparten el mismo procesador, resultados que se presentan en la Figura 8.7(a). En todas las instancias resueltas, cualquier algoritmo heterogéneo obtiene tasas de éxito mayores que PHoGA, excepto $\text{Het}_{BILX+Trad-1}Var$. Esta situación se puede explicar por el modelo de búsqueda mejorado que presentan los algoritmos heterogéneos, mediante la combinación de distintos operadores y, en algunos casos, también a diferentes tasas, lo que justifica la contribución de esta propuesta. Si analizamos la tasa de éxito cuando los diferentes algoritmos se ejecutan en 16 CPUs (cada subalgoritmo tiene una máquina dedicada) de la Figura 8.7(b) podemos observar que los enfoques heterogéneos también superan a PHoGA, exceptuando nuevamente a $\text{Het}_{BILX+Trad-1}Var$. En media, cualquiera de las PHMs obtienen una tasa de éxito por encima del 60 % en forma independiente de usar 1 o 16 procesadores (exceptuando a $\text{Het}_{BILX+Trad-1}Var$), mientras que PHoGA para 1 procesador no supera el 45 % y en 16 procesadores apenas alcanza el 50 % de éxitos.

La siguiente medida que analizar es el esfuerzo numérico para resolver las distintas instancias de 2SPP. Los resultados de los distintos algoritmos en 1 y 16 CPUs se muestran en

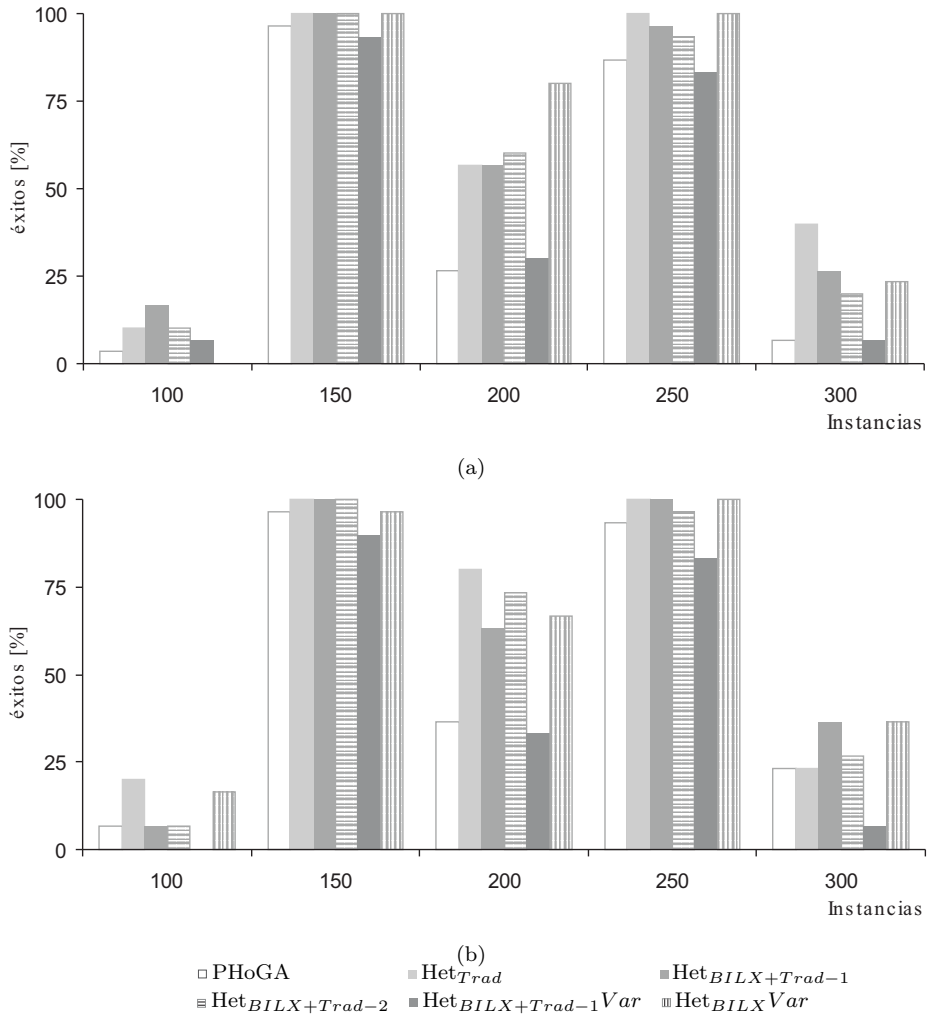


Figura 8.7: Tasa de éxito de las opciones heterogéneas y PHoGA: (a) 1 CPU y (b) 16 CPUs

la Tabla 8.4. Nos encontramos con que un algoritmo es más eficiente que el resto dependiendo de la instancia y del número de CPUs. Por lo tanto, no podemos concluir nada respecto de la superioridad de alguna de las versiones heterogéneas, ya que cualquiera parece igualmente adecuada y eficaz para los casos de prueba considerados.

Ahora comencemos con el estudio del comportamiento paralelo. La Tabla 8.5 incluye los valores de *speedup* de los algoritmos analizados. En este caso, comparamos los mismos algoritmos distribuidos ejecutándose tanto en secuencial como en paralelo (1 contra 16 procesadores), siguiendo una definición ortodoxa de *speedup* (ver Sección 7.1). Lo que primero se

Tabla 8.4: Número de evaluaciones de PHoGA y de las variantes propuestas de PHMs

Inst	**PHoGA**	**Heterogeneidad Basada en operadores**			**Heterogeneidad Basada en parámetros**	
		Het$_{Trad}$	**Het**$_{BILX+Trad-1}$	**Het**$_{BILX+Trad-2}$	**Het**$_{BILX}Var$	**Het**$_{BILX Trad-1}Var$
			1 proc			
100	48,00	360,67	2609,00	1837,00	4097,50	0,00
150	194,00	74,27	316,23	215,77	273,32	112,67
200	1786,38	810,35	873,82	1090,17	1401,67	1219,71
250	836,35	810,95	621,31	819,79	679,40	627,50
300	1272,50	1705,17	1530,00	2141,33	2179,50	1631,00
			16 proc			
100	1615,00	2064,17	1365,50	956,50	0,00	1626,80
150	192,93	78,50	86,80	122,43	118,00	76,24
200	1342,73	1289,96	1641,16	1449,09	1016,60	942,90
250	440,25	262,00	205,93	410,03	527,52	426,17
300	2077,43	886,29	1951,45	2050,38	463,50	1652,36

observa es que, en promedio, las opciones heterogéneas presentan valores de *speedup* ligeramente mejores que los obtenidos por PHoGA, indicando que los algoritmos heterogéneos son más propicios para la paralelización. El valor de *speedup* es relativamente alto, y en la mayoría de los casos se encuentra cercano al lineal, con excepción de Het$_{BILX}$Var. Este último algoritmo presenta los valores de *speedup* más altos, resultando super lineales, excepto para la instancia con M=100 para la cual no alcanza la calidad de soluciones predefinidas bajo 16 procesadores. Un resultado sorprendente, un valor por encima de 4 × #procesadores, es el presentado por Het$_{BILX}$Var para la instancia con M=300. Valores de *speedup* superlineal pueden ser sorprendentes, pero es un hecho conocido e informado en metaheurísticas [10]. Sin embargo, la mayoría de las veces, la fuente de *speedup* superlineal no es sólo por contar con más procesadores, sino también por disponer de más discos que permite una memoria virtual más rápida y especialmente más espacio en caches. Por lo tanto, las subpoblaciones de un algoritmo distribuido al ser de menor tamaño encajan en la caché de los procesadores en paralelo; si bien este no es el caso de las poblaciones del algoritmo ejecutado en secuencia (en 1 procesador), mucho más grandes. Esto produce como consecuencia que obtenemos, en paralelo, un acceso a los individuos que es mucho más rápido que en la ejecución de un algoritmo distribuido en secuencial, lo que provoca un crecimiento repentino en la velocidad de los cálculos conduciendo a un *speedup* superlineal.

Tabla 8.5: *Speedup* de PHoGA y de las variantes propuestas de PHMs

Inst	**PHoGA**	**Heterogeneidad Basada en operadores**			**Heterogeneidad Basada en parámetros**	
		Het$_{Trad}$	**Het**$_{BILX+Trad-1}$	**Het**$_{BILX+Trad-2}$	**Het**$_{BILX}Var$	**Het**$_{BILX\ Trad-1}Var$
100	3,56	4,20	14,97	22,82	-	0,00
150	5,06	14,46	9,75	9,22	18,08	17,22
200	14,58	7,85	7,46	10,91	18,76	16,62
250	14,56	30,19	21,77	11,86	11,65	7,68
300	7,55	28,26	11,82	14,11	64,99	12,97

Análisis detallado de la dinámica de los algoritmos

Esta sección está dedicada al análisis del comportamiento de dos algoritmos heterogéneos: Het$_{BILX+Trad-2}$ y Het$_{BILX}$Var, tomados como ejemplo. Nuestro objetivo es ofrecer una traza de la búsqueda realizada por los algoritmos heterogéneos. Este análisis se elabora teniendo en cuenta la variación de la diversidad genética durante el proceso evolutivo, como así también la variación del *fitness* medio de la población mientras se realiza la evolución. Para realizar esta clase de estudio, utilizamos como criterio de parada el alcanzar un máximo número de evaluaciones. Debido a la naturaleza estocástica del proceso evolutivo, todos los resultados mostrados aquí se obtienen promediando 30 ejecuciones independientes de la misma experimentación. Las gráficas muestran el comportamiento de las 16 subpoblaciones de los algoritmos distribuidos. El valor de entropía se calcula de la misma manera como explicado en la Sección 5.8.1 de este trabajo.

Comencemos con uno de los algoritmos heterogéneos seleccionados, Het$_{BILX+Trad-2}$. Las Figuras 8.8 y 8.9 muestran los gráficos de la variación de la entropía y *fitness* medio respecto del esfuerzo computacional para la instancia con M=200 (similares resultados se obtienen para las otras instancias). Observamos que las subpoblaciones con numeración par (aquellas islas que no aplican el operador BILX) mantienen niveles de entropía más altos (y por consiguiente mayor diversidad genética) que las subpoblaciones con numeración impar (islas que sí usan BILX). Esta era una de las razones por las cuales proponíamos entremezclar islas con diferentes niveles de diversidad genética. En cuanto a la aptitud media de la población, observamos que, en promedio, las calidades de las soluciones son más altas en las subpoblaciones con numeración impar, traduciéndose a valores bajos de *fitness*. Cabe

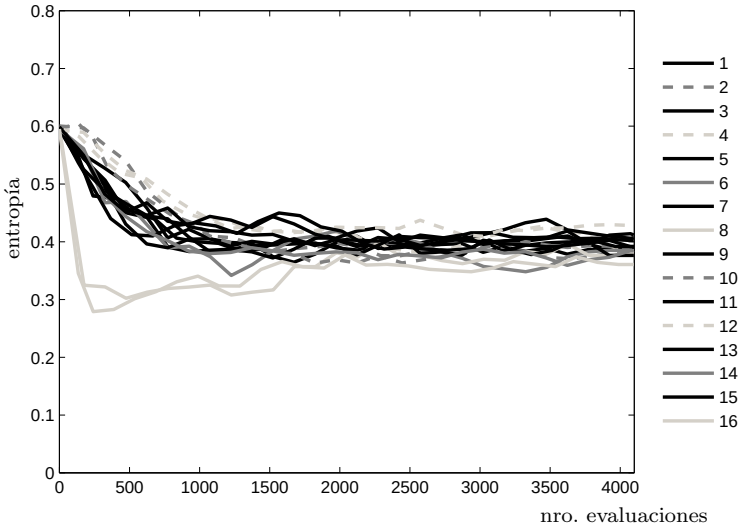


Figura 8.8: Entropía versus esfuerzo computacional de cada isla de $\text{Het}_{BILX+Trad-2}$ (instancia con M=200)

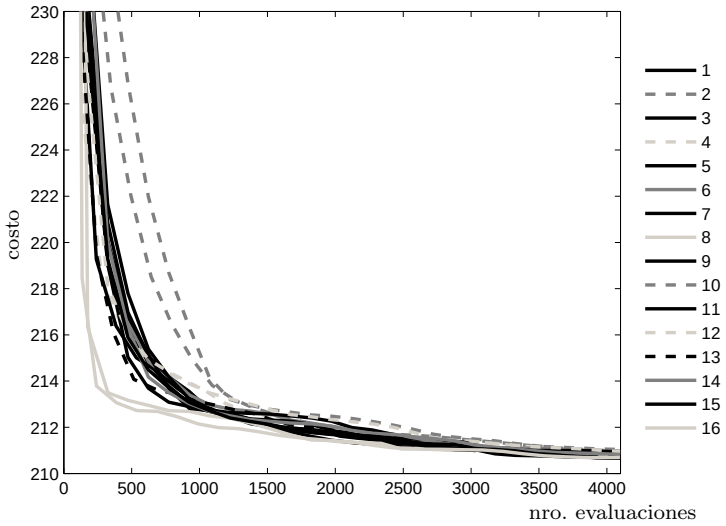


Figura 8.9: Promedio poblacional versus esfuerzo computacional de cada isla de $\text{Het}_{BILX+Trad-2}$ (instancia con M=200)

destacar que los picos que se presentan en las curvas de entropía se deben al proceso de migración y significa que los individuos recibidos aportan material genético ya perdido en la subpoblación. Este fenómeno es fácilmente comprensible si recordamos que el tamaño de las subpoblaciones es pequeño. Además, no parece evidente la existencia de una correlación entre la capacidad de las subpoblaciones utilizando BILX para encontrar soluciones de buena calidad y la diversidad genotípica presente. La entropía media se mantiene alrededor del valor 0,4, sin decrementarse, en las etapas finales del proceso de búsqueda, una característica importante de resaltar.

Otro aspecto que hemos analizado en el comportamiento del algoritmo $\text{Het}_{BILX+Trad-2}$ es la relación entre el *fitness* de la solución que se recibe (inmigrante) de una subpoblación vecina con respecto del *fitness* de las existentes en la subpoblación receptora. La Figura 8.10 muestra el porcentaje de veces (eje y) en que al menos una de las soluciones en la subpoblación es mejor que la solución recibida por cada una de las subpoblaciones (eje x). De esta figura se desprende que las subpoblaciones impares presentan bajos porcentajes, indicando que las soluciones existentes en las mismas poseen mejor *fitness* que las inmigrantes. Este es un escenario esperado debido a los buenos patrones de empaquetado obtenidos por un GA con la aplicación de BILX, ya mencionado en la Sección 5.8.2.

Continuando con el análisis planteado en el párrafo anterior, ahora nos centramos en la influencia de las soluciones inmigrantes respecto de la diversidad genética de la subpoblación destino. Este escenario se muestra en la Figura 8.11, donde las barras grises representan el porcentaje de veces que el valor de entropía mejora con la inserción de la nueva solución recibida, mientras que las barras más oscuras denotan el porcentaje de veces que el valor de entropía decrementa. La razón de una pérdida de diversidad genética frente a una recepción está justificada en el hecho que la solución inmigrante tiene material genético similar o idéntico a uno o más individuos de la población receptora. Se puede observar una favorable influencia de la migración en la diversidad porque en todas las subpoblaciones las barras grises son más altas que las barras de color gris oscuro.

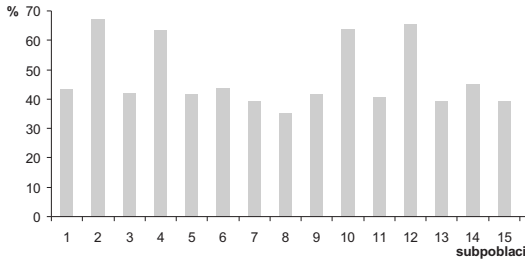


Figura 8.10: Porcentaje de veces que la solución recibida fue mejor que alguna solución de la subpoblación destino (instancia con M=200)

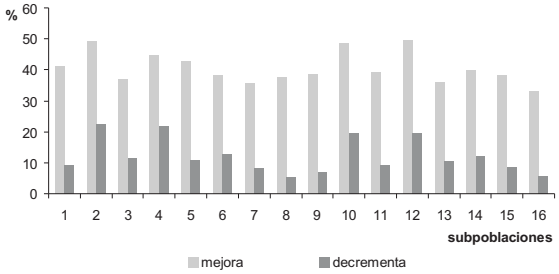


Figura 8.11: Variación de la entropía por subpoblación en función de la recepción de soluciones (instancia con M=200)

A partir de este punto hacemos referencia al análisis del comportamiento exhibido por Het$_{BILX}$Var. La Figura 8.12 muestra la evolución de entropía de cada subpoblación. Observamos que la diversidad genética disminuye tras una etapa inicial del proceso de búsqueda y luego tiende a estancarse. Otra observación es que las subpoblaciones con numeración impar mantienen alta diversidad genética, comportamiento que era previsible debido a que al aplicar el operador MFF_Adj con más frecuencia ayuda a mantener la diversidad genética (ver Sección 5.8.3 para más detalles). Respecto de la variación del *fitness* medio en función del esfuerzo, no se observan diferencias entre las distintas subpoblaciones (Figura 8.13. En cuanto a las diferencias en los valores de *fitness* entre los migrantes y las individuos en la población (véase Figura 8.14), se presenta una situación similar en todas las subpoblaciones. La Figura 8.15 muestra la influencia de las recepciones en los valores de la entropías en las distintas subpoblaciones, en este caso nuevamente los mayores porcentajes de mejora se corresponden con las subpoblaciones utilizando el MFF_Adj con bajas probabilidades. La mejora en esas subpoblaciones es un resultado esperado debido a los GAs sin MFF_Adj pierden diversidad en forma muy repentina, como se observó en la Sección 5.8.3. Recapitulando, el operador MFF_Adj reorganiza las piezas con el fin de reducir el espacio desperdiciado dentro de cada nivel del patrón de empaquetado, de esta manera el movimiento de piezas ayuda para mantener la diversidad durante la búsqueda en buenos niveles.

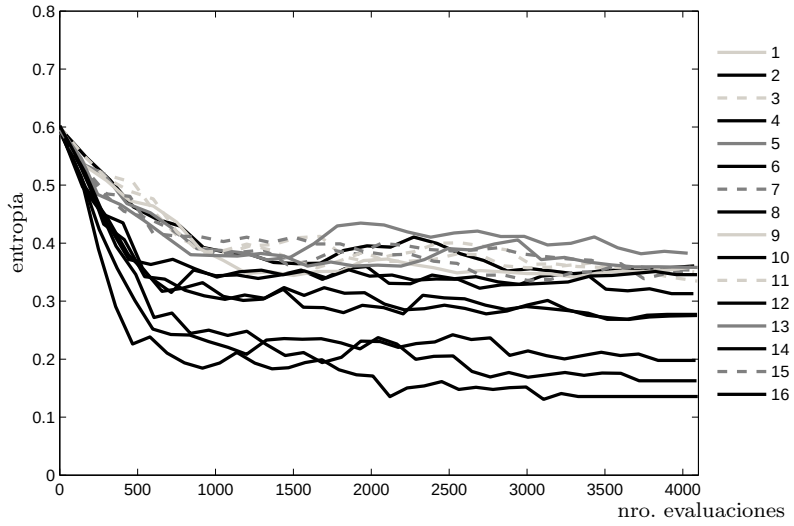


Figura 8.12: Entropía promedio contra esfuerzo computacional de cada isla de Het$_{BILX}$Var (instancia con M=200)

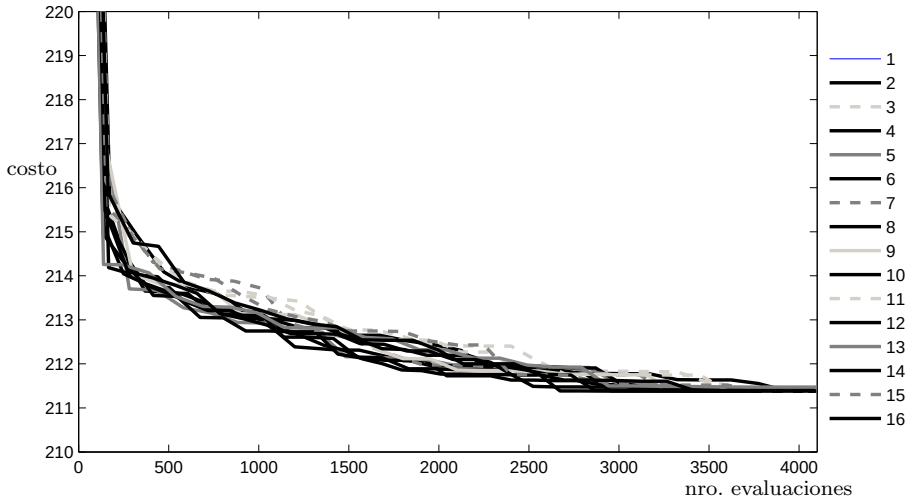


Figura 8.13: Población promedio contra esfuerzo computacional de cada isla de Het$_{BILX}$Var (instancia con M=200)

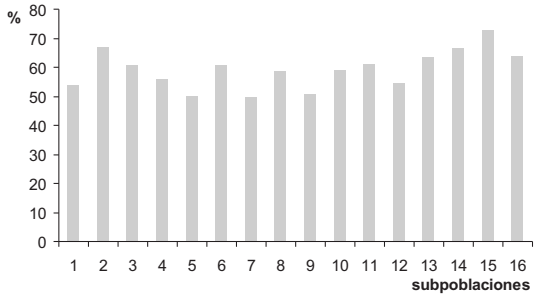


Figura 8.14: Porcentaje de veces que la solución recibida fue mejor que alguna solución de la subpoblación receptora (instancia con M=200)

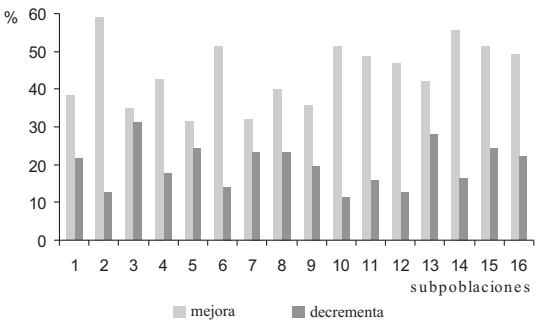


Figura 8.15: Variación de la entropía por subpoblación en función de la recepción de soluciones (instancia con M=200)

8.2. Propuesta heterogénea al usar distintos algoritmos de búsqueda

En esta sección detallamos las propuestas heterogéneas desarrolladas utilizando, en este caso, distintos mecanismos de búsqueda en cada hilo de búsqueda: GAs y ACO, es decir, se produce una cooperación entre ambas metaheurísticas para encontrar la mejor solución.

Con el creciente número de nuevas metaheurísticas, las búsquedas colaborativas entre los diferentes métodos se han convertido en más habituales. En general, si se utilizan metaheurísticas basadas en trayectoria en esta colaboración, la comunicación se realiza a través de un conjunto centralizado de soluciones. Las metaheurísticas basadas en población normalmente colaboran mediante la transferencia de soluciones entre las subpoblaciones.

Por lo general, las PHMs que aparecen en la literatura que caen dentro de esta clasificación, combinan metaheurísticas poblacionales, principalmente GAs, con basadas en trayectoria tal como SA o TS [38, 93]. También se encuentran metaheurísticas SA y TS con métodos de búsqueda local [173]. En particular, hay pocas propuestas heterogéneas paralelas que en cada hilo de búsqueda usen metaheurísticas poblacionales. Una de ellas es la propuesta en [1], la cual está formada por dos hilos de búsqueda: uno usa un GA y el otro un algoritmo ACO, denominada GAACO; ambos se ejecutan en paralelo y ocurre intercambio de información (en la forma de soluciones) cuando alguno de ellos encuentra

una mejor solución después de una iteración. Comparan el GAACO con implementaciones de GA y ACO tradicionales, obteniendo mejoras en la calidad de los resultados para los problemas planteados.

Como hemos descrito en capítulos anteriores, los modelos que hemos desarrollado para resolver 2SPP tanto de GA como de ACO comparten la misma representación del problema, por lo que es posible el intercambio de información entre ellas bajo la forma de soluciones (permutaciones de piezas). La configuración desarrollada está basada en una idea muy simple, a la que denominamos Het_{GA+ACS}. El modelo paralelo está basado en una concepción de islas cooperantes respetando una topología anillo, donde las islas ejecutando GA y ACO están intercaladas. La propuesta es que por medio de la cooperación de ambos algoritmos se obtenga una retroalimentación de la distintas características de búsqueda que ambas metaheurísticas realizan. Cada isla ya sea usando GA como ACO intercambian sus mejores soluciones halladas en la iteración actual, con una frecuencia previamente establecida como parámetro. El GA la incorpora a su población en el caso de que la solución recibida sea mejor que el peor individuo de la población actual. El ACO, por su parte, además de actualizar el rastro de feromonas, utiliza esta solución para ayudar a las hormigas de la siguiente iteración en su proceso de construcción de soluciones (como se describió en la Sección 7.3).

Como hemos visto en el capítulo anterior, el tiempo total de búsqueda de dGA es bastante menor que dACS, por lo tanto hemos propuesto dos alternativas heterogéneas variando la frecuencia en la cual las colonias envían sus mejores soluciones a las subpoblaciones vecinas, a fin de mejorar la cooperación entre las islas. Het_{GA+ACS} realiza un intercambio de soluciones cada 100 evaluaciones tanto en las islas con GA como con ACS. Por otra parte, $\text{Het}_{GA+ACS}dm$ se caracteriza por tener distintas frecuencias de migración en las distintas islas, dependiendo del tipo de metaheurística que usen. En este caso las colonias de ACS envían con mayor frecuencia (cada 50 evaluaciones) sus mejores soluciones a las islas vecinas en el anillo (un GA), mientras que las islas con GAs mantienen su frecuencia cada 100 evaluaciones.

8.2.1. Caso homogéneo base

Como en la experimentación realizada previamente con las propuestas heterogéneas, en este caso comparamos la calidad de las soluciones y la eficiencia de Het_{GA+ACS} y $\text{Het}_{GA+ACS}dm$ con los algoritmos homogéneos, a fin de mostrar la robustez de los algoritmos heterogéneos como método de búsqueda alternativo. Utilizamos dos algoritmos homogéneos:

- PHoGA: como descrito en la sección 8.2.1
- PHoACS: un algoritmo homogéneo como se describió en la Sección 7.3.

8.2.2. Configuración de los algoritmos

En esta sección explicamos las configuraciones utilizadas para los algoritmos Het_{GA+ACS} y $\text{Het}_{GA+ACS}dm$ para resolver las instancias de 2SPP. El máximo número de evaluaciones que realiza cada algoritmo es de 2^{16}. El esquema de distribución de los algoritmos está basado en una topología anillo.

Cada subpoblación del GA está compuesta por 32 individuos. Por defecto, la población inicial se genera en forma aleatoria. El operador de recombinación utilizado BILX se aplica con una probabilidad de 0,8 mientras que la probabilidad de usar el operador de mutación SE es de 0,1. El operador de relocalización se aplica a todas las nuevas soluciones generadas.

Cada colonia de ACO está compuesta por 8 hormigas. Cada hormiga inicia la construcción de una solución con una pieza seleccionada en forma aleatoria. Los valores de los parámetros son los siguientes: $q_0 = 0{,}9$, $\rho = 0{,}8$, γ=0,96, ξ=0,1 y $\beta = 2$. Los valores iniciales de feromonas τ_0 son fijados a τ_{min}. Se aplica búsqueda local a todas las soluciones generadas por las hormigas.

8.2.3. Resultados experimentales

En esta sección estudiaremos el comportamiento de los algoritmos heterogéneos Het_{GA+ACS} para abordar las instancias planteadas del problema de empaquetado en estudio. Estos Het_{GA+ACS} se comparan con dos versiones homogéneas PHoGA y PHoACS. Los resultados de los experimentos se muestran en las siguientes secciones.

Tabla 8.6: Resultados experimentales obtenidos por PHoGA, PHoACS y Het$_{GA+ACS}$

Inst	**PHoGA**		**PHoACS**		**Het$_{GA+ACS}$**		**Het$_{GA+ACS}dm$**	
	mejor	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**	***mejor***	***media*$\pm\sigma$**
100.00	215.75	217.58 ±0,58	217.63	218.59 ±0,32	215.57	218.15 ±1,08	**214.76**	218.41 ±0,90
150.00	**211.90**	214.83 ±0,76	216.67	217.93 ±0,73	216.71	218.19 ±0,76	215.77	217.95 ±0,83
200.00	**207.87**	211.51 ±1,12	210.79	213.63 ±1,37	212.52	214.41 ±1,00	210.83	214.16 ±1,20
250.00	210.63	212.36 ±0,83	208.11	209.21 ±0,59	208.42	209.75 ±0,55	**208.06**	209.81 ±0,78
300.00	**210.73**	212.31 ±0,80	211.65	214.03 ±0,78	211.79	214.49 ±0,86	213.13	214.40 ±0,65

Resultados con esfuerzo predefinido

Inicialmente estudiaremos el comportamiento de los algoritmos para medir la calidad de las soluciones de Het$_{GA+ACS}$ y Het$_{GA+ACS}dm$. Todos los resultados provistos representan un promedio sobre 30 ejecuciones independientes. El criterio de parada adoptado en todos los algoritmos es parar el algoritmo tras un número de evaluaciones predefinidas. Este número máximo de evaluaciones es 2^{16}, 4096 en cada isla, como en los experimentos previos.

En la Tabla 8.6 mostramos los mejores valores alcanzados en las 30 ejecuciones junto con el promedio de esos valores y su desvío típico. Como podemos observar, en tres de las cinco instancias los mejores valores los obtienen PHoGA mientras que en las restantes lo hace Het$_{GA+ACS}dm$. Además las diferencias entre PHoGA y el resto de los algoritmos en todos los valores son estadísticamente significativas (las pruebas ANOVA realizadas arrojaron valores p bien cercanos a cero), excepto en la instancia con M=250 donde Het$_{GA+ACS}dm$ presenta las diferencias. Es decir la colaboración entre las islas con distintos mecanismos de búsqueda no es fructífera en general.

Analizando el esfuerzo numérico, en la Figura 8.16 se presenta una marcada diferencia entre el esfuerzo número que necesita PHoGA y el resto de las opciones tanto heterogéneas como homogénea, con diferencias significativas en las pruebas estadísticas realizadas. Entre las opciones heterogéneas se observa una tendencia a que resulte más favorable que las colonias envíen con mayor frecuencia sus mejores soluciones a sus islas vecinas con GA, esta característica se observa tanto en los mejores valores obtenidos como en el número de evaluaciones para encontrar el mejor valor. Esto tendría su razón de ser si se analizan

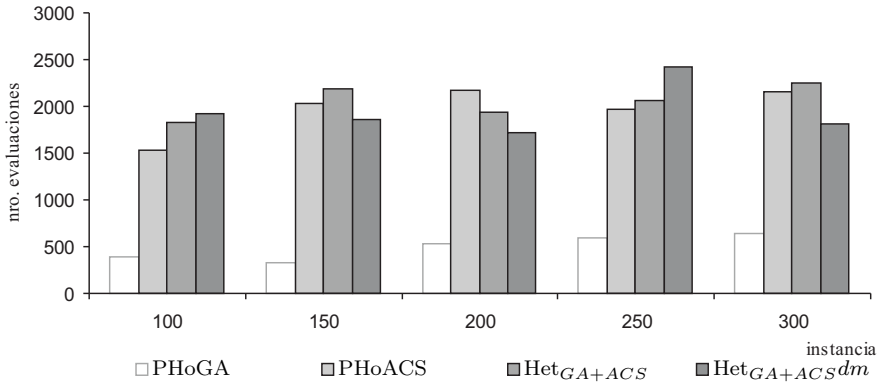


Figura 8.16: Número de evaluaciones promedio de los algoritmos Het$_{GA+ACS}$ para alcanzar el mejor valor

los tiempos de ejecución de PHoGA y PHoACS (ver Figura 8.17), las primeras resultan en opciones más rápidas, entonces que la colonia envíe más tempranamente sus mejores soluciones puede producir una colaboración más positiva.

Efectividad y eficiencia de las opciones heterogéneas

En esta sección realizaremos el estudio de Het$_{GA+ACS}$ mostrando el esfuerzo computacional de los algoritmos para encontrar una solución preestablecida (ver Tabla 8.3).

Iniciamos el estudio con la tasa de éxitos que exhiben los algoritmos homogéneos y heterogéneos Het$_{GA+ACS}$. En la Figura 8.18(a) se muestra los resultados cuando los algoritmos se ejecutan en una CPU. En todas las instancias resueltas, tanto PHoGA como Het$_{GA+ACS}dm$ obtienen tasas similares de éxito. Es de remarcar lo que ocurre en la instancia con M=250: tanto PHoACS como las dos variantes heterogéneas alcanzan en las 30 ejecuciones realizadas la solución objetivo. Situación similar se presenta cuando se analiza la tasa de éxito cuando los diferentes algoritmos se ejecutan en 16 CPUs (Figura 8.18(b)). En este caso nuevamente, podemos inferir que el hecho que las colonias envíen con mayor frecuencia sus mejores soluciones (Het$_{GA+ACS}dm$) produce un efecto positivo en las islas ejecutando GAs, que en el caso de utilizar Het$_{GA+ACS}dm$. El peor comportamiento lo presenta PHoACS, el cual para la mayoría de las instancias no alcanza los valores propuestos.

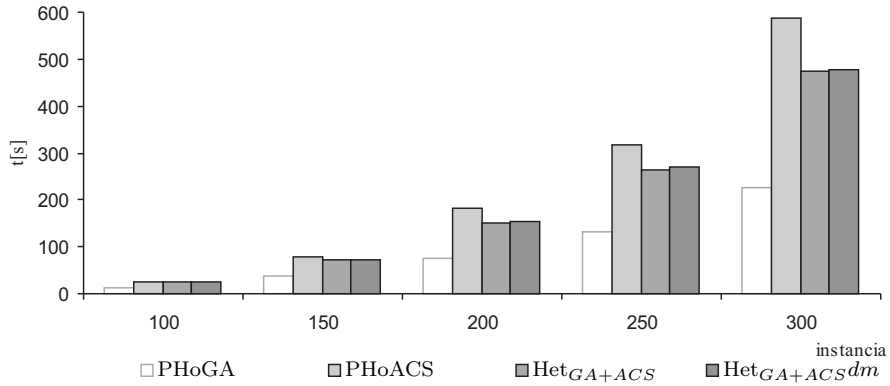


Figura 8.17: Tiempo de ejecución de los algoritmos Het$_{GA+ACS}$

Tabla 8.7: *Speedup* de PhoGA, PHoACS y Het$_{GA+ACS}$

Inst	**PHoGA**	**PHoACS**	**Het$_{GA+ACS}$**	**Het$_{GA+ACS}dm$**
100	3,56	35,97	9,63	12,10
150	5,06	0,00	14,07	10,61
200	14,58	0,00	8,98	8,04
250	14,56	23,68	18,16	14,59
300	7,55	0,00	0,00	3,32

En media, se observan valores cercanos al 50 % de éxito para los algoritmos ejecutados en 16 procesadores, mientras que al ejecutarlos en 1 procesador no se supera el 44 %.

Concluimos la sección mostrando los valores de *speedup* exhibidos por los distintos algoritmos paralelos en la Tabla 8.7. La mayoría de los algoritmos presenta valores de speedup por debajo de la linealidad, exceptuando PHoACS en las dos únicas instancias donde encuentra soluciones con la calidad prefijada. Este último algoritmo en particular presenta valores muy altos de speedup, por encima de la linealidad.

8.3. Conclusiones

En este capítulo presentamos una solución a 2SPP por medio de algoritmos paralelos heterogéneos. La principal característica de nuestros algoritmos heterogéneos es la utilización de múltiples hilos de búsqueda usando diferentes operadores de recombinación (PHMs basadas en operadores), valores de parámetros (PHMs basadas en en los valores de los

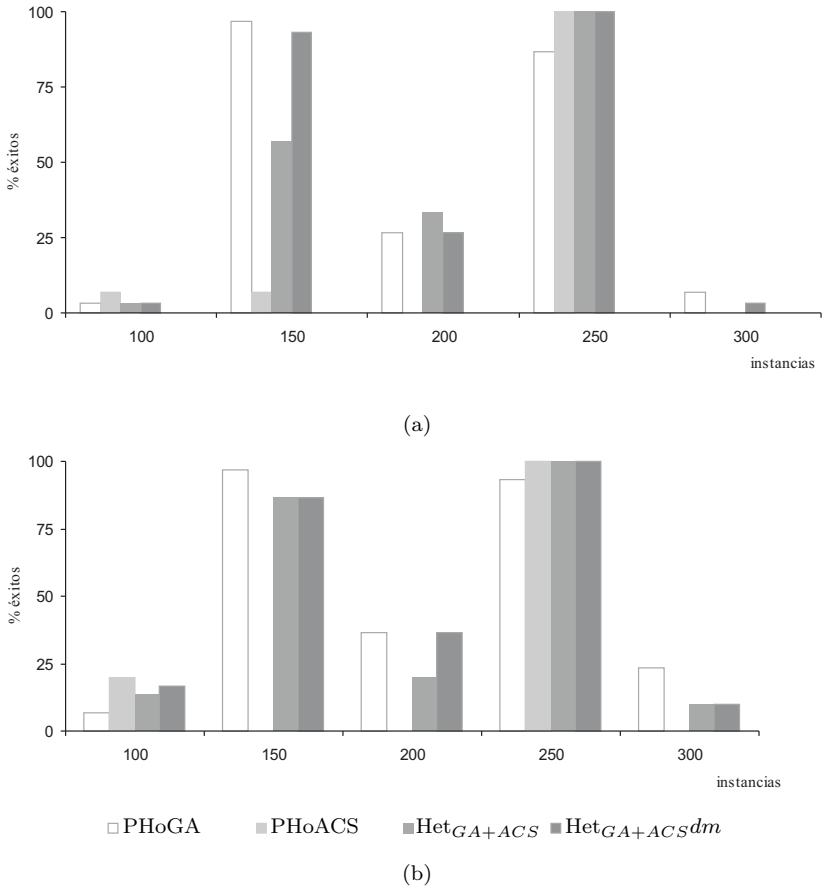


Figura 8.18: Tasa de éxito de las opciones Het$_{GA+ACS}$, PHoGA y PHoACS: (a) 1 CPU y (b) 16 CPUs

parámetros) y distintos algoritmos de búsqueda.

En primer instancia, hemos analizado PHMs basadas en operadores y basadas en parámetros imponiendo un número predefinido de evaluaciones, para demostrar su eficacia para resolver las instancias de 2SPP. Los resultados muestran que en general los PHMs se comportan de manera similar a PHoGA. Sin embargo, PHMs basadas en operadores son ligeramente más rápidas que PHoGA. De todas las opciones de PHMs basadas en operadores, Het$_{Trad}$ presenta el mejor comportamiento tanto en eficiencia como en eficacia. Por su parte, Het$_{BILX}Var$ es la mejor PHM basada en variación de parámetros, pero con tiempos

de ejecución elevados en relación con las restantes PHMs analizadas. Los algoritmos heterogéneos son más adecuados para la paralelización que PHoGA, debido a los valores altos de *speedup* alcanzados, en particular aquellos alcanzados por Het$_{BILX}Var$ que superan la linealidad.

En una segunda etapa hemos mostrado los resultados obtenidos por PHMs que usan distintos algoritmos de búsqueda. Los resultados muestran que PHoGA y Het$_{GA+ACS}dm$ poseen una eficacia similar en general. Pero PHOGA presenta diferencias significativas con el resto de los PHMs en todos los valores, en este caso la colaboración entre islas con distintos mecanismos de búsqueda no es fructífera en general. De los PHMs propuestos Het$_{GA+ACS}dm$ presenta mejores soluciones que Het$_{GA+ACS}$, lo que supone que es más favorable que las colonias envíen con mayor frecuencia sus mejores soluciones a sus islas vecinas, esto se podría justificar por las diferencias en tiempos de ejecución entre dGAs y dACS.

Las características de nuestros algoritmos, especialmente las de PHMs basadas en operador, han demostrado dar lugar a técnicas rápidas obteniendo buena calidad de resultados, lo que representa un futuro interesante en este ámbito. Por otra parte, estos algoritmos mejoran resultados ya reportados en capítulos previos. Así, el objetivo de obtener un método mejorado para resolver 2SPP se logró mediante el diseño de nuestras PHMs basadas en operador.

Para dar cierre a este trabajo, nos centraremos en mostrar el mejor valor encontrado para cada instancia, indicando en cada caso el algoritmo que lo ha obtenido, para establecer el estado del arte en este conjunto de instancias. Este resumen se presenta en la Tabla 8.8, de estos resultados se observa que en tres instancias los mejores valores fueron hallados por un algoritmo genético secuencial utilizando alguna variante del operador de relocalización. Para la instancia con M=250 el mejor algoritmo siempre resultó un ACS. La instancia con M=200 fue mejor resuelta por una metaheurística paralela heterogénea.

Habiendo determinado los mejores algoritmos para cada instancia, resta analizar si existen diferencias significativas en los resultados reportados por cada uno de ellos al considerar el conjunto de instancias analizadas. Para ello, en la Tabla 8.9 se extraen los valores medios

Tabla 8.8: Resumen de los mejores valores reportados por algún algoritmo para cada instancia

Inst	**Mejor solución**	**Algoritmo**
100	211,34	GA+SA+FF
150	210,89	GA+MFF_Adj
200	207,61	$\text{Het}_{BILX+Trad-1}Var$
250	207,54	$dACS_{mem}$
300	206,41	$GA_{Rseed}BF$

Tabla 8.9: Valores medios de las mejores soluciones obtenidas por GA+SA+FF, GA+MFF_Adj, $\text{Het}_{BILX+Trad-1}Var$, $dACS_{mem}$ y $GA_{Rseed}BF$

Inst	**GA+SA+FF**	**GA+MFF_Adj**	**Het**$_{BILX+Trad-1}Var$	$dACS_{mem}$	$GA_{Rseed}BF$	**prueba**
100	215,62 $\pm 1,70$	216,10 $\pm 1,45$	216,94 $\pm 0,95$	218,03 $\pm 0,96$	220,00 $\pm 1,76$	+
150	214,36 $\pm 0,56$	216,02 $\pm 1,77$	214,62 $\pm 0,35$	218,51 $\pm 0,96$	216,11 $\pm 0,76$	+
200	211,88 $\pm 1,11$	211,00 $\pm 1,81$	210,53 $\pm 0,94$	214,86 $\pm 1,16$	210,96 $\pm 1,52$	+
250	212,71 $\pm 0,55$	213,19 $\pm 1,16$	212,29 $\pm 0,43$	210,13 $\pm 0,70$	212,92 $\pm 1,13$	+
300	213,04 $\pm 1,12$	211,25 $\pm 1,84$	211,46 $\pm 0,79$	214,62 $\pm 0,92$	212,63 $\pm 2,16$	+

de las mejores soluciones junto con su desviación típica de cada uno de los algoritmos. Se realizaron estudios estadísticos indicando que existen diferencias significativas entre los cinco algoritmos para todas las instancias (símbolo "+" en la columna **prueba**). Si bien no se puede decir cuál es el algoritmo más adecuado para resolver estas instancias, sí se puede indicar que, al realizar el estudio estadístico de comparaciones múltiples, $\text{Het}_{BILX+Trad-1}Var$ es el algoritmo que figura en las primeras posiciones del ordenamiento para todas las instancias. Por otra parte, si se considera que es un algoritmo paralelo, corre en ventaja con el resto de los algoritmos si se analizan sus tiempos de ejecución.

Parte III

Conclusiones y trabajo futuro

Conclusiones

Esta tesis doctoral ha estado dedicada a la resolución de problemas de corte y empaquetado utilizando técnicas metaheurísticas. En particular el problema que se ha abordado es el de *strip packing* (2SPP), que consiste en empaquetar un conjunto de piezas rectangulares en una plancha de material cuya característica es que una de sus dimensiones (por lo general el alto) no está definida. El objetivo es entonces encontrar una distribución de todas las piezas, sujeta a un conjunto de restricciones, en la plancha de modo tal de minimizar la altura necesaria para empaquetar todas las piezas. Este problema tiene una aplicación real, ya que está presente en numerosas situaciones en la industria como el corte de vidrio, entre otros.

Primero, hemos introducimos las metaheurísticas junto con su clasificación y el problema de empaquetado en cuestión. Luego discutimos diferentes alternativas presentes en la literatura que resuelven 2SPP con metaheurísticas. De allí se desprende que las metaheurísticas más utilizadas en este caso son los algoritmos genéticos (GAs). Este último tiempo también se empieza con la aplicación, y con buen resultado, de otras metaheurísticas más nuevas como lo son algoritmos de optimización basados en colonia de hormigas (ACO) y GRASP. Por esta razón adoptamos GAs y ACO como motores de búsqueda de buenas soluciones a 2SPP. Por cada una de las metaheurísticas hemos realizado una pormenorizada descripción de su adecuación para resolver 2SPP.

Hemos desarrollado nuestro propio conjunto de datos de prueba generado de una manera estructurada para poder realizar la experimentación de forma coherente, exhaustiva en cuanto a los valores de los atributos de las piezas. Además para evitar disparidades en la complejidad de los datos de prueba existentes. Por otra parte es de destacar la ausencia,

en los papers existentes en la literatura, de los datos que medimos en este estudio y de las características del problema de *strip packing* consideradas, como hemos resaltado en la revisión de trabajos existentes en la literatura, lo que representa un aporte importante de este trabajo.

Hemos realizado un estudio de distintas alternativas de GA para seleccionar una configuración adecuada a 2SPP. Los resultados muestran que GA usando BILX y SE (operadores propuestos en este trabajo) ha resultado ser muy eficiente para resolver 2SPP comparado con GA aplicando operadores tradicionales de permutaciones, no sólo por la bondad de sus resultados sino también por el menor número de evaluaciones que necesita. Esto sugiere que el diseño de operadores genéticos que incorporan información específica del problema en cuestión funcionan especialmente bien comparados con operadores tradicionales. Hemos diseñado distintas reglas simples para generar semillas, las cuales se incorporan en las poblaciones iniciales de GA. Este simple y rápido mecanismo ha mostrado incrementar la bondad de las poblaciones iniciales de GA y además obtener buenos patrones de empaquetado finales en un número menor de evaluaciones.

En esta tesis se presenta un estudio en colonias de hormigas para determinar la correcta definición tanto de la preferencia heurística como de los rastros de feromona que usan las hormigas para construir sus soluciones para 2SPP. Los resultados han mostrado una clara ventaja al usar $\eta_j = h_j$, justificado por la forma en que se construyen los patrones de empaquetado. Por otra parte, hemos usado la propiedad de que C&P es un problema de agrupamiento para definir el significado del rastro de feromonas: se ha dado preferencia a la asignación de dos piezas en un mismo nivel; esta alternativa ha permitido obtener buenos resultados para 2SPP, como era de esperar, al compararla con la alternativa de codificar la preferencia de que dos piezas estén consecutivas en patrón de empaquetado. Ambos componentes: información heurística y definición del rastro de feromonas han mostrado ser claves para un adecuado funcionamiento del proceso de búsqueda de ACS propuesto. Los resultados han mostrado que los algoritmos basados en colonias de hormigas han sido los más rápidos en encontrar soluciones de calidad y los mejores cuando se consideran instancias más grandes. Además para favorecer la explotación de información previa hallada

por las hormigas, se ha diseñado un ACS con memoria externa (ACSMem) que ha obtenido soluciones de calidad semejante a ACS pero con exhibiendo tiempos de cómputo menores, una característica muy importante para una propuesta nueva.

La hibridación con heurísticas específicas del problema ha resultado ser una estrategia muy prometedora para obtener soluciones competitivas tanto en GAs como en ACSs, intensificando la búsqueda alrededor de algunas regiones del espacio de soluciones. Por lo general los resultados muestran que los GAs hibridados con MFF han obtenido los mejores patrones de empaquetado para 2SPP comparados con algoritmos ACSs en tres de las cinco instancias analizadas. La excepción la constituye instancia con $M = 250$, para la cual los motores de búsqueda basados en ACS han mostrado una versatilidad especial en casi todas las variantes propuestas a lo largo de este trabajo de tesis. En particular, el mejor valor para esta instancia es reportado por algoritmo multicolonia con memoria externa.

Las características de los modelos propuestos distribuidos tanto de GA como de ACS han mostrado ser técnicas rápidas que producen buenos patrones de empaquetado, lo que representa un avance prometedor en este ámbito. Los algoritmos distribuidos han sido capaces de obtener un mejor rendimiento numérico (menor esfuerzo) con similares niveles de precisión comparado con sus versiones secuenciales. Observación que representa un avance prometedor en este ámbito. La resolución de 2SPP con nuestros modelos distribuidos ha permitido obtener valores de *speedup* cercanos al lineal en la mayoría de los casos, observándose también valores superlineales, indicando que los modelos paralelos propuestos han tomado ventaja de los elementos de procesamiento adicionales.

Finalmente, hemos incursionado en el ámbito de metaheurísticas paralelas heterogéneas (PHMs) donde cada uno de los hilos de búsqueda utiliza un procedimiento de búsqueda distinto, ya sea a nivel algorítmico o a nivel de configuración de parámetros. Los resultados han revelado que las PHMs basadas en operadores y basadas en variación de parámetros se comportan de manera similar a PHoGA en cuanto a calidad de resultados. Además, las PHMs basadas en operadores resultaron técnicas más rápidas cuando se las compara con PHoGA y PHMs basadas en variación de parámetros. Respecto al comportamiento paralelos, las PHMs basadas en operadores y basadas en variación de parámetros han exhibido

tasas de éxito mayores a PHoGA, resultando además más adecuadas para paralelizar que PHoGA por los altos valores de *speedup*, en particular los de Het$_{BILX}Var$. Ha sucedido algo notable y es que PHoACS ha obtenido tiempos de ejecución más elevados que PHoGA y el resto de las PHMs, con pobre calidad de soluciones.

Como resumen de este trabajo de tesis doctoral, podemos indicar que los mejores valores numéricos para cada una de las instancias están dadas por alguno de los algoritmos genéticos desarrollados usando el operador de relocalización, que ha mostrado brindar un correcto equilibrio entre exploración y explotación. La excepción se presenta en la instancia con M=250, para la cual las versiones multicolonia obtienen los mejores patrones de empaquetado. Si bien no se puede decir cuál es el algoritmo más adecuado para resolver el conjunto de instancias, Het$_{BILX+Trad-1}Var$ es el algoritmo que figura en las primeras posiciones del ordenamiento para todas las instancias. Por otra parte, si se considera que es un algoritmo paralelo, corre en ventaja con el resto de los algoritmos si se analizan sus tiempos de ejecución. Por lo tanto, el objetivo de obtener un método mejorado para resolver 2SPP ha sido alcanzado al diseñar nuestros PHMs. Los resultados obtenidos han sido siempre validados por estudios estadísticos, lo cual da sustento a las conclusiones aquí vertidas dando rigurosidad a los estudios existentes.

Trabajos futuros

La incorporación de información representativa del problema de empaquetado dentro de la dinámica de las metaheurísticas ha probado ser fuente de información para que la búsqueda se dirija a regiones del espacio con soluciones cercanas a la óptima. Esta incorporación de información se puede dar en operadores genéticos, en semillas para la inicialización de la población, en el proceso de construcción de la solución que realizan las hormigas, etc. Estas ideas se podrían trasladar a otros problemas de optimización donde las soluciones se representen por medio de permutaciones.

También, como se desprende de la parte experimental, sería interesante incrementar el número y tipo de instancias para evaluar el comportamiento de los distintos algoritmos propuestos. Como expresamos oportunamente, todas las instancias utilizadas fueron generadas bajo la misma relación de aspecto.

El trabajo futuro en sistemas heterogéneos es altamente prometedor. Planeamos extender el concepto de heterogeneidad a los parámetros que afectan a la dinámica de los métodos paralelos distribuidos. En concreto, los parámetros que definen la topología y la frecuencia del intercambio de información son los que más condicionan el comportamiento del método y, en consecuencia un buen ajuste de estos parámetros es vital para un rendimiento óptimo. Por lo tanto, una extensión muy interesante es estudiar cómo incorporar esta información dentro del propio algoritmo para que cada hilo de búsqueda tome decisiones, durante la ejecución, de acuerdo a ella y permita un comportamiento más eficiente y eficaz.

Parte IV

Apéndices

Apéndice A

Relación de publicaciones que sustentan la tesis doctoral

En este apéndice se presenta el conjunto de trabajos que han sido publicados como fruto de las investigaciones desarrolladas a lo largo de esta tesis doctoral. Estas publicaciones avalan el interés, la validez y las aportaciones de esta tesis doctoral en la literatura, ya que estos trabajos han aparecido publicados en foros de prestigio y, por lo tanto, han sido sometidos a procesos de revisión por reconocidos investigadores especializados. A continuación se muestran las referencias de todas las publicaciones.

Revistas indexadas

- C. Salto, E. Alba and J.M. Molina. Analysis of Distributed Genetic Algorithms for Solving Cutting Problems. *International Transactions in Operational Research*, Volume 13/issue number 5, pág 403-423, septiembre de 2006.

- C. Salto, E. Alba, J.M. Molina and G. Leguizamón, Greedy Seeding Procedure for GAs Solving a Strip Packing Problem. *Revista Iberoamericana de Inteligencia Artificial*, Número Especial artículos seleccionados del Workshop de Agentes y Sistemas Inteligentes, Congreso Argentino de Ciencias de la Computación, edición 2007. Volume 40, pág. 73-85, diciembre 2008.

Capítulos de libros

- C. Salto, J.M. Molina and E. Alba, Greedy Seeding and Problem Specific Operators for GAs Solving Strip Packing Problems, in *Optimization Techniques for Solving Complex Problems*, E. Alba, C. Blum, P. Isasi, C. León, and J.A. Gómez (Editores), Wiley, pág. 385-405, 2008.

- C. Salto, G. Leguizamón, E. Alba, and J.M. Molina, Evolutionary and ACO Based Approaches for Two-dimensional Strip Packing Problem, in *Natural Intelligence for Scheduling, Planning, and Packing Problems*, Springer-Verlag en la serie *Studies in Computational Intelligence*, vol 250. Raymond Chiong and S. Dhakal (editores). En prensa. 2009.

Congresos internacionales publicados en la serie *Lecture Notes in Computer Science*

- C. Salto, E. Alba and J.M Molina, Analysis of Distributed Genetic Algorithms for Solving a Strip Packing Problem, en *6th International Conference on Large-Scale Scientific Computations (LSSC07)*, volumen 4818 de LNCS, pág. 609-617, 2007.

Congresos internacionales

- C. Salto, G. Leguizamón, E. Alba and J.M. Molina, Hybrid Ant Colony System to Solve a 2-Dimensional Strip Packing Problem, en *Hybrid Intelligent Systems (HIS08)*, pág 708-713. 2008.

- C. Salto, E. Alba and J.M Molina, Evolutionary Algorithms for Level Strip Packing Problem, en *Workshop on Nature Inspired Cooperative Strategies (NICSO 2006)*, pág. 137-148, 2006.

- C. Salto, E. Alba and J.M Molina, Sequential versus Distributed Evolutionary Approaches for the Two-dimensional Guillotine Cutting Problem, en *International Conference on Industrial Logistics (ICIL 2005)*, pág. 291-300, 2005.

Congresos nacionales

- C. Salto, E. Alba, and G. Leguizamón, External Memory in a Hybrid Ant Colony System for a 2D Strip Packing, en *Congreso Argentino de Ciencias de la Computación (CACIC 2009)*, 2009. En prensa.

- C. Salto, E. Alba, J.M. Molina and G. Leguizamón, Greedy Seeding Procedure for GAs Solving a Strip Packing Problem, en *Congreso Argentino de Ciencias de la Computación (CACIC07)*, pág 1512-1523. 2007.

- C. Salto, E. Alba and J.M Molina, A comparison of different recombination operators for the 2-dimensional strip packing problem, *XII Congreso Argentino de Ciencias de la Computación (CACIC 2006)*, pág. 1126-1138, 2006.

Trabajos en evaluación

- C. Salto, G. Leguizamón and E. Alba, Heterogeneous Distributed Genetic Algorithms for a 2D Strip Packing, enviado a la revista *Engineering in Optimization*, en agosto de 2009.

Bibliografía

[1] A. Acan. GAACO: A GA + ACO hybrid for faster and better search capability. *ANTS 2002, LNCS 2463*, pages 300–301, 2002.

[2] A. Acan. An external memory implementation in ant colony optimization. *Ant Colony Optimization and Swarm Intelligence (ANTS2004)*, pages 73–82, 2004.

[3] A. Acan. An external partial permutation memory for ant colony optimization. *EvoCOP 2005, LNCS 3448*, pages 1–11, 2005.

[4] P. Adamidis and V. Petridis. Co-operating populations with different evolution behaviour. *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pages 188–191, 1996.

[5] R. Ahuja and J. Orlin. Developing fitter genetic algorithms. *INFORMS Journal on Computing*, 9(3):251–253, 1997.

[6] R. Ahuja, J. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27(3):917–934, 2000.

[7] R. Aiex, S. Binato, and M. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.

[8] R. Aiex, M. Resende, P. Pardalos, and G. Toraldo. GRASP with path-relinking for the three-index assignment problem. Technical report, AT&T Labs Research Technical Report, Shannon Laboratory, USA, February 2001.

[9] A. Al-Yamani, S. Sait, and H. Barada. HPTS: Heterogeneous parallel tabu search for VLSI placement. In *Proc. of the 2002 Congress on Evolutionary Computation*, pages 351–355, 2002.

[10] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters, Elsevier*, 82(1):7–13, 2002.

[11] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005.

[12] E. Alba and C. Cotta. Evolución de estructuras de datos complejas. *Informática y Automática*, 30(3):42–60, 1997.

[13] E. Alba, G. Leguizamon, and G. Ordoñez. Two models of parallel ACO for the minimum tardy task problem. *Int. Journal High Performance Systems Architecture*, 1:50–59, 2007.

[14] E. Alba, F. Luna, and A.J. Nebro. Advances in parallel heterogeneous genetic algorithms for continuous optimization. *International Journal of Applied Mathematics and Computer Science*, 14(3):101–117, 2004.

[15] E. Alba, F. Luna, A.J. Nebro, and J.M. Troya. Parallel heterogeneous genetic algorithms for continuous optimization. *Parallel Computing*, 30(5-6):699–719, 2004.

[16] E. Alba, J. Luna, L.M. Moreno, C. Pablos, J. Petit, A. Rojas, F. Xhafa, F. Almeida, M.J. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, and C. León. *MALLBA: A Library of Skeletons for Combinatorial Optimisation*, volume 2400 of *LNCS*, pages 927–932. Springer, 2002.

[17] E. Alba, A.J. Nebro, and J.M. Troya. Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel and Distributed Computing*, 62:1362–1385, 2002.

[18] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transaction on Evolutionary Computation*, 6(5):443–462, 2002.

[19] E. Alba and J. Troya. *Statistics and Computing*, volume 12, chapter Improving flexibility and efficiency by adding parallelism to genetic algorithms, pages 91–114. Kluwer Academic Publishers, 2002.

[20] R. Alvarez-Valdez, A. Pararreño, and J. Tamarit. Reactive GRASP for the strip-packing problem. *Computers and Operations Research*, 35(4):1065–1083, 2008.

[21] P. András, A. András, and Z. Szabó. A genetic solution for the cutting stock problem. *Proceedings of the 1st Online Workshop on Soft Computing (WSC1)*, pages 87–92, 1996.

[22] N. Ansai and E. Hou. *Computational intelligence for optimization.* Dordrecht: Kluwer Academic Press, 1997.

[23] A. Ramesh Babu and N. Ramesh Babu. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. *International Journal of Productions Research*, 37(7):1625–1643, 1999.

[24] V. Bachelet, P. Preux, and E. Talbi. Parallel hybrid meta-heuristics: application to the quadratic assignment problem. In *Proceedings of the Parallel Optimization Colloquium*, 1996.

[25] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming.* Oxford University Press, New York, 1996.

[26] B. Baker, E. Coffman Jr., and R. Rivest. Orthogonal packing in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[27] R. Battiti and M. Protasi. *Handbook of Combinatorial Optimization*, volume 1, chapter Aproximate algorithms and heuristics for MAX-SAT, pages 77–148. Kluwer Academic Publishers, 1998.

[28] T. Bäck, D. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. Oxford University Press, New York, 1997.

[29] J. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[30] J.D. Beltrán, J.E. Calderón, R.J. Cabrera, J.A.M. Pérez, and J.M. Moreno-Vega. GRASP/VNS hybrid for the strip packing problem. *Proceedings of Hybrid Metaheuristics*, pages 79–90, 2004.

[31] J. Berkey and P. Wang. Two-dimensional finite bin packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[32] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(2):1161–1172, 2004.

[33] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[34] A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172(3):814–837, 2006.

[35] A. Bortfeldt and H. Gehring. New large benchmark instances for the two-dimensional strip packing problem with rectangular pieces. In *HICSS*. IEEE Computer Society, 2006.

[36] M. Boschetti and V. Maniezzo. The two-dimensional finite bin packing problem. Part II. *4OR*, 1(2):135–148, 2003.

[37] M. Boschetti and V. Maniezzo. An ant system heuristic for the two-dimensional finite bin packing problem: preliminary results. *Chapter 7 of book Multidisciplinary Methods for Analysis Optimization and Control of Complex Systems*, pages 233–247, 2005.

[38] A. Le Bouthillier and T. Crainic. A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research*, 32(7):1685–1708, 2004.

[39] A. Le Bouthillier, T. Crainic, and R. Keller. Co-operative parallel method for vehicle routing problem with time windows. *4th Metaheuristics International Conference MIC´2001*, pages 277–279, 2001.

[40] O. Bräysy. A reactive variable neighborhood search algorithm for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[41] B. Bullnheimer, R. Hartl, and C. Strauss. A new rank-based version of the ant system: a computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[42] E. Burke, P. Cowling, and R. Keuthen. Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem. *Lecture Notes in Computer Science*, 2037:203–212, 2001.

[43] E. Burke and G. Kendall. Applying ant algorithms and the no fit polygon to the nesting problem. *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence, AI99 LNCS*, pages 453–464, 1999.

[44] E. Burke and G. Kendall. Applying evolutionary algorithms and the no fit polygon to the nesting problem. *Proceedings of the International Conference on Artificial Intelligence (IC-AI 99)*, pages 51–57, 1999.

[45] E. Burke and G. Kendall. Applying simulated annealing and the no fit polygon to the nesting problem. *Proceedings of the World Manufacturing Congress*, pages 27–30, 1999.

[46] E.K. Burke, G. Kendall, and G. Whitwell. Metaheuristic enhancements of the best-fit heuristic for the orthogonal stock cutting problem. *INFORMS Journal on Computing*, 52:655–671, 2004.

[47] E.K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52:655–671, 2004.

[48] V. Campos, F. Glover, M. Laguna, and R. Martí. An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization*, pages 397–414, 2001.

[49] V. Cerney. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Optimization Theory an Applications*, 45:41–51, 1985.

[50] B. Chazelle. The botton-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, 32(8):697–707, 1983.

[51] J. Chicano. *Metaheurísticas e Ingeniería del Software.* PhD thesis, University of Málaga, 2007.

[52] Y. Chunyu and L. Xinbao;. Solution to 2D rectangular strip packing problems based on ACOs. *International Workshop on Intelligent Systems and Applications, ISA 2009*, pages 1–4, 2009.

[53] C. Coello Coello, E. Alba, G. Luque, and A. Hernandez Aguirre. Comparing different serial and parallel heuristics to design combinational logic circuits. *NASA/DoD Conference on Evolvable Hardware (EH'03)*, page 3, 2003.

[54] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:801–826, 1980.

[55] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[56] D.T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.

[57] O. Cordón, I. Fernández de Viana, F. Herrera, and L. Moreno. A new ACO model integrating evolutionary computation concepts: The best-worst ant system. In M. Dorigo, M. Middendorf, and T. Stützle, editors, *Abstract proceedings of ANTS2000 - From Ant Colonies to Artifical Ants: A series of International Workshops on Ant Algorithms*, pages 22–29. IRIDIA, Université Libre de Bruxelles, Belgium, 2000.

[58] O. Cordón, F. Herrera, and L. Moreno. Integración de conceptos de computación evolutiva en un nuevo modelo de colonias de hormigas. *VII Conferencia de la Asocación Española para la Inteligencia Artificial*, II:98–105, 1999.

[59] F. Corno, P. Primetto, M. Rebaudengo, M. Sonza Reorda, and S. Bisotto. Optimizing area loss in flat glass cutting. *GALESIA97,IEE/IEEE International Conference on Genetic ALgorithms in Engineering Systems: Innovations and Applications, Glasgow (UK)*, 1997.

[60] T. Crainic and M. Gendreau. Cooperative parallel tabu search for capacited network design. *Journal of Heuristics*, 8:601–627, 2002.

[61] T. Crainic, M. Toulose, and M. Gendreau. Parallel asynchronous tabu search for multicommodity location-allocation with balancing requirements. *Annals of Operations Research*, 63:277–299, 1996.

[62] T.G. Crainic and M. Tolouse. *Fleet Management and Logistics*, chapter Parallel Metaheuristics, pages 205–251. Kluwer Academic Publisher, 2003.

[63] T.G. Crainic and M. Tolouse. *Handbook of Metaheuristics*, chapter Parallel Strategies for Metaheuristics, pages 475–514. Kluwer Academic Publisher, 2003.

[64] V. Cung, S.L. Martins, C.C Ribeiro, and C. Roucairol. *Essays and Surveys in Metaheuristics*, chapter Strategies for the Parallel Implementation of Metaheuristics, pages 263–308. Kluwer, 2003.

[65] C. Darwin. *On the Origin of Species by Means of Natural Selection*. Londres, 1859.

[66] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[67] K. DeJong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[68] T. Dereli and G. Sena Daş. A hybrid simulated-annealing algorithm for two-dimensional strip packing problem. *Proceedings of 5th International Symposium on Intelligent Manufacturing Systems, LNCS 4431*, pages 959–966, 2007.

[69] H. Dickhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[70] H. Dickhoff and U. Finke. *Cutting and packing in production and distribution*. Springer Verlag, Berlin, 1992.

[71] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnica di Milano, Italy, 1992.

[72] M. Dorigo and G. Di Caro. *New Ideas in Optimization*, chapter The ant colony optimization meta-heuristic, pages 11–32. D. Corne and M. Dorigo and F. Glover, editors. McGraw Hill, London, 1999.

[73] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[74] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics*, 26(1):29–41, 1996.

[75] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In F. Glover and G. Kochenberger., editors, *Handbook of Metaheuristics*. 2002.

[76] M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, 2004.

[77] K.A. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, 68:389–399, 1993.

[78] K.A. Dowsland and W.B. Dowsland. Packing problems. *European Journal of Operational Research*, 56:2–14, 1992.

[79] Grzegorz Dudek. Genetic algorithm with integer representation of unit start-up and shut-down times for the unit commitment problem. *European Transactions on Electrical Power*, 17(5):500–511, 2006.

[80] J. Eggermont and T. Lenaerts. Non-stationary function optimization using evolutionary algorithms with a case-based memory. Technical Report 2001-11, Leiden University Advanced Computer Sceice (LIACS), 2001.

[81] A. Eiben, P.-E Raué, and Z. Ruttkay. Genetic algoritms with multi-parent recombination. In Y. Davidor, H.-P Schwefel, and R. Manner, editors, *Proceedings for the 3rd Conference on Parallel Problem Solving from Nature*, volume LNCS 866, pages 78–87, 1994.

[82] A. Eiben, I. Sprinkhuizen-Kuyper, and B. Thijssen. Competing crossovers in an adaptive GA framework. *Proc of the Fifth IEEE Conference on Evolutionary Computation*, pages 787–792, 1998.

[83] L. Faina. Application of simulated annealing to the cutting stock problem. *European Journal of Operational Research*, 114:542–556, 1999.

[84] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[85] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods & Software*, 7:1033–1058, 2002.

[86] K. Fleszar and K.S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29(7):821–839, 2002.

[87] L. Fogel. Toward inductive inference automata. *Proceedings of the International Federation for Information Processing Congress*, pages 395–399, 1962.

[88] L. Fogel, A. Owens, and M Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[89] C. Fonlupt, P. Preux, and D. Robilliard. Preventing premature convergence via cooperating genetic algorithms. In *Proc. of the 3rd Int. Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, and Rough Sets (MENDEL'1997)*, pages 50–55, 1997.

[90] L.M. Gambardella and M. Dorigo. Ant colony system colony hybridized with a new local search for the sequential ordering problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

[91] L.M. Gambardella, E. Taillard, and M. Dorigo. *New Ideas in Optimization*, chapter MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, pages 63–76. McGraw-Hill, 1999.

[92] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979.

[93] H. Gehring and H. Homberger. A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific Journal of Operational Research*, 18:35–47, 2001.

[94] P.C. Gilmore and R.E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operational Research*, 13:94–119, 1965.

[95] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13:533–549, 1986.

[96] F. Glover. Scatter Search and Path Relinking. pages 297–316, 1999.

[97] F. Glover and G.A.Kochenberg, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, London, 2003.

[98] F. Glover and M. Laguna. *Tabu search*. Modern Heuristic Techniques for Combinatorial Problems, Oxford, England. Blackwell Scientific Publishing. C. Reeves, 1993.

[99] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[100] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[101] D. Goldberg and R. Lingle. Alleles, loci, and the TSP. *Proceedings of the 1rst International Conference on Genetic Algorithms*, pages 154–159, 1985.

[102] D. Goldberg and J. Richardson. *Genetic Algorithms and their Applications*, chapter Genetic algorithms with sharing for multimodal function optimization, pages 41–49. Lawrence Erlbaum Associates, Hillsdale, 1987.

[103] A. Gomes and J. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811–829, 2006.

[104] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76:579–581, 1989.

[105] J.J. Grefenstette. *Genetic Algorithms and Simulated Annealing*, chapter Incorporating problem specific knowledge into genetic algorithms, pages 42–60. Morgan Kaufmann Publishers, 1987.

[106] G. Gudise and G. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. *Proceedings of the IEEE Swarm Intelligence Symposium 2003*, pages 110–117, 2003.

[107] M. Guntsch and M. Middendorf. A population based approach for ACO. *Applications of Evolutionary Computing - EvoWorkshops2002 LNCS 2279*, pages 72–81, 2002.

[108] R.W. Haessler and P.E. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

[109] Jean-Philippe Hamiez, Julien Robet, and Jin-Kao Hao. A tabu search algorithm with direct representation for strip packing. In Carlos Cotta and Peter I. Cowling, editors, *EvoCOP*, volume 5482 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2009.

[110] P. Hansen and N. Mladenoviç. Variable neighborhood search for the *p*-median. *Location Science*, 5:207–226, 1997.

[111] P. Hansen and N. Mladenoviç. *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, chapter An introduction to variable neighborhood search, pages 433–458. S. Voss, S. Martello, I.H. Osman and C. Roucairol, editors. Kluwer Academic Publishers, 1999.

[112] P. Hansen, N. Mladenoviç, and D. Pérez Brito. Variable neighborhood decompostion search. *Journal of Heuristics*, 7(4):335–350, 2001.

[113] E.A. Herbert and K.A. Downsland. A family of genetic algorithms for the pallet loading problem. *Annals of Operations Research*, 63:415–436, 1996.

[114] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. Technical Report DECSAI-97-01-03, 1997.

[115] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transaction on Evolutionary Computation*, 4(1):43–63, 2000.

[116] Francisco Herrera, Manuel Lozano, and Jose L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[117] M. Hifi and R. Hallah. A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes. *International Transactions in Operational Research*, 10(3):195–216, 2003.

[118] R. Hintterding, Z. Michalewicz, and T.C. Peachey. Self-adaptive genetic algorithm for numeric functions. *Parallel Problem Solving from Nature (PPSN IV)*, pages 420–429, 1996.

[119] A. Hinxman. The trim loss and assortment problems. *European Journal of Operational Research*, 5:8–18, 1980.

[120] J. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[121] H.H. Hoos and T. Stützle. Local search algorithms for SAT: an empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[122] E. Hopper. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. PhD thesis, University of Wales, Cardiff, U.K., 2000.

[123] E. Hopper and B. Turton. Application of genetic algorithms to packing problems - a review. *Proceedings of the Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing, Springer Verlag, London*, pages 279–288, 1997.

[124] E. Hopper and B. Turton. A genetic algorithms for a 2D industrial packing problem. *Computers in Engineering*, 37:375–378, 1999.

[125] E. Hopper and B. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128(1):34–57, 2001.

[126] E. Hopper and B. Turton. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review*, 16:257–300, 2001.

[127] J. Hu and E. Goodman. The hierarchical fair competition HFC model for parallel evolutionary algorithms. In *Proc. of the 2002 Congress on Evolutionary Computation CEC2002*, pages 49–54. IEEE Press, 2002.

[128] S. Hwang, C. Kao, and J. Horng. On solving rectangle bin packing problems using genetic algorithms. *IEEE International Conference on Systems, Man, and Cybernetics - Humans, Information and Technology*, 2:1583–1590, 1994.

[129] M. Iori, S. Martello, and M. Monaci. *Optimization and Industry: New Frontiers*, chapter Metaheuristic algorithms for the strip packing problem, pages 159–179. Kluwer Academi Publishers, 2003.

[130] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88:165–181, 1996.

[131] A. Karp and H. Flatt. Measuring parallel processor performance. *Communications of he ACM*, 33(5):539–543, 1990.

[132] J. Kennedy and R. C Eberhart. Particle swarm optimization. *Proc. IEEE International Conf. on Neural Networks*, IV:1942–1948, 1995.

[133] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 4598:671–680, 1983.

[134] B. Kröger. Guillotineable bin-packing: a genetic approach. *European Journal of Operational Research*, 84:645–661, 1995.

[135] B. Kröger, Schwenderling P., and O. Vornberger. Genetic packing of rectangles on transputers. *Transputing, part. 2, IOS Press, Amsterdam*, pages 593–608, 1991.

[136] B. Kröger, Schwenderling P., and O. Vornberger. Parallel genetic packing of rectangles. *Parallel Solving from Nature 1st Workshop, Springer Verlag*, pages 160–164, 1991.

[137] B. Kröger, P. Schwenderling, and O. Vornberger. *Parallel Genetic Algorithms: Theory and Applications,*, chapter Parallel genetic packing on transputers, pages 151–185. IOS Pres, Amsterdam, 1993.

[138] F. Krüger, D. Merkle, and M. Middendorf. Studies on a parallel ant system for the BSP model. 1998.

[139] P.J.M. Val Laarhoven, E.H.L Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Operational Research*, 40(1):113–125, 1992.

[140] M. Laguna, H.R. Lourenço, and R. Martí. *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research.*, chapter Assigning proctors to exams with scatter search, pages 215–227. Kluwer Academic Publishers, 2000.

[141] P. Larrañaga and J. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Genetic Algorithms and Evolutionary Computation, Series. Springer, 2001.

[142] S. Lee and K. Lee. Synchronous and asynchronous parallel simulated anneaing with multiple markov chains. *IEEE Trans. on Parallel and Distributed Systems*, 7(10):993–1008, 1996.

[143] N. Lesh, H. Marks, A. Mc.Mahon, and M. Mitzenmacher. New heuristic and interactive approaches to 2D rectangular strip packing. *ACM Journal of Experimental Algorithmics*, 10:1–18, 2005.

[144] N. Lesh and M. Mitzenmacher. Bubble search: a simple heuristic for improving priority-based greedy algorithms. *Information Processing Letters*, 97:161–169, 2006.

[145] T.W. Leung, C.H. Yung, and C.K. Chan. Application of genetic algorithm and simulated annealing to the 2-dimensional non-guillotine cutting stock problem. *IFORS '99, Beijing China*, 1999.

[146] T.W. Leung, C.H. Yung, and M.D. Troutt. Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. *Computers and Industrial Engineering*, 40:201–214, 2001.

[147] J. Levine and F. Ducatelle. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, (55):705–716, 2004.

[148] S. Lin, W. Punch, and E. Goodman. Coarse Grain Parallel Genetic Algorithms: categorization and new approach. *Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing*, 1994.

[149] D. Liu and H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*, 112:413–420, 1999.

[150] Rui Liu, Xianlong Hong, Sheqin Dong, Yici Cai, Jun Gu, and Chung-Kuan Cheng. Module placement with boundary constraints using o-tree representation. *IEEE International Symposium on Circuits and Systems (ISCAS2002)*, 2:871–874, 2002.

[151] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141:241–252, 2002.

[152] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[153] H. Ramalhino Lourenco, O. Martin, and T. Stützle. A beginner´s introduction to iterated local search. *Proceedings of MIC 2001*, pages 1–6, July 2001.

[154] H. Ramalhino Lourenço, O.C. Martin, and T. Stützle. *F. Glover and G. Kochenberger (eds) Handbook of Metaheuristics*, chapter Iterated local search, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[155] F. Luna, E. Alba, and A.J. Nebro. *Parallel Metaheuristics. A new class of algorithms*, chapter Parallel heterogeneous metaheuristics, pages 395–422. Wiley, 2005.

[156] M. Lundy and H. Teng. Convergence of an annealing algorithm. *Mathematical Programming*, 34:11–124, 1986.

[157] G. Luque. *Resolución de Problemas Combinatorios con Aplicación Real en Sistemas Distribuidos.* PhD thesis, Universidad de Málaga, 2006.

[158] S. Mahfoud. *Niching methods for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.

[159] S. Martello, S. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[160] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44:388–399, 1998.

[161] G. Mendel. *Versuche über Pflanzen-Hybriden*, volume 4. Verhandlungen des Naturforschedes Vereines in Brünn, 1865.

[162] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[163] R. Michels and M. Middendorf. *New Ideas in Optimization*, chapter An ant system for the shortest common supersequence problem, pages 51–61. McGraw-Hill, 1999.

[164] M. Middendorf, F. Reischle, and H. Schmeck. Multicolony ant system algorithms. *Journal of Heuristics (Special issue on Parallel Metaheuristics*, 8(3):305–320, 2002.

[165] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998.

[166] N. Mladenoviç and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

[167] J. Montgomery and M. Randall. The accumulated experience ant colony for the travelling salesman problem. *International Journal of Computational Intelligence and Applications*, 3(2):189–198, 2003.

[168] R. Morabito and S. Morales. A simple and effective recursive procedure for the manufacturer's pallet loading problem. *Journal of the Operational Research Society*, 49(8):819–828, 1998.

[169] H. Mühlenbein and G. Paafi. From recombination of genes to the estimation of distributions I. Binary parameters. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 178–187, London, UK, 1996. Springer-Verlag.

[170] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–633, 1991.

[171] 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 BV, 2003.

[172] N.Ñtene and J.H. van Vuuren. A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem. *Discrete Optimization*, 6(2):174–188, 2009.

[173] V.Ñwana, D. Darby-Dowman, and G. Mitra. A co-operative parallel heuristic for mixed zero-one linear programming: Combining simulated annealing with branch and bound. *European Journal of Operational Research*, 164(1):2–23, 2004.

[174] S. Oh, C. Lee, and J. Lee. A new distributed evolutionary algorithm for optimization in nonstationary environments. In *Proc. of the 2002 Congress on Evolutionary Computation*, pages 378–383. IEEE Press, 2002.

[175] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the traveling salesman problem. *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 224–230, 1987.

[176] M. Omran, A. Salman, and A. Engelbrecht. Image classification using particle swarm optimization. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning 2002*, pages 370–374, 2002.

[177] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Dover Publications, Inc., New York, 1982.

[178] J.C. Potts, T.D. Giddens, and S.B. Yadav. The development and evaluation of an improved genetic algorithm based on migration and artificial selection. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(1):73–86, 1994.

[179] J. Puchinger and G. Raidl. An evolutionary algorithm for column generation in integer programming: An effective approach for 2D bin packing. In X. Yao et al, editor, *PPSN*, volume 3242 of *LNCS*, pages 642–651. Springer, 2004.

[180] J. Puchinger and G.R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operational Research*, 183(3):1304–1327, 2007.

[181] J. Puchinger, G.R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In *EvoCOP 2004*, volume LNCS 3004, pages 162–173, 2004.

[182] Jakob Puchinger. *Combining Metaheuristics and Integer Programming for Solving Cutting and Packing Problems*. PhD thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, January 2006.

[183] A.T. Rahmani and N. Ono. An evolutionary approach to two-dimensional guillotine cutting problem. *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pages 148–151, 1995.

[184] C. Ramsey and J. Grefenstette. Case-based initialization of GAs. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, 1993.

[185] M. Randall and A. Lewis. A parallel implemetnation of ant colony optimization. *Journal lf Parallel and Distributed Computing*, 62:1421–1432, 2002.

[186] V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith. *Modern heuristic search methods*. New York: Wiley, 1996.

[187] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[188] C. Reeves and J. Rowe. *Genetic Algorithms: Principles and Perspectives. A Guide to GA theory*. Kluwer Academic Publishers, 2002.

[189] C.R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, UK,, 1993.

[190] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[191] M.G.C. Resende and C.C. Ribeiro. *F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics*, chapter Greedy randomized adaptive search procedures, pages 219–249. Kluwer Academic Publishers, 2003.

[192] M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 41:104–114, 2003.

[193] M.C. Riff, X. Bonnaire, and B.Ñeveu. A revision of recent approaches for two-dimensional strip-packing problems. *Enginnering Applications of Artificial Intelligence*, 22:823–827, 2009.

[194] A. Rogers and A. Prügel-Bennett. *Foundations of Genetic Algorithms 5*, chapter Modelling the dynamics of a steady-state genetic algorithm, pages 57–68. Morgan Kaufmann, San Francisco, CA, 1999.

[195] A. Rogers and A. Prügel-Bennett. Genetic drift in genetic algorithm selection schemes. *Transactions on Evolutionary Computation*, 3(4):298–303, 1999.

[196] C. Salto, J.M. Molina, and E. Alba. Analysis of distributed genetic algorithms for solving cutting problems. *International Transactions in Operational Research*, 13(5):403–423, 2006.

[197] C. Salto, J.M. Molina, and E. Alba. Evolutionary algorithms for the level strip packing problem. *Proceedings of NICSO*, pages 137–148, 2006.

[198] V. Schnecke and O. Vornberger. An adaptive parallel genetic algorithm for VLSI-layout optimization. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 859–868, 1996.

[199] H.P. Schwefel. *Evolution and optimum seeking.* New York: Wiley, 1994.

[200] D. Smith. Bin-packing with adaptive search. *Proceedings of an International Conference on Genetic Algorithms and their Applications*, pages 202–206, 1985.

[201] A. Socke and Z. Bingul. Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problem. *Engineering Applications of Artificial Intelligence*, 19:557–567, 2006.

[202] T. Starkweather, S. McDaniel, K. Mathias, C. Whitley, and D. Whitley. A comparison of genetic sequencing operators. *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 69–76, 1991.

[203] T. Stützle and H.H. Hoos. Analyzing the run-time behaviour of iterated local search for the TSP. *Proceedings of the 3rd Metaheuristics International Conference (MIC 1999)*, pages 449–453, 1999.

[204] T. Stützle and H.H. Hoos. MAX-MIN ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[205] P.E. Sweeney and E.R. Paternoster. Cutting and packing problems: a categorized, application-oriented research bibliography. *Journal of the Operational Research Society*, 43:691–706, 1992.

[206] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991.

[207] E. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[208] E. Talbi, J. Geib, and D. Kebbal. A fault-tolerant parallel heuristic for assignment problems. *Future Generations Computer Systems*, 14:425–438, 1998.

[209] E. Talbi and Hervé Meunier. Hierarchical parallel approach for GSM mobile network design. *Journal of Parallel and Distributed Computing*, 66(2):274–290, 2006.

[210] R. Tanese. Parallel genetic algorithm for a hypercube. In *Proc. of the Second International Conference on Genetic Algorithms*, pages 434–439. Morgan Kaufmann, 1987.

[211] R. Tanese. Distributed genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439, 1989.

[212] D. Thiruvady, B. Meyer, and A. Ernst. Strip packing with hybrid ACO: Placement order is learnable. *IEEE Congress on Evolutionary Computation (CEC 2008)*, pages 1207–1213, 2008.

[213] S. Tsutsui and Y. Fujimoto. Forking genetic algorithm with blocking and shrinking modes (fGA). In S. Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 206–213. IEEE Press, 1993.

[214] R. Valdés, A. Parajón, and J. Tamarit. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers and Operations Research*, 29(7):925–947, 2002.

[215] F. Vanderbeck. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. *Management Science*, 47(6):864–879, 2001.

[216] F. Vavak and T.C. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. *International Conference on Evolutionary Computation*, pages 192–195, 1996.

[217] M. Vose. *The simple genetic algorihtm: foundations and theory.* MIT Press, 1999.

[218] G. Wang, E. Goodman, and W. Punch. Simltaneous multi-level evolution. Technical report, GARAGe Technical Report, Department of Computer Science and Case Center for Computer-Aided Engineering & Manufacturing, Michigan State University, 1996.

[219] P.Y. Wang and C.L. Valenzuela. Data set generation for rectangular placement problems. *EJOR*, 134:378–391, 2001.

[220] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: the genetic edge recombination operator. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 133–140, 1989.

[221] R. Whittaker. A fast algorithm for the greedy interchange for large-scale clustering and median location problems. *INFORMS*, 21:95–108, 1983.

[222] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130, 2007.

[223] Y. Wu, W. Huang, S. Lau, C.K. Wong, and G.H. Young. An effective quasi-human based heuristic for solving the rectangle packing problem. *European Journal of Operational Research*, 141:341–358, 2002.

[224] W. Zhang Z. X. Xie and Yang. Solving numerical optimization problems by simulating particulates in potential field with cooperative agents. *International Conference on Artificial Intelligence*, 2002.

[225] C.H. Dagli y P. Poshyanonda. New approaches to nesting rectangular patterns. *Journal of Intelligent Manufacturing*, 8:177–190, 1997.

[226] L.H.W. Yeung and W.K.S. Tang. A hybrid genetic approach for garment cutting in the clothing industry. *IEEE Transactions on Industrial Electronics*, 50(3):449–455, June 2003.

[227] W. Yi, Q. Liu, and Y. He. Dynamic distributed genetic algorithms. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1132–1136. IEEE Press, 2000.

[228] D. Zhang, Y. Kang, and A. Deng. A new heuristic recursive algorithm for the strip rectangular packing problem. *Computers and Operations Research*, 33(8):2209–2217, 2006.

[229] D. Zhang, Y. Liu, S. Chen, and X. Xie. A meta-heuristic algorithm for the strip rectangular packing problem. In Lipo Wang, Ke Chen, and Yew-Soon Ong, editors, *ICNC (3)*, volume 3612 of *Lecture Notes in Computer Science*, pages 1235–1241. Springer, 2005.

[230] D-F Zhang, S-D Chen, and Y-J Liu. Improved heuristic recursive strategy based on genetic algorithm for the strip rectangular packing problem. *Acata Automatica Sinica*, 33(9):911–916, 2007.

[231] M. Zlochin and M. Dorigo. Model-based search for combinatorial optimization: a comparative study. *Proceedings of PPSN-VII, Seventh International Conference on Parallel Problem Solving from Nature*, LNCS 2439:651–661, 2002.

Índice alfabético