UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

EVALUACIÓN DE UN ALGORITMO GENÉTICO CELULAR PARA EL PROBLEMA DE CORTE DE PIEZAS

DIEGO ABELARDO SOTO JARA

Trabajo de titulación presentado en conformidad a los requisitos para obtener el Grado de...

Comisión integrada por los profesores:

Dr. Víctor Parada Daza

Dr. Nombre

Dr. Nombre

Dr. Nombre

Santiago de Chile

2013

Contenido

Capítulo 1 INTRODUCCIÓN

1.1 Antecedentes y motivación

Los problemas de optimización combinatoria, se componen de un espacio de soluciones, un conjunto de restricciones, una función de evaluación y un extremo u óptimo (mínimo o máximo), y consisten en encontrar dicho óptimo según la función de evaluación. Para encontrar el óptimo, basta enumerar todas las soluciones posibles para luego elegir la mejor. Sin embargo, resulta evidente la impracticabilidad de esta aproximación cuando los espacios de soluciones crecen.

Un enfoque para abordar estos problemas, es el que proporciona la optimización clásica, mediante algoritmos exactos, donde destacan principalmente los métodos basados en ramificación y poda, y planos cortantes. Sin embargo, los problemas de optimización combinatoria son usualmente problemas pertenecientes a la clase *NP-Hard* (Garey y Johnson, 1979), luego para una instancia del problema de un tamaño suficientemente grande, el tiempo requerido para encontrar una solución óptima mediante algoritmos exactos se vuelve inadmisible (Laurence, 1999).

Debido a lo anterior, se han elaborado algoritmos aproximados, basados en heurísticas, las cuales son un conjunto de reglas que recogen conocimiento específico del problema particular que se está abordando. Las heurísticas no garantizan optimalidad pero funcionan bien en la práctica. Por otra parte, las denominadas metaheurísticas, son una clase de técnicas generales de optimización correspondientes a plantillas de las cuales pueden ser derivados optimizadores específicos (Laurence, 1999).

Tanto las heurísticas como metaheurísticas son métodos aproximados, sin embargo idealmente encuentran soluciones cercanas al óptimo en un tiempo razonable para instancias grandes del problema. Esta flexibilidad a menudo se da en la práctica, en donde para problemas de gran escala basta encontrar una solución lo suficientemente "buena" para ser aceptada. Por su parte, el éxito de las metaheurísticas puede ser explicado debido a su generalidad, al poder amoldarse al problema que se desea modelar, y por su equilibrio entre calidad de solución y esfuerzo computacional.

Las ideas para generar los algoritmos metaheurísticos se han basado por ejemplo en la física, dando origen al método Simulated Annealing (Kirkpatrick, Gelatt Jr, & Vecchi, 1983); en biología, originando métodos basados en colonias de hormigas (Dorigo & Stützle, 2004); la evolución natural, dando origen a los algoritmos evolutivos (Affenzeller, Winkler, Wagner, & Beham, 2009); entre otros.

En particular dentro de los algoritmos evolutivos, los algoritmos genéticos AG, han sido exitosos en la resolución de una gran variedad de problemas de diversa complejidad. Por tal razón, han sido ampliamente estudiados tanto desde un enfoque teórico como empírico. Con el tiempo, diversas extensiones del AG simple, principalmente basadas en generalizaciones de alguno de sus aspectos, han dado como resultado algoritmos cada vez más competitivos en la resolución de distintos problemas.

En particular, generalizaciones respecto a la estructuración de la población permiten clasificar los AG como no estructurados o *panmícticos*, que corresponden a AG con una sola gran población, y AG estructurados, o de población descentralizada, que corresponden a AG con varias subpoblaciones de menor tamaño. Dentro de los AG estructurados, se distinguen los AG distribuidos, los cuales estructuran la población en islas independientes débilmente conectadas, y los AG celulares, los cuales estructuran la población basados en aislamiento por distancia.

Una característica de los AG no estructurados es su alta explotación; cuando la evolución encuentra una buena solución, esta es rápidamente intensificada, lo que lleva al algoritmo a atascarse fácilmente en óptimos locales. Los AG estructurados poseen un mejor balance entre explotación y exploración, es decir poseen un mejor comportamiento exploratorio, aunque penalizando la capacidad de explotación. Este comportamiento es más apropiado en problemas complejos de mayor escala, donde esta búsqueda más suave guía a una buena solución.

Por su parte, dentro de los AG estructurados los AG celulares poseen incluso un mejor balance entre explotación y exploración que los AG distribuidos, ya que la superposición de pequeñas vecindades produce baja difusión de soluciones. Además sus parámetros específicos, como grado de superposición entre vecindades, forma y tamaño de vecindad y población, tipo de actualización, entre otros, proveen de mayor control sobre los efectos en la evolución. Por lo tanto, se piensa que la estructuración de la población utilizando conceptos celulares permitirá crear mejores algoritmos para la resolución de problemas de carácter geométrico, en particular el problema de corte de piezas.

Los problemas de corte y empaque pertenecen a la clase NP-Hard. Estos problemas, han sido estudiado durante varios años, por lo que mucho conocimiento se ha generado en torno a ellos y muchas formas de abordarlos han sido propuestas. Sin embargo, debido a su fuerte dureza, métodos más eficientes para su resolución siguen siendo desarrollados en la actualidad, y mientras persista la interrogante *P=NP*, este seguirá siendo un problema abierto para el campo de la optimización.

Sin embargo soluciones competitivas son obtenidas al

Existe evidencia de que un enfoque celular supera enfoques no estructurados en varios problemas académicos, como, así también como problemas de una dureza comparable, como el *TSP* o el *VRP* . Por lo que la aplicación de este enfoque para abordar este problema resulta promisorio, considerando que la mejora consiste en la aplicación de conceptos independientes del problema

1.2 Descripción del problema

Los problemas de corte y empaque pertenecen a una antigua y bien conocida familia, llamada *C&P* (cutting y packing). Esta es una familia de problemas naturales de optimización combinatoria, presentes en numerosas aplicaciones en el mundo real de la informática, ingeniería industrial, logística, fabricación, proceso de producción, etc. (Hifi y M'Halla, 2003). En términos abstractos, la estructura general de los problemas de *C&P* puede resumirse de la siguiente manera. Dados:

- Un conjunto de figuras pequeñas
- Un conjunto de regiones contenedoras

El objetivo es encontrar la mejor asignación posible (de acuerdo a algún criterio) de figuras pequeñas en las regiones contenedoras, respetando que:

- Las figuras se encuentren totalmente contenidas en las regiones
- Las figuras no se superpongan

Lo anterior implica resolver varios sub-problemas simultáneamente: un problema de selección de las regiones contenedoras, un problema de selección de las figuras pequeñas, un problema de agrupamiento de las figuras pequeñas seleccionadas,

un problema de distribución en la asignación las figuras pequeñas en las distintas regiones contenedoras y un problema de disposición de las figuras pequeñas en cada una de las regiones contenedoras (Wascher et al., 2007).

La complejidad de los problemas de C&P está fuertemente relacionada con la forma geométrica de los elementos que intervienen en la asignación. Así, la componente geométrica es el principal criterio para la clasificación de éstos (Beraudo et al., 2004).

A continuación se muestran algunos ejemplos de *C&P*. En la FIGURA 1.1 se muestran dos casos, en el primero hay una asignación de figuras regulares (rectángulos) y en el segundo una asignación de figuras irregulares (asimetrías y concavidades), ambas a una región contenedora rectangular. En la FIGURA 1.2 se muestra una asignación de figuras rectangulares a una región contenedora de largo infinito, (Ej.: corte de un rollo de género en industria textil), asignación a varias regiones contenedoras de distintas formas (Ej.: industria maderera, industria de vidrio) y asignación a regiones irregulares (Ej.: industria del cuero).

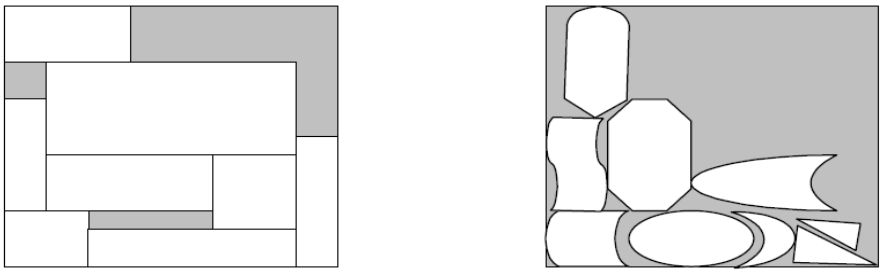


FIGURA 1.1 *Asignación de figuras regulares e irregulares a regiones rectangulares*

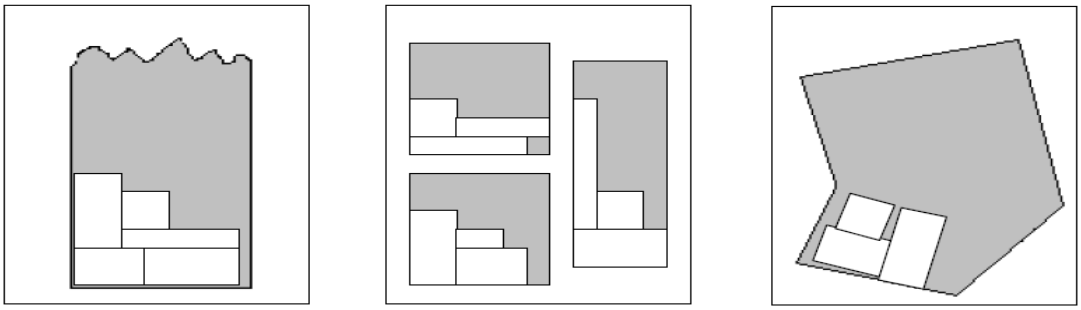


FIGURA 1.2 *Distintas formas de regiones contenedoras*

Existe una gran variedad de problemas de *C&P*, por lo que se han propuesto algunas tipologías, de manera de organizar y estructurar los problemas de una forma lógica, sin considerar sus aplicaciones ni disciplinas. (Dyckhoff, 1990) propone una tipología, la cual se resume en la FIGURA 1.3, donde los problemas se dividen

inicialmente en *2D* y *3D*, luego en regulares e irregulares, dentro de los regulares se encuentran las figuras rectangulares y otros tipos, y en los irregulares aparecen los polígonos y otros sin un patrón específico, finalmente los rectangulares se dividen en *no-guillotinables* y *guillotinables*.

En particular, los problemas de *C&P* en *2D* son de gran relevancia en la producción y logística. Problemas de empaque en *2D* aparecen, por ejemplo, cuando varios artículos deben ser empacados en pallets en capas horizontales, o el posicionamiento eficiente de componentes en microchips. Problemas de corte en *2D* se encuentran en la customización de material en las industrias de vidrio, metal, madera y papel.

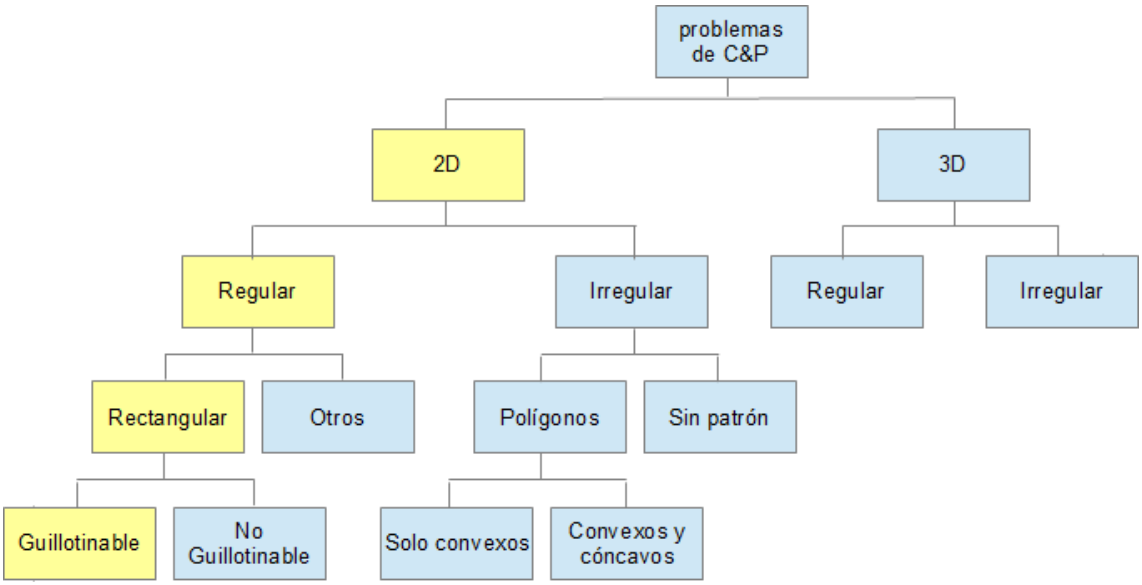


FIGURA 1.3 *Clasificación de problemas de corte y empaque (Dyckhoff 1990)*

Dentro de los problemas de corte en *2D*, una variante clásica es el *TDC* (two-dimensional cutting) (Hifi y M'Halla, 2003), el cual considera una demanda de ítems rectangulares con tamaño y valor propios, que deben ser cortados de una placa rectangular, y tiene como objetivo determinar un patrón de corte (combinación de items a cortar), que maximice el valor total cortado. Si el valor de un ítem está dado por su área, el objetivo es maximizar el área total utilizada. Un patrón de corte, también llamado plan de corte, es considerado como factible si cada ítem se corta ortogonal (es decir, paralelo a los bordes de la placa), está completamente contenido en la placa, y si no hay superposición entre los ítems.

(Wascher et al., 2007) propone una tipología actualizada de problemas de *C&P*, la cual se ilustra en la FIGURA 1.4. Según esta tipología, el *TDC* cae dentro del tipo de

problema de *maximización de output* (Gonçalves y Resende, 2011), en donde el conjunto de regiones contenedoras no es suficiente para acomodar todos los items, por lo que todas ellas deben ser usadas. En contraste, los problemas de *minimización de input* son aquellos donde el conjunto de regiones contenedoras es suficiente para acomodar todos los items, luego todos los ítems deben ser producidos, utilizando la menor cantidad de regiones posibles. En este último tipo de problemas cae el denominado *CSP* (cutting stock problem), el cual describe el procedimiento de cortar el número requerido de items de un conjunto de placas rectangulares. El *TDC* puede ser usado como un problema auxiliar del *CSP*, ya que para resolver este último, el primero debe resolverse varias veces como un subproblema (Yoon et al., 2013).

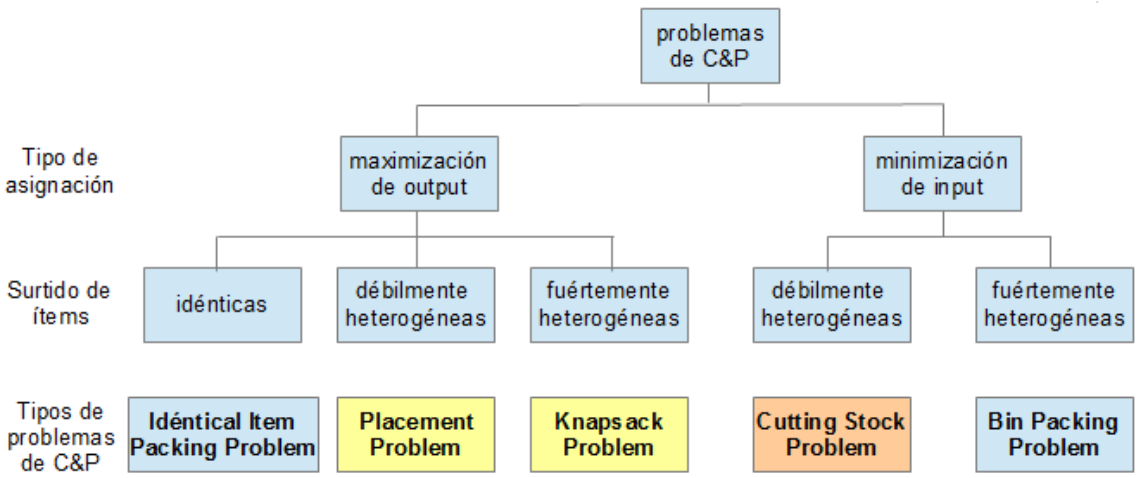


FIGURA 1.4 *Clasificación de problemas de corte y empaque (Wascher 2007)*

Es importante señalar que, desatendiendo la operación de corte, la definición del *TDC* coincide con la del *2D-KP* (two-dimensional knapsack), en donde los ítems, en vez de ser cortados, deben ser empacados en un contenedor rectangular. De hecho, de acuerdo a (Beasley, 2000) los problemas de corte en *2D* también se conocen comúnmente como problemas de empaque, ya que se pueden ver como:

a) el problema de cortar piezas más pequeñas de una placa rectangular
b) el problema de empacar piezas más pequeñas en un contenedor rectangular

Debido a lo anterior, todas las consideraciones que se enuncien se pueden aplicar tanto al *TDC* como a su contraparte de empaque, *2D-KP*. El problema es fuertemente

NP-duro, ya que en el caso especial en que todos los ítems tienen la misma altura, el problema de probar si todos ellos caben en el contenedor es equivalente al bien conocido *1D-BP* (one-dimensional bin packing) (Caprara y Monaci, 2003), el cual es fuertemente *NP-duro* (Garey y Johnson, 1979).

El tipo de ítem se define por sus dimensiones y por su valor. La demanda de ítems está dada por un conjunto de tipos de ítems. Con respecto al número de copias por tipo, de acuerdo a (Beasley, 2000) se distinguen las siguientes variantes:

- Irrestricto (Unconstrained) (*UTDC*): El número de copias por tipo no es fijo. Un patrón de corte puede por lo tanto, tener un número ilimitado de copias.
- Restricto (Constrained) (*CTDC*): Se fija un límite superior p_i para el ítem i. Un patrón de corte puede por lo tanto, contener como máximo p_i ítems de tipo i.
- Doblemente restricto (Doubly Constrained) (*DCTDC*): Se fija un límite superior p_i y un límite inferior q_j. Un patrón de corte puede contener como máximo p_i y como mínimo q_j items de tipo i.

Para complementar esta clasificación es interesante mencionar la clasificación de Fayard (Álvarez-Valdés et al., 2002), la cual además considera el valor de los items. Según esta clasificación se distinguen cuatro versiones para el *TDC*:

- Irrestricto no ponderado (Unconstrained unweighted) (*UU_TDC*): El valor de cada ítem es equivalente a su área. Luego el objetivo es maximizar el área ocupada, equivalente a minimizar la pérdida.
- Irrestricto ponderado (Unconstrained weighted) (*UW_TDC*): El valor de cada ítem es independiente de sus áreas. Luego el objetivo es maximizar el valor total de los ítems a cortar.
- Restricto no ponderado (Constrained unweighted) (*CU_TDC*): Cada pieza tiene un límite superior de demanda b_i, que restringe la cantidad de piezas de tipo i a cortar.
- Restricto ponderado (Constrained weighted) (*CW_TDC*): Es el caso más general.

Desde un punto de vista práctico, el *UTDC* es un caso especial del C*TDC*, donde el número de veces que cada pieza puede aparecer en el patrón, está naturalmente

restringido por el número de veces que cabe en la placa. Sin embargo el *UTDC* es generalmente más fácil de resolver que el C*TDC* (Cui y Huang, 2012). De hecho, a menudo es usado como un problema auxiliar del C*TDC* (Hifi y Zissimopoulos, 1997). En general, las variantes restrictas del *TDC* son más interesantes para las aplicaciones, y se ha dedicado mayor investigación a éstas (Álvarez-Valdés et al., 2005). Una explicación para esto es que, en la práctica, a menudo no se tienen cantidades ilimitadas de cada tipo de ítem, sino que estas están especificadas, por ejemplo en una orden de fabricación, o limitadas, por ejemplo por el inventario actualmente disponible.

Por otra parte, si el valor de los ítems equivale a su área, la notación de Beasley es suficiente para describir el problema. Estos casos, correspondientes a las versiones no ponderadas, aparecen típicamente en el corte de placas de acero o vidrio en piezas de ciertos tamaños requeridos, debiendo reducir las pérdidas al mínimo. En contraste, las versiones ponderadas aparecen, por ejemplo, en una línea de producción, donde los valores podrían definir prioridades para ciertas piezas o incluso imponer que ciertas piezas, ya existentes en el patrón actual, deberían aparecer también en el siguiente patrón (Cung et al., 2000).

<table>
<tr><th colspan="2">surtido de ítems / objetos grandes</th><th>idénticos</th><th>débilmente heterogéneas</th><th>fuértemente heterogéneas</th></tr>
<tr><td rowspan="3">todas las dimensiones fijas</td><td>un objeto grande</td><td>Identical Item Packing Problem
IIPP</td><td>Single Large Object Placement Problem
SLOPP</td><td>Single Knapsack Problem
SKP</td></tr>
<tr><td>idénticos</td><td rowspan="2"></td><td>Multiple Identical Large Object Placement Problem
MILOPP</td><td>Multiple Identical Knapsack Problem
MIKP</td></tr>
<tr><td>heterogéneos</td><td>Multiple Heterogeneous Large Object Placement Problem
MHLOPP</td><td>Multiple Heterogeneous Knapsack Problem
MHKP</td></tr>
</table>

FIGURA 1.5 *Landscape de tipos de problemas intermedios: maximización de output (Wascher 2007)*

Con respecto al surtido de la demanda de ítems, es usual distinguir las variantes: homogénea (solo un tipo de ítem), débilmente heterogénea (pocos tipos de ítems y muchas copias por tipo), y fuertemente heterogénea (muchos tipos de ítems y pocas

copias por tipo). De acuerdo a la tipología de Wascher, existe un *SLOPP* (Single Large Object Placement Problem) con una demanda débilmente heterogénea, y un *SKP* (Single Knapsack Problem) con una demanda fuertemente heterogénea. La FIGURA 1.5 muestra esta clasificación.

Del total de publicaciones revisadas, pocas identifican el *TDC* dentro de la tipología de Wascher. En particular, (Chen, 2007) señala que el *TDC* es conocido formalmente como *SLOPP*, no obstante, (Bortfeldt y Winter, 2008) señalan que para las variantes *CTDC* y *DCTDC* pueden haber ambos *SLOPP* y SKP, en tanto que para la variante *UTDC* solo debería ser asumido *SLOPP*. Por lo tanto una revisión de publicaciones en torno al *TDC* debería considerar tanto *SLOPP* como *SKP*. Para una revisión de publicaciones y otros recursos relacionados a los problemas de *C&P* el lector puede dirigirse al sitio de *ESICUP*.

Adicionalmente, se añaden las siguientes restricciones, las cuales provienen de las limitaciones prácticas y/o tecnológicas de la aplicación en cuestión:

- **Restricción de orientación:** Fija la orientación de todos los ítems, y prohíbe su rotación. Esto implica que un ítem de dimensiones *(a,b)* es distinto de un ítem de dimensiones *(b,a)*.

- **Restricción de corte de guillotina:** Exige que los cortes sean realizados de lado a lado sin parar, ya sea horizontal o verticalmente, tal como se puede ver en la FIGURA 1.6.

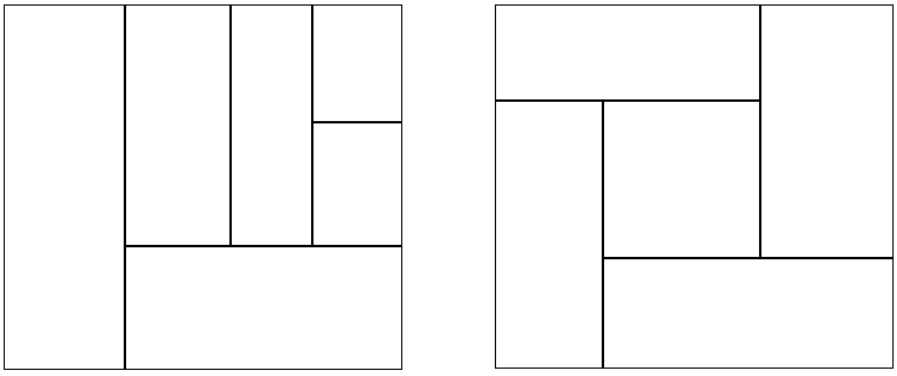


FIGURA 1.6 *Ejemplos de patrón guillotinable y no-guillotinable*

La restricción de orientación aparece, por ejemplo, por la calidad de la superficie del material (como resultado de la laminación), también en el diseño de páginas de periódicos, donde los elementos deben tener una orientación fija. Cabe señalar que el tratamiento de ítems con rotación es rara vez encontrado y una restricción de orientación es casi siempre asumida. Sin embargo (Bortfeldt y Winter, 2008) señalan que desde el punto de vista sistemático el caso sin una restricción de orientación (con rotación) es primario, mientras que el caso con una restricción de orientación (sin rotación) representa un problema derivado y por lo tanto secundario. Por esta razón, las variantes de problemas sin esta restricción deberían ser considerados.

Por otra parte, la restricción de guillotina aparece, por ejemplo, cuando se utilizan hojas de guillotina para dividir la placa en piezas rectangulares, lo cual es muy común en las industrias de fabricación. Igualmente, en una situación de empaque, a menudo se requiere que los elementos se puedan descargar en etapas, extrayendo simultáneamente todos los artículos empacados en la misma vertical u horizontal.

En la práctica, maximizar el valor cortado o empacado a menudo no es suficiente. También son importantes otros aspectos, como los tiempos de operación, o la vida útil de la maquinaria utilizada. (Cui y Huang, 2012) señalan que la elección del patrón debe considerar el compromiso entre dos aspectos: utilización del material y complejidad del patrón; generalmente patrones simples conducen a una baja utilización del material pero cortos tiempos de operación. Por el contrario, patrones más complejos, conducen a una mejor utilización del material, a costa de mayores tiempos de operación.

Debido a lo anterior, formulaciones más orientadas a las aplicaciones, imponen una restricción adicional, que consiste en restringir el número de etapas a una cantidad acotada, de manera de simplificar el patrón y así reducir los tiempos de operación. Esta restricción se conoce como *k*-etapas, donde *k* denota el número máximo de etapas permitido para realizar la operación de corte o descarga. Típicamente, esta restricción fija el número de etapas a 2 o 3, con lo que se obtienen patrones simples, aunque en ocasiones, con baja utilización del material. La FIGURA 1.7 muestra que 4 etapas son necesarias para obtener el patrón de la FIGURA 1.6. En la Figura 1.8 se muestra un ejemplo de patrón en 2-etapas y un patrón en 3-etapas.

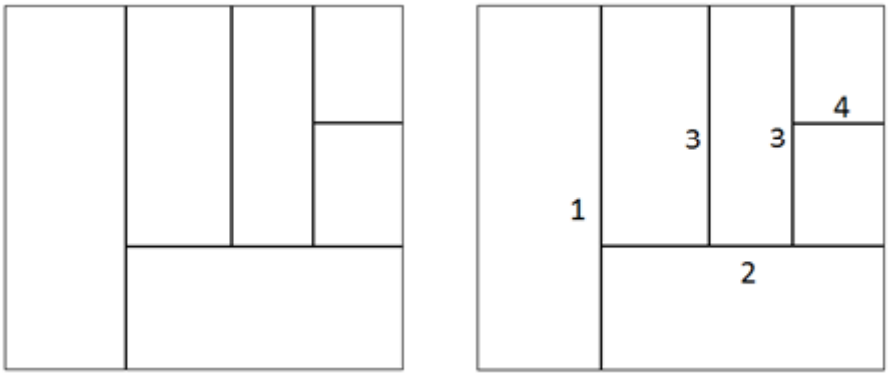


FIGURA 1.7 *Número de etapas necesarias para obtener el patrón de la FIGURA 1.6*

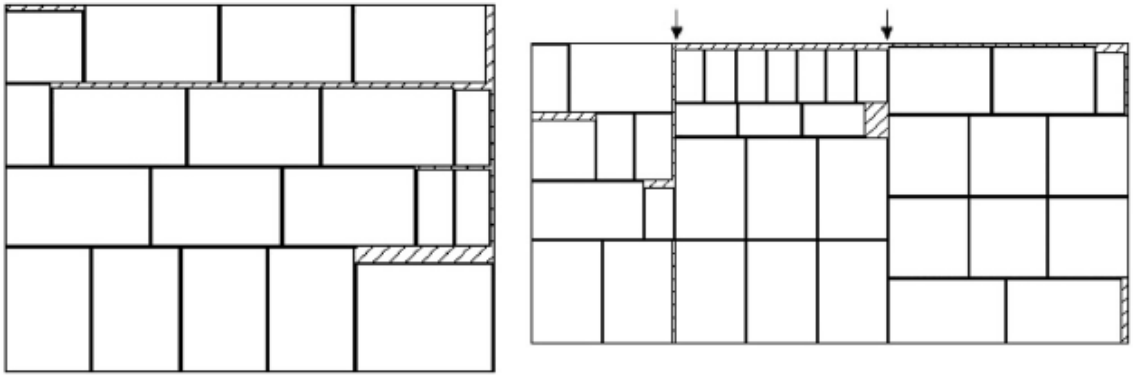


Figura 1.8 *Ejemplos de patrones en 2-etapas y 3-etapas*

Desde un punto de vista computacional, la restricción de *k*-etapas simplifica el problema (Dolatabadi et al., 2012). Por una parte, debido a que el espacio combinatorio es mucho más reducido, al... Por ejemplo, es posible formularlo como un problema de programación entera, extendiendo la formulación del bien conocido *1D-CP*. Propuesta por ...Hasta la fecha no se conocen formulaciones matemáticas para la variante sin etapas, por lo que métodos de enumeración o métodos de búsqueda. Es por ello que, en términos de dureza, resulta más interesante el problema sin la restricción de *k*-etapas.

La variante que se aborda en este trabajo de titulación corresponde al *CTDC* con las restricciones de orientación y guillotina. Este es uno de los problemas más interesantes, ya que es computacionalmente más difícil de resolver que cualquier otra variante de *TDC* (por ejemplo es más difícil que el *UTDC* o el *TDC* en *k*-etapas). (Yoon et al., 2013) El problema puede ser caracterizado de la siguiente forma:

Las dimensiones de la placa *R* son *L x W* (length y width); el *i*-ésimo item tiene dimensiones l_i *x* w_i, valor v_i (equivalente a su área $si=l_i*w_i$), y un límite superior de

demanda b_i, $i=1,...,m$. El objetivo es cortar la placa en xi piezas del tipo i, de manera que $0<=x_i<=b_i$, $i=1,...,m$ y la utilidad total $\Sigma s_i x_i$ sea maximizada. Además se asume que:

1. Todos los items tienen orientación fija
2. Todos los cortes aplicados son de tipo guillotina
3. Los parámetros L; W, l_i, w_i, y b_i, i=1,...,m, son enteros no negativos

El vector $(x_1,...,x_n)$ corresponde a un patrón de corte, si es posible producir x_i piezas de tipo i, $i=1,...,m$, en la placa R sin superposición.

Tanto la fuerte dureza inherente del *CTDC* como sus variadas aplicaciones, lo convierten en un problema muy interesante, por lo que la investigación de nuevas técnicas y métodos más eficientes para abordarlo continúan siendo desarrollados en la actualidad. Así mismo, constituye un buen candidato...

1.3 Solución propuesta

1.3.1 Características de la solución

La solución propuesta consiste en modelar el problema mediante un enfoque evolutivo, a través de un algoritmo genético. Existen varias formas de hacer esto, sin embargo, dado que se desea evaluar el comportamiento en la búsqueda genética realizada, el modelo debe permitir que el algoritmo tenga la potencialidad de explorar cualquier región del espacio de búsqueda, guiándose únicamente por la evolución. Se adopta la representación propuesta por (Flores, 2012), la cual está basada en un esquema genotipo-fenotipo, en donde se abstrae la interpretación del cromosoma del núcleo de procesamiento genético. Sobre esta capa común de modelamiento, dos modelos de estructuración de población son evaluados para el problema.

1.3.2 Propósito de la solución

El propósito de este trabajo es determinar con un nivel de confianza suficiente, si un enfoque de estructuración celular es más apropiado que un enfoque no estructurado, para tipos de problemas de carácter geométrico.

1.4 Objetivos y alcances del proyecto

1.4.1 Objetivo general

Evaluar los efectos de la estructuración de la población en un AG mediante un enfoque celular, aplicado a problemas de carácter geométrico, en particular para el problema de corte de piezas guillotinable bidimensional restricto

1.4.2 Objetivos específicos

i. Realizar un estado del arte sobre algoritmos genéticos celulares.
ii. Realizar un estado del arte sobre algoritmos genéticos aplicados al problema en estudio o similares.
iii. Diseñar e implementar las representaciones y funciones constructoras para el problema en estudio.
iv. Diseñar e implementar un algoritmo genético y un algoritmo genético celular que resuelvan el problema en estudio.
v. Identificar conjuntos de instancias de prueba.
vi. Diseñar un experimento computacional que permita evaluar los algoritmos propuestos.
vii. Analizar los resultados obtenidos.

1.4.3 Alcances

En este trabajo, dos modelos de población en *AG* son comparados, evaluando tanto su comportamiento numérico, como en la convergencia de la evolución hacia el óptimo. Para ello, se diseña cada uno de los *AG* a evaluar. Primero, se verifican experimentalmente mediante una optimización de parámetros, los mecanismos de una estructuración celular. Luego, se determina estadísticamente, si el enfoque celular es superior a un enfoque no estructurado. Adicionalmente, se investigan fuentes de sesgo en la búsqueda genética en el desempeño de los algoritmos, y se proponen distintas consideraciones para reducirlo, y eventualmente mejorar su desempeño.

1.5 Metodologías y herramientas utilizadas

En este trabajo se utiliza una plataforma computacional, *JCELL*, desarrollada por el grupo de investigación *NEO* (Networking y Emerging Optimization), para evolucionar individuos para el *CTDC*. Se realiza la evolución bajo dos enfoques: un enfoque no estructurado, consistente en un *AG* generacional, y un enfoque estructurado, consistente en un *AG* celular. Para la configuración de cada algoritmo se usa una plataforma de optimización de parámetros, *ParamILS*
. Para responder las preguntas de investigación se realiza un experimento computacional, en el cual cada *AG* es ejecutado sobre un conjunto de instancias ampliamente usadas en la literatura, cuyos óptimos son conocidos. Así, se evalúa la calidad de las soluciones, en términos de precisión respecto al valor óptimo. Los resultados obtenidos mediante el experimento computacional son analizados. Para sustentar la hipótesis se realizan pruebas estadísticas apropiadas a la distribución de los datos. Los test estadísticos son realizados mediante la suite estadística *IBM SPSS*.

1.6 Organización del documento

El Capítulo 2 presenta la revisión de la literatura respecto al problema de corte de piezas bidimensional restricto, primero en un contexto general y luego desde un enfoque evolutivo. Luego se presentan algunos antecedentes históricos del origen y desarrollo de los algoritmos genéticos celulares. El Capítulo 3, describe la metodología evolutiva utilizada, donde se enuncian los conceptos de algoritmos genéticos, representación, operadores y estructuración de población, centrándose en el enfoque celular. El Capítulo 4 describe procedimiento experimental realizado. En éste,. Se indica el modelamiento del problema, los datos de prueba seleccionados, la configuración de parámetros realizada para los algoritmos propuestos, el diseño experimental y la plataforma evolutiva utilizada. En el Capítulo 5, se presentan los resultados obtenidos, estos son analizados intentando dar respuesta a las preguntas de investigación. Finalmente, el Capítulo 6 presenta las conclusiones del estudio, donde se resumen los principales hallazgos y se enumeran las consideraciones para trabajos futuros.

Capítulo 2 ESTADO DEL ARTE

2.1. El problema de corte de piezas guillotinable bidimensional restricto (CTDC)

Dada la antigüedad, y debido a sus diversas aplicaciones, el problema ha sido ampliamente abordado, por lo que no resulta extraño que la gama de trabajos existentes, sea muy extensa y variada. En la literatura se reconocen diversos enfoques, los cuales se pueden clasificar en: exactos, heurísticos y metaheurísticos. Los enfoques exactos están representados principalmente por *tree-search* o *branch-and-bound*; también existen métodos exactos basados en *dynamic optimization* y otros son basados en modelos. Todos ellos también pueden ser encontrados en métodos heurísticos, sin garantía de optimalidad. Los enfoques metaheurísticos incluyen algoritmos genéticos (*AG*), Simulated Annealing (*SA*), Tabú Search (*TS*) y Greedy Randomized Adaptive Search Procedure (*GRASP*).

Dentro de los métodos exactos, debido a las limitaciones computacionales, varios estudios se han enfocado en clases especiales de patrones de corte (Yoon et al., 2013), tales como los: *p-group*, *k-staged* (Scheithauer 2004, Alves 2010), *t-shaped* (Cui, 2007). Sin embargo, pocos estudios han considerado algoritmos exactos para el *CTDC* general (sin etapas). Existen dos enfoques algorítmicos para abordar este problema: *top-down* y *bottom-up*. El enfoque *top-down* genera todos los patrones posibles mediante el corte sucesivo de subplacas. Así, el problema se define como la búsqueda en un árbol como el que se muestra en la FIGURA 2.1, donde las ramas representan cortes horizontales o verticales, y los nodos representan la subplaca resultante del corte.

En contraste, el enfoque *bottom-up*, se basa en la observación de que cualquier patrón que satisfaga la restricción de guillotina, puede ser obtenido mediante la construcción horizontal o vertical de rectángulos, como se muestra en la FIGURA 2.2. Todas las posibles combinaciones de rectángulos más pequeños son generadas para obtener rectángulos más grandes hasta que no puedan obtenerse más patrones de guillotina. Ambos enfoques pueden usar *upper* y *lower-bounds* en procedimientos de búsqueda, para descartar ramas no prometedoras.

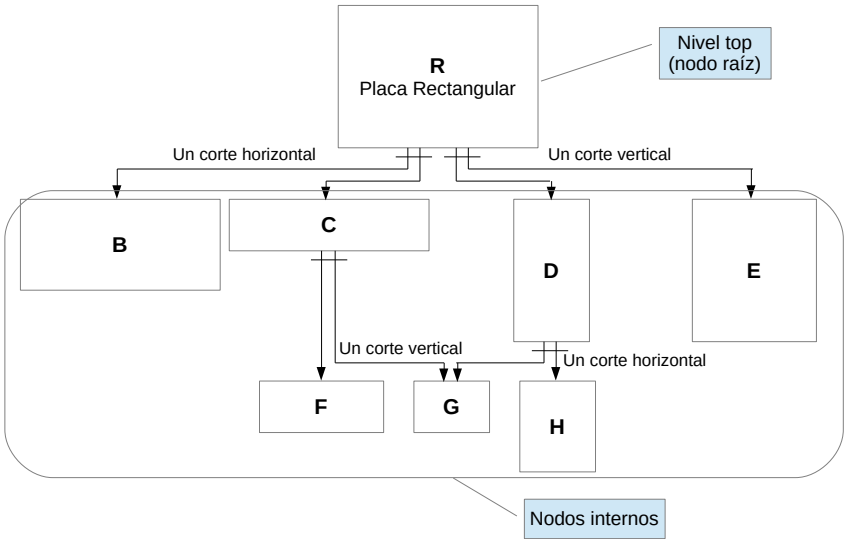


FIGURA 2.1 *Enfoque top-down: Proceso de búsqueda en el árbol*

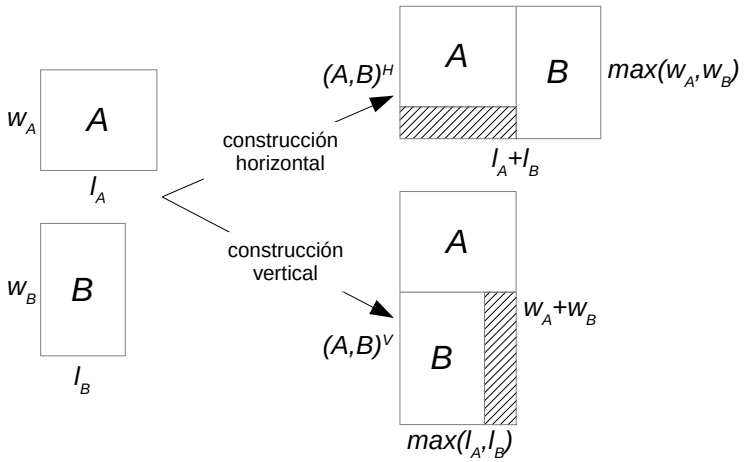


FIGURA 2.2 *Enfoque bottom-up: Construcción horizontal o vertical de patrones*

El *CTDC* ha sido resuelto de manera exacta por cada uno de estos enfoques. (Christofides y Whitlock, 1977) propusieron originalmente el enfoque *top-down*; desarrollaron un procedimiento *tree-search* basado en una estrategia de búsqueda *depth-first*, resolviendo instancias pequeñas de manera exacta. Además proponen un procedimiento de discretización o normalización, el cual considera solamente combinaciones lineales de las dimensiones de las piezas ordenadas. Esto permite lidiar con un número finito de cortes en métodos de enumeración. (Hifi y Zissimopoulos, 1997) propusieron un algoritmo mejorado, el cual utiliza *lower* y *upper-bounds* más efectivos. Aplicaron una estrategia de búsqueda *depth-first*, la cual requiere una pequeña cantidad de memoria, a expensas de mayor tiempo computacional y dificultades en presencia de restricciones adicionales.

Por su parte, el enfoque *bottom-up*, propuesto originalmente por (Wang, 1983), fue generalizado por (Viswanathan y Bagchi, 1993); desarrollaron un método *branch-and-bound* basado en una estrategia de búsqueda *best-first*, donde cada nodo corresponde a un patrón de corte *R* cuya área es *g(R)* y *P* el área en la placa sin ocupar, (ver FIGURA 2.3). La estrategia *best-first* consiste en expandir el nodo más promisorio, utilizando para ello, una estimación del *upper-bound* relativo a P^1. Se generan combinaciones verticales u horizontales del patrón seleccionado con los patrones correspondientes a los nodos hoja. (Hifi, 1997) mejoró este algoritmo mediante la aplicación de *lower* y *upper-bounds* más efectivos. (Cung et al., 2000) mejoró su eficiencia añadiendo algunas estrategias de poda y almacenando "códigos de patrones" en los nodos, para remover patrones repetidos, los cuales comparten el mismo tamaño y la misma combinación de piezas. *best-first* requiere más memoria, pero menor tiempo de cómputo que *depth-first*, además es flexible ante restricciones adicionales. Recientemente (Yoon et al., 2013) proponen varias mejoras al algoritmo anterior; un nuevo método más eficiente es usado para remover patrones duplicados, se usa una estrategia eficiente de poda, se proponen dos nuevos *upper-bounds* y se proponen dos métodos para prevenir la formación de patrones dominados. El algoritmo es comparado con el de (Cung et al., 2000), el cual había sido previamente el algoritmo exacto más eficiente para el *CTDC*. Además, para las instancias de escalas mayores, se incluye una comparación con el algoritmo heurístico *TDH*, el cual está basado en un enfoque *top-down* con *hill-climbing*. El algoritmo propuesto reduce de manera dramática la cantidad de nodos generados en el árbol de ramificación, gracias a técnicas eficientes de poda y a

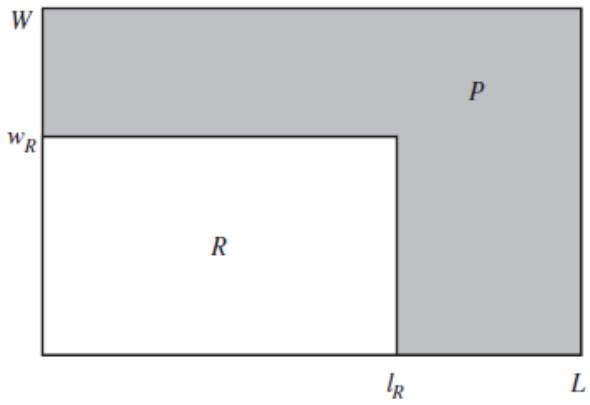


FIGURA 2.3 *Branch-and-bound: estado de un nodo; tamaño del patrón R y la placa*

[1] Corresponde al mínimo entre distintas soluciones candidatas. Ej: la solución de la ecuación recursiva del problema de programación dinámica formulado por Gilmore y Gomory (1966), la solución óptima para el problema de programación dinámica del UTDC, entre otras.

upper-bounds más estrictos. Además, obtiene soluciones óptimas para instancias de larga escala, previamente desconocidas. Sin embargo, no es capaz de resolver algunas instancias de larga escala debido a falta de memoria.

Dentro de los métodos basados en modelos, un enfoque novedoso es el que propone la teoría de grafos. Recientemente (Clautiaux et al., 2013) presentan un modelo teórico llamado *grafo de guillotina*. Este modelo usa grafos de coloración dirigidos, en donde los circuitos están relacionados a combinaciones horizontales o verticales. La idea es que cada *grafo de guillotina* puede ser asociado con una clase específica de patrón, llamada *clase corte-guillotina*, esto permite enfocarse en subconjuntos dominantes de soluciones, evitando redundancias en métodos de búsqueda.

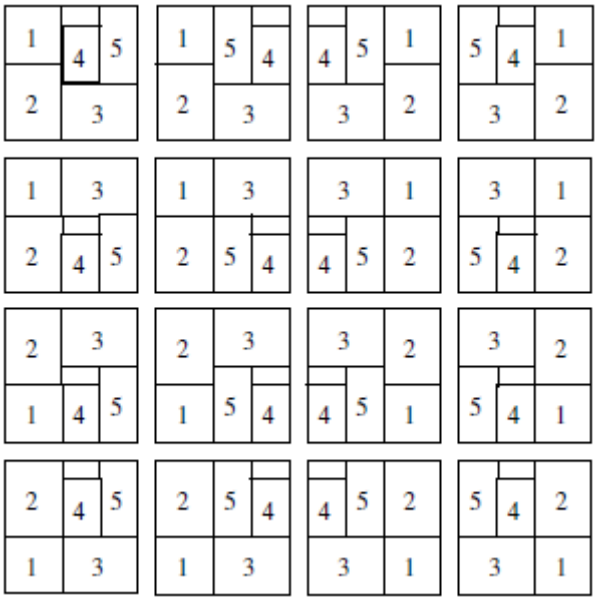


FIGURA 2.4 *Algunos elementos de una Clase Corte-Guillotina*

Las *clases corte-guillotina* describen conjuntos de patrones equivalentes (ver FIGURA 2.4), mediante una expresión denominada Recursive Multi Build (*RMB*), la cual se define como: un solo ítem, o una composición de combinaciones verticales horizontales sucesivas de *RMB*. Dicha expresión permite representar una gran cantidad de secuencias de combinaciones, cuyos patrones resultantes comparten la misma estructura combinatoria y pueden ser considerados como la misma solución. Una implementación computacional eficiente para representar los *RMB* es mediante grafos bi-coloreados dirigidos, en donde cada nodo corresponde a un *RMB* y cada arco, según su color, representa una combinación vertical u horizontal entre *RMB*. En este grafo, los

ciclos monocromáticos representan *RMB* que pueden ser reducidos mediante una contracción del circuito. Inicialmente cada nodo corresponde a un *RMB* compuesto de un solo ítem, los cuales se van reduciendo en sucesivas contracciones de circuitos monocromáticos. Si es posible reducir el grafo a un único nodo se habla de un grafo de guillotina (ver FIGURA 2.5).

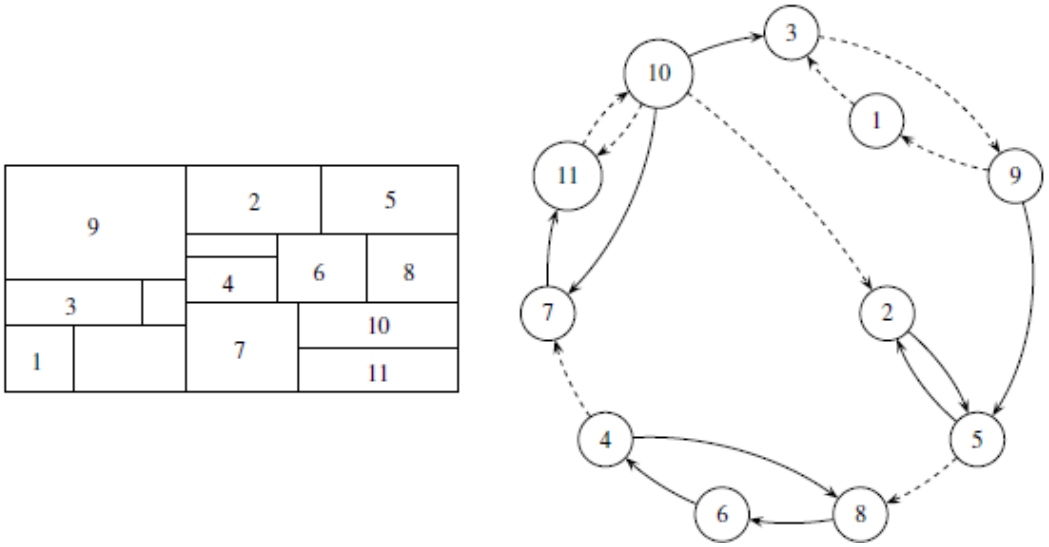


FIGURA 2.5 *Modelando la Clase Corte-Guillotina asociada al patrón con un grafo de guillotina*

En general, los algoritmos exactos no son un método práctico para resolver el *CTDC*, debido a la complejidad y a la explosión combinatoria, luego el costo computacional se torna exorbitante para instancias de mayor dureza (Cui y Chen, 2012). En estos escenarios, a menudo se usan heurísticas. Dentro de los métodos heurísticos, (Wang, 1983) Propuso un algoritmo constructivo usando el ya mencionado enfoque *bottom-up*, el cual genera patrones agregando sucesivamente piezas o grupos de piezas (soluciones parciales), generando así, nuevas soluciones parciales. Para evitar la explosión combinatoria, se eliminan las soluciones duplicadas y se define un criterio de "aspiración" para cada solución parcial, mediante la fijación de un parámetro β, el cual debe satisfacer $T_p<\beta WH$, donde T_p es la pérdida total de un patrón P. Este parámetro permite establecer un balance entre tiempo de computación y garantía de optimalidad. Posteriormente el algoritmo fue mejorado independientemente por (Vasko, 1989), descartando soluciones parciales que no puedan seguir siendo combinadas vertical u horizontalmente, y por (Oliveira y Ferreira, 1990) el cual cambia el criterio de "aspiración" para rechazar lo antes posible las soluciones parciales con pérdida total superior a βWH. Para ello, se considera un *lower-bound* asociado a la pérdida externa de

la solución parcial en conjunto con la pérdida interna, para decidir sobre la aceptación de la solución parcial. (Morabito y Arenales, 1992) presentan un enfoque basado en el modelo *And/or-Graph* (grafo y/o), el cual representa cada posible patrón como un camino completo en un grafo y/o, donde los nodos corresponden al rectángulo inicial, los rectángulos intermedios, las piezas, o las pérdidas, y los arcos corresponden a los cortes a los rectángulos. Propusieron una búsqueda *depth-first*, usando solo cortes normalizados y aplicando reglas de simetría, orden en los cortes y estrategias *hill-climbing* para reducir la búsqueda. En este trabajo resuelven instancias de larga escala para el *UTDC*. Unos años más tarde, (Morabito y Arenales, 1996) extendieron su enfoque previo, al *CTDC*, obteniendo buenos resultados. (Fayard et al., 1998) diseñaron un algoritmo basado en la resolución de una serie de problemas de la mochila usando *dynamic programming* para el *UTDC*. Además mostraron cómo su enfoque también podía ser usado para resolver de manera aproximada el *CTDC*. (Álvarez-Valdés et al., 2002) desarrollaron varios algoritmos heurísticos de propósito general para resolver las cuatro variantes del *TDC*. Propusieron dos procedimientos constructivos basados en limites simples mediante la resolución del problema de la mochila unidimensional. Usaron dichos algoritmos constructivos como *building-blocks* para procedimientos más complejos. Desarrollaron un algoritmo *GRASP* y un algoritmo *TS*. De este trabajo destaca *TS*, el cual es capaz de obtener resultados de alta calidad en tiempos moderados. (Hifi, 2004) presentó un algoritmo híbrido (denominado *TDH*) para el *CTDC*, en el cual, una búsqueda *depth-first* usando estrategias *hill-climbing* y *dynamic programming* son combinados. El algoritmo puede producir buenas soluciones para las instancias de larga escala. (Cui, 2007) presentó dos algoritmos exactos para el *CTDC*. basados en *branch-and-bround* combinado con técnicas de *dynamic programming*: uno para generar patrones *T-shape* para simplificar el proceso de corte, el otro para generar patrones en *3-etapas homogéneos*. Muestran cómo estos algoritmos pueden ser usados como heurísticas para generar patrones generales para el *CTDC*. (Chen, 2007) presentó un algoritmo heurístico recursivo (*REC*) para el *CTDC*, cuya formulación se basa en ubicar las piezas en la esquina inferior izquierda del bloque, de lo cual se obtienen dos formas de dividir la región inutilizada, vertical u horizontalmente. Luego el valor del bloque equivale al valor de la pieza ubicada más el valor de las pérdidas producidas (que también son bloques). Para reducir los tiempos computacionales usa *upper y lower-bounds*, se ordenan las piezas de acuerdo a un orden decreciente de sus valores y se restringe el tiempo computacional destinado al bloque de prueba actual. El algoritmo

REC produce buenas soluciones en corto tiempo para problemas de varias escalas. (Morabito y Pureza, 2008) proponen un método heurístico para el *CTDC*, el cual usa una relajación del espacio de estados de una formulación mediante dynamic programming para el *CTDC* en *k* etapas. Proponen un algoritmo de optimización subgradiente, el cual fija un valor de *k* suficientemente grande, y el cual incluye una heurística interna que convierte soluciones no factibles dadas en un paso dado del algoritmo, en soluciones factibles. Utilizan un enfoque And/or-graph para inicializar el *lower-bound* al comienzo del algoritmo. Se concluye que el algoritmo es competitivo comparado a otros métodos propuestos en la literatura y requiere cortos tiempos computacionales para proveer la mejor solución. Recientemente, (Cui y Chen, 2012) proponen una heurística simple, basada en una clase de patrón denominada *patrón de bloque extendido*, el cual contiene un número *k* de piezas del mismo tipo (piezas principales), ubicadas en la esquina inferior izquierda de la subplaca correspondiente. Se establecen 4 casos de *patrón de bloque extendido* (FIGURA 2.6), donde las piezas principales forman una fila o una columna y si el área no ocupada en la placa se divide en forma horizontal o vertical, respectivamente. Se propone un algoritmo goloso, denominado *HCEB* (heuristic for constrained extended blocks patterns), el cual va construyendo la solución de acuerdo a la fórmula de recursión *F(x,y)*, equivalente al maximo entre *F(x-1,y)*, *F(x,y-1)* y los valores correspondientes a cada unos de los *patrones de bloque extendidos*, previamente explicados. El valor inicial de *F(x,y)* es el mayor entre *F(x-1,y)* y *F(x,y-1).* y los valores de los 4 tipos de patrones antes descritos, los cuales son considerados para mejorar la solución. Antes de considerar un tipo de patrón se verifica si su valor es superado por un *upper bound* relativo y si la frecuencia de la pieza *i* no excede la restricción del problema. Si estas condiciones se cumplen se intenta mejorar el patrón mediante una reasignación de piezas a subplacas. *HCEB* opera de la siguiente manera, primero inicializa F(x,y) y n(x,y,j) a 0 para x e y inferiores a la dimensiones mínimas de las piezas. Las dimensiones de las subplacas se van enumerando de acuerdo a un orden ascendente del tamaño de las piezas. Luego para cada subplaca el algoritmo evalúa F(x,y) para cada pieza disponible.

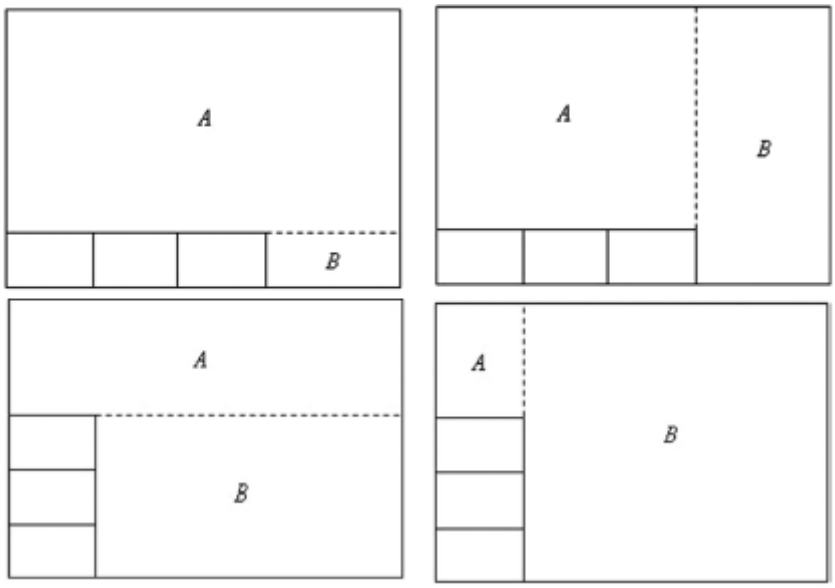


FIGURA 2.6 *Patrón de bloque extendido: los cuatro casos*

HCEB es comparado con otros tres algoritmos basados en enfoques heurísticos: *TS500*, un algoritmo basado en búsqueda tabú, *TDH2*, un algoritmo que usa programación dinámica y hill climbing, y *REC*, un algoritmo recursivo. Para instancias de larga escala, *HCEB* es superior al resto. Se concluye que el algoritmo provee un buen balance entre calidad de solución y tiempo computacional. Por otra parte, debido a su simpleza es sencillo de implementar. Además puede ser usado como una buena solución inicial para algoritmos exactos.

2.2. Algoritmos genéticos aplicados al CTDC

Debido a la relevancia que tienen para este estudio, se dedica esta sección para exclusivamente presentar los principales avances en este tipo de problemas desde el enfoque evolutivo, en particular los *AG*. Estableciendo así, un referente sobre cómo el problema ha sido abordado mediante esta metodología, así como los principales resultados, los cuales serán de vital importancia, tanto para el diseño del *AG* propuesto, como para el análisis de los resultados obtenidos.

De acuerdo a (Hopper, 2000) existen básicamente tres enfoques para abordar el problema. El más común es el enfoque en dos etapas, donde el *AG* es usado para explorar y manipular el espacio de soluciones, dadas por secuencias de elementos, y un segundo procedimiento, consistente en una rutina de colocación, es usado para evaluar las soluciones generadas. Un ejemplo de este enfoque se da en (Leung et al., 2003),

quienes aplicaron un algoritmo genético simple (*GA*) al *2D-KP*, donde una representación de permutación, correspondiente a la secuencia de piezas que debe ser empacada/cortada, constituye el genotipo, y una estrategia de colocación (*DP*) es usada como algoritmo decodificador, para así obtener un *patrón* válido, o fenotipo. Este trabajo, también constituye evidencia de que se ha intentado mejorar el comportamiento en la búsqueda del algoritmo para este problema, ya que la principal motivación fue la de intentar aliviar el problema de *convergencia prematura*. En su investigación, ellos encontraron que *GA* usualmente producía buenos resultados, sin embargo convergía muy rápido, de manera que buenos resultados eran producidos en etapas tempranas de la evolución. Propusieron un algoritmo genético mixto con simmulated annealing (*SAGA*); Introdujeron un operador que induce competencia entre los padres y los hijos; si los hijos son mejores que sus padres, estos se aceptan, pero si no, aún pueden aceptarse con cierta probabilidad. Hicieron variar esta probabilidad según la temperatura en un enfoque *simulated annealing*. En su estudio observan que *SAGA* se comporta un poco mejor que *GA*, porque, mientras que *GA* converge a buenas soluciones, este rápidamente se homogeniza, en tanto que *SAGA* inicialmente mantiene peores soluciones, sin embargo estas son capaces de evolucionar a individuos superiores en etapas maduras de la evolución (ver FIGURA 2.7).

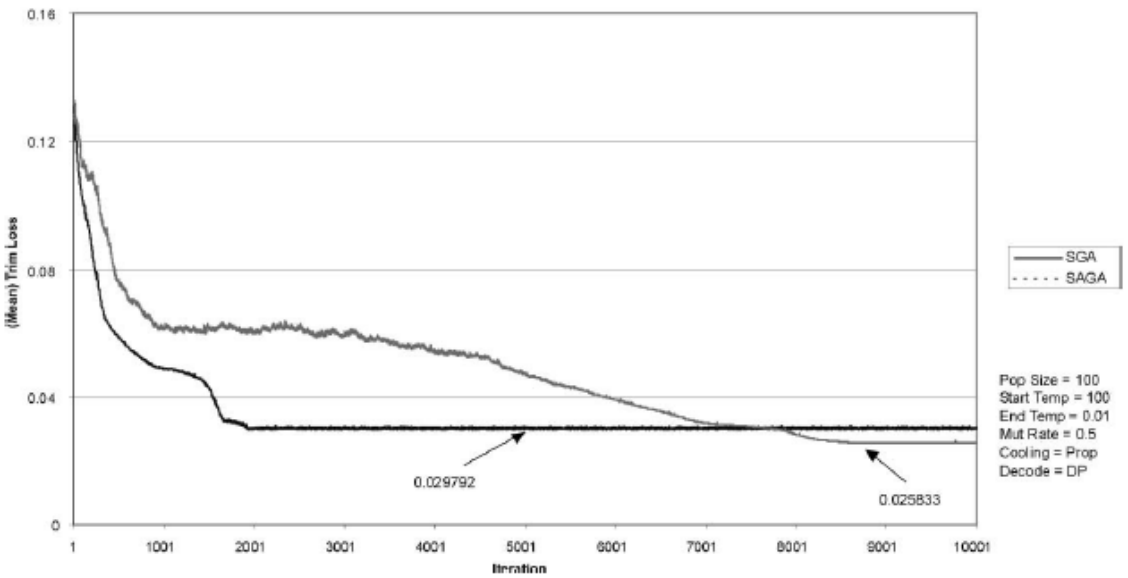


FIGURA 2.7 *GA vs SAGA Pérdida promedio en la población versus Iteraciones*

Un problema aparente del enfoque en dos etapas es su fuerte dependencia del decodificador, ya que el conocimiento del dominio del problema está oculto en la rutina

de colocación. Por otra parte, un decodificador podría limitar al AG, al no soportar la herencia de ciertas características en la descendencia (Hopper, 2000). En esta línea, (Hopper y Turton, 2000) realizan una investigación empírica de algoritmos heurísticos y metaheurísticos para el *2D-KP*. Consideran dos heurísticas: bottom-left (*BL*), la cual es simple y eficiente en tiempo, y bottom-left fill (*BLF*), la cual es capaz de rellenar espacios pero a costa de mayor tiempo computacional. Estas heurísticas son hibridizadas con tres metaheurísticas: algoritmo genético *AG*, simulated annealing *SA*, naïve evolution *NE* y una heurística de búsqueda local (*hill-climbing*). Su estudio compara los algoritmos híbridos en términos de calidad de solución y tiempo computacional. Además, con el fin de mostrar la efectividad de estos, su rendimiento es comparado con búsqueda aleatoria y rutinas heurísticas de empaque. Sus resultados muestran que en términos de calidad de solución, las metaheuristicas superan a las heurísticas, siendo *SA* la mejor de todas. Sin embargo *GA* y *NE* son mejores en términos de tiempo computacional. Debido a que las diferencias en el desempeño entre los algoritmos híbridos usando *BL* y *BLF* son debido a la heurística mejorada, el decodificador tiene mayor efecto en el desempeño de la técnica híbrida que la metaheurística en sí. Esto parece sugerir enfoques donde mayor conocimiento del patrón sea incorporado en la metaheuristica en vez del decodificador. Sin embargo señalan que este último tipo de representación, estudiado por otros investigadores, no necesariamente logró mayores densidades empacadas que las técnicas híbridas para los problemas probados.

Un segundo enfoque intenta incorporar más información acerca del patrón dentro de la estructura de datos del AG, aunque se necesitan ciertas reglas adicionales para fijar la posición en el patrón. Por ejemplo, (Ono y Ikeda, 1998) proponen un AG que utiliza una parte binaria, para representar el tipo de combinación (vertical u horizontal), y una parte numérica, representa las piezas. Así, el cromosoma es interpretado como un árbol binario, leído en notación post-fija (ver FIGURA 2.8). Se imponen ciertas restricciones sobre la cantidad de genes binarios respecto a genes numéricos y su posición relativa en el cromosoma para representar una solución válida para el problema. De acuerdo a la ecuación, todas las piezas son integradas recursivamente mediante un algoritmo de patrón, utilizando el enfoque *bottom-up*.

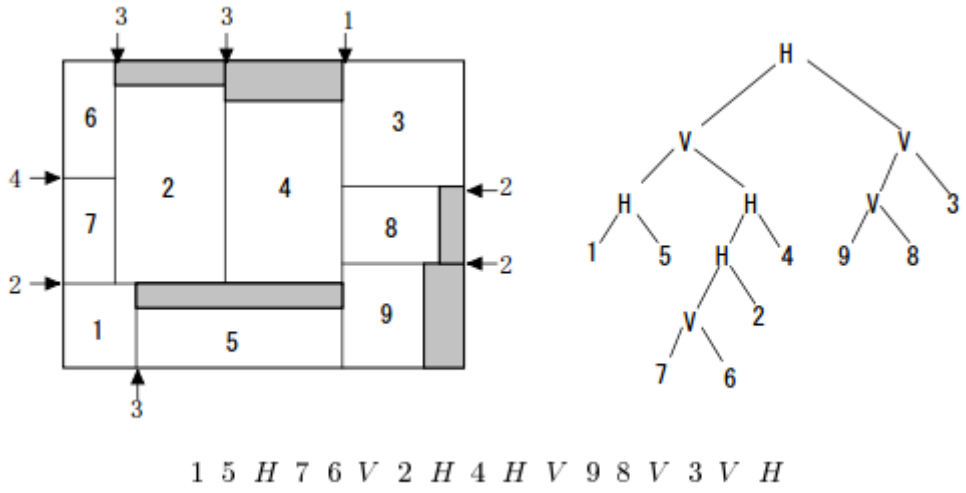


FIGURA 2.8 *Cromosoma y su interpretación como patrón guillotinable*

Otro caso donde se incorpora información del problema en la representación se encuentra en (Beasley, 2000), en el cual utiliza una representación binaria/real para el problema sin la restricción de guillotina. La parte binaria indica si la p-ésima copia de la pieza es cortada o no de la placa. Mientras que la parte real indica las coordenadas del centro de la pieza correspondiente, las cuales son aproximadas al entero más cercano. El *fitness* está dado por dos medidas: *fitness* es la función objetivo original, y *unfitness* indica en qué grado, un individuo está violando la restricción de superposición. Unos años después (Beraudo et al., 2004) adoptan la representación de (Beasley, 2000), con la diferencia de que las coordenadas no son aproximadas a enteros. El *fitness* está dado por la diferencia entre las piezas a cortar que pudieron ser colocadas en el patrón y las piezas a cortar que no pudieron ser colocadas, y es usado como un mecanismo de penalización. El objetivo es maximizar dicha diferencia, ya que una solución que contenga piezas que no pudieron ser cortadas, es menos deseable que otras donde todas las piezas hayan podido ser cortadas.

A diferencia de los dos primeros, un tercer enfoque traslada el proceso de búsqueda genética al dominio de patrones. Debido a que las operaciones genéticas son realizadas directamente en los patrones, este método no requiere una técnica de decodificación. Tiene la ventaja de que la implementación de los conceptos metaheurísticos (vecindad, operadores genéticos) tienen una significación y aplicación directa para el problema. Sin embargo, a diferencia de un enfoque en dos etapas, donde las restricciones geométricas son aplicadas por un algoritmo decodificador, en este caso la representación debe hacerse cargo de estas. Usualmente alguno de los siguientes

mecanismos son usados para ello: rechazo, reparación o penalización. Un ejemplo de este enfoque se encuentra en (Bortfeldt y Winter, 2008), quienes introducen un algoritmo genético llamado Strip Packing Genetic Algorithm Layer (*SPGAL*). El *AG* genera planes con una estructura de capas, donde el largo de cada capa está dado por el largo de una pieza definitoria, de manera que cada pieza se encuentra completamente contenida en una capa (ver FIGURA 2.9). Para medir la calidad de una capa, utilizan el filling rate (*fr*), que equivale al cuociente entre la suma de todas las áreas de las piezas contenidas en una capa y el área de la capa. Mientras que el *fitness* equivale al área total de todas las piezas empacadas.

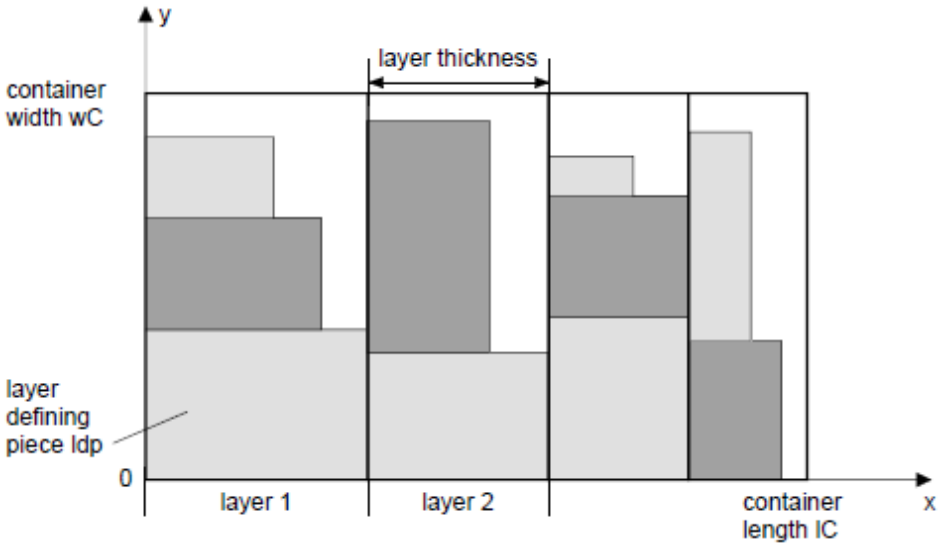


FIGURA 2.9 *Estructura de capas de un plan de empaque*

Para generar la población inicial utilizan la heurística Best Fit Decreasing Height (*BFDH*) con algunas mejoras. Para el cruzamiento, las capas con mejores *fr* son extraídas de los padres para combinar las buenas características de estos. Utilizan un procedimiento heurístico de completado de capas, el cual consiste en un algoritmo *tree search* que agrega capas y en cada paso selecciona la que tienen mayor *fr*. Además, consideran una post-optimización para mejorar aún más la mejor solución encontrada al final de la búsqueda genética. Esta reduce las pérdidas resultantes en los bordes de las capas.

Una de las aproximaciones evolutivas más recientes es propuesta en (Gonçalves y Resende, 2011), quienes presentan un AG basado en claves aleatorias, hibridizado con una nuevo procedimiento de colocación. La representación usa un alfabeto de números reales aleatorios entre 0 y 1 (claves aleatorias). El cromosoma está compuesto de dos

partes: la primera contiene la secuencia de piezas a empacar, mientras que la segunda indica el tipo de procedimiento de colocación usado para colocar cada pieza. Los autores justifican una representación en dos etapas debido a la dificultad de una representación directa de patrones de corte, en particular la dificultad de desarrollar operadores de cruzamiento y mutación. Para generar la secuencias de piezas, las *N* piezas del problema se biyectan con los primeros *N* genes del cromosoma, Luego de ser generados, estos son ordenados de manera ascendente, obteniendo así, una secuencia (ver FIGURA 2.10). Para los *N* siguientes genes, un valor menor o igual a 0.5 indica que se debe usar la heurística *BL*. En caso contrario se debe emplear la heurística *LB*.

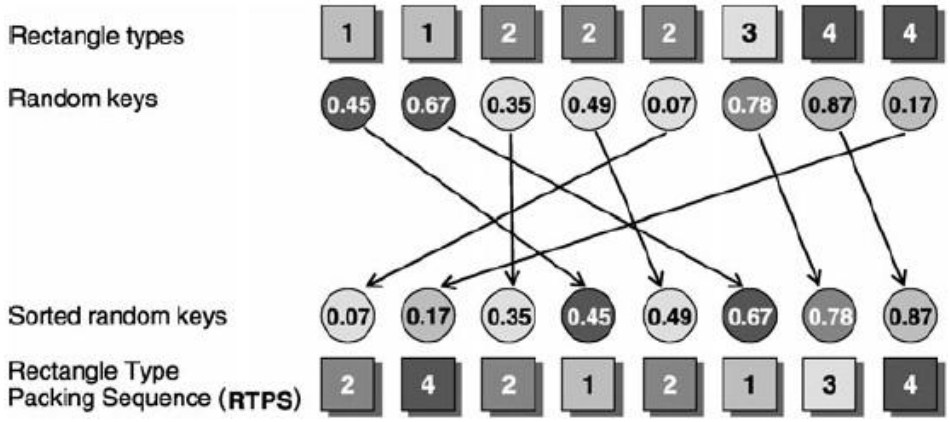


FIGURA 2.10 *Decodificación del cromosoma en representación de códigos aleatorios*

Para el proceso evolutivo emplean una estrategia elitista, preservando los mejores individuos en una porción de la población, *TOP*, reservada para ellos (ver FIGURA 2.11). Para el cruzamiento utilizan *cruzamiento uniforme* parametrizado, donde uno de los padres es seleccionado del pool *TOP*, y el otro es seleccionado al azar de la población. El aportador de cada alelo se determina lanzando una moneda cargada al mejor individuo, según una probabilidad de cruzamiento, la cual fijan experimentalmente en 0.7. En lugar de emplear un operador de mutación, en cada generación se generan nuevos individuos aleatorios que reemplazan al pool de los peores individuos de la población, *BOT*.

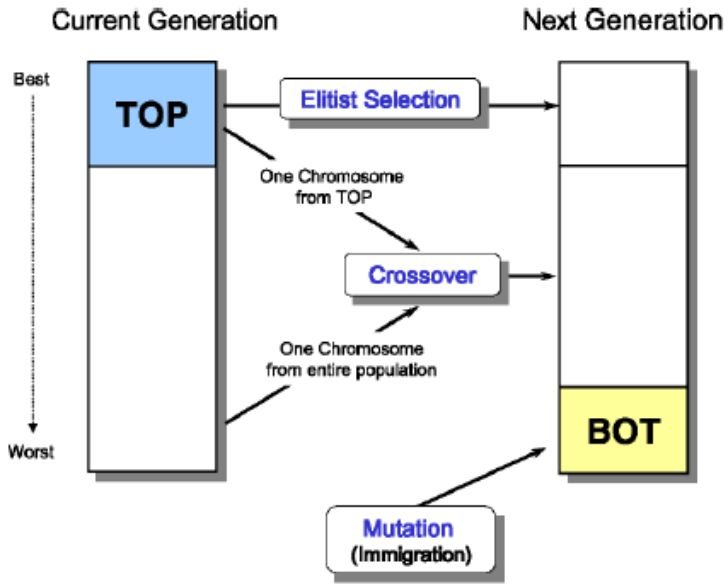


FIGURA 2.11 *Proceso de transición entre generaciones consecutivas*

La población inicial no es totalmente aleatoria. Se introducen cuatro cromosomas no aleatorios, cuyas secuencias de piezas están en un orden descendente de sus valores, considerando las siguientes combinaciones heurísticas: *random*, todas *BL*, todas *LB*, y *LB* alternado con *BL*. Resultados experimentales muestran que la inclusión de estos cuatro individuos mejora de manera significativa la calidad de las soluciones obtenidas. Proponen una modificación a la función de *fitness* natural (valor total de las piezas cortadas), la cual incorpora información sobre el potencial de mejoramiento del patrón; así, dos patrones con el mismo valor total, pueden tener diferentes potenciales de mejoramiento, dados por la forma en que las pérdidas están distribuidas; si están concentradas en una zona, será más probable la inserción de piezas en ella. (ver FIGURA 2.12). Para sus experimento configuran el *GA* mediante un estudio piloto, de donde obtienen los siguientes ajustes: *TOP*= 25%, *BOT*=15%, *CProb*=0.7. El tamaño de

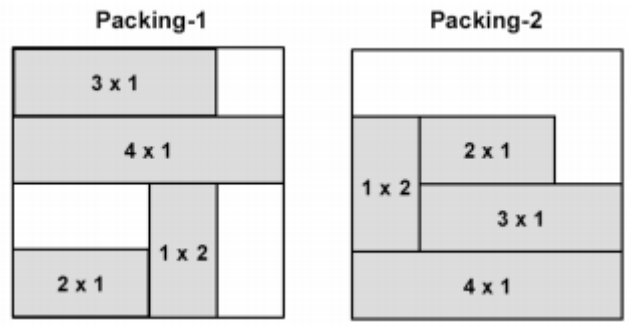


FIGURA 2.12 *Dos patrones con mismo valor total y diferentes potenciales de mejora*

población es referenciado al tamaño del problema, dado por la cantidad de piezas. Así, determinaron como tamaño 15 veces la cantidad de rectángulos de la instancia.

Realizan una implementación paralela, en la que paralelizan la evaluación del *fitness*, debido a su alto consumo de cómputo. Para ello emplean un esquema distribuido, en el cual los mejores dos individuos son compartidos de manera síncrona, con una frecuencia de migración determinada experimentalmente, al resto de las subpoblaciones. Comparan el algoritmo con otros enfoques, consistentes en una heurística basada en poblaciones *PH*, un algoritmo genético *GA*, *GRASP* y *TABU*. Concluyen que el enfoque propuesto es efectivo y robusto comparado con los otros enfoques.

2.3. Algoritmos genéticos celulares

En esta sección se presentan algunos antecedentes históricos del origen y desarrollo de los algoritmos genéticos celulares. Además se citan algunas evidencias, tanto teóricas como empíricas, que sustentan la noción de que los conceptos celulares permiten mejorar las características en la búsqueda realizada por un algoritmo genético, para ciertos escenarios y tipos de problemas.

Los algoritmos evolutivos celulares fueron diseñados inicialmente para funcionar en máquinas masivamente paralelas, En el caso más simple, un solo individuo era asignado a un procesador, y el apareamiento entre individuos estaba restringido al individuo más cercano. (Bethke, 1976) hizo el estudio teórico de un AG en una máquina paralela SIMD, analizando la eficiencia del uso de la capacidad de procesamiento. Concluyó que la máxima eficiencia es obtenida cuando la evaluación de la función objetivo es mucho más costosa que los operadores genéticos, lo cual sucede a menudo.

El primer modelo celular conocido (cGA) es el propuesto por (Robertson, 1987), implementado en un computador CM1. En este modelo, todos los operadores genéticos eran ejecutados en paralelo. El principal resultado de este trabajo es que el tiempo de ejecución era independiente del tamaño de la población.

(Muhlenbein et al., 1988) publicaron un trabajo donde un cGA en máquinas masivamente paralelas fue propuesto para el TSP. Incorporaron un paso de búsqueda local para mejorar las soluciones generadas. Así, es considerado como el primer cGA híbrido publicado.

El término algoritmo genético celular no fue usado sino hasta 1993, cuando (Whitley, 1993) lo propuso por primera vez en un trabajo donde un modelo de autómata celular era aplicado a un algoritmo genético.

Si bien todos estos cGAs fueron inicialmente diseñados para trabajar en máquinas masivamente paralelas, debido a la rápida pérdida de popularidad sufrida por estas máquinas, el modelo fue adoptado después para trabajar en máquinas mono-procesador, lo que evidencia que el modelo celular es independiente de la arquitectura sobre la cual está implementado (Alba y Dorronsoro, 2008).

Una forma simple para caracterizar la búsqueda realizada por un cGA es usar la *presión de selección*, la cual es una medida de la velocidad de difusión de las buenas soluciones a través de la población. Algunos trabajos teóricos comparan los algoritmos de acuerdo a la presión de selección mostrada, y en algunos casos incluso intentan modelar matemáticamente su comportamiento.

(Sarma y De Jong, 1996) realizaron un estudio teórico sobre la presión de selección inducida por los cGAs con diferentes operadores de selección, y tamaños y formas de vecindades. Para estudiar el efecto del tamaño de la vecindad en la presión de selección, propusieron una definición del radio de la vecindad como medida de su tamaño. Más aún, observaron el mismo efecto al cambiar el tamaño de la población, con lo cual propusieron una nueva medida llamado *ratio*, definida como la relación entre el radio de la vecindad y la población. Descubrieron que el ratio es un factor clave para controlar la presión de selección del algoritmo. Así, dos algoritmos con diferentes tamaños de población y vecindades, pero con el mismo *ratio*, tienen una presión de selección similar. Finalmente propusieron el uso de una función logística para aproximar la curva de presión de selección de los cGAs. El modelo propuesto pareció ser un buen enfoque para cGAs con poblaciones cuadradas, pero después fue demostrado que tiene ciertas deficiencias cuando se usan poblaciones rectangulares.

(Sprave, 1999) propuso una descripción unificada de cualquier tipo de EA (algoritmo evolutivo) con ambas, poblaciones estructuradas como no estructuradas, basado en el concepto de hipergrafo. Un hipergrafo es una extensión de un grafo canónico, donde el concepto de arco es generalizado: en lugar de unión entre pares de vértices son uniones de subconjuntos de vértices. Usando el concepto de hipergrafo, Sprave desarrolló un método para estimar la curva de crecimiento de presión de selección de un GA. Este método está basado en el cálculo del diámetro de la estructura

de la población y la probabilidad de la distribución inducida por el operador de selección.

(Gorges-Schleuter, 1999) estudió las curvas de crecimiento para un modelo celular de estrategia evolutiva (ES) con poblaciones estructuradas en formas toroidales o anillo. En su estudio, observó que el modelo celular (tanto el toroidal como anillo) tienen menor presión de selección que el ES equivalente con población no estructurada. Más aun, comparando los dos modelos, concluyó que, usando el mismo tamaño de vecindad, estructurar la población en forma de anillo permite una menor presión de selección que al usar una población toroidal.

(Giacobini et al., 2003) propusieron modelos cuantitativos para estimar el takeover time (el tiempo para colonizar la población mediante copias del mejor individuo bajo los efectos de la selección), para cGAs síncronos y asíncronos con una población estructurada en forma de anillo, y usando una vecindad compuesta por los dos individuos más cercanos al individuo considerado. Este trabajo fué extendido con el fin de encontrar modelos matemáticos precisos para fijar las curvas de presión de selección de cGAs síncronos y asíncronos. Posteriormente, los mismos autores propusieron ciertas recurrencias probabilisticas para modelar el comportamiento de la presión de selección de cGAs síncronos y asíncronos con poblaciones cuadradas, toroidales y lineales (anillo) para dos esquemas de selección diferentes. Este modelo no es completamente preciso cuando son usados otros esquemas de selección.

(Simoncini et al., 2006) propusieron un nuevo operador de selección para cGAs llamado *selección anisotrópica*, para ajustar la presión de selección del algoritmo. Esta consiste en permitir la selección de individuos de la vecindad con distintas probabilidades de acuerdo a su posición. De esa manera, los autores promueven la aparición de nichos en la población.

Finalmente en (Dorronsoro y Alba, 2007) fue presentada una ecuación matemática más precisa para modelar las curvas de presión de selección de cGAs con poblaciones rectangulares y cuadradas. El modelo propuesto fue

En la literatura existen resultados que sugieren, pero no analizan, que la forma de la grilla en la población realmente influye en la calidad de la búsqueda realizada por el algoritmo. Como se mencionó anteriormente el concepto de ratio es relevante porque algoritmos con ratios similares muestran un comportamiento similar en la búsqueda.

(Alba y Troya, 2000) publicaron un estudio cuantitativo de las mejoras obtenidas en la eficiencia de un cGA al usar grillas no cuadradas. En este trabajo, el

comportamiento de algunos cGAs con diferentes formas de grilla fue analizado en distintos problemas, concluyendo que el uso de grillas no cuadradas promueve un comportamiento eficiente en los algoritmos. Más aun, redefinieron el concepto de radio como la dispersión de un conjunto de patrones, la cual es más precisa que la definición previa de Sarma y De Jong. Adicionalmente, proponen cambiar dinámicamente la forma de la población, (ratio dinámico), y así autoajustar el balance entre exploración y explotación.

En esta misma línea, (Dorronsoro y Alba, 2005) desarrollaron un nuevo modelo adaptivo en el cual la forma de la población es modificada automáticamente para regular el balance entre exploración y explotación. Diferentes versiones del nuevo algoritmo adaptivo fueron comparados a los algoritmos con ratio estático, y las superaron a todas ellas en todos los casos.

A continuación se citan algunos importantes trabajos que se centran en el análisis del comportamiento de los cGAs, como el proceso evolutivo de los individuos en la población, o la complejidad del algoritmo de acuerdo a los operadores usados.

(Collins y Jefferson, 1991) caracterizan la diferencia entre los GA no estructurados y los cGAs de acuerdo a ciertos factores, como la diversidad del genotipo y fenotipo, la velocidad de convergencia, o la robustez del algoritmo, concluyendo que el apareamiento local realizado por los cGAs es más apropiado para la evolución artificial. Demuestran que para un problema particular con dos óptimos, un GA no estructurado raramente encuentra ambas soluciones, mientras que cGA generalmente las encuentra. Esto es debido a la lenta difusión de las mejores soluciones producidas por el cGA, la diversidad es mantenida por más tiempo en la población, formando pequeños nichos (grupos de individuos similares), representando diferentes áreas de búsqueda del algoritmo. Este trabajo motivó a otros autores a usar cGAs para encontrar óptimos múltiples para problemas. De este y otros trabajos similares se concluyó que un comportamiento característico de los cGAs es la formación de diversos nichos en la población donde el ciclo reproductivo tiende a promover la especialización de los individuos al interior de ellos. Así, los cGAs mantienen diversas rutas de búsqueda hacia diferentes soluciones, donde cada uno de estos nichos puede ser visto como una ruta de explotación del espacio de búsqueda.

(Manderick y Spiessens, 1991) publicaron un estudio comparativo de la complejidad temporal entre su cGA y un GA secuencial. mostrando que para el cGA esta aumenta linealmente de acuerdo al largo del genotipo. Por el contrario, la

complejidad de un GA secuencial aumenta polinomialmente de acuerdo al tamaño de la población multiplicado por el largo del genotipo. En este artículo los autores deducen el número esperado de individuos al usar los métodos de selección comunes en cGAs, mostrando que la selección proporcional es la que tiene menor presión de selección.

(Sarma y De Jong, 1995) compararon varios cGAs usando diferentes esquemas de selección y observaron que dos de las selecciones estudiadas se comportaron de manera distinta aún teniendo presiones de seleccións equivalentes, desmintiendo así, la asunción de que presiones de seleccións equivalentes implican comportamientos similares de búsqueda.

(Gordon et al., 1994) estudiaron siete cGAs con diferentes vecindades en problemas de optimización discretos y continuos. En su trabajo concluyeron que las vecindades más grandes trabajan mejor con problemas más simples, pero al contrario, con problemas más complejos es mejor el uso de vecindades más pequeñas.

Más recientemente, (Alba et al., 2002) realizaron un estudio comparativo del comportamiento de cGAs con políticas de actualización síncronas y asíncronas. Los resultados obtenidos muestran que los cGAs asíncronos ejercen una mayor presión de selección que los síncronos, por lo que convergen más rápido, y generalmente, encuentran la solución más pronto que los síncronos en los problemas menos complejos estudiados. Por el contrario, en el caso de los problemas más duros, los cGAs síncronos parecen ser los que ofrecen una mejor eficiencia, ya que los asíncronos se atascan en óptimos locales más frecuentemente.

Del estado del arte se desprende que, en general, soluciones competitivas son obtenidas al aplicar conocimiento especifico del problema, lo cual es atribuible a su fuerte componente geométrica. Por ejemplo, tanto para métodos exactos como aproximados, se suelen usar *lower-bounds* de alta precisión (respecto al óptimo), provistos por heurísticas o relajaciones del problema en la restricción de las piezas, o en el número de etapas para realizar los cortes. También suelen utilizarse heurísticas orientadas a ciertas clases especiales de patrones, los cuales exhiben propiedades geométricas que se traducen en buenas soluciones, como por ejemplo los patrones de bloque y bloque extendido. A menudo un preordenamiento de las piezas en función de sus características geométricas, como su área, ancho, largo, entrega ...Es posible obtener buenas soluciones mediante técnicas generales, y la aplicación de conceptos independientes del problema.

El problema ha sido ampliamente abordado mediante el enfoque evolutivo, en particular mediante *AG*. Respecto a los distintos enfoques

Se ha reconocido la existencia de problemáticas en el comportamiento de la búsqueda respecto a la convergencia del algoritmo, y han habido diferentes intentos de aliviar este problema. han habido intentos por mejorar la búsqueda realizada por el AG

El enfoque celular por su parte ha sido aplicado a diversos problemas, algunos de ellos son *TSP* y *VRP*. Aunque para estos problemas se usaron enfoques híbridos, combinándolo con operadores de búsqueda local. Una característica del enfoque celular, es que gracias a permite fácilmente modificar el comportamiento en la búsqueda del algoritmo,

Hasta ahora, la estructuración de población usando un enfoque celular, no se ha usado como un mecanismo para mejorar la búsqueda del algoritmo genético para este problema...

Capítulo 3 ALGORITMOS GENÉTICOS CELULARES

Este capítulo introduce las principales nociones de la metodología evolutiva. Además, se describen los principios de estructuración bajo un enfoque celular y sus principales mecanismos, para de esta manera obtener una comprensión y capacidad de verificación de los conceptos que subyacen en la hipótesis planteada.

3.1 Algoritmos evolutivos

Los algoritmos evolutivos son técnicas de optimización basadas en poblaciones, diseñados para la búsqueda de valores óptimos en espacios complejos. Estas técnicas están basadas en ciertos procesos biológicos naturales, como la selección natural y la herencia genética, entre otros. La FIGURA 3.1 muestra el funcionamiento de un algoritmo evolutivo canónico. Se puede apreciar su estructura iterativa, en la cual sucesivamente se va evolucionando una población actual de individuos. La evolución es el resultado de la aplicación de operadores estocásticos, como selección, cruzamiento y mutación, a fin de obtener una generación completa de nuevos individuos. La población inicial usualmente es generada de manera aleatoria, aunque también es usual emplear algunas semillas para mejorar el comportamiento de la búsqueda (Alba y Dorronsoro, 2008). Una evaluación del *fitness* asigna un valor a cada individuo, el cual representa su aptitud

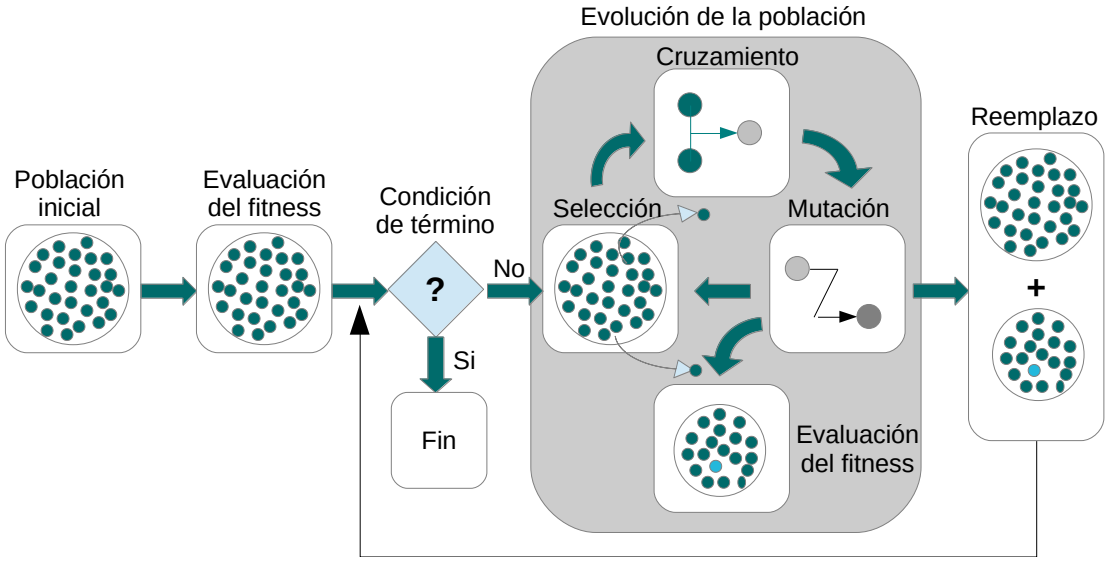


FIGURA 3.1 *Funcionamiento básico de un AG*

para el problema en estudio. Esta evaluación puede ser realizada por una función objetivo, (como una expresión matemática o una simulación computacional). El *fitness* es usado para decidir cuáles individuos son mejores y cuáles son peores. La condición de término usualmente es alcanzar un número máximo de iteraciones del algoritmo, o encontrar una solución para el problema (o una aproximación a esta, si es conocida de antemano).

Los algoritmos evolutivos, y en general las técnicas metaheurísticas, son plantillas algorítmicas que deben ser personalizadas a la aplicación objetivo. Así, la tarea de diseño se centrará en dos aspectos: Por una parte, decidir y diseñar la representación genética de una solución candidata para el problema en estudio, y por otra, decidir los operadores genéticos apropiados, dada la representación adoptada. A continuación, se describen brevemente algunas de las representaciones y operadores más ampliamente utilizados.

3.1.1 Representación de individuos

El primer paso en la construcción de un algoritmo evolutivo es decidir una representación genética de una solución candidata para el problema. Esto involucra definir el genotipo y el mapeo desde el genotipo al fenotipo. Es importante elegir la representación correcta para el problema a resolver, lo cual es una de las partes más difíciles en el diseño de un buen algoritmo evolutivo, a menudo esto se logra con la práctica y un buen conocimiento del dominio de la aplicación (Eiben y Smith, 2003).

- ***Representación binaria***

Históricamente, esta es una de las primeras representaciones, y muchos AG la han adoptado de manera independiente al problema que han tratado de resolver. Aquí, el genotipo consiste en un *string* binario. Para ciertos problemas, en particular aquellos que incluyen variables de decisión booleanas, el mapeo desde el genotipo al fenotipo es natural, pero frecuentemente, *strings* binarios son usados para codificar información no binaria. Estos casos, sugieren el uso de otros tipos de representación, los cuales eventualmente conducen a mejores resultados (Eiben y Smith, 2003).

- ***Representación entera***

Las representaciones binarias no siempre son las más apropiadas si el problema se puede mapear a una representación donde los diferentes genes puedan tomar un conjunto de valores. Por ejemplo, supóngase que se intenta evolucionar una ruta en una grilla cuadrada. Luego, los valores pueden ser restringidos al conjunto {0,1,2,3} representando los puntos cardinales. Una codificación entera es probablemente más apropiada que una binaria.

- ***Representación real***

A menudo, la manera más sensible de representar una solución candidata para un problema, es tener un *string* de valores reales. Esto ocurre cuando los valores a representar como genes, provienen de una distribución más bien contínua.

- ***Representación de permutación***

Muchos problemas naturalmente toman la forma de decidir el orden en el cual una secuencia de eventos debería ocurrir. Aunque existen otras formas de representación, como funciones decodificadoras, basadas en representaciones enteras sin restricciones, o "llaves flotantes", basadas en representaciones reales, la representación más natural para tales problemas es la permutación de un conjunto de enteros.

3.1.2 Operadores genéticos

Los operadores genéticos constituyen el núcleo evolutivo del AG. Inspirados en procesos biológicos como la selección natural, la reproducción sexual, y la mutación, a continuación se explican los operadores genéticos más comunes.

- ***Selección***

Parte de la evolución está determinada por la selección natural de los individuos en su adaptación al entorno. Inevitablemente, algunos individuos son mejores que otros,

éstos son más propensos a sobrevivir, aprender y difundir su material genético. La selección es una función de la aptitud, es decir, qué tan bien compite cada individuo en su entorno. La intensidad que tenga el mecanismo de selección para discriminar cuán apto es un individuo en relación al resto, determinará cuál es el tiempo necesario para que el individuo más sobresaliente se difunda por la población completa, esto se denomina *presión de selección*, y cada operador posee una presión característica.

a. Selección por ranking: Ordena la población basándose en el *fitness* y asigna probabilidades de selección de acuerdo a su ranking. El mapeo desde la posición en el ranking a la probabilidad de selección es arbitrario, y puede hacerse de varias formas, a menudo se usa un mapeo lineal o exponencial. La ecuación 3.1 corresponde a la probabilidad de seleccionar un individuo en el caso de un ranking lineal. Donde *n* es el tamaño del ranking y *j* es la posición del individuo *i* en el ranking. En general un mapeo lineal limita la *presión de selección*, mientras que un mapeo exponencial pone mayor énfasis en seleccionar individuos cuyo *fitness* supera la media.

$$p_i = \frac{2 \cdot (n - j)}{n \cdot (n - 1)} \tag{3.1}$$

b. Selección por ruleta: .Se simula el giro de una ruleta, la cual es particionada en proporción a los *fitness* de cada individuo. La ruleta se gira n veces para seleccionar *n* individuos. La ecuación 3.1 corresponde a la probabilidad de seleccionar un individuo, la cual es proporcional a su *fitness*. Para simular el giro de la ruleta, se asume algún orden en la población (ranking o aleatorio). Cada vez que se gira la ruleta se genera un valor aleatorio *r* uniforme en [0,1], luego para cada individuo en la población, si su probabilidad de selección p_i supera a *r*, entonces es seleccionado.

$$p_i = \frac{fitness(i)}{\sum_{j \epsilon pob} fitness(j)} \tag{3.2}$$

c. Selección por torneo: Un grupo *k* de individuos, típicamente dos, son seleccionados aleatoriamente y el mejor es considerado como el padre. A diferencia de los operadores anteriores, la selección por torneo no requiere conocimiento global de la

población. En lugar de ello, solo depende de una relación de orden que permita rankear cualquier par (o más) individuos. Similar a un esquema de ranking, utiliza un *fitness* relativo en vez de absoluto. La probabilidad de que un individuo sea seleccionado depende de:

- Su ranking en la población. Efectivamente esto es estimado sin necesidad de ordenar la población completa.
- El tamaño *k* del torneo. Mientras más individuos considere, mayor es la oportunidad de que contenga individuos superiores a la media.
- La probabilidad *p* de que el individuo más apto del torneo sea seleccionado.
- Si los individuos son escogidos con o sin reemplazo.

- ***Cruzamiento***

Tal como en la reproducción sexual, el cruzamiento considera el apareamiento entre dos individuos[2], produciendo crías que contienen una combinación de la información de sus padres. La importancia del cruzamiento es su capacidad de reunir propiedades de soluciones candidatas que pueden estar localizadas en regiones distantes entre sí. Esto favorece la exploración, ya que permite descubrir áreas prometedoras en el espacio de búsqueda, mediante grandes saltos a áreas que se encuentran entre dos padres.

a) Cruzamiento de 1 punto: Se escoge una posición aleatoria en el cromosoma, luego se dividen ambos padres en ese punto y se crean dos hijos intercambiando las colas.

b) Cruzamiento de *n* puntos: Es la generalización del cruzamiento de 1 punto, en la cual el genotipo es fragmentado en más de dos segmentos de genes continuos, luego los hijos son creados tomando segmentos alternados de los padres.

c) Cruzamiento uniforme: Cada gen es tratado de manera independiente, y por cada uno se escoge aleatoriamente de qué padre debe ser heredado

[2] En un contexto computacional, donde no existen las limitaciones biológicas, esto podría ser fácilmente generalizado a *n* padres y *n* hijos

Una característica que exhibe el cruzamiento de *n* puntos es el sesgo posicional, ya que tiende a mantener juntos genes que están cercanos el uno del otro en el genotipo. En general a medida que la cantidad de puntos de cruce aumenta, este sesgo tiende a disminuir. En contraste, el cruzamiento uniforme no exhibe sesgo posicional, sin embargo tiene una fuerte tendencia a transmitir el 50% de los genes de cada padre y en contra de transmitir un mayor número de genes coadaptados. Esto es conocido como sesgo distribucional.

- ***Mutación***

Así como en el caso biológico, los individuos pueden mutar ocasionalmente experimentando pequeños cambios en su información genética. La mutación permite la explotación de áreas prometedoras, realizando pequeños movimientos dentro de dicha región. También es un mecanismo que ayuda a mantener cierto grado de diversidad en la población (evitando atascarse en óptimos locales).

a) Mutación binaria**:** El operador más comúnmente usado para representaciones binarias considera cada gen separadamente y permite a cada bit cambiar con una pequeña probabilidad p_m. Luego para una codificación de largo L, en promedio Lp_m

Los operadores genéticos revisados en esta sección, corresponden a los típicamente usados para representaciones binarias. Sin embargo, también existen operadores para los otros tipos de representaciones indicados. Una revisión de estos puede ser encontrada en (Eiben y Smith, 2003).

3.2 El modelo celular

Originalmente, los algoritmos genéticos conciben la población como un único y gran conjunto de individuos (FIGURA 3.2 a la izquierda), dentro del cual, la evolución ocurre mediante sucesivos ciclos reproductivos de manera libre, en un contexto global. En este escenario, la presión de selección, fuerza motriz de la evolución, actúa de manera que los individuos más aptos rápidamente comienzan a dominar la población, difundiendo su material genético, y produciendo así, una rápida pérdida en la diversidad

genética. En otras palabras, en un AG existe poco control sobre la presión de selección, provocando una alta rapidez de difusión de buenos individuos.

Otra manera de caracterizar el comportamiento antes descrito, es mediante el reconocimiento de dos fuerzas contrapuestas que coexisten en la evolución: explotación y exploración. La explotación corresponde a la intensificación de individuos, promoviendo una búsqueda en profundidad sobre el espacio de soluciones. En contraste, la exploración está dada por la diversificación de individuos, promoviendo una búsqueda en anchura sobre el espacio de soluciones. Un exceso de explotación hace que el algoritmo profundice rápidamente en una región del espacio, descartando otras regiones. Este es el caso de los algoritmos genéticos, donde una presión de selección no controlada promueve una rápida intensificación de buenos individuos, profundizando en una región del espacio y descartando el resto. En contraste, un exceso de exploración hace que el algoritmo visite distintas regiones sin profundizar en ninguna de ellas, por lo que buenos individuos no llegan a ser especializados. La idea es poder establecer un balance entre ambas fuerzas.

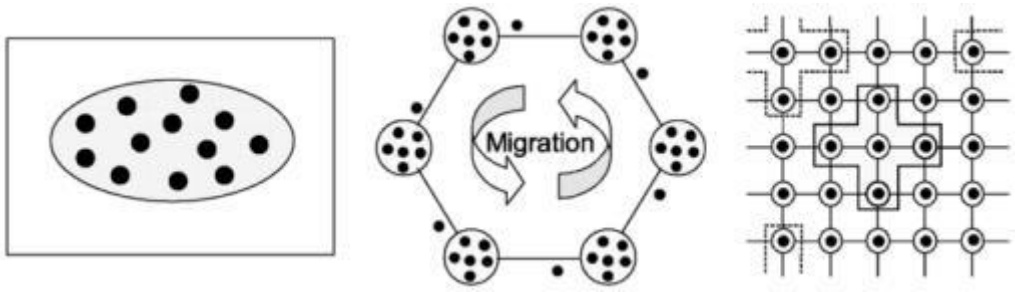


FIGURA 3.2 *Distintos enfoques de estructuración. De izquierda a derecha: Población no estructurada, Población distribuida y Población celular*

Como una forma de mejorar el comportamiento altamente explotativo de los algoritmos genéticos, surge el concepto de descentralización, donde la población es de algún modo estructurada, con la idea de que el aislamiento de poblaciones permite una mayor diferenciación genética, y de que el uso de poblaciones descentralizadas provee un mejor muestreo del espacio de búsqueda, mejorando así el comportamiento en la búsqueda realizada por el algoritmo.

Un primer enfoque de estructuración es el distribuido, en el cual la población es dividida en un conjunto de islas, las cuales se comunican mediante intercambios puntuales de individuos (migración), con el objetivo de introducir cierta diversidad en las subpoblaciones (ver FIGURA 3.2 al centro). Esta evolución independiente dentro de

cada isla promueve la diversidad en la población global. Debido a que los ciclos reproductivos ocurren en contextos más acotados (islas), existe cierta influencia sobre la presión de selección global, restringiendo la difusión desencadenada, propia de una población no estructurada. El intercambio de individuos conduce a una aceleración en la convergencia dentro de cada isla, ya que la introducción de buenos individuos produce grandes saltos evolutivos dados por

Un segundo enfoque es el celular, el cual estructura la población basándose en el concepto de aislamiento por distancia, donde solo individuos cercanos pueden interactuar. Cada individuo, es concebido como una célula, con una determinada posición en una grilla interconectada (ver FIGURA 3.2 a la derecha). Surge así el concepto de vecindad, la cual corresponde a una agrupación de individuos, dentro del cual ocurre el ciclo reproductivo. La superposición entre vecindades provee un mecanismo implícito de comunicación entre grupos, ya que vecindades adyacentes comparten algunos individuos.

En un algoritmo genético celular, la exploración se da por la superposición de vecindades, mientras que la explotación ocurre al interior de cada vecindad. La FIGURA 3.3 a la izquierda muestra una grilla bidimensional de individuos, topología típica en una población estructurada mediante un enfoque celular. Se puede apreciar que los individuos ubicados en los límites están conectados con los opuestos en la misma fila/columna. El efecto adquirido es una grilla toroidal, donde todos los individuos tienen el mismo número de vecinos. A la derecha se muestran seis formas de vecindad típicamente usadas: Aquellas de la forma *Ln* (lineales), distribuyen los individuos según una disposición axial, mientras que aquellas de la forma *Cn* (compactas), distribuyen los individuos en direcciones horizontales, verticales y diagonales. Las más comúnmente usadas son *L5*, también llamada *NEWS* o de *Von Newmann*, y *C9*, también conocida como de *Moore*.

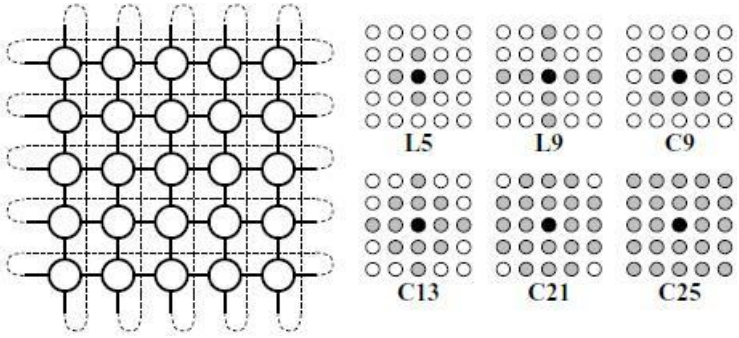


FIGURA 3.3 *Población toroidal y vecindades típicas: L5 o NEWS, y C9 o Moore*

La propia definición de la población bajo un enfoque celular provee dos mecanismos que permiten una mayor influencia y control sobre la *presión de selección*, estableciendo un balance entre exploración y explotación del algoritmo. Estos mecanismos son el *ratio* y la *política de actualización*.

3.2.1 Ratio

Para caracterizar formalmente la población se define el concepto de *radio*. De acuerdo a Alba y Troya, 2000 el *radio* equivale a la dispersión de n^* puntos en una elipse centrada en $(\bar{x}, \bar{y})$. Como se indica en (Alba y Dorronsoro, 2008), esta definición toma en cuenta el caso de grillas no cuadradas. Sea n^* la cantidad de puntos de la grilla, cuyas coordenadas son (x_i, y_i), luego:

$$radio = \sqrt{\frac{\sum(x_i - \bar{x})^2 + \sum(y_i - \bar{y})^2}{n^*}}\ , \bar{x} = \frac{\sum_{i=1}^{n^*} x_i}{n^*}, \bar{y} = \frac{\sum_{i=1}^{n^*} y_i}{n^*} \tag{3.3}$$

La ecuación 3.3 también provee un valor para el radio de la vecindad. La relación entre la vecindad y la población puede ser cuantificada mediante el *ratio* (ecuación 3.4), el cual es una medida de la relación entre la dispersión de los individuos en la vecindad respecto a la dispersión de los individuos en la población.

$$ratio = \frac{radio_{vecindad}}{radio_{población}} \tag{3.4}$$

La FIGURA 3.4 muestra el cálculo del radio para el tipo de vecindad *NEWS*. En este caso, la vecindad se compone de 5 individuos, luego $n^* = 5$. Para simplificar los cálculos, por conveniencia se centra la vecindad en el origen, con lo cual $\bar{x} = \bar{y} = 0$. Luego, usando la ecuación 3.3:

$$rad_{NEWS} = \sqrt{\frac{[(-1)^2 + (0)^2 + (1)^2] + [(-1)^2 + (0)^2 + (1)^2]}{5}}$$

$$rad_{NEWS} = \sqrt{\frac{2+2}{5}} = \sqrt{\frac{4}{5}} = 0{,}8944 \tag{3.6}$$

Considérese una población de 25 individuos, distribuidos en una grilla cuadrada de 5x5 individuos. El cálculo de su radio se realiza de la misma forma. En este caso $n^* = 25$. Para sintetizar los cálculos es útil considerar las simetrías axiales de la grilla. Con lo cual se amplifica por cuatro, los cálculos acotados a un cuadrante. Luego:

$$rad_1 = \sqrt{\frac{4 * [(1)^2 + (2)^2]}{25}} = \sqrt{\frac{4 * 5}{25}} = \sqrt{\frac{20}{25}} = 0{,}8944 \tag{3.7}$$

En una segunda instancia, considérese distribuir los individuos en una grilla rectangular, por ejemplo, de 4x8 individuos. En este caso, $n^* = 24$. Además dada la simetría axial, se amplifica por dos, los cálculos acotados a cada hemisferio. Luego:

$$rad_2 = \sqrt{\frac{2 * \{[(1)^2 + (2)^2 + (3)^2 + (4)^2] + [(1)^2]\}}{24}}$$

$$rad_2 = \sqrt{\frac{2 * 31}{24}} = \sqrt{\frac{62}{24}} = 1{,}6073 \tag{3.8}$$

Al comparar los radios de cada grilla, se puede ver que una grilla rectangular posee un mayor radio que una cuadrada, ya que la mayor dispersión de puntos respecto a uno de los ejes tiene un mayor efecto en la dispersión total, que la disminución en la dispersión del eje ortogonal. Así, los ratios resultantes usando la misma vecindad son:

$$ratio_1 = \frac{rad_{NEWS}}{rad_2} = \frac{0{,}8944}{0{,}8944} = 1, \qquad ratio_2 = \frac{rad_{NEWS}}{rad_1} = \frac{0{,}8944}{1{,}6073} = 0{,}556 \tag{3.9}$$

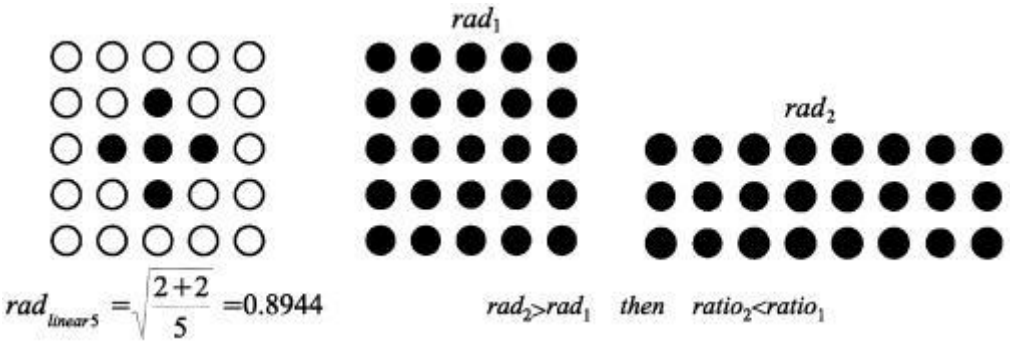


FIGURA 3.4 *A la izquierda, radio de la vecindad NEWS. A la derecha, grillas de 5x5=25 y 3x8≈25. El mismo número de individuos puede generar dos ratios diferentes*

Como se muestra en la figura, para la misma vecindad, una grilla rectangular produce un menor ratio que una grilla cuadrada. Esto queda evidenciado en las ecuaciones 3.9. Disminuir el valor del *ratio* reduce la intensidad de la selección global en la población, lo cual promueve la exploración. Esto permite una mayor diversidad en la población (Alba and Dorronsoro, 2008).

El concepto de *ratio* puede interpretarse intuitivamente mediante la idea de que, mientras más dispersos se encuentren los individuos en la población, mayor resistencia habrá a la difusión de buenos individuos inducida por la presión de selección, ya que resulta más difícil esparcir el material genético a través de un espacio extendido y estrecho, que por un espacio contraído y ancho. Más aún, mientras menos dispersos se encuentren los individuos en la vecindad, la difusión será aún más lenta, debido a que la superposición entre vecindades será mínima, limitando aún más la rapidez de la difusión de buenos individuos. Promoviendo aún más la exploración del algoritmo.

La FIGURA 3.5 muestra las distintas curvas de crecimiento en la proporción del mejor individuo inducidas por distintos *ratios*, al variar el *radio* de una población de 1024 individuos. Estas curvas se obtienen al evolucionar una población considerando únicamente el efecto de la selección. Como se observa, un menor *ratio* induce una menor presión de selección. La mayor presión de selección es inducida cuando los individuos se disponen de forma cuadrada (*32x32*), presentando la menor dispersión posible. A medida que la disposición de los individuos es más rectangular, disminuye el *ratio*, por consiguiente la presión de selección es menor. En el otro extremo, cuando los

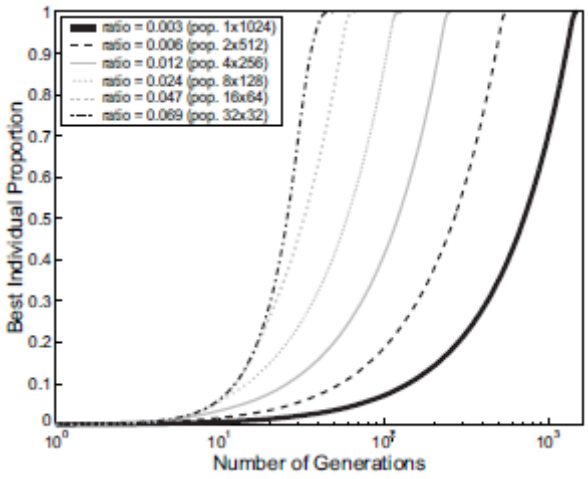


FIGURA 3.5 *Presión de selección inducida por distintos ratios*

individuos se disponen de forma lineal (*1x1024*), se maximiza la dispersión y por consiguiente, se minimiza la presión de selección.

La FIGURA 3.6 muestra dos tipos de vecindad. Como se puede apreciar, el *radio* de la vecindad afecta directamente el grado de superposición entre vecindades. Lo que incide a su vez en la rapidez de difusión de buenas soluciones. Luego es posible ajustar el balance entre exploración y explotación mediante el *radio* de la vecindad.

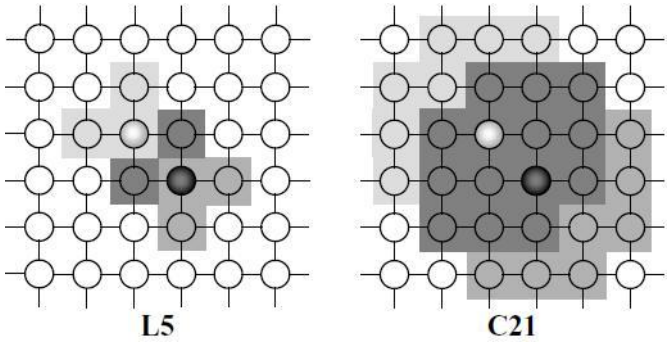


FIGURA 3.6 *Vecindades más grandes inducen un mayor nivel de migración implícita*

3.2.2 Política de actualización

En un contexto de estructuración celular, el ciclo reproductivo ocurre al interior de cada vecindad. Como se puede ver en la FIGURA 3.7, la selección de los progenitores tiene lugar dentro de una vecindad, estos generan un descendiente (mediante cruzamiento y mutación), el cual finalmente reemplaza al individuo actual.

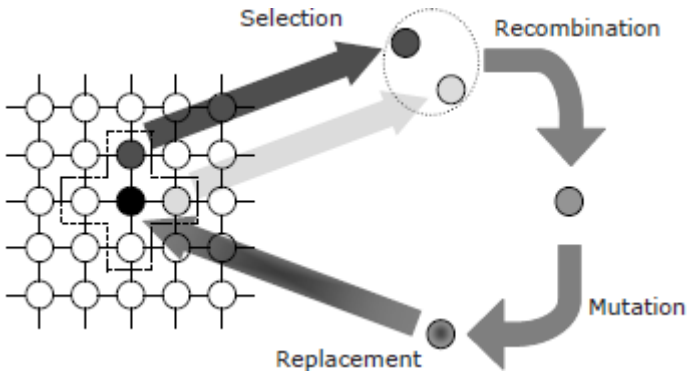


FIGURA 3.7 *Ciclo reproductivo de cada individuo*

La política de actualización determina cómo se aplica el ciclo reproductivo a los individuos. En un primer nivel se distingue una política síncrona, en la cual todos los

individuos son actualizados simultáneamente, usando para ello, una población auxiliar. En contraste, en las políticas asíncronas, los individuos son actualizados de acuerdo a una secuencia de actualización. Para determinar en qué orden los individuos son actualizados se establecen diversas políticas:

- Line Sweep (*ls*): Actualiza los individuos en el orden natural de *1* a *n.*
- Fixed Random Sweep (*frs*): Se genera una secuencia de actualización de acuerdo a una probabilidad uniforme sin reemplazo. Esta secuencia es fija durante la evolución.
- New Random Sweep (*nrs*): Similar a la anterior, con la diferencia de que se genera una nueva permutación aleatoria en cada generación.
- Uniform Choice (*uc*): Se genera una secuencia de actualización de acuerdo a una probabilidad uniforme con reemplazo.

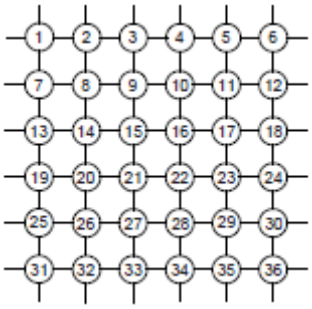


FIGURA 3.8 *Enumeración de individuos en la población*

Cada una de las políticas antes descritas inducen distintas intensidades en la presión de selección global. Una política síncrona induce una menor presión de selección frente a políticas asíncronas, ya que en el primer caso, todos los individuos evolucionan de manera simultánea en un paso generacional, mientras que en el caso asíncrono, individuos más competitivos son introducidos en la población durante el paso a la siguiente generación, produciendo un mayor diferencial evolutivo entre la generación actual y la siguiente. Dentro de las políticas asíncronas, la mayor presión de selección es inducida por la política *ls*, ya que cada ciclo reproductivo selecciona individuos que ya han sido evolucionados en el paso generacional actual. Mientras que para el resto de las políticas, esta situación no necesariamente aplica para todos los ciclos reproductivos, dependiendo en estos casos, de factores estocásticos.

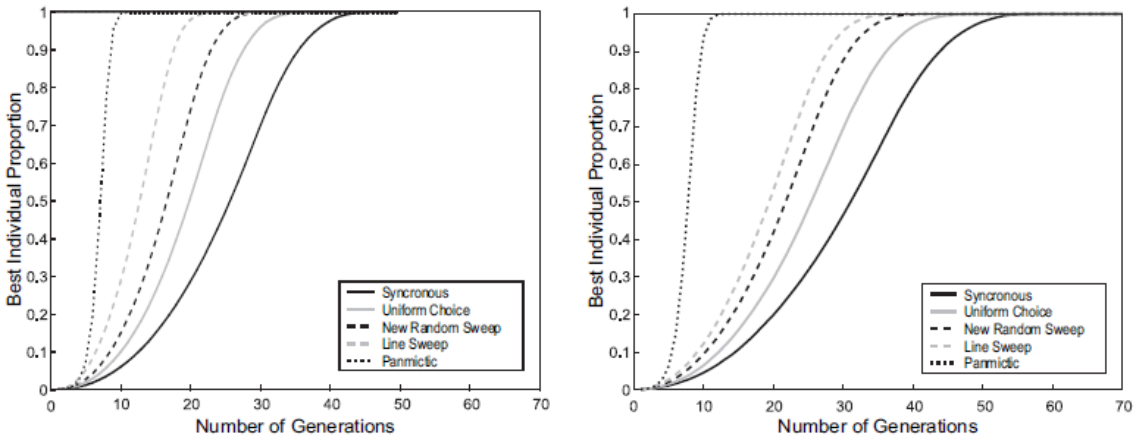


FIGURA 3.9 *Presión de selección inducida por distintas políticas de actualización. A la izquierda usando "torneo-binario" y a la derecha usando "ruleta"*

La FIGURA 3.9 muestra las curvas de crecimiento en la proporción del mejor individuo, donde se evidencian las nociones previamente expuestas, siendo las políticas asíncronas, inductoras de una mayor presión de selección frente a la política síncrona. Más aún, dentro de las políticas asíncronas, la que induce una mayor presión de selección es la política *ls*, seguida por *nrs*, siendo *uc* la menor. Cabe destacar la importante brecha existente entre cualquiera de las políticas de actualización y la curva de crecimiento en un algoritmo genético no estructurado, cuyo comportamiento altamente explotativo genera una curva mucho más inclinada, indicando con ello, que el mejor individuo rápidamente coloniza la población completa. Finalmente, es importante notar las diferencias en las presiones de selección globales, ejercidas por cada operador de selección, donde se aprecia que una mayor presión es ejercida por una selección por torneo binario, frente a una presión ejercida por una selección por ruleta, en cuyo caso, las distintas políticas de actualización tienen una influencia mucho mayor.

El algoritmo 3.1 corresponde al seudocódigo de un algoritmo genético celular. Como se puede observar, los operadores genéticos son aplicados dentro de la vecindad de los individuos, de manera que individuos pertenecientes a diferentes vecindades no están autorizados para interactuar. Como se puede ver, en el caso síncrono, los individuos que componen la siguiente generación son almacenados en una población auxiliar, y cuando se completa, esta reemplaza en un paso atómico la población actual. Así, todos los individuos son actualizados simultáneamente y la creación de individuos está hecha solo desde los individuos en la población presente (no aquellos creados durante la misma iteración).

ALGORITMO 3.1 Pseudo-código de un algoritmo genético celular síncrono

```
1.  proc Evolucionar(agc)        // Parámetros del algoritmo en 'agc'
2.  GenerarPoblacionInicial(agc.pop);
3.  Evaluacion(agc.pop);
4.  Mientras(!condicionDeTermino()) hacer
5.        Para individuo ← 1 hasta agc.tamPop hacer
6.              vecinos ← CalcularVecindad(agc,posicion(individuo));
7.              padres ← Seleccion(vecinos);
8.              descendiente ← Cruzamiento(agc.Pc,padres);
9.              descendiente ← Mutacion(agc.Pm,descendiente);
10.             Evaluacion(descendiente);
11.             Reemplazo(posicion(individuo),aux_pop,descendiente);
12.       Fin-Para
13.       agc.pop ← aux_pop;
14. Fin-Mientras
15. Fin proc Evolucionar
```

Capítulo 4 MATERIALES Y MÉTODOS

Este capítulo detalla el modelamiento utilizado, así como los pasos experimentales, los cuales permiten asegurar la replicabilidad de la experimentación computacional realizada en este trabajo.

4.1 Modelamiento del problema

4.1.1 Genotipo-fenotipo

Los *AG* que abordan problemas de *C&P*, a menudo utilizan un enfoque basado en decodificadores de secuencias para la representación del problema. Una ventaja de este enfoque es su simpleza, al separar la lógica del dominio del problema del núcleo de procesamiento genético. Como se señaló en la sección 2.2, debido a la fuerte dependencia del decodificador, esta aproximación puede no parecer la más apropiada para este tipo de problemas, sin embargo, los resultados empíricos no son concluyentes al respecto (Hopper y Turton, 2000). Más allá de esto, la principal razón por la cual se

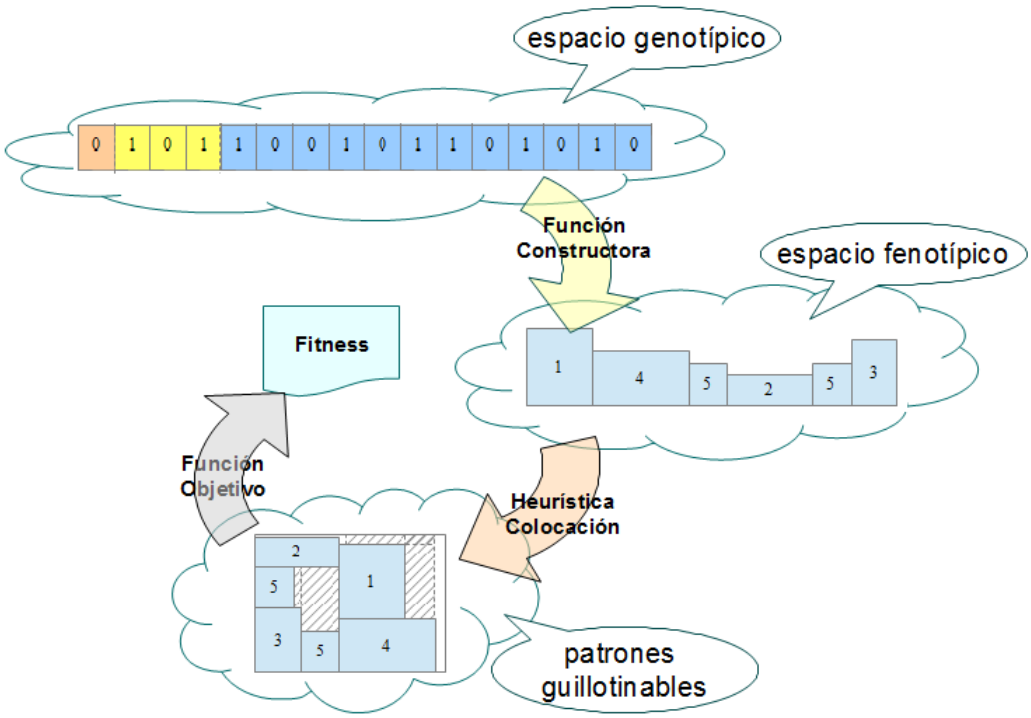


FIGURA 4.1 *Esquema general: Función constructora + heurística colocación*

ha adoptado este enfoque, es por su consistencia respecto a la aplicación de conceptos independientes del problema en la mejora del comportamiento en la búsqueda del *AG*.

En este esquema, la idea es que la metaheurística se encargue de realizar la búsqueda en el espacio de soluciones, mientras que una heurística de colocación se encarga de realizar la traducción a una solución válida e interpretable para el problema. El esquema general es descrito en la FIGURA 4.1, se parte de un espacio genotípico, donde tiene lugar el proceso evolutivo. Cada genotipo es mapeado mediante la función constructora a un fenotipo, consistente en una secuencia de piezas. El fenotipo es traducido a un patrón guillotinable mediante una heurística de colocación. El patrón finalmente es evaluado por una función objetivo para obtener el *fitness* de la solución candidata.

4.1.2 Representación

La representación utilizada está basada en la propuesta por Flores, 2012, la cual utiliza una codificación binaria para representar diversas secuencias de piezas[3]. Para ello, el cromosoma se divide en dos partes: una parte, compuesta por genes de presencia/ausencia, indica cuáles piezas están presentes en la secuencia, y una parte compuesta por genes de ordenamiento, codifica qué orden tienen las piezas presentes, en la secuencia.

Para los genes de presencia, se hace una biyección entre cada una de las copias de los distintos tipos de pieza y cada gen del cromosoma, de esa manera, cada posición referencia a una copia. De esto se deduce que la cantidad de genes de presencia es equivalente a la suma de todas las copias de cada tipo de pieza. Para cada gen, un valor 1 indica que la copia de la pieza está presente en la secuencia, y un valor 0 indica que no lo está. Considérese el ejemplo de la FIGURA 4.2, con una demanda de 6 tipos de piezas. Para cada tipo se definen sus dimensiones y su límite superior. Así, la pieza de tipo 1 tiene 3 copias, la de tipo 2 tiene 2 copias, la de tipo 3 tiene 1 copia, etc.

Las piezas son serializadas en un arreglo que mantiene cada una de las copias, y que es usado como referencia para interpretar cada gen de presencia. En la FIGURA 4.3 se ilustra la interpretación de un cromosoma de ejemplo, según el cual: la primera copia

[3] La representación adoptada no es capaz de generar todas las secuencias posibles. Sin embargo, se asume que es capaz de generar una diversidad de secuencias suficiente como para obtener soluciones de calidad razonable

de la pieza de tipo 1 está presente en la secuencia, la segunda y tercera copias no lo están, la primera copia de la pieza de tipo 2 si lo está, etc.

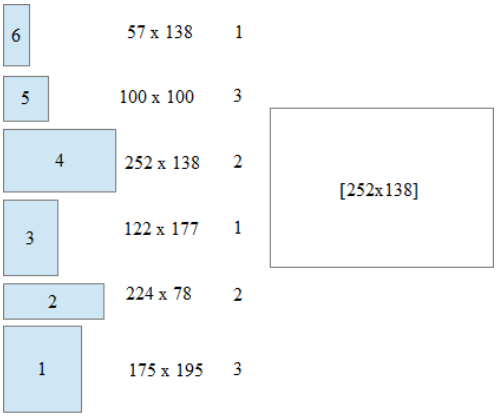


FIGURA 4.2 *Instancia de ejemplo problema*

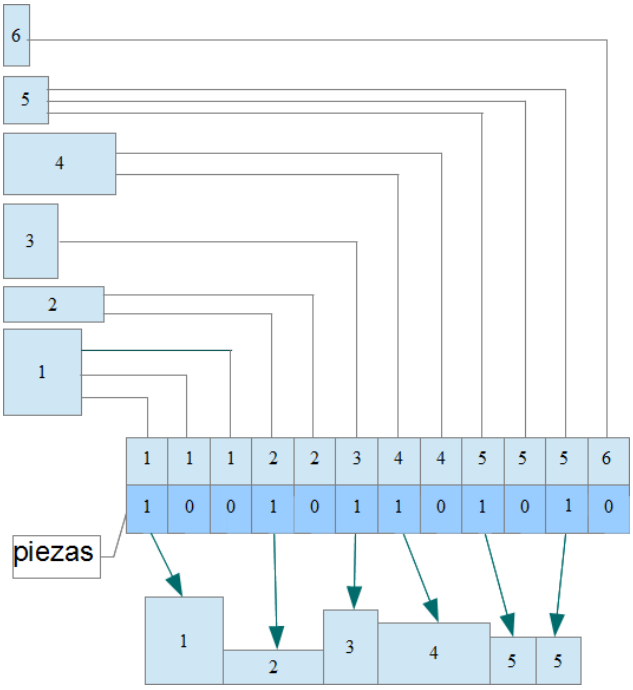


FIGURA 4.3 *Genes de presencia e interpretación*

Los genes de ordenamiento determinan el orden de las piezas mediante una lectura de los genes de presencia. La idea es que durante la lectura, cada vez que se encuentre una pieza marcada como presente (*gen=1*), esta sea agregada a la secuencia ordenada. Para que la lectura genere diversidad de ordenamientos se definen dos parámetros: sentido y salto. El sentido establece si la lectura se realiza desde la primera posición hasta la última, o viceversa. El salto establece cada cuántas posiciones debe realizarse una lectura. Para el sentido solo se necesita 1 bit ya que tiene dos valores posibles: 0 para un sentido del inicio al final, y 1 para un sentido opuesto. Para el salto, si el largo del cromosoma de piezas es *n*, se necesitan *ld(n)* bits para representar todos los saltos posibles que se pueden generar.

El cromosoma completo es el que se muestra en la FIGURA 4.4. El largo total para codificar una solución es: *n+ld(n)+1*, donde *n* corresponde a la cantidad total de piezas, *ld(n)* al salto en la lectura de las piezas, y *1* al sentido de la lectura, la cual se realiza mediante saltos sucesivos en forma cíclica hasta haber visitado todas las piezas. Es por esto que el cromosoma se interpreta conceptualmente en una disposición toroidal, obteniéndose una ruleta de piezas como la que se muestra en la FIGURA 4.5, en donde la

ruleta interna contiene el arreglo con las distintas copias de cada pieza, y la ruleta externa contiene los genes de presencia. A diferencia del resto de los genes, los cuales tienen una interpretación directa, para obtener el valor del salto se hace una conversión de binario a entero[4]. La interpretación completa del genotipo se describe mediante el cromosoma del ejemplo: Las primeras copias de las piezas de tipo 1, 2, 3, 4, y la primera y tercera copia de la pieza de tipo 5 están presentes en la secuencia. La lectura de la ruleta de piezas debe ser en el sentido de las manecillas del reloj y en saltos sucesivos de 5 posiciones.

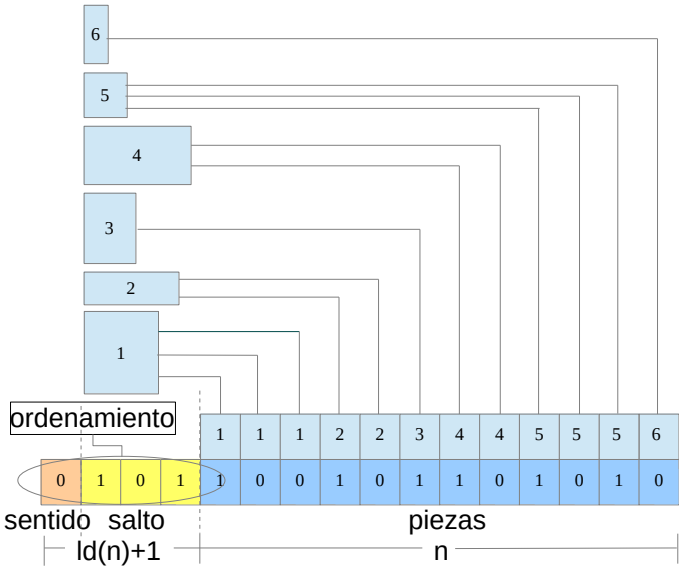


FIGURA 4.4 *Cromosoma completo y su composición*

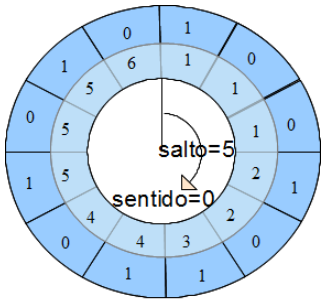


FIGURA 4.5 *Interpretación conceptual del cromosoma como "Ruleta de piezas"*

4.1.3 Función Constructora

Para decodificar el string binario en una secuencia de piezas, lo primero que realiza la función constructora es descomponer el cromosoma de acuerdo a los desplazamientos asociados a cada parte; el sentido corresponde a la 1a posición, el salto, de la posición *1* hasta *1+ld(n),* y las piezas desde la posición *1+ld(n)* hasta *1+ld(n)+n*. De esta manera se obtienen los genes de piezas y los parámetros necesarios para a continuación proceder con su lectura.

El ALGORITMO 3.2 muestra el seudocódigo de la función constructora, que recibe como parámetro el genotipo, además existe una estructura global, "piezas", que

[4] De acuerdo a A. E Eiben 2003 p. 40-41, debido a la significación numérica de bits en una codificación binaria, una codificación de Gray puede ser más apropiada para representar el salto.

corresponde a un arreglo que almacena las distintas piezas del problema. Las líneas 1-6 corresponden a la obtención de los distintos parámetros e inicialización de variables para realizar la lectura. Las líneas 7-20 corresponden a los ciclos de lectura, al final de cada ciclo interno se verifica si se ha cumplido un ciclo en la sucesión de saltos, de ser así, se desliza el desplazamiento en una posición. Este mecanismo permite visitar las piezas que no han sido visitadas hasta la sucesión de saltos actual.

ALGORITMO 3.2 Pseudo-código de la función constructora

```
1.  crom_piezas ← copiarRango(genotipo,ld(n)+1,n+ld(n)+1);
2.  sentido ← genotipo[0];
3.  Si(sentido=1) revertir(piezas); Fin-Si
4.  salto ← genotipo.binario_a_decimal(1,ld(n));
5.  desplazamiento ← salto;
6.  origen ← salto;
7.  Mientras(crom_piezas.contenga(1)) hacer
8.      Mientras(desplazamiento<crom_piezas.largo) hacer
9.          Si(crom_piezas[desplazamiento]=1)
10.             fenotipo.agregar(piezas[desplazamiento];
11.             crom_piezas[desplazamiento] ← 0;
12.         Fin-SI
13.         desplazamiento ← desplazamiento + salto;
14.     Fin-Mientras
15.     desplazamiento ← desplazamiento - crom_piezas.largo;
16.     Si(desplazamiento=origen)
17.         desplazamiento ← desplazamiento + 1;
18.         origen ← desplazamiento;
19.     Fin-Si
20. Fin-Mientras
21. devolver fenotipo;
```

A continuación, se incluye la traza para la solución candidata de ejemplo, la cual es ilustrada en la FIGURA 4.6. Inicialmente se realiza la descomposición del cromosoma, extrayendo las piezas, el sentido y salto. Como el sentido es 0 la lectura se hace en el sentido de las manecillas del reloj, la conversión a entero del salto corresponde a 5, así se fija el desplazamiento y el origen en 5. Luego comienza el ciclo de lectura. Un color amarillo simboliza qué pieza es visitada en el ciclo de lectura actual, mientras que un color rojo indica que la pieza ya ha sido previamente visitada.

El desplazamiento inicial corresponde a la posición 5. Esta posición, en el arreglo "piezas" corresponde a la pieza 3, por lo que la primera pieza leída es la 3, cuyo gen de presencia es igual a 1 (presente), por lo tanto se agrega a la secuencia ordenada.

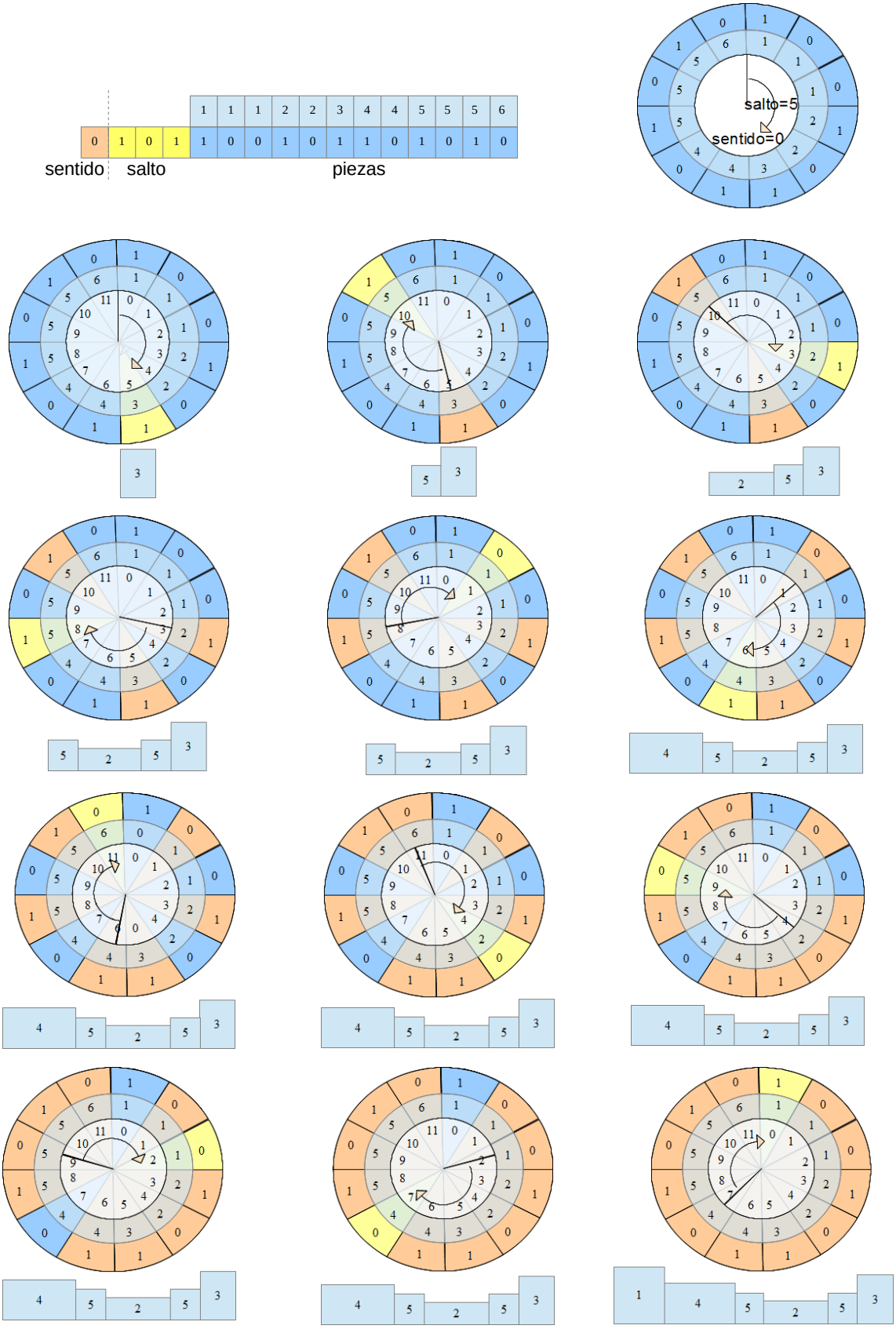


FIGURA 4.6 *Traza función constructora para la solución candidata de ejemplo*

Luego, el desplazamiento se incrementa en un salto (5+5=10), este desplazamiento corresponde a la pieza 5, la cual está presente, por lo tanto se agrega a la secuencia. Al incrementar el desplazamiento (10+5=15), se sobrepasa el largo del cromosoma de piezas (12), por lo que es normalizado (15-12=3), este desplazamiento corresponde a la pieza 2, la cual está presente, por lo tanto se agrega a la secuencia, y así sucesivamente hasta haber visitado todas las piezas. En este caso, al cumplirse un ciclo en la sucesión de saltos, todas las piezas ya han sido visitadas, por lo que el ciclo de lectura termina y no es necesario deslizar el desplazamiento.

4.1.4. Heurística de colocación

Una vez obtenida la secuencia de piezas, la heurística de colocación realiza la traducción a un patrón guillotinable. La heurística utilizada está basada en la *Combinación Heurística VH* (Romero, 2003), la cual emplea un enfoque *bottom-up*, donde cualquier patrón que satisfaga la condición de guillotina puede ser obtenido mediante construcción horizontal o vertical de rectángulos. Una combinación vertical consiste en ubicar una pieza arriba de un patrón existente, creando un nuevo patrón de corte, tal como muestra la FIGURA 4.7. Una combinación horizontal consiste en ubicar una pieza a la derecha de un patrón existente, creando un nuevo patrón de corte, tal como muestra la FIGURA 4.8.

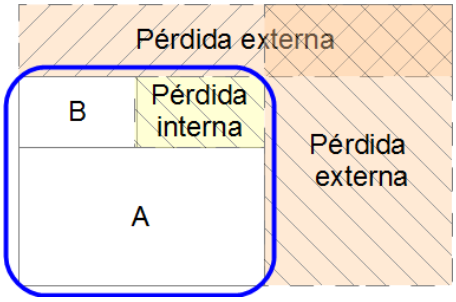


FIGURA 4.7 *Combinación vertical de las piezas A y B*

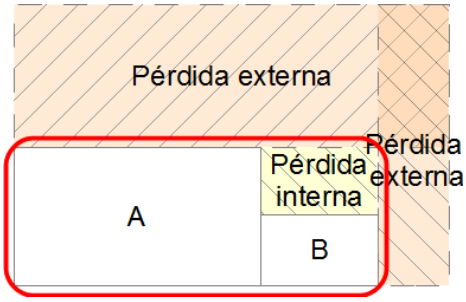


FIGURA 4.8 *Combinación horizontal de las piezas A y B*

Como se observa en las figuras, cada combinación tiene asociada una pérdida interna, correspondiente al área rectangular que contiene las piezas y que queda inutilizada. A su vez, el patrón resultante también tiene asociada una pérdida externa, correspondiente al área de la placa que queda inutilizada. En este trabajo, se extiende el

concepto de combinación vertical-horizontal, de manera que las pérdidas internas son pérdidas, también verticales u horizontales, asociadas a una pieza en particular. Del mismo modo, las pérdidas externas son pérdidas verticales u horizontales asociadas al patrón de corte. Bajo este concepto, en la FIGURA 4.7 la pérdida interna es una pérdida horizontal asociada a la pieza B, la cual no tiene pérdida vertical asociada. De manera similar, en la FIGURA 4.8 la pérdida interna es una pérdida vertical asociada a la pieza B, la cual no tiene pérdida horizontal asociada.

Como un intento de mejora, una modificación que se hizo a la heurística, fue evitar dividir el área inutilizada como resultado de una inserción, donde en el caso general, se producen dos pérdidas secantes, como se muestra en las FIGURA 4.9. La idea es mantener ambas pérdidas considerando el área de intersección, hasta que una de ellas sea utilizada. Solo entonces, si procede, se reduce el área de la otra pérdida, como muestra la FIGURA 4.10. Con esto se pretende eliminar la arbitrariedad de dividir el área inutilizada vertical u horizontalmente, promoviendo la reducción de cualquier fuente de sesgo en la búsqueda genética

FIGURA 4.9 *Pérdidas secantes asociadas a la pieza A*

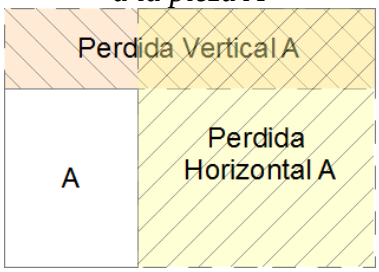
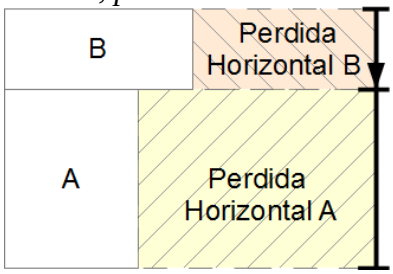


FIGURA 4.10 *Reducción de pérdida horizontal A, producto de una inserción*



El ALGORITMO 3.3 muestra el seudocódigo de la heurística de colocación, que recibe como parámetro el fenotipo, (dado por una secuencia de piezas), y retorna un patrón guillotinable. El patrón es una estructura global que almacena las piezas que han sido colocadas en la placa. Adicionalmente, dos estructuras, *"listaPerdidasInternas"* y *"listaPerdidasExternas"*, almacenan las pérdidas asociadas a cada pieza y las pérdidas asociadas al patrón, respectivamente. La heurística recorre la secuencia del inicio hasta el final mediante la variable *cursor*, y para cada pieza evalúa cada uno de los siguientes casos en orden:

- Si la placa está vacía, intenta ubicar la pieza actual en el origen. Como esta acción genera pérdida externa se crean las pérdida externas.
- Si la placa no está vacía:
    - busca (en *listaPerdidasInternas*) una pérdida interna donde quepa la pieza. Esta búsqueda puede ser de acuerdo a algún criterio. En este trabajo se ha usado el criterio first-fit[5]. Luego se actualizan las pérdidas internas.
    - Si no es posible insertar la pieza en alguna pérdida interna, se busca (en *listaPerdidasExternas*) una pérdida externa donde quepa la pieza. Luego se actualizan las pérdidas externas.
    - Si no es posible insertar la pieza en alguna pérdida externa, se descarta y se lee la pieza siguiente en la secuencia.

ALGORITMO 3.3 Pseudo-código de la heurística de colocación

```
1.  cursor ← 0
2.  Mientras(cursor < fenotipo.size()) hacer
3.      pieza ← fenotipo[cursor];
4.      Si(placa.vacia)
5.          Si(pieza.cabe(placa))
6.              insertarPieza(patrón,pieza,placa);
7.              actualizarPerdidasExternas();
8.          fin-si
9.      Si-no
10.         perdidaInterna ← firstFit(pieza,listaPerdidasInternas);
11.         Si(perdidaInterna != null)
12.             insertarPieza(patrón,pieza,perdidaInterna);
13.             actualizarPerdidasInternas();
14.         Si-no
15.             perdidaExterna ← firstFit(pieza,listaPerdidasExternas)
16.             Si(perdidaExterna != null)
17.                 insertarPieza(patrón,pieza,perdidaExterna);
18.                 actualizarPerdidasExternas();
19.             Fin-Si
20.         Fin-Si
21.     Fin-Si
22.     cursor ← cursor + 1;
23. Fin-Mientras
24. devolver patrón;
```

[5] Retorna la primera pérdida donde quepa la pieza. Nótese que dependiendo del resultado de la búsqueda (pérdida vertical u horizontal), la inserción de la pieza producirá una combinación vertical u horizontal respectivamente.

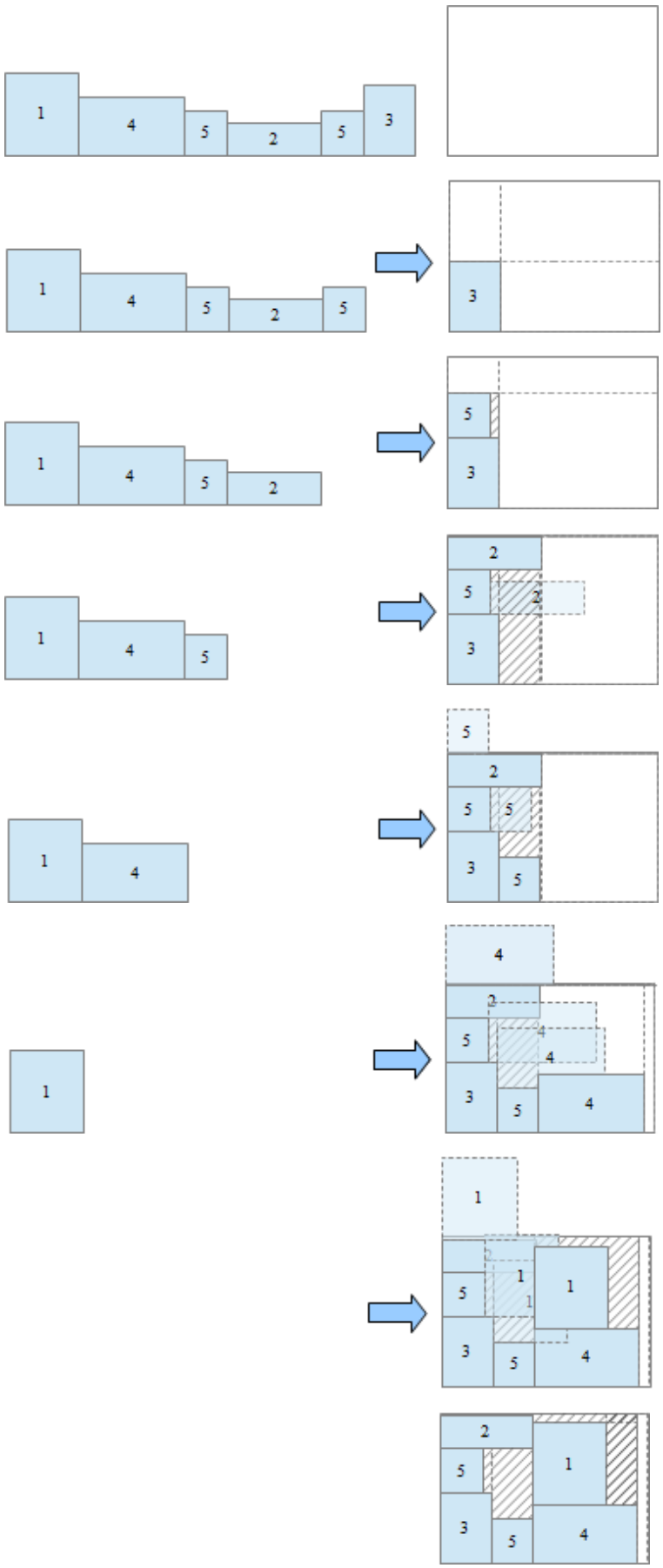


FIGURA 4.11 *Traza heurística colocación para la solución candidata de ejemplo*

Para explicar su funcionamiento, se incluye la traza para la solución candidata de ejemplo, la cual es ilustrada en la FIGURA 4.11. Inicialmente, como la placa está vacía, se inserta la pieza 3 en el origen, produciendo una pérdida externa vertical y horizontal asociadas al patrón de corte. La siguiente pieza en la secuencia es la 5, como la placa no está vacía y no existen pérdidas internas, se buscan pérdidas externas capaces de contener a la pieza 5, la cual es insertada en la pérdida externa vertical, generando una pérdida interna horizontal asociada a ella, además se reduce la pérdida externa vertical existente. La siguiente pieza en la secuencia es la 2, la cual no cabe en ninguna pérdida interna, por lo que es ubicada en la pérdida externa vertical, generando una pérdida interna horizontal asociada a la pieza 3, y una pérdida interna vertical asociada a la pieza 2. Se actualizan las pérdidas externas. La siguiente pieza en la secuencia es la 5, la cual no cabe ni en la pérdida horizontal asociada a la pieza 5 ni en la pérdida vertical asociada a la pieza 2, sin embargo cabe en la pérdida horizontal asociada a la pieza 3. Se actualizan las pérdidas internas; se elimina la pérdida utilizada y se crea una nueva pérdida vertical, asociada a la pieza insertada, y así sucesivamente hasta haber leído todas las piezas.

4.1.5 Función objetivo

Finalmente, el patrón guillotinable es evaluado por la función objetivo para obtener el *fitness* de la solución candidata, el cual representa su competitividad para el problema en cuestión. Como el objetivo es maximizar el área total utilizada en la placa, el *fitness* está dado por la suma de las áreas de las piezas presentes en el patrón.

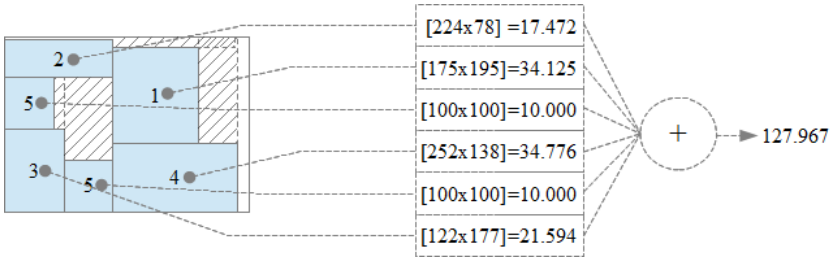


FIGURA 4.12 *Función objetivo y fitness de la solución candidata*

4.1.6 Efectos de los operadores genéticos sobre la representación adoptada

Una cuestión esencial en el desempeño general de un *AG* es si efectivamente, buenos padres son capaces de producir hijos de *fitness* comparable e incluso mejor. En este sentido, el objetivo de un *AG* es combinar la información genética esencial, la cual se encuentra inicialmente esparcida en muchos individuos, y debe ser reunida en un solo cromosoma en las etapas finales del proceso evolutivo (Affenzeller, 2004). Lo anterior lleva a formularse, entre otras preguntas: ¿Cuál de los operadores de cruzamiento disponibles es el más apropiado para cierto problema en una cierta representación?. La respuesta a esta pregunta queda relegada al usuario, quien debe decidir cuidadosamente un operador apropiado. En esta sección se analizan algunos efectos de los operadores descritos previamente, teniendo como antecedentes tanto el problema a resolver como la representación utilizada.

Un problema que surge al utilizar un enfoque basado en decodificadores de secuencias, donde se separa el procesamiento genético del dominio del problema, es la dificultad en la transmisión de información desde un dominio a otro. Un hecho que surge de esta dificultad, es que generalmente, solo los primeros genes del cromosoma tienden a ser los relevantes, ya que representan las piezas que forman parte del patrón de corte, y que son las que pudieron ser colocadas exitosamente por el decodificador. Así, el resto de los genes no aportan información para el problema y son irrelevantes para el *fitness* del individuo.

Es posible intuir que el uso de un operador que exhiba sesgo posicional, no es apropiado en este escenario, ya que las propiedades de los individuos están concentradas en los primeros genes, y el ensamble de información genética relevante se vuelve un proceso errático e incluso puede llegar a anularse. Considérese el caso extremo del cruzamiento de 1-punto. La FIGURA 4.13 muestra el apareamiento de dos individuos, cuyos genes relevantes están destacados (azul para el 1er padre y rojo para el 2o padre). Como se puede observar, al efectuar el cruzamiento, solo información irrelevante es intercambiada, dando como resultado hijos, que desde el punto de vista del problema, son idénticos a los padres. En este escenario, la función explorativa del cruzamiento queda anulada, y la evolución queda relegada únicamente a la mutación.

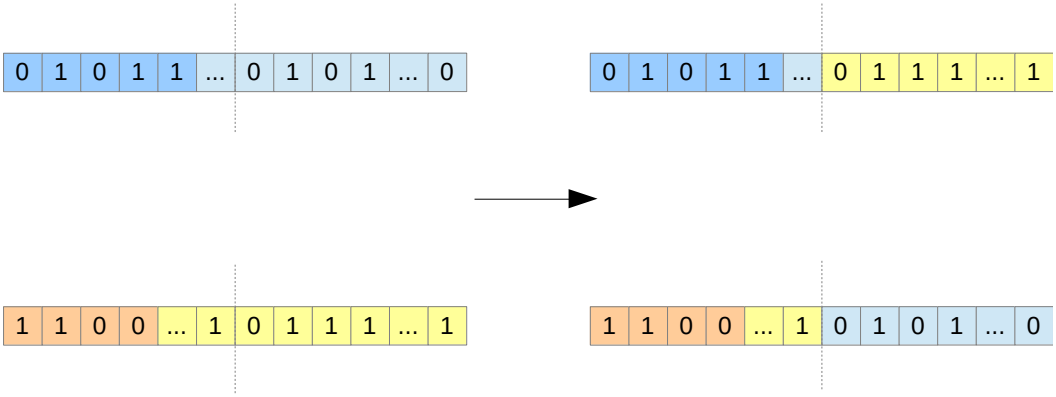


FIGURA 4.13 *Cruzamiento de 1-punto sobre una representación basada en orden*

Una situación diferente ocurre al considerar un cruzamiento uniforme, en el cual no existe sesgo posicional. En este escenario, la tendencia a transmitir el 50% de información de cada padre, es más apropiada para la función explorativa del cruzamiento, debido a que existe una alta probabilidad de que los hijos efectivamente hereden información genética relevante, proveniente de ambos padres, como se muestra en la FIGURA 4.14. Aún cuando el cruzamiento uniforme permite la herencia de genes relevantes, una desventaja de su uso, es que existe una alta probabilidad de disrupción de secuencias coadaptadas al problema.

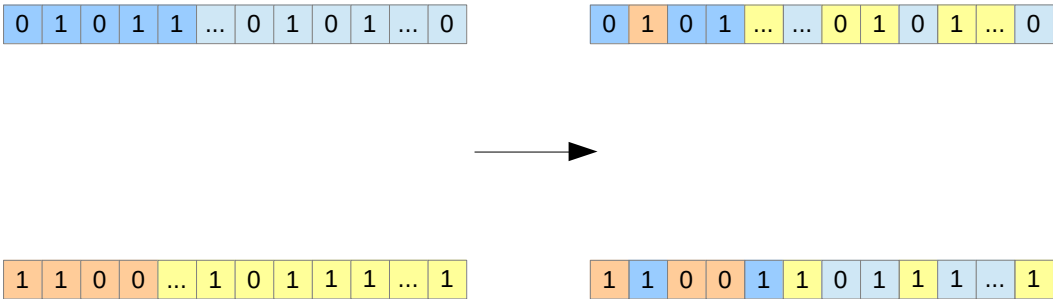


FIGURA 4.14 *Cruzamiento uniforme sobre una representación basada en orden*

A continuación se describen algunas dificultades encontradas al utilizar la representación propuesta, dada su estructura relacional, donde ciertos genes dependen de otros, los cuales determinan cómo los primeros deben ser procesados. Aún cuando estas dificultades no invalidan el uso de la representación adoptada, se sospecha que introducen efectos indeseados en la evolución, los cuales podrían ser evitados mediante el uso de otro tipo de representación.

Una dificultad que aparece debido a la dependencia entre genes es que, si hay buenos genes, correspondientes a segmentos de secuencias que se traducen en buenos patrones, estos están coadaptados con otros genes, los cuales determinan el orden en que los primeros deben ser leídos. De esta manera, buenas características en un padre no se traducen en buenas características en los hijos. Para ilustrar esta situación, obsérvese la FIGURA 4.15, en donde genes coadaptados a los genes de ordenamiento respectivos, son intercambiados en el cruzamiento. Como los genes de ordenamiento después de la cruza, representan una configuración distinta a la que tenían antes, los segmentos intercambiados representan secuencias distintas a las de origen, y así, gran parte de la información genética relevante se pierde.

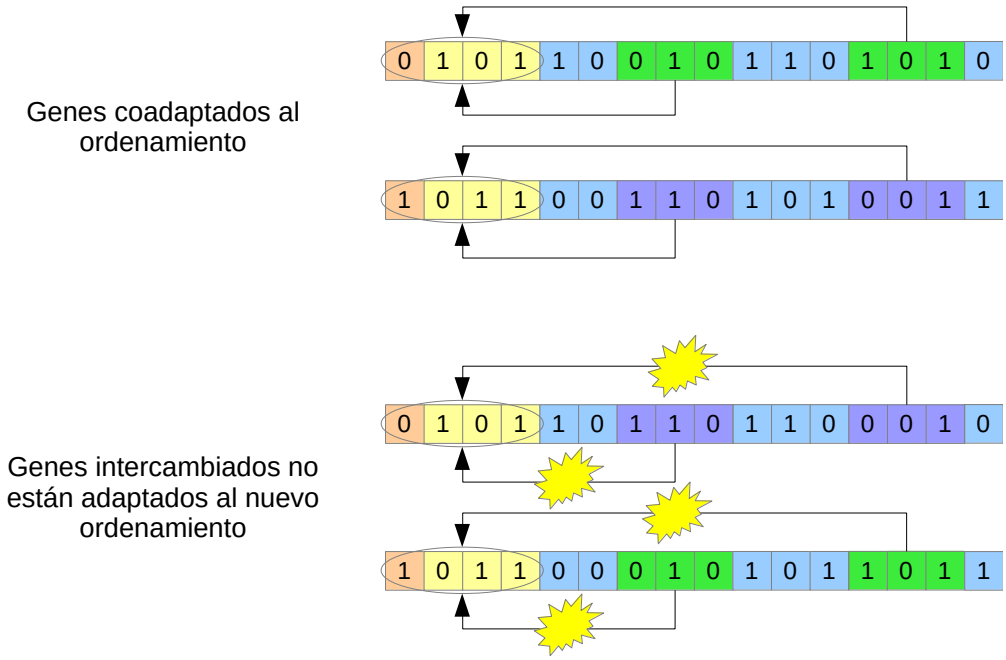


FIGURA 4.15 *Efecto del cruzamiento en representación adoptada*

Otro problema proviene de mantener explícitamente información acerca del ordenamiento de los elementos. Ya que pequeños cambios en el genotipo, se pueden traducir en grandes cambios en el fenotipo. Considérese una situación de mutación, en donde un solo gen de un individuo es mutado. Si este gen corresponde a un gen de ordenamiento, modifica el procesamiento completo del resto de los genes correspondientes a las piezas. Para ilustrar esta situación considérese la FIGURA 4.16, en donde un gen de salto ha mutado, convirtiendo un salto equivalente de 5 posiciones, a uno de 1 posición. Se observa que este pequeño cambio impacta considerablemente en

el patrón resultante, obteniéndose en este caso, un patrón muy distinto. Lo anterior claramente tiene efectos indeseados en la función explotativa de la mutación, donde pequeñas variaciones deben ser introducidas sobre soluciones promisorias.

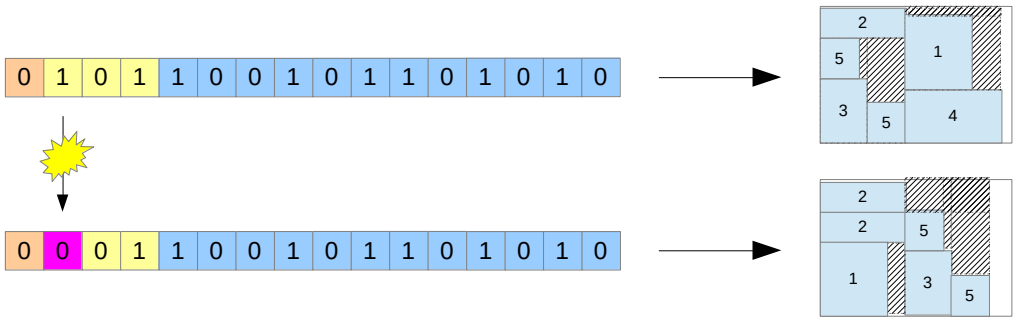


FIGURA 4.16 *Efecto de la mutación en un gen de ordenamiento para la representación adoptada*

4.2 Datos de prueba

Para evaluar el desempeño de los algoritmos propuestos, se han seleccionado varios conjuntos de instancias, ampliamente usados en la literatura. De acuerdo a lo discutido en la sección 1.2, las instancias corresponden al *TDC* o *2D-KP* con las restricciones de guillotina y rotación, cuyos valores óptimos son conocidos. Solo se consideran aquellas instancias correspondientes a la variante restricta no ponderada, la cual ha sido abordada en este trabajo. A continuación se describe brevemente cada uno de los conjuntos, señalando las publicaciones desde donde han sido extraídas. Todas ellas, fueron obtenidas del repositorio:

ftp://cermsem.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting

Para efectos de clasificación, las instancias, en total 49, han sido divididas en tres grupos, basándose en la agrupación propuesta por (Yoon et al., 2013).

Un primer grupo comprende varios conjuntos de instancias pequeñas y algunas de tamaño medio, la mayoría hasta n=20 y algunas hasta n=35 tipos de pieza. Estas corresponden a: *W*, *Wang1*, *Wang2* y *Wang3* (Wang, 1983), *OF1* y *OF2* (Oliveira y Ferreira, 1990), *STS2s* y *STS4s* (Tschoke y Holthofer, 1995), *2s*, *3s*, *A1s*, *A2s*, *A3*, *A4* y *A5* (Hifi, 1997), *CHL1s*-*CHL4s*, *CHL5*, *CHL6* y *CHL7* (Cung et al., 2000). Además, se incluyen once instancias de tamaño medio, generadas aleatoriamente, con n=25 hasta

n=50 tipos de pieza. Estas corresponden a las instancias *CU1*-*CU11* (Álvarez-Valdés et al., 2002). La TABLA 4.1 muestra un resumen para cada instancia, el cual incluye las dimensiones de la placa, el número de tipos de pieza, el valor óptimo y la pérdida asociada al patrón.

TABLA 4.1 *Primer grupo, correspondiente a instancias de tamaño pequeño y medio*

Instancia	*W*	*L*	*n*	*Óptimo*	*Pérdida*
W	70	40	20	2721	79
wang1	33	69	20	2277	0
wang2	39	70	20	2694	36
wang3	40	70	20	2721	79
OF1	70	40	10	2737	63
OF2	70	40	10	2690	110
STS2s	55	85	30	4653	22
STS4s	99	99	20	9770	31
2s	40	70	10	2778	22
3s	40	70	20	2721	79
A1s	50	60	20	2950	50
A2s	60	60	20	3535	65
A3	70	80	20	5451	149
A4	90	70	20	6179	121
A5	132	100	20	12985	215
CHL1s	132	100	30	13099	101
CHL2s	62	55	10	3279	131
CHL3s	157	121	15	7402	11595
CHL4s	207	231	15	13932	33885
CHL5	20	20	10	390	10
CHL6	130	130	30	16869	31
CHL7	130	130	35	16881	19
CU1	100	125	25	12330	170
CU2	150	175	35	26100	150
CU3	134	125	45	16679	71
CU4	285	354	45	99366	1524
CU5	456	385	50	173364	2196
CU6	356	447	45	158572	560
CU7	563	458	25	247150	10704
CU8	587	756	35	432714	11058
CU9	856	785	25	657055	14905
CU10	794	985	40	773772	8318
CU11	977	953	50	924696	6385

Un segundo grupo comprende seis instancias difíciles generadas aleatoriamente (TABLA 4.2), con *n*=10 hasta *n*=40. Estas corresponden a las instancias *Hchl3s-Hchl8s* (Cung et al., 2000).

TABLA 4.2 *Segundo grupo, correspondiente a instancias difíciles*

Instancia	*W*	*L*	*n*	*Óptimo*	*Pérdida*
Hchl3s	127	98	10	12215	231
Hchl4s	127	98	10	12006	452
Hchl5s	205	223	25	45410	354
Hchl6s	253	244	22	61040	692
Hchl7s	263	241	40	63112	271
Hchl8s	49	20	10	911	69

Por último, un tercer grupo comprende diez instancias de larga escala generadas aleatoriamente (TABLA 4.3), con *n*=27 hasta *n*=56. Estas corresponden a las instancias *ATP30-ATP39* (Álvarez-Valdés et al., 2002)

TABLA 4.3 *Tercer grupo, correspondiente a instancias de larga escala*

Instancia	*W*	*L*	*n*	*Óptimo*	*Pérdida*
ATP30	927	152	38	140904	0
ATP31	856	964	51	823976	1208
ATP32	307	124	56	38068	0
ATP33	241	983	44	236611	292
ATP34	795	456	27	361398	2436
ATP35	960	649	29	621021	2243
ATP36	537	244	28	130744	284
ATP37	440	881	43	387276	364
ATP38	731	358	40	261395	544
ATP39	538	501	33	268750	788

4.3 Configuración de algoritmos

En esta sección se describe la metodología utilizada para la determinación de los parámetros de los algoritmos, los cuales son comparados en la fase experimental.

Si bien, un conocimiento acabado del problema y la representación utilizada son valiosos para la comprensión y el ajuste de ciertos aspectos de los algoritmos a evaluar, el teorema del *no free lunch* establece que no existe una configuración que entregue un mejor funcionamiento del algoritmo para todos los problemas e instancias existentes.

Debido a que existe un gran número de combinaciones posibles, el problema de encontrar la mejor configuración de parámetros para un escenario particular, constituye por sí solo un problema de optimización, por lo tanto, su automatización es de gran importancia práctica en varios contextos (Hutter et al., 2009). La FIGURA 4.17 ilustra el problema de configuración de parámetros. Un escenario de configuración incluye un algoritmo y una colección de instancias del problema. Un configurador ejecuta el algoritmo objetivo con un determinado ajuste de parámetros sobre alguna o todas las instancias, recibe información sobre el rendimiento de estas ejecuciones, y utiliza esta información para decidir las siguientes configuraciones de parámetros evaluar.

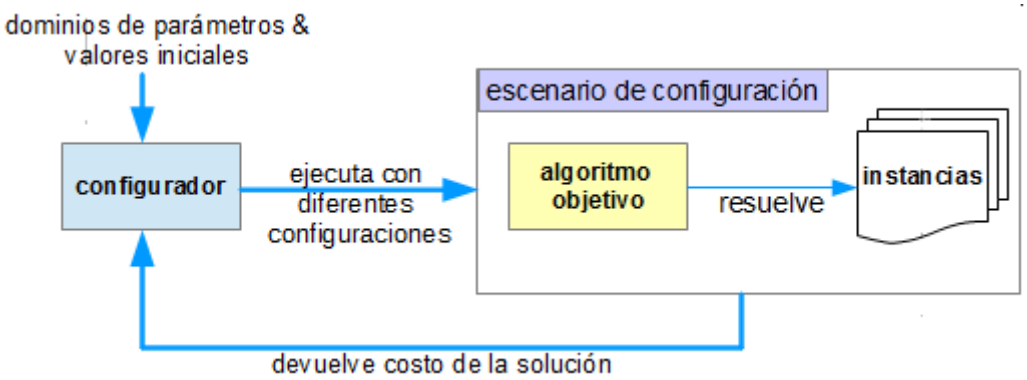


FIGURA 4.17 *Problema de configuración de algoritmos*

4.3.1 El configurador

ParamILS es un framework para la optimización de parámetros, el cual utiliza la metaheurística basada en trayectoria, Iterated Local Search (Hutter et al., 2009). Funciona para cualquier algoritmo cuyos parámetros puedan ser discretizados. *ParamILS* busca a través del espacio de posibles configuraciones de parámetros, evaluando configuraciones mediante la ejecución del algoritmo objetivo en un conjunto de instancias de prueba. Para ello se debe definir:

- Un algoritmo ejecutable *A* (algoritmo objetivo)
- Todos los parámetros y sus valores posibles
- Un conjunto de instancias de prueba *S*

Existen distintas maneras de medir el desempeño del algoritmo objetivo. Por ejemplo, puede interesar minimizar el costo de los recursos computacionales

consumidos (como tiempo de ejecución, memoria, ancho de banda), o maximizar la calidad de la solución encontrada. Para tal efecto, *ParamILS* permite elegir de una variedad de objetivos de optimización, desde minimizar el tiempo medio de ejecución hasta maximizar la mediana de las calidades de solución. Así, *ParamILS* ejecuta el algoritmo *A* sobre una muestra de instancias de *S*, en busca de la configuración que obtenga el mejor rendimiento general.

4.3.2 Plan de configuración y diseño de escenarios de sintonización

Dos algoritmos objetivo, correspondientes a cada *AG* a evaluar, deben ser configurados para optimizar su desempeño sobre el conjunto de instancias seleccionado. Al mismo tiempo, se desea realizar comparaciones entre los desempeños de cada uno de ellos. Para mantener la consistencia en dichas comparaciones, se define un plan de configuración, el cual prioriza el algoritmo con una menor cantidad de parámetros, comunes para ambos. De este modo, una configuración posterior asume el ajuste de los parámetros previamente obtenidos, y solo requiere la obtención de parámetros adicionales y específicos para el posterior algoritmo.

La TABLA 4.4 muestra los valores posibles que se consideran para cada parámetro, un ticket indica que el parámetro es obtenido en una configuración previa y un guión indica que el parámetro no aplica para ese algoritmo. Dado que *AG* posee parámetros comunes, este es el primero en ser configurado, de cuyo resultado se obtiene parte de los parámetros para *AGc*. En particular, una previa obtención del tamaño de población permite fijar el área de la grilla bidimensional[6], de manera que los valores posibles para este parámetro son sus dimensiones. Además, se añade la cardinalidad de los espacios de configuración, esto permite estimar los tiempos de espera para configurar cada algoritmo.

ParamILS no restringe los valores de los parámetros a dominios únicamente numéricos, por el contrario, asume que todos los parámetros son variables categóricas. Gracias a esta característica, se incluyen en la configuración, parámetros como los distintos operadores genéticos, y parámetros específicos de *AGc*.

[6] Estructura geométrica típica en algoritmos genéticos celulares

TABLA 4.4 *Valores posibles para cada parámetro de cada algoritmo objetivo*

Parámetro	*AG*	*AGc*
tamaño población	50; 100; 150; 200; 250; 300; 350; 400; 450; 500	–
probabilidad cruzamiento	0,6; 0,7; 0,8; 0,9; 1,0	✔
probabilidad mutación	0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9; 1,0	✔
operador selección 1	torneo-binario; ruleta; ranking-lineal	✔
operador selección 2	torneo-binario; ruleta; ranking-lineal	✔
operador cruzamiento	1-punto; 2-puntos; uniforme; probabilístico	✔
operador mutación	por-bit	✔
Grilla	–	20x20; 10x40; 5x80; 4x100; 2x200; 1x400
Vecindad	–	Linear5; Compact9; Compact13
política actualización	–	ls; frs; nrs; uc; ss; synchronous
# total configuraciones	4500	108

Toda la información necesaria para realizar la configuración se especifica en un archivo, conceptualmente llamado *escenario de sintonización*. A continuación se especifican los principales parámetros de un *escenario de sintonización*, y cómo fueron definidos para la configuración de los algoritmos objetivo.

- **"algo"**: Un algoritmo ejecutable o una llamada a un script de envoltura en torno al algoritmo, que cumpla con el formato I/O de ParamILS. Para este trabajo se desarrolló un script de envoltura, el cual entre otras tareas, realiza la normalización de la mejor solución encontrada por el algoritmo respecto al valor óptimo de la instancia respectiva.
- **"run_obj"**: Un escalar que cuantifique cuán buena es una ejecución del algoritmo. Algunos ejemplos son: tiempo de ejecución, aproximación de la solución, etc. Para efectos de este estudio, se prioriza optimizar la calidad de la solución, luego se usó una expresión que mide la aproximación de la mejor solución al óptimo, qual=óptimo/best especificando para cada instancia su valor óptimo correspondiente.

- **"overall_obj"**: Una medida de resumen que permita describir la tendencia en la calidad de varias ejecuciones del algoritmo. Algunos ejemplos son: promedio, mediana, q90, etc. En este trabajo se utiliza el promedio.
- **"cutoff_time"**: El tiempo transcurrido tras el cual una ejecución del algoritmo será terminada sin éxito. (Ej: No alcanzar el óptimo). El cutoff_time fue ajustado experimentalmente en 260s, tiempo en el cual el AG-generacional efectúa 150.000 evaluaciones sin encontrar el óptimo.
- **"cutoff_length"**: La longitud tras la cual una ejecución del algoritmo será terminada sin éxito. Esta longitud puede ser definida como la cantidad de decisiones en un árbol de búsqueda o la cantidad de flips en un algoritmo SLS. Para los algoritmos a evaluar se ha definido como la cantidad de evaluaciones de la función objetivo, la cual se ajustó experimentalmente en 150.000 evaluaciones, longitud tras la cual los AG en general no mejoraban la mejor solución encontrada.
- **"tuner_timeout"**: Tiempo de espera del configurador, tras el cual comienza la validación de la mejor configuración de parámetros encontrada. Los tiempos de espera fueron estimados para cada algoritmo en base al "cutoff_time" y a los tamaños de sus espacios de configuración, de manera de alcanzar a realizar como mínimo 10 ejecuciones por instancia, en el caso pesimista de no encontrar el óptimo en ninguna ejecución.
- **"instance_file & test_instance_file"**: ParamILS utiliza un conjunto de instancias de entrenamiento y un conjunto de instancias de prueba, para validar la configuración obtenida en el entrenamiento. Para ello, el conjunto de 49 instancias se particionó en 50% para entrenamiento y el resto para prueba. Para que la distribución fuera uniforme, las instancias fueron repartidas equitativamente en cada conjunto.

La TABLA 4.5 muestra los parámetros correspondientes a la mejor configuración obtenida por *paramILS* para cada algoritmo. Es importante observar cómo estos resultados tienden a verificar los conceptos que subyacen en la hipótesis planteada. En primer lugar, el valor obtenido para el operador de selección es de ruleta, siendo este el operador con una menor presión de selección de los tres considerados (Alba y Dorronsoro, 2008). Esto es consistente con el hecho de que una menor presión de selección promueve la exploración del algoritmo, comportamiento apropiado para espacios de búsqueda complejos, como el problema en estudio.

TABLA 4.5 *Valores obtenidos por ParamILS para cada algoritmo objetivo*

	AG	*AG celular*
tamaño población	400	–
probabilidad cruzamiento	1,0	✔
probabilidad mutación	0,9	✔
operador selección 1	Ruleta	✔
operador selección 2	Ruleta	✔
operador cruzamiento	Uniforme	✔
operador mutación	por-bit	✔
Grilla	–	1x400
Vecindad	–	Compact9
política actualización	–	Uc

Respecto a los parámetros para *AGc*, el valor obtenido para la topología de la población es de *1x400*. Este valor se interpreta como una distribución lineal o en anillo, la cual maximiza la exploración del algoritmo, ya que en esta disposición, los individuos presentan la mayor dispersión posible, minimizando la rapidez de difusión de buenas soluciones. El valor obtenido para la topología de la vecindad es *Compact9*, la que promueve una mayor rapidez de difusión que la topología *NEWS*, debido a una mayor superposición de individuos, aunque más lenta que la topología *Compact13*. Esta rapidez intermedia tiene sentido, si se considera que la difusión debe ser lo suficientemente lenta para prevenir convergencia prematura, pero si es demasiado lenta, buenos individuos no alcanzarían a intensificarse antes de que finalice la evolución. El valor obtenido para la política de actualización es *uc* (uniform choice), que corresponde la política asíncrona que induce la menor presión de selección de las consideradas, lo cual tiene sentido, ya que una menor presión de selección promueve la capacidad de exploración del algoritmo.

4.4 Diseño experimental

Para verificar la hipótesis, es necesario realizar una comparación empírica, evaluando el desempeño mostrado por cada uno de los algoritmos. La evaluación se centra en dos aspectos. Por una parte se analiza el comportamiento numérico, seleccionando para ello, métricas relacionadas al *fitness* obtenido por cada algoritmo. Un segundo aspecto se centra en el comportamiento en la búsqueda realizada, para ello se usan las *curvas de convergencia*. A continuación se describen cada uno de estos elementos de evaluación y otros aspectos considerados para la realización del experimento.

Debido a la naturaleza no determinista de los algoritmos a evaluar, se realiza un número independiente de ejecuciones, usando para cada una de ellas, una semilla generada de manera aleatoria. Mientras mayor sea el número de ejecuciones, mayor robustez tendrán los resultados frente al azar, y la tendencia de estos, estará sustentada en mayor medida por las cualidades del algoritmo, que por componentes aleatorias, como la semilla utilizada. En la literatura se sugiere como mínimo un número de 30 ejecuciones, aunque este número suele llegar a 100 (Alba y Dorronsoro, 2008). Para este trabajo se considera la realización de 30 ejecuciones independientes de cada algoritmo para cada una de las instancias de prueba.

Las métricas usadas para evaluar el comportamiento numérico son *Mean*, *%GAP* y *Best*. *Mean* corresponde al promedio de los *fitness* obtenidos en las 30 ejecuciones (ecuación 4.1). Esta métrica es importante para efectos comparativos, ya que su valor representa la tendencia en la calidad obtenida por el algoritmo para una instancia en particular.

$$Mean = \overline{fitness_i} \quad , i = 1 \dots 30 \tag{4.1}$$

Una comparación transversal a las distintas instancias sugiere la utilización de una métrica normalizada. Por ello, se usa una medida del error del *Mean* respecto al óptimo (o mejor valor conocido) de la instancia. Esta métrica corresponde al error porcentual, denotada como *%GAP* (ecuación 4.2), y es una medida de la precisión alcanzada por el algoritmo; mientras menor sea el *%GAP* mayor es la precisión alcanzada.

$$\%GAP = 100 \cdot \frac{(Optimo - Mean)}{Optimo} \tag{4.2}$$

La tercera métrica, *Best*, corresponde al máximo de los *fitness* obtenidos en las 30 ejecuciones (ecuación 4.3),. Quizá no tan importante en la comparación entre los algoritmos, ya que eventualmente puede no ser representativa del comportamiento en las 30 ejecuciones, sin embargo constituye el mejor exponente al momento de efectuar una comparación con otros enfoques, o para efectos de evaluar la calidad de solución alcanzada, expresada como el patrón de corte correspondiente.

$$Best = Max\{fitness_i\} \quad , i = 1 \dots 30 \tag{4.3}$$

Para obtener resultados concluyentes en la comparación de los algoritmos, se realizan pruebas estadísticas sobre los resultados obtenidos. Considerando como variable dependiente el *%GAP*, por ser esta una métrica transversal a las instancias, y como variable independiente el algoritmo utilizado. La prueba estadística tiene por objeto contrastar una hipótesis acerca de las diferencias encontradas entre los *%GAP* de los algoritmos. La verificación de esta hipótesis permitirá concluir si existen diferencias estadísticamente significativas en la precisión, que pueden ser explicadas por el algoritmo utilizado.

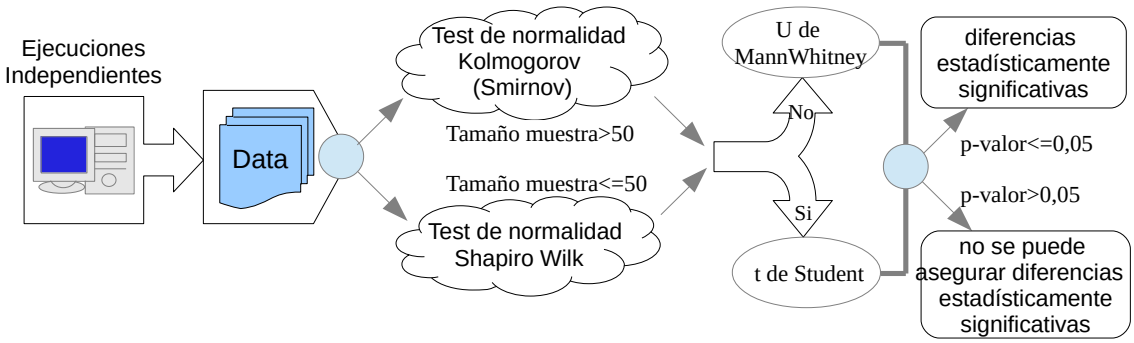


FIGURA 4.18 *Esquema del proceso de evaluación de resultados*

La

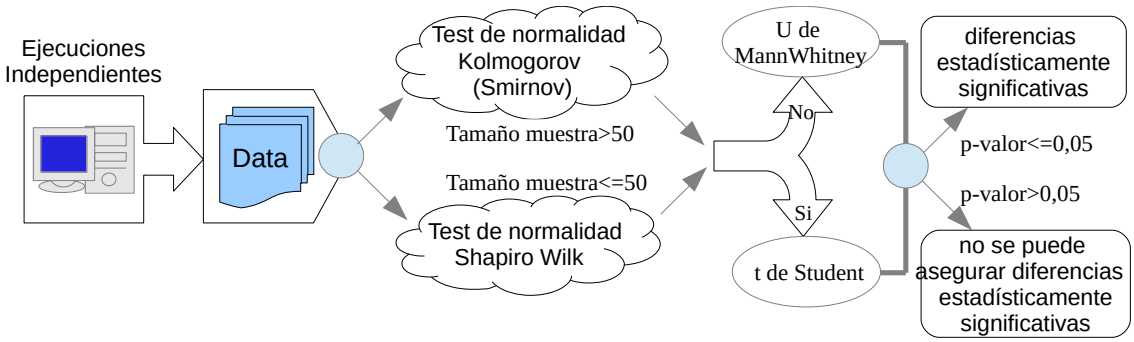


FIGURA 4.18 ilustra la metodología aplicada para realizar la comparación entre los algoritmos. Para realizar la prueba estadística apropiada, previamente se debe determinar si los datos se ajustan a una distribución normal. Dependiendo del tamaño de la muestra, se aplica la prueba de normalidad correspondiente; si la muestra es mayor a 50 observaciones se aplica el test de *kolmogorov-smirnov*, en caso contrario se debe aplicar el test de *shapiro-wilk*. Estas pruebas contrastan la hipótesis de que los datos se ajustan a una distribución normal. Si se acepta dicha hipótesis se debe aplicar la prueba paramétrica *t de Student* para comparar dos muestras independientes. Si por el contrario, se rechaza la hipótesis, se debe aplicar la prueba no paramétrica *U de Mann Whitney*. Se considera un nivel de confianza del 95% (nivel de significación o p-valor bajo 0.05). Esto permite garantizar que las diferencias de los algoritmos comparados son o no significativas con una probabilidad del 95%.

Para evaluar el comportamiento en la búsqueda realizada por los algoritmos, se generan las curvas de convergencia para cada ejecución. Si el *AG* ha sido correctamente implementado, la población evolucionará a lo largo de sucesivas generaciones de forma que la adaptación del mejor y el promedio general tenderán hacia el optimo global. La convergencia es la progresión hacia la uniformidad. Para generar las curvas de

TABLA 4.6 *Datos tabulados para obtener curvas de convergencia*

X	Y
i	$Max\{fitness_j\}\ , j \in Población_i$

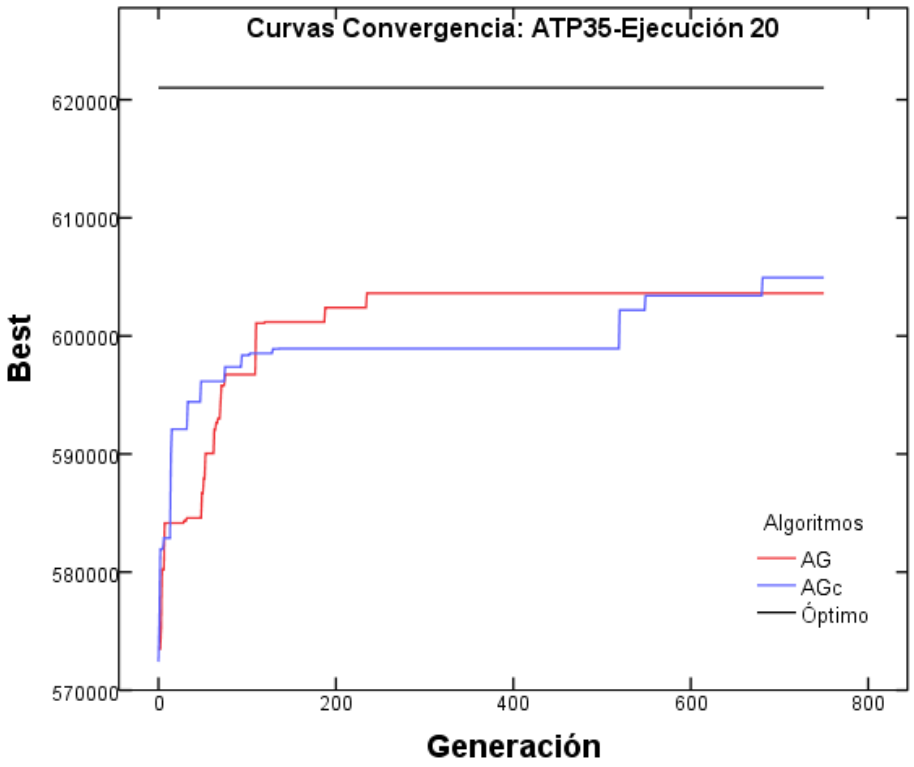
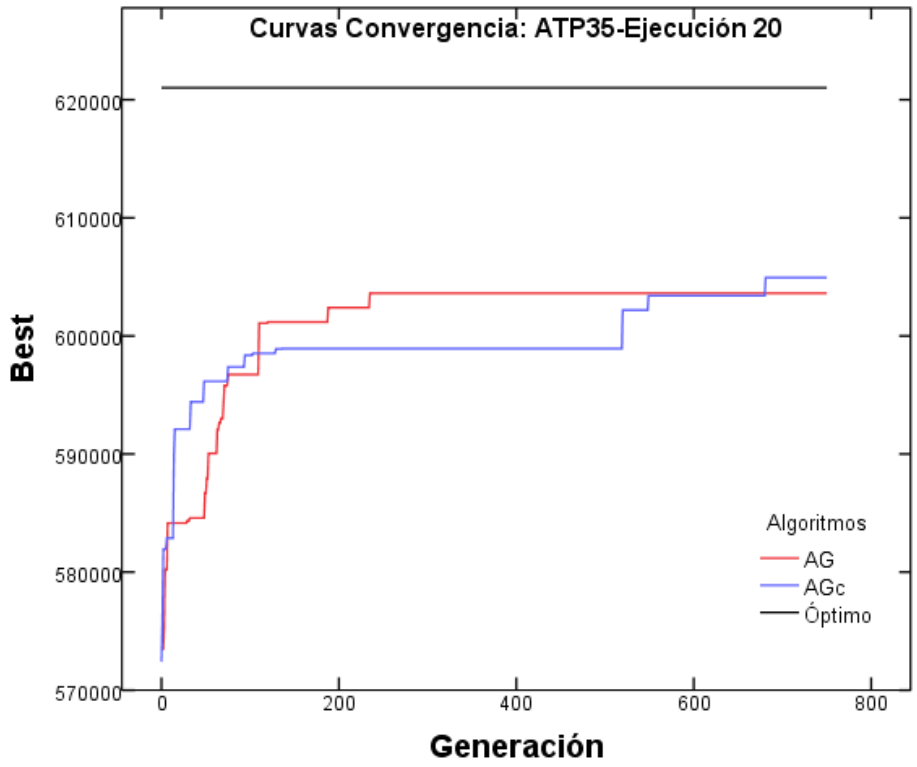


FIGURA 4.19 *Ejemplo de curvas de convergencia de una ejecución*

convergencia se tabula el número de la generación y el *fitness* del mejor individuo en la población para cada generación, como muestra la TABLA 4.6. Luego estos datos son graficados, obteniéndose las curvas de convergencia para cada algoritmo, como muestra



la

FIGURA **4.19**.

La FIGURA 4.20 muestra un esquema general del diseño experimental. Los distintos *AG* son ejecutados 30 veces para cada una de las instancias de prueba. Cada ejecución genera distintas salidas: Para un análisis del comportamiento numérico, se genera el *Best*, *Mean*, y a partir del *Mean* y el óptimo de cada instancia se obtiene el *%GAP*. Estas métricas son tabuladas, mediante el software utilitario *Excel 2007*, en tablas comparativas para llevar a cabo el análisis numérico. Los *%GAP* obtenidos por cada algoritmo son sometidos a una prueba estadística apropiada a su distribución, mediante la suite estadística *SPSS 19*, para obtener resultados concluyentes en la comparación de las precisiones de los algoritmos. Los datos tabulados para generar las curvas de convergencia son procesados para obtener el promedio en las curvas de convergencia en función del *%GAP*. Finalmente estos datos son introducidos en *SPSS 19* para obtener los gráficos con las curvas de convergencia. Por último, los patrones son generados como archivos de texto con un formato determinado, para ser procesados por la herramienta de visualización de patrones *Dibuja* (Flores, 2012), la cual fue personalizada para incluir las etiquetas de las piezas en el patrón resultante.

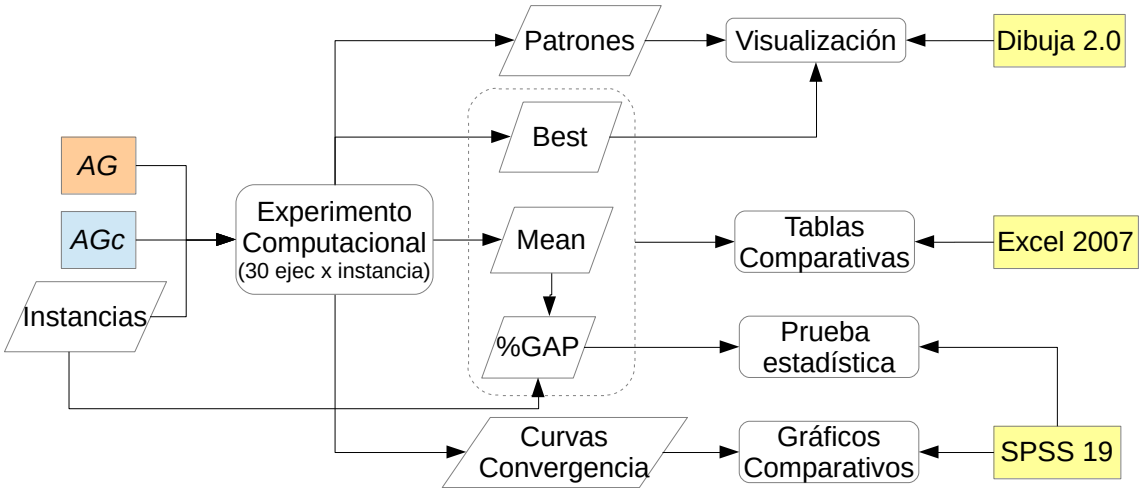


FIGURA 4.20 *Esquema general del diseño experimental*

4.5 Plataforma evolutiva *JCell*

JCell es un framework evolutivo orientado a objetos, desarrollado por el grupo *Neo* . Dentro de él, se encuentran implementados muchos de los conceptos y técnicas típicos de la computación evolutiva. Originalmente contiene distintas familias de *AG*, tanto estructurados como no estructurados. Dentro de los no estructurados se encuentran: *AG* generacional y steady-state. Dentro de los estructurados, *AG* distribuido y celular. Asimismo, distintos tipos de representaciones, así como operadores genéticos, se encuentran de manera nativa dentro de la plataforma, lo cual brinda un significativo ahorro de trabajo para el experimentador. Así, el usuario únicamente debe implementar el problema en cuestión, mediante la definición de una función objetivo apropiada, dentro de una clase que herede de la clase *Problem*.

Usualmente, todos los ajustes del algoritmo se especifican en un archivo de configuración, leído y validado por una instancia de la clase *ReadConf,* la cual es llamada desde el método *main* por defecto, residente en la clase *JCell*. Sin embargo, si se desea personalizar el *I/O* para comunicarse con otros módulos de software se utiliza un *main* distinto, el cual se encuentra en la clase *JCellIsland*. Este fue el caso al realizar la configuración de parámetros, ya que se necesitó comunicar el ejecutable generado por *JCell* con la plataforma de configuración *paramILS.*

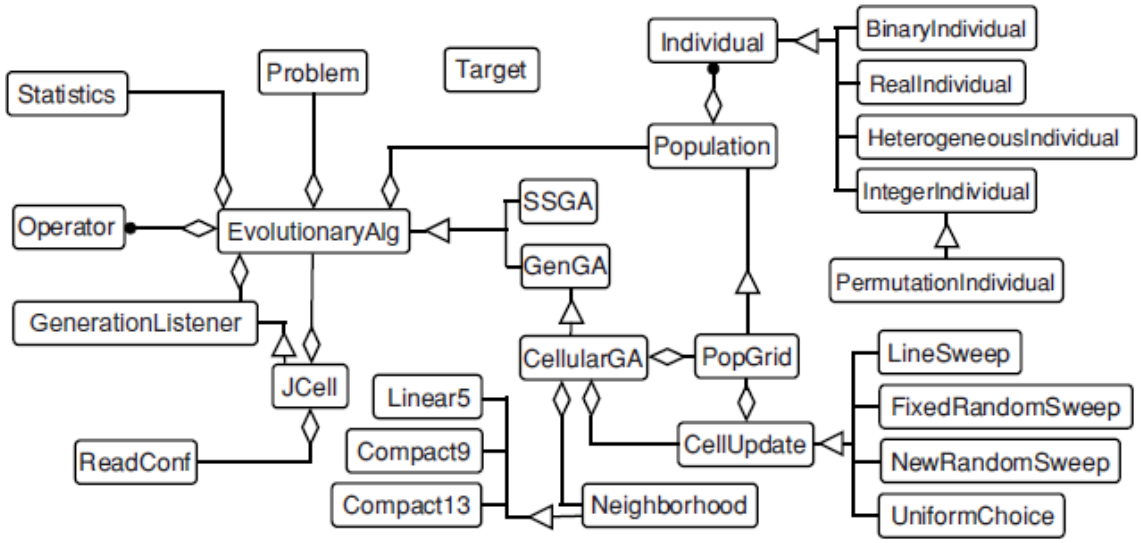


FIGURA 4.21 *Breve descripción de diseño orientado a objetos de JCell*

Una completa revisión de *JCell* puede encontrarse en (Alba y Dorronsoro, 2008). El hardware donde se ejecuta *JCell* contiene un procesador Intel Core i5-3337U 1.8 GHz, 6 Gb RAM y sistema operativo Windows 8 de 64 bits.

Capítulo 5 RESULTADOS COMPUTACIONALES Y ANÁLISIS

De acuerdo con los objetivos del estudio, este capítulo presenta los resultados computacionales obtenidos por un enfoque de estructuración celular, mediante el algoritmo genético celular propuesto, denotado como *AGc*, versus un enfoque no estructurado, correspondiente al algoritmo genético, denotado como *AG*.

Para efectos de análisis y presentación, las 49 instancias fueron divididas, basándose en la agrupación propuesta en (Yoon et al., 2013), en 3 grupos:

- 33 instancias de tamaño pequeño y mediano (TABLA 5.1)
- 6 instancias difíciles (TABLA 5.2)

10 instancias de tamaño grande (

- TABLA 5.3)

Para cada tabla, la primera columna corresponde a la instancia, para la cual se especifican las dimensiones de la placa (*W*) y (*L*), el número de piezas distintas (*NP*), y el óptimo o mejor valor conocido. Las siguientes columnas corresponden a cada algoritmo comparado, para los cuales se especifica la mejor solución obtenida en las 30 ejecuciones (*Best*), el promedio de las 30 ejecuciones (*Mean*), y el margen de error porcentual del *Mean* respecto al óptimo (*%GAP*), siendo esta última, una medida de la precisión alcanzada por el algoritmo; mientras menor sea el *%GAP* más preciso es el algoritmo. Una celda oscurecida indica que el *Best* alcanza un valor óptimo o supera el mejor conocido. Un asterisco indica que el valor obtenido para el algoritmo marcado es superior a su contraparte, para la métrica indicada.

La TABLA 5.1 muestra los resultados para las instancias del primer grupo. Se observa que *AGc* presenta un *Mean* superior al de *AG* para casi la totalidad de las instancias, a excepción de 5 de ellas, donde son iguales. Esta superioridad alcanza un valor máximo para la instancia *CU10¸* donde *AGc* supera por 11323 unidades de ganancia a *AG*, siendo en promedio para este grupo, 773 unidades superior.

Así mismo, para casi todos los casos, *AGc* presenta un *Best* superior a *AG*, a excepción de la instancia *CU10*, en donde *AG* supera a *AGc*. La superioridad en el *Best*

de *AGc* alcanza un valor máximo para la instancia *CU11*, donde *AGc* supera por 6661 unidades de ganancia a *AG*, siendo en promedio para este grupo 327 unidades superior.

TABLA 5.1 *Resultados computacionales para las instancias pequeñas y medianas*

Instancia					*AG*			*AGc*		
	W	*L*	*NP*	*Optimo*	*Best*	*Mean*	*%GAP*	*Best*	*Mean*	*%GAP*
2s	40	70	10	2778	2743	2736	1,53	2743	*2741	1,32
3s	40	70	20	2721	2721	2720	0,03	2721	*2721	0,00
A1s	50	60	20	2950	2950	2950	0,00	2950	2950	0,00
A2s	60	60	20	3535	3535	3428	3,01	3535	*3455	2,26
A3	70	80	20	5451	5374	5342	2,00	*5431	*5374	1,41
A4	90	70	20	6179	6160	6028	2,45	6160	*6073	1,72
A5	132	100	20	12985	12780	12699	2,21	*12985	*12796	1,46
CHL1s	132	100	30	13099	12846	12807	2,23	*13014	*12839	1,99
CHL2s	62	55	10	3279	3244	3239	1,22	3279	3279	0,00
CHL3s	157	121	15	7402	7402	7402	0,00	7402	7402	0,00
CHL4s	207	231	15	13932	13932	13932	0,00	13932	13932	0,00
CHL5	20	20	10	390	390	390	0,00	390	390	0,00
CHL6	130	130	30	16869	16646	16434	2,58	*16736	*16618	1,49
CHL7	130	130	35	16881	16408	16383	2,95	*16595	*16492	2,31
CU1	100	125	25	12330	12200	12098	1,88	12200	*12182	1,20
CU2	150	175	35	26100	25596	24845	4,81	*25806	*25362	2,83
CU3	134	125	45	16679	16360	16256	2,54	*16472	*16356	1,93
CU4	285	354	45	99366	98910	96961	2,42	98910	*97870	1,51
CU5	456	385	50	173364	170452	168071	3,05	*172441	*170169	1,84
CU6	356	447	45	158572	157380	152387	3,90	*157764	*154287	2,70
CU7	563	458	25	247150	247150	244758	0,97	247150	*245415	0,70
CU8	587	756	35	432714	432198	423541	2,12	432198	*425087	1,76
CU9	856	785	25	657055	649282	645894	1,70	649282	*647432	1,46
CU10	794	985	40	773772	*760475	743979	3,85	759639	*755302	2,39
CU11	977	953	50	924696	906305	895956	3,11	*912966	*900050	2,67
OF1	70	40	10	2737	2714	2714	0,84	2714	2714	0,84
OF2	70	40	10	2690	2650	2650	1,49	2650	2650	1,49
STS2s	55	85	30	4653	4625	4623	0,65	4625	*4625	0,60
STS4s	99	99	20	9770	9450	9341	4,39	*9578	*9426	3,52
W	70	40	20	2721	2721	2720	0,03	2721	*2721	0,00
wang1	33	69	20	2277	2277	2277	0,00	2277	2277	0,00
wang2	39	70	20	2694	2694	2694	0,00	2694	2694	0,00
wang3	40	70	20	2721	2721	2720	0,03	2721	*2721	0,00

Para 12 de las 33 instancias ambos algoritmos alcanzan valores óptimos. *AGc* alcanza un valor óptimo para una instancia adicional (*A5*), en la que *AG* no lo hace. En general, los casos donde se alcanzó el óptimo, el *Best* es muy cercano o igual al *Mean*, lo que indica que son instancias en donde los algoritmos fácilmente encuentran la solución óptima.

Para todas las instancias de este grupo, los %*GAP* obtenidos por *AGc* son inferiores o iguales a los obtenidos por *AG*; siendo para 8 instancias iguales, y para las restantes 25 inferiores. La diferencia en el *GAP* alcanza un valor máximo de 1,98% para la instancia *CU2*, siendo en promedio 0,50%. En promedio el %*GAP* obtenido por *AG* equivale a 1,76% y el obtenido por *AGc* equivale a un 1,25%. Por lo tanto *AGc* es capaz de obtener resultados en promedio un 0,50% más precisos que *AG* para este grupo.

La TABLA 5.2 muestra los resultados para las instancias del segundo grupo. Se observa que *AGc* presenta un *Mean* superior al de *AG* para casi la totalidad de las instancias, a excepción de la instancia *Hchl8s*, donde son iguales. La diferencia en el *Mean* alcanza un valor máximo para la instancia *Hchl5s*¸ donde *AGc* supera por 1082 unidades de ganancia a *AG*, siendo en promedio para este grupo, 497 unidades superior.

Para 5 de las 6 instancias *AGc* presenta un *Best* superior a *AG*. Para la restante los *Best* alcanzados fueron equivalentes. La diferencia en el *Best* alcanza su valor máximo para la instancia *Hchl5s*, donde *AGc* supera por 1020 unidades de ganancia a *AG*, siendo en promedio para este grupo 326 unidades superior.

Para ninguna de las instancias de este grupo los algoritmos son capaces de obtener valores óptimos.

TABLA 5.2 *Resultados computacionales para las instancias difíciles*

Instancia					*AG*			*AGc*		
	W	*L*	*NP*	*Optimo*	*Best*	*Mean*	*%GAP*	*Best*	*Mean*	*%GAP*
Hchl3s	127	98	10	12215	12006	11868	2,84	*12153	*11981	1,92
Hchl4s	127	98	10	12006	11323	10981	8,54	*11543	*11339	5,56
Hchl5s	205	223	25	45410	43173	42858	5,62	*44193	*43940	3,24
Hchl6s	253	244	22	61040	59184	58405	4,32	*59668	*59327	3,09
Hchl7s	263	241	40	63112	62088	61131	3,14	*62175	*61640	2,81
Hchl8s	49	20	10	911	845	845	7,24	845	845	7,24

Para casi todas las instancias de este grupo, los %*GAP* obtenidos por *AGc* son inferiores o iguales a los obtenidos por *AG*; excepto para la instancia *Hchl8s* donde son iguales y equivalentes a 7,24. La diferencia en el %*GAP*, alcanza un valor máximo de 2,98% para la instancia *Hchl4s*, siendo el promedio de un 1,31%. En promedio el %*GAP* obtenido por AG equivale a 5,28% y el obtenido por *AGc* equivale a un 3,98%. Por lo tanto *AGc* es capaz de obtener resultados en promedio un 1,31% más precisos que AG para este grupo.

La

TABLA 5.3 muestra los resultados para las instancias del tercer grupo. Se observa que *AGc* presenta un *Mean* superior al de *AG* para casi la totalidad de las instancias, a excepción de la instancia *ATP39*, donde *AG* es superior. La superioridad de *AGc* alcanza su valor máximo para la instancia *ATP35*¸ donde *AGc* supera por 4554 unidades de ganancia a *AG*, siendo en promedio para este grupo, 1081 unidades superior.

Para 7 de las 10 instancias *AGc* presenta un *Best* superior a *AG*. De las 3 restantes, dos son equivalentes, y para la restante *AG* es superior. La mayor diferencia en el *Best* se registra para la instancia *Hchl5s*, donde *AGc* supera por 615 unidades de ganancia a *AG*, siendo en promedio para este grupo 233 unidades superior.

TABLA 5.3 *Resultados computacionales para las instancias grandes*

Instancia					*AG*			*AGc*		
	W	*L*	*NP*	*Optimo*	*Best*	*Mean*	*%GAP*	*Best*	*Mean*	*%GAP*
ATP30	927	152	38	140904	138436	137206	2,62	*138556	*137617	2,33
ATP31	856	964	51	823976	814069	807318	2,02	*815672	*810503	1,64
ATP32	307	124	56	38068	37498	37234	2,19	*37524	*37374	1,82
ATP33	241	983	44	236611	231359	228581	3,39	*232202	*230225	2,70
ATP34	795	456	27	361398	356396	351671	2,69	356396	*353582	2,16
ATP35	960	649	29	621021	*614090	602345	3,01	611807	*606899	2,27
ATP36	537	244	28	130744	127902	126794	3,02	*128656	*127536	2,45
ATP37	440	881	43	387276	379390	375132	3,14	*380338	*376893	2,68
ATP38	731	358	40	261395	256911	253530	3,01	*257328	*254553	2,62
ATP39	538	501	33	268750	265612	*263237	2,05	265612	263123	2,09

Para todas las instancias de este grupo, los %*GAP* obtenidos por *AGc* son inferiores a *AG*; excepto para la instancia ATP39, donde *AG* es ligeramente inferior. La

diferencia en el %*GAP*, alcanza un valor máximo de 1,22% para la instancia *ATP35*, siendo en promedio de un 0,57%. En promedio el *GAP* obtenido por AG equivale a 2,60% y el obtenido por *AGc* equivale a un 2,03%. Por lo tanto *AGc* es capaz de obtener resultados en promedio un 0,57% más precisos que AG para este grupo.

Para los 3 grupos analizados se observa que *AGc* alcanza una mayor precisión que *AG*. Estas diferencias muestran ser más pequeñas (*Δ%GAP*=0,50%) para el grupo1, donde las instancias son pequeñas y medianas, mayores (*Δ%GAP*=0,57%) para el grupo3, correspondientes a instancias grandes, y mucho mayores (*Δ%GAP*=1,31%) para el grupo2, correspondientes a instancias difíciles. Numéricamente, las diferencias parecen ser poco significativas, de hecho en promedio el *Δ%GAP* para el total de instancias es de 0,59% aprox. Una prueba estadística permitirá eliminar la subjetividad de esta apreciación. Para realizar la prueba apropiada, primero se debe determinar si existe normalidad en la distribución de los datos. Debido al tamaño de la muestra (<50), se usa la prueba de *Shapiro-Wilk* para contrastar la hipótesis de normalidad. En la TABLA 5.5 Se observa que para ambos algoritmos el nivel de significación es menor a 0,05, luego se rechaza la hipótesis nula de que los datos se ajustan a una distribución normal. Las FIGURA 5.1 y 4.2 evidencian la no normalidad de los datos, donde existe una alta frecuencia de *%GAP* concentrados en los valores pequeños.

TABLA 5.4 *Resultados test de normalidad para el %GAP exhibido por cada algoritmo*

	Algoritmo	*Kolmogorov-Smirnov*[a]			*Shapiro-Wilk*		
		Estadístico	*gl*	*Sig.*	*Estadístico*	*Gl*	*Sig.*
%GAP	*AG*	,144	49	,012	,929	49	,006
	AGc	,167	49	,001	,915	49	,002

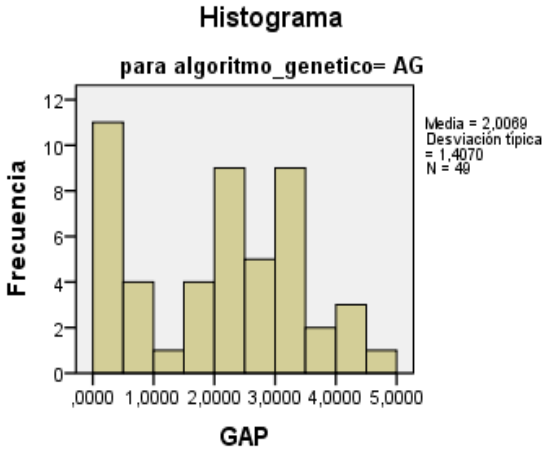


FIGURA 5.1 *Histograma para el %GAP en AG*

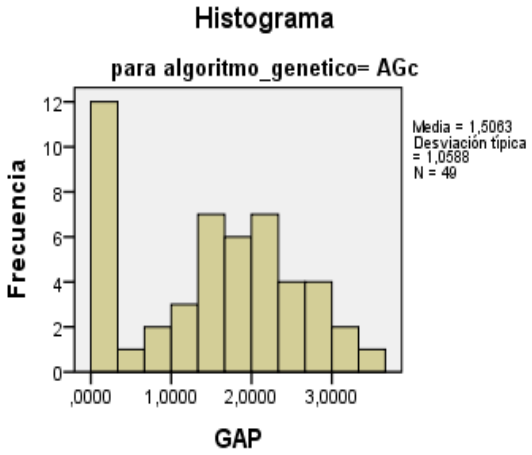


FIGURA 5.2 *Histograma para el %GAP en AGc*

Dado que los datos no exhiben normalidad, se debe usar una prueba no paramétrica para comparar dos muestras independientes. Se aplica la prueba *U de Mann-Withney* para contrastar la hipótesis nula de que no existen diferencias significativas entre el *%GAP* mostrado por ambos grupos. Los resultados de esta prueba se muestran en la TABLA 4.4. Como se puede observar, el nivel de significación es de 0,032 < 0,05. Luego se rechaza la hipótesis nula y es posible concluir que existen diferencias estadísticamente significativas entre las precisiones de los grupos comparados, que pueden ser explicadas por el algoritmo utilizado.

TABLA 5.5 *Resultados prueba U de Mann-Whitney para el %GAP*

	%GAP
U de Mann-Whitney	899,000
W de Wilcoxon	2124,000
Z	-2,147
Sig. asintót. (bilateral)	,032

Otro aspecto importante a evaluar, es el comportamiento en la búsqueda realizada por los algoritmos, lo que permite deducir cuál es la influencia en la presión de selección inducida por cada uno de ellos. Se ha seleccionado una instancia de cada grupo, donde se grafica el promedio de las curvas de convergencia del *%GAP* del mejor individuo, para las 30 ejecuciones, obteniendo así, la tendencia en el comportamiento de la convergencia para la instancia en cuestión. Se ha utilizado el *%GAP* en lugar de usar directamente el *fitness* (ganancia del patrón), ya que es una métrica normalizada que permite realizar un análisis transversal a las instancias. Para cada gráfico, en el eje *y* se

muestra el *%GAP* obtenido por el promedio del mejor individuo en las 30 ejecuciones en cada generación, dada por el eje *x*.

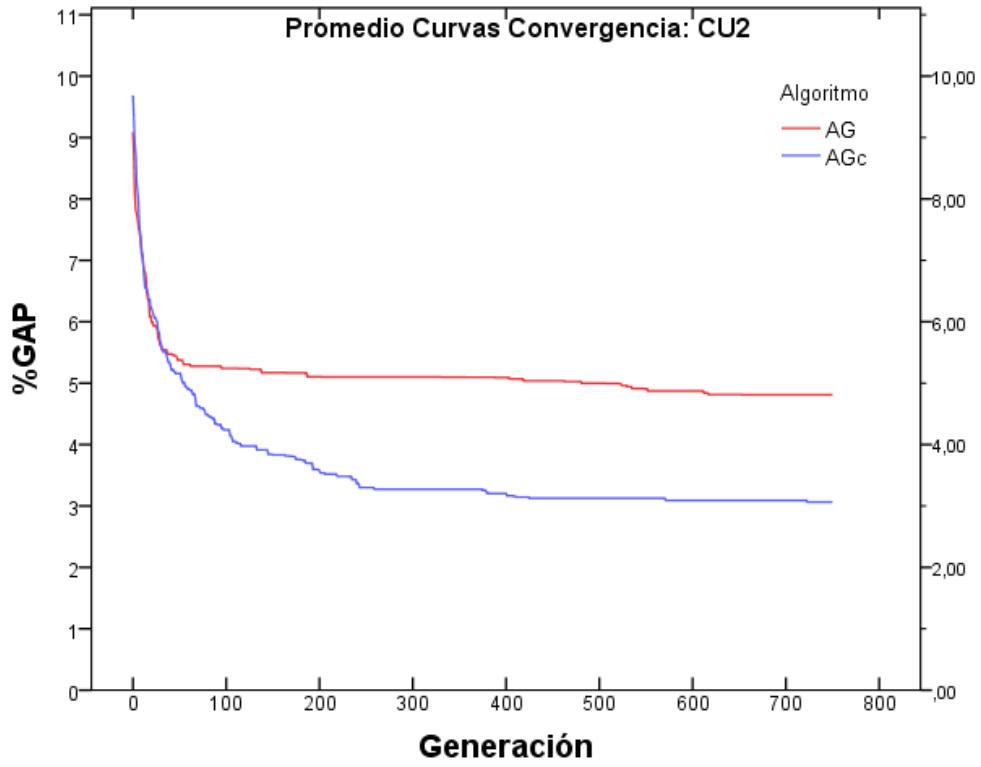


FIGURA 5.3 *Promedio de curvas de convergencia de AG y AGc para la instancia CU2*

La

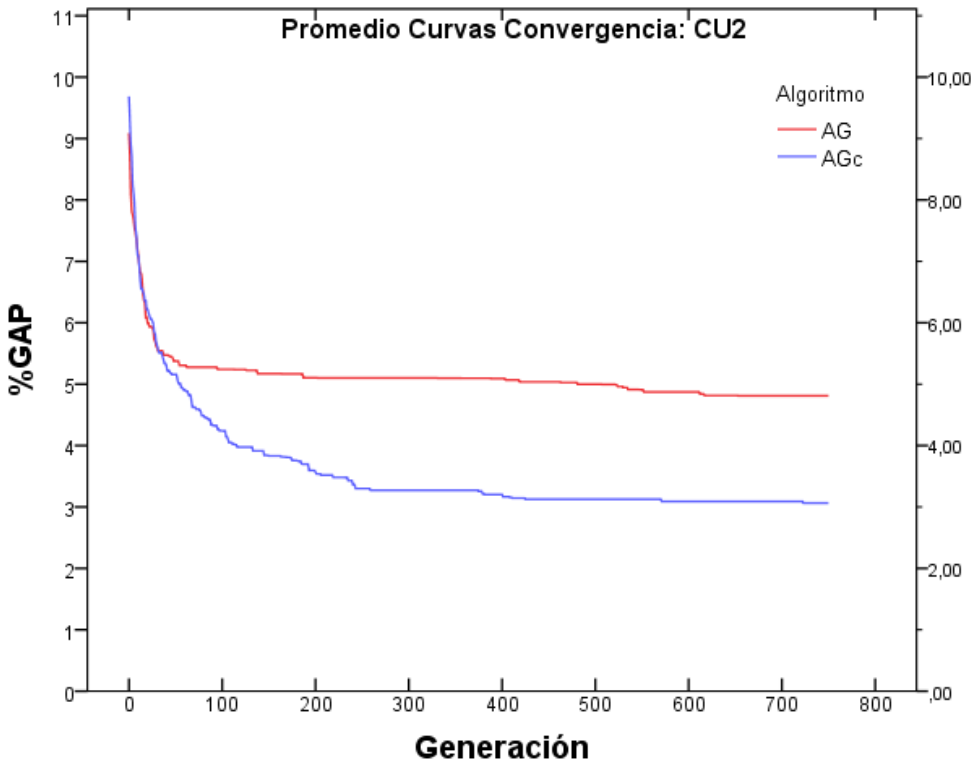


FIGURA 5.3 muestra las curvas de convergencia para la instancia *CU2* (perteneciente al primer grupo). Se observa que al cabo de pocas generaciones *AGc* es capaz de promover la creación de individuos de mayor precisión que *AG*. En tanto que este último tiende a converger abruptamente de manera prematura, teniendo dificultades para superar la precisión del mejor individuo encontrado. Por lo anterior, la brecha entre *AG* y *AGc* crece rápidamente a medida que progresa la evolución, hasta que *AGc* comienza a converger, de manera mucho más suave que *AG*. Al final de la evolución *AG* alcanza un *%GAP* de 4,81% y *AGc* alcanza un *%GAP* de 3,06%.

La

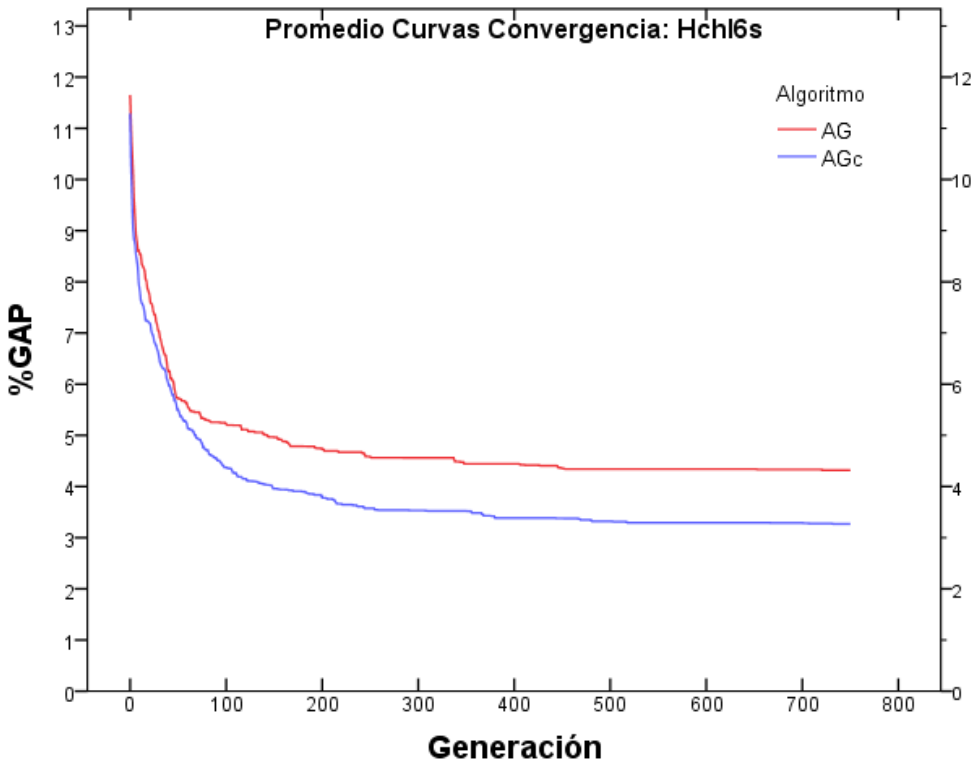


FIGURA 5.4 muestra las curvas de convergencia para la instancia *Hchl6s* (perteneciente al segundo grupo). Similarmente, aunque en menor medida que en el caso anterior, en etapas tempranas de la evolución *AGc* empieza a diferenciarse de *AG*, obteniendo soluciones de mayor precisión y mostrando una curva más suave en la convergencia. Al final de la evolución *AG* alcanza un *%GAP* de 4,32% y *AGc* alcanza un *%GAP* de 3,27%, donde se observa que ambos algoritmos han alcanzado un estado de madurez evolutiva, ya que las curvas se han estabilizado en sus valores respectivos.

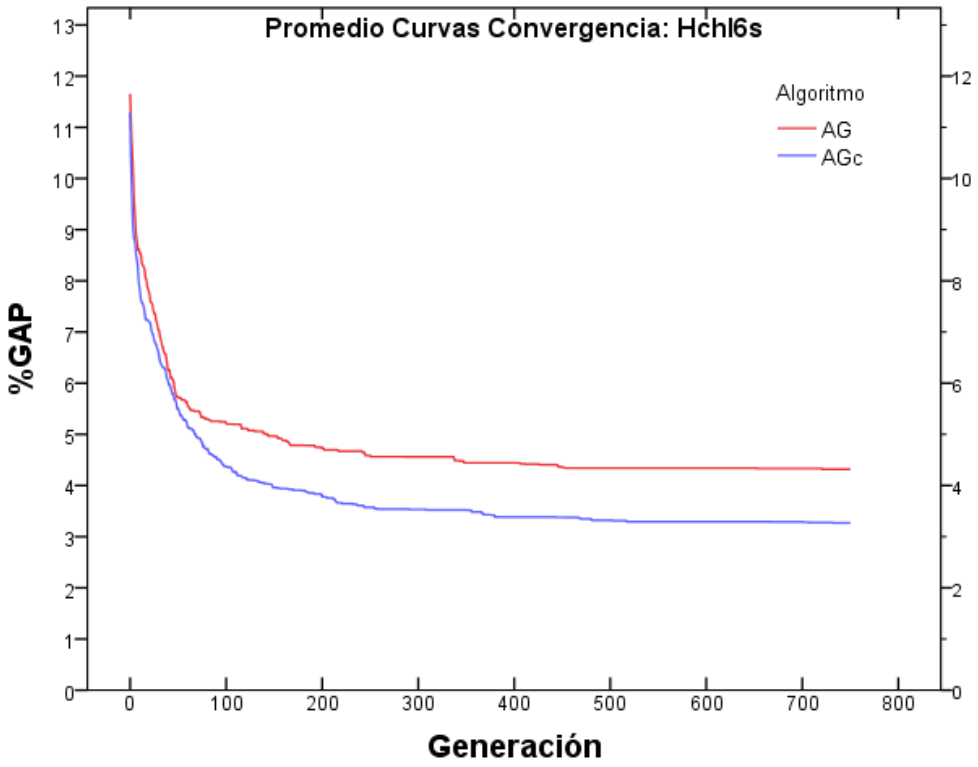


FIGURA 5.4 *Promedio de curvas de convergencia de AG y AGc para la instancia Hchl6s*

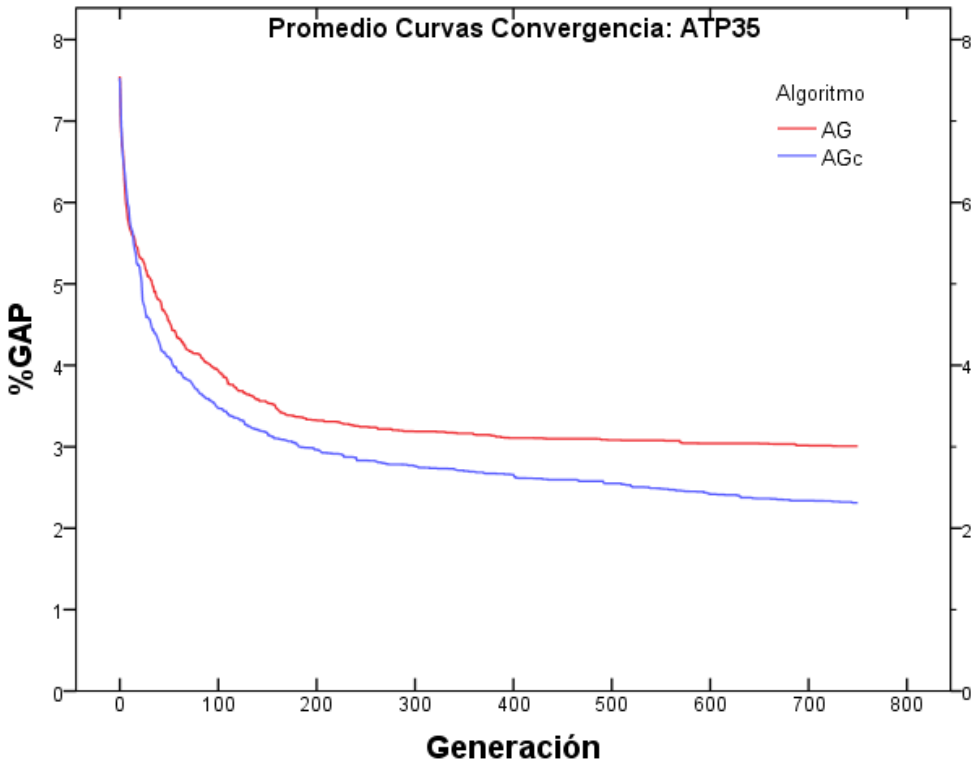


FIGURA 5.5 *Promedio de curvas de convergencia de AG y AGc para la instancia ATP35*

La

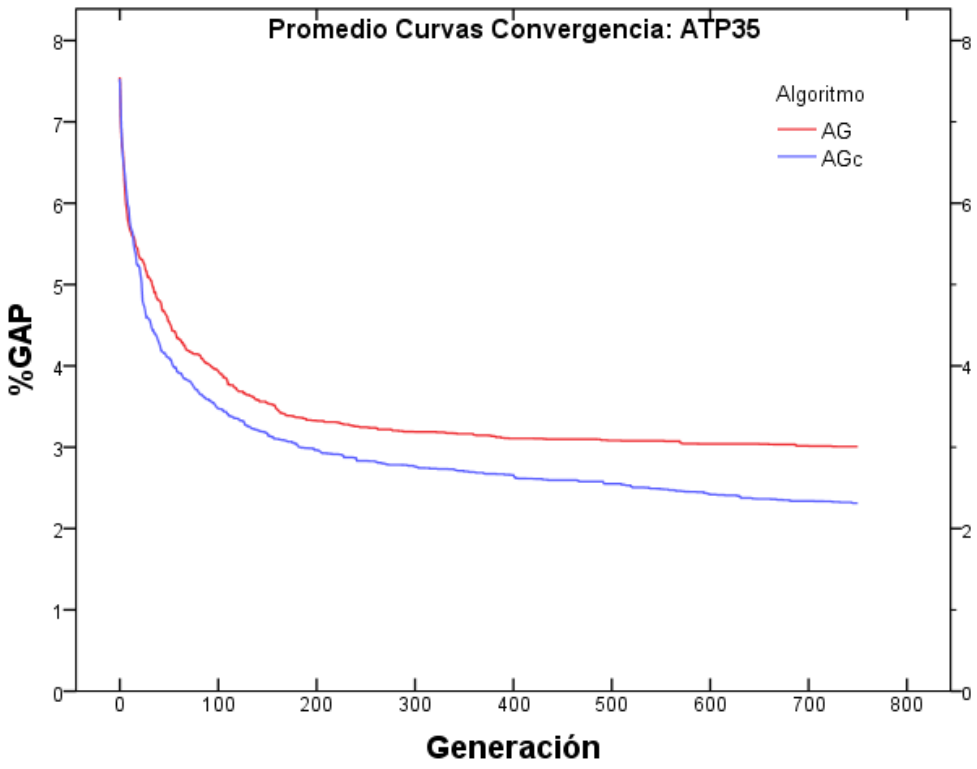


FIGURA 5.5 muestra las curvas de convergencia para la instancia *ATP35* (perteneciente al tercer grupo). Una vez más, aunque en menor medida que los casos anteriores, *AGc* rápidamente supera a *AG*, obteniendo soluciones de mayor precisión. Al final de la evolución, *AG* alcanza un *%GAP* de 4,81% y *AGc* alcanza un *%GAP* de 3,06%. Aparentemente, ambos algoritmos aún no han terminado de converger, y una mayor cantidad de generaciones permitiría a la evolución mejorar un poco más los individuos.

Para las tres instancias analizadas, *AGc* muestra una convergencia más suave que *AG*, lo que le permite alcanzar individuos de mayor precisión. *AG* converge de manera muy abrupta para la instancia *CU2*, menos abrupta para la instancia *Hchl6s*, y menos abrupta para la instancia *ATP35*, tendiendo las curvas de ambos algoritmos a exhibir un comportamiento cada vez más similar. Por su parte, *AGc* muestra ser más robusto, ya que transversalmente a las tres instancias analizadas, las curvas tienden a mantener un comportamiento similar de convergencia más suave. Esto puede explicarse debido a que *AGc* promueve la mantención de la diversidad genética, mediante su influencia en la presión de selección global, por lo que es más difícil que se atasque en óptimos locales en etapas tempranas de la evolución.

La tendencia en la convergencia de los algoritmos muestra que, prácticamente durante toda la evolución, el enfoque celular provee mejores individuos que un enfoque

no estructurado. Contrario a la presunción inicial de que una presión de selección no controlada (rápida intensificación de soluciones promisorias), debería superar en etapas tempranas de la evolución, a una presión de selección más controlada (preservación de diversidad, a costa de una difusión más lenta de soluciones promisorias), los cuales son superados en etapas maduras. En este caso, dicha superación tiende a ocurrir en etapas tempranas de la evolución. Es posible que este hecho se deba a que *AG* en el proceso de intensificación no es capaz de encontrar mejores individuos debido a una componente de sesgo en la búsqueda genética, lo cual se ve atenuado por la mantención de diversidad del enfoque celular, que se traduce en el alcance de mejores individuos, en el caso de *AGc*.

Aún cuando la prueba estadística realizada sugiere diferencias significativas entre las precisiones de los algoritmos, esto parece no ser suficiente para probar la hipótesis. Después de todo, el nivel de significación obtenido es ligeramente inferior a 0,05 (0,032), y de hecho, si se utiliza un intervalo de confianza más estricto, como 0,01, los resultados obligarían a concluir lo contrario. Así entonces, persiste la interrogante de si las diferencias obtenidas entre los algoritmos pudieran ser mayores, o más precisamente, si existen factores externos que hayan mitigado estas diferencias. Un hecho que apunta en esta dirección, es que las soluciones no parecen poder superar ciertos límites. Ejemplos claros de esto, son las instancias *2s*, *OF1* y *OF2*, las cuales corresponden a instancias pequeñas, donde ambos *AG* encuentran fácilmente soluciones suboptimales, que no son capaces de superar en las 30 ejecuciones. Un antecedente que se debe recordar, es que se ha empleado un enfoque basado en decodificadores de secuencias. En este escenario, existen dos posibles fuentes de sesgo en la búsqueda genética:

1. **La representación:** Se utiliza una codificación binaria para generar las secuencias de piezas, en lugar de usar directamente una representación más natural para ello. Existe una serie de dificultades en el uso de esta representación bajo el enfoque adoptado, algunas de ellas son discutidas en la sección 3.2.6. Quizá la dificultad más significativa, para efectos de calidad en la solución, es la presumible incapacidad de la representación para generar todas las posibles secuencias. Así, el *AG* no puede alcanzar eventualmente mejores soluciones, dadas por aquellas secuencias que no puede generar.

2. **El decodificador:** Una característica del enfoque en dos etapas, es su fuerte dependencia del decodificador, ya que la calidad de la solución depende en gran medida de la eficiencia que este tenga, respecto a la densidad del patrón construido. En teoría, la heurística de colocación empleada, únicamente tiene en consideración la restricción de guillotina y no utiliza reglas adicionales, que pudieran mejorar los patrones resultantes, precisamente para eliminar fuentes de sesgo en la búsqueda genética y relegar la tarea del mejoramiento de patrones a la evolución. Sin embargo, puede que aún existan limitaciones, por ejemplo al no considerar algún tipo de encaje de piezas, no previsto en el diseño.

Para despejar las sospechas referentes a la primera fuente de sesgo, un experimento adicional fue llevado a cabo para *AGc*, el cual consistió en sustituir la representación binaria por una representación de permutación, en conjunto con una función constructora apropiada para la nueva representación. Para el cruzamiento se utilizó *OX*, y para la mutación *swap mutation*, manteniendo el resto de los ajustes fijos. Para diferenciarlo de *AGc*, el algoritmo modificado se denota como *AGc1*. Los resultados del experimento adicional se incluyen en el anexo x.x. En resumen, *AGc1* es capaz de alcanzar valores óptimos para 6 instancias adicionales, entre ellas: *2s*, *OF1*, *OF2*, y aumentar la precisión del *%GAP* en un 0,51%. Por lo tanto, el uso de una representación más apropiada tiene un importante efecto, disminuyendo el sesgo en la búsqueda genética, y aumentando así, la precisión del algoritmo.

Para despejar las sospechas referentes a la segunda fuente de sesgo, se realizó una traza en la que se intenta construir un patrón óptimo mediante el decodificador empleado. El patrón fue obtenido de (Yoon et al., 2013), y corresponde a un patrón óptimo para la instancia *Hchl5s*. Las trazas se encuentran en el anexo x.x. Gracias a este ejercicio, se descubrieron dos fuentes de sesgo en el decodificador:

1. La heurística no considera un caso de inserción de piezas en el patrón. Este tipo de encaje corresponde a una heurística *bottom-left*, la cual no necesariamente viola la restricción de guillotina.

2. El patrón óptimo que se intentó construir, se obtiene al efectuar una determinada secuencia de combinaciones verticales y horizontales. En contraste, la heurística

impone que una combinación vertical siempre tiene precedencia sobre una horizontal, siendo este orden establecido de manera arbitraria.

Un experimento adicional fue llevado a cabo para *AGc1*, el cual ya es una modificación de *AGc*, agregando un encaje tipo *bottom-left* en la heurística de colocación. El seudocódigo se detalla en el anexo x.x. Para diferenciarlo de *AGc1*, el algoritmo con la heurística modificada se denota como *AGc2*. Los resultados se incluyen en el anexo x.x. En resumen, el algoritmo aumenta la precisión del *%GAP* en un 0,19%, sin embargo no es capaz de encontrar soluciones óptimas adicionales, posiblemente debido a la segunda fuente de sesgo mencionada.

El no determinismo subyacente en la elección del mejor tipo de combinación a realizar para cada pieza (vertical u horizontal), sugiere que esta información es apropiada para ser incorporada al procesamiento genético, y de esa forma poder ser evolucionada. Para ello se vislumbran dos alternativas:

- Utilizar una representación heterogénea, donde una parte entera indique la secuencia de piezas, y una parte binaria indique el tipo de combinación. Este tipo de cromosoma heterogéneo se encuentra implementado en la plataforma evolutiva *JCell*, por lo que su integración no supone una gran cantidad de trabajo adicional. Sin embargo pueden existir dificultades al aplicar los operadores genéticos estándar (similares a las explicadas en).

- Mantener un cromosoma para las piezas y otro para las combinaciones. En este caso, la plataforma debe ser extendida, implementando un individuo con múltiples cromosomas. En este caso se mantiene la aplicación de los operadores estándar.

Al reducir las fuentes de sesgo mencionadas anteriormente, existen dos escenarios posibles; En el primer caso, debido a que el sesgo radica en la capa común de modelamiento, ambos algoritmos mejoran su desempeño, aumentando su precisión en una proporción similar, por lo que las diferencias tienden a mantenerse. En el segundo escenario, la mejora en la capacidad de exploración del algoritmo inducida por la reducción del sesgo en la búsqueda, permite explotar de mejor manera las cualidades de la estructuración bajo un enfoque celular, provocando que *AGc* mejore sus resultados en

una proporción mayor que *AG*, obteniendo así, una mayor diferencia en los resultados obtenidos por cada algoritmo.

A continuación se incluyen algunas comparaciones entre el mejor de los algoritmos obtenidos, *AGc2*, el cual corresponde al algoritmo genético celular basado en la representación de permutación y la heurística extendida, con distintos algoritmos propuestos en la literatura. El objetivo es evaluar la competitividad del enfoque evolutivo celular adoptado, frente a otros enfoques existentes. Para cada tabla se especifica el *Best* de cada algoritmo, que en el caso de *AGc2* corresponde a la mejor solución de entre las 30 ejecuciones. Las comparaciones están organizadas en 3 grupos. Primero, para 11 instancias de tamaño medio, *AGc2* es comparado con *GRASP* (Álvarez-Valdés et al., 2002), y *SPGAL* (Bortfeldt y Winter, 2008), un algoritmo genético específico para el problema, el cual realiza la búsqueda genética directamente en el espacio de patrones de corte. Para los dos grupos restantes, correspondientes a 6 instancias difíciles y 10 instancias grandes, *AGc2* es comparado con *GRASP* y *HCEB*, una heurística basada en una clase especial de patrón, el cual es el enfoque heurístico más competitivo hasta ahora.

La TABLA 5.6 muestra las comparaciones para el primer grupo. No se aprecia una superioridad absoluta de un algoritmo por sobre el resto. Para 5 de las 11 instancias *GRASP* y SPGAL son superiores al resto, mientras que para 4 de las 11 instancias *AGc2* supera al resto. Al comparar ambos enfoques evolutivos, *AGc2* supera a *SPGAL* para 5 instancias y *SPGAL* supera a *AGc2* para otras 4 instancias, para el resto de instancias empatan, por lo que puede considerarse que ambos enfoques son similares en cuanto a precisión. Sin embargo, se debe rescatar el hecho de que, a pesar de que *AGc2* no es especializado para el problema en cuestión, aún es capaz de igualar y superar a *SPGAL* para algunas instancias, siendo este último un *AG* mucho más especializado, ya que utiliza operadores específicos y post-optimizaciones. *GRASP* encuentra valores óptimos para 5 de las 11 instancias, *SPGAL* y *AGc2* lo hacen para 4 de las 11 instancias. En este sentido, *GRASP* muestra ser el superior de los 3 algoritmos comparados para este grupo.

TABLA 5.6 *Comparación entre GRASP, SPGAL, y AGc2 para instancias medianas*

Instancia					*GRASP*	*SPGAL*	*AGc2*
	W	*L*	*NP*	*Optimo*			
CU1	100	125	25	12330	12312	*12330	*12330
CU2	150	175	35	26100	*26100	*26100	25817

CU3	134	125	45	16679	16652	16598	*16653
CU4	285	354	45	99366	*99264	98764	99039
CU5	456	385	50	173364	*173364	171935	172441
CU6	356	447	45	158572	158572	158572	158572
CU7	563	458	25	247150	*247150	246143	*247150
CU8	587	756	35	432714	*432714	431126	432198
CU9	856	785	25	657055	651597	*657055	*657055
CU10	794	985	40	773772	767580	*772118	771002
CU11	977	953	50	924696	909898	*918304	916035

La TABLA 5.7 muestra las comparaciones para el segundo grupo. Se aprecia una superioridad por parte de *HCEB*, superando al resto para 5 de las 6 instancias, y presentando para dos casos, valores óptimos. Los otros algoritmos no alcanzan valores óptimos para ninguna de las instancias de este grupo. *AGc2* es superior a *GRASP* para 4 de las 6 instancias, mientras que *GRASP* supera a *AGc2* para las restantes 2.

TABLA 5.7 *Comparación entre GRASP, HCEB, y AGc2 para instancias difíciles*

Instancia					*GRASP*	*HCEB*	*AGc2*
	W	*L*	*NP*	*Optimo*			
Hchl3s	127	98	10	12215	12159	*12214	12121
Hchl4s	127	98	10	12006	11621	*11964	*11964
Hchl5s	205	223	25	45410	44346	*45410	44958
Hchl6s	253	244	22	61040	60403	*61040	60491
Hchl7s	263	241	40	63112	62547	*63102	62655
Hchl8s	49	20	10	911	*904	876	894

La TABLA 5.8 muestra las comparaciones para el tercer grupo. En este caso, se aprecia una marcada superioridad de *HCEB* por sobre el resto de los algoritmos, superándolos para todas las instancias. *HCEB* alcanza valores óptimos para 9 de las 10 instancias, mientras que el resto no alcanza valores óptimos para ninguna de ellas. Aún así, *AGc2* muestra ser superior a *GRASP*, superándolo para 8 de las 10 instancias, mientras que *GRASP* supera a *AGc2* para las restantes 2. A pesar de lo anterior, en general *AGc2* logra resultados de una precisión razonable para la mayoría de las instancias, mostrando márgenes relativamente moderados respecto a los valores óptimos.

TABLA 5.8 *Comparación entre GRASP, HCEB, AGc2 para instancias grandes*

Instancia	*GRASP*	*HCEB*	*AGc2*

	W	*L*	*NP*	*Optimo*			
ATP30	927	152	38	140904	138863	*140904	139759
ATP31	856	964	51	823976	801767	*823976	816745
ATP32	307	124	56	38068	37786	*38068	37714
ATP33	241	983	44	236611	231178	*236611	234935
ATP34	795	456	27	361398	353822	*361357	358823
ATP35	960	649	29	621021	607864	621021	615343
ATP36	537	244	28	130744	129634	130744	129087
ATP37	440	881	43	387276	379329	387276	382224
ATP38	731	358	40	261395	252605	261395	258243
ATP39	538	501	33	268750	263076	268750	267593

Con el fin de apreciar la precisión obtenida en términos cualitativos, a continuación se incluyen patrones óptimos para algunas instancias, obtenidos de (Yoon et al., 2013), en conjunto con los mejores patrones obtenidos por *AGc2* para cada una de ellas.

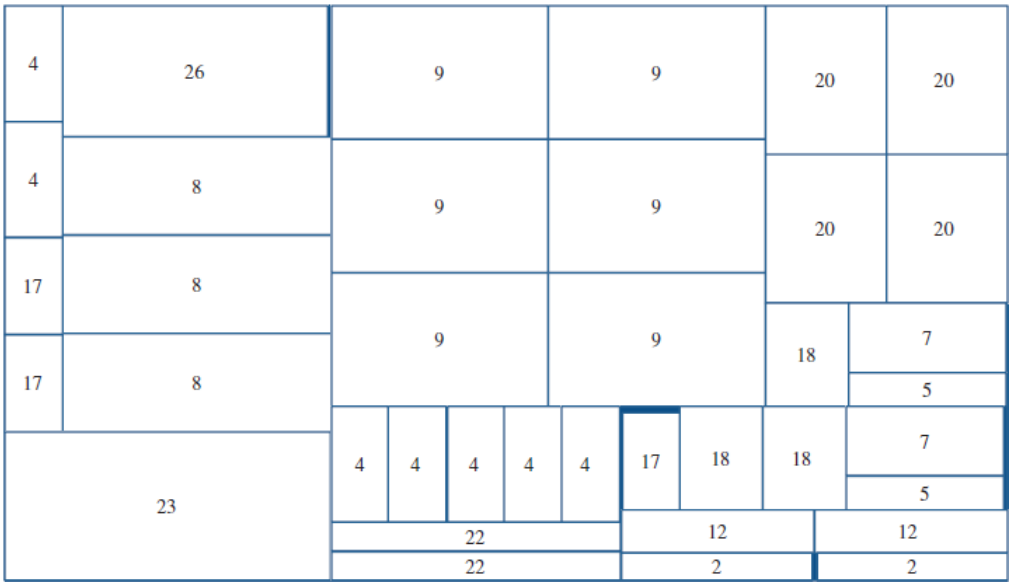


FIGURA 5.6 *Un patrón óptimo para la instancia ATP34, con W=795, L=456, NP=27 y ganancia igual a 361368*

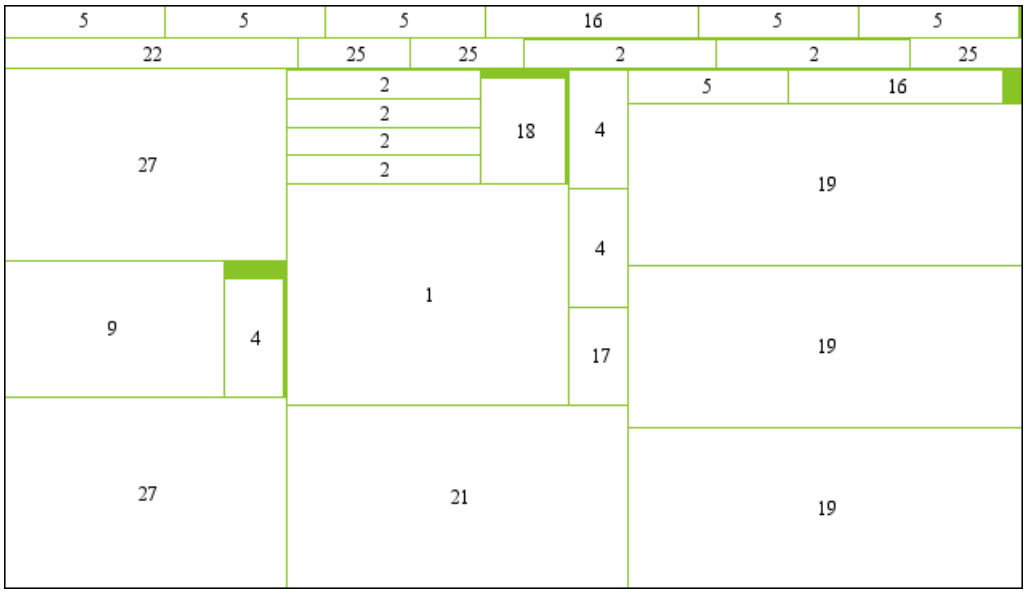


FIGURA 5.7 *Mejor patrón obtenido por AGc2 para la instancia ATP34, con W=795, L=456, NP=27, ganancia igual a 359243 y %GAP=0,60%*

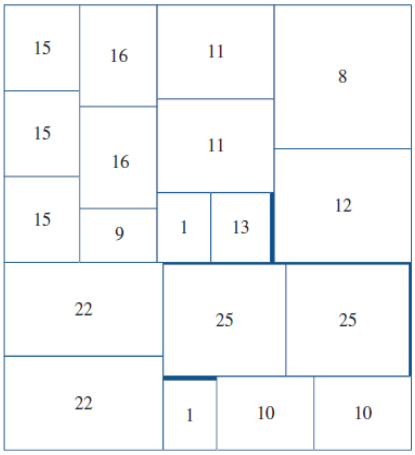


FIGURA 5.8 *Un patrón óptimo para la instancia Hchl5s, con W=205, L=223, NP=25 y ganancia igual a 45410*

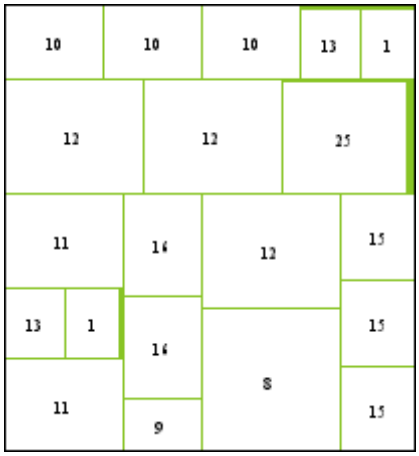


FIGURA 5.9 *Mejor patrón obtenido por AGc2 para la instancia Hchl5s, con W=205, L=223, NP=25, ganancia igual a 45147 y %GAP=0,58%*

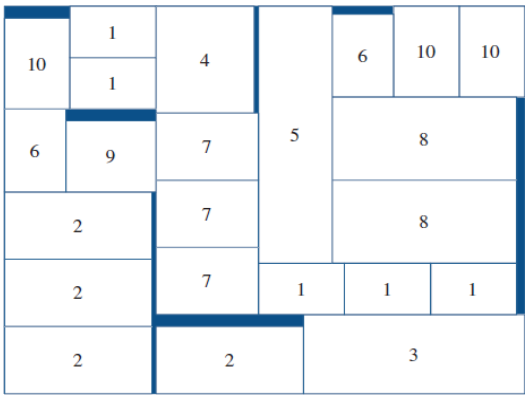


FIGURA 5.10 *Un patrón óptimo para la instancia Hchl4s, con W=127, L=98, NP=10 y ganancia igual a 12006*

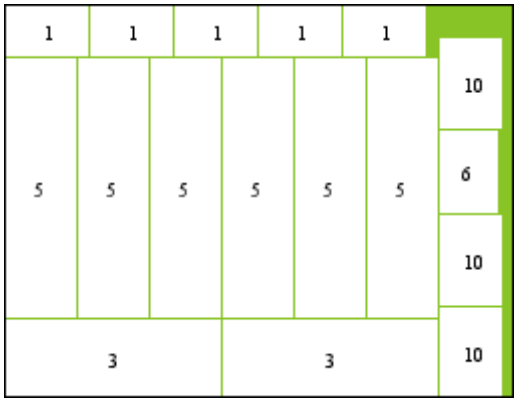


FIGURA 5.11 *Mejor patrón obtenido por AG2c para la instancia Hchl4s, con W=127, L=98, NP=10, ganancia igual a 11964 y %GAP=0,35%*

Capítulo 6 CONCLUSIONES

En este trabajo, un modelo de estructuración de población bajo un enfoque celular, es evaluado en un algoritmo genético, para el problema de corte de piezas bidimensional restricto, con las restricciones de guillotina y orientación. La hipótesis plantea que los mecanismos de estructuración celular permiten mejorar el comportamiento en la búsqueda realizada por el algoritmo, en particular, retardando la convergencia del algoritmo, al mismo tiempo que manteniendo la diversidad genética. Como consecuencia, resultados más competitivos podrían obtenerse en términos de calidad en la solución encontrada.

Para probar la hipótesis, dos algoritmos genéticos son comparados. Uno no estructurado, correspondiente a un AG generacional (*AG*), el cual considera una sola gran población, y un AG celular (*AGc*), el cual estructura la población basándose en el concepto de aislamiento por distancia, donde solo individuos cercanos (pertenecientes a la misma vecindad), pueden interactuar.

Para configurar los algoritmos, una herramienta de optimización de parámetros, *paramILS*, es utilizada. Las mejores configuraciones obtenidas en esta primera fase experimental, verifican empíricamente ciertos conceptos subyacentes en la hipótesis. En concreto, el operador de selección obtenido corresponde a selección por ruleta, la cual ejerce una menor presión sobre la evolución, promoviendo así, la exploración del algoritmo. Para los parámetros de *AGc*, en primer lugar se obtiene una población de *1x400* individuos, lo que verifica el principio de que maximizar la dispersión de individuos promueve la exploración del algoritmo. En segundo lugar, se obtiene una vecindad *Compact9*, cuya rapidez de difusión intermedia es consistente con un balance entre exploración y explotación. Por último, se obtiene una política de actualización *uc*, lo cual también está alineado con el resto de parámetros, ya que es la política asíncrona que induce una menor presión de selección de las consideradas.

Para la fase experimental, 49 instancias ampliamente utilizadas en la literatura son identificadas. Estas son clasificadas en 3 grupos, consistentes en: 33 instancias pequeñas y medias, 6 instancias difíciles y 10 instancias grandes. El diseño experimental consiste en la realización de un número independiente de ejecuciones (30) para cada una de las instancias, de donde se establecen 3 métricas para medir el comportamiento numérico de los algoritmos: *Mean*, correspondiente al promedio de los

fitness en las 30 ejecuciones. *%GAP*, correspondiente al error porcentual del *Mean* respecto al valor óptimo, y *Best*, correspondiente al máximo de los *fitness* obtenidos en las 30 ejecuciones. Además, para evaluar el comportamiento en la búsqueda realizada, se generan las curvas de convergencia de los algoritmos respectivos.

Los resultados numéricos muestran que, en general, *AGc* obtiene resultados más competitivos que *AG*, presentando en promedio *Mean* y *Best* superiores y *%GAP* inferiores para los 3 grupos de instancias. En concreto, *AGc* es en promedio 0,50% más preciso para el primer grupo, alcanzando un *%GAP* promedio de 1,25%. Para el segundo grupo *AGc* es 1,31% más preciso, alcanzando un *%GAP* promedio de 3,98%. Para el tercer grupo *AGc* es 0,57% más preciso, obteniendo un *%GAP* promedio de 2,03%. De estas diferencias, resalta el segundo grupo, correspondiente a instancias difíciles, donde las mayores diferencias se deben más por una importante disminución en la precisión de *AG*, que por una precisión de *AGc*. Así, *AGc* presenta mayor robustez, al mantener la precisión dentro de márgenes relativamente moderados, de manera transversal a los 3 grupos evaluados. Respecto a obtención de valores óptimos, ambos algoritmos obtienen 12 valores óptimos para el primer grupo. *AGc* obtiene 1 adicional. En contraste, para los grupos restantes no son capaces de obtener ningún óptimo. Finalmente, una prueba estadística permite concluir, con un 95% de confianza, que existen diferencias estadísticamente significativas en las precisiones obtenidas entre los algoritmos comparados.

Para evaluar el comportamiento en la búsqueda realizada por los algoritmos, 3 curvas de convergencia, representantes de cada grupo, son seleccionadas. Cada curva grafica el promedio del *%GAP* del mejor individuo en cada generación en las 30 ejecuciones, lo que da la tendencia para la instancia en cuestión. Para las tres instancias analizadas, *AGc* muestra una convergencia más suave que *AG*, lo que le permite alcanzar individuos de mayor precisión. *AG* muestra un comportamiento más inestable, al mostrar una convergencia más abrupta para la instancia del primer grupo, y menos abrupta para las restantes. Por su parte, *AGc* muestra ser más robusto, ya que transversalmente a las tres instancias analizadas, las curvas tienden a mantener un comportamiento similar de convergencia más suave. Esto puede explicarse debido a que *AGc* promueve la exploración, favoreciendo la mantención de la diversidad genética, por lo que es más difícil que se atasque en óptimos locales en etapas tempranas de la evolución.

El sesgo en la búsqueda genética afecta significativamente el desempeño de los algoritmos. Se identifican dos fuentes de sesgo. La primera está dada por la representación adoptada, al no ser capaz de generar todas las posibles secuencias de piezas. Una representación de permutación es capaz de aumentar la precisión de *AGc* en un 0,52%, alcanzando valores óptimos para 6 instancias adicionales pertenecientes al primer grupo. La segunda fuente de sesgo radica en el decodificador. Específicamente al no ser capaz de realizar un encaje de piezas tipo *bottom-left*, el cual no necesariamente viola la restricción de guillotina. La inclusión de este tipo de encaje es capaz de aumentar la precisión en un 0,19% adicional. Una fuente de sesgo adicional proviene del hecho de considerar una precedencia fija en el tipo de combinación a realizar, por lo que se plantea incorporar esta información a la evolución.

Para medir la competitividad del enfoque evolutivo celular, el mejor de los algoritmos obtenidos, *AGc2*, es comparado con otros enfoques existentes en la literatura. En particular, para instancias de tamaño medio, es comparado con *GRASP* y *SPGAL*, un algoritmo genético específico para el problema. *AGc2* muestra ser competitivo frente a los otros enfoques, presentando una precisión similar. Para instancias difíciles y grandes, *AGc2* es comparado con *GRASP* y *HCEB*, siendo el enfoque heurístico más competitivo hasta ahora. En estos casos *HCEB* muestra ser superior al resto. Sin embargo *AGc2* en general muestra ser superior a *GRASP*.

Se concluye que los mecanismos de estructuración celular son una herramienta efectiva de ajuste en el comportamiento en la búsqueda realizada por el algoritmo genético, permitiendo un control en la presión de selección y un balance entre explotación y exploración del algoritmo. Lo que eventualmente conduce a una mejora significativa en el desempeño del algoritmo respecto a la calidad de la solución.

Como trabajo futuro se plantea incorporar la información del tipo de combinación en la evolución. Para ello se vislumbran dos alternativas:

- Utilizar una representación heterogénea, donde una parte entera indique la secuencia de piezas, y una parte binaria indique el tipo de combinación.
- Mantener un cromosoma para las piezas y otro para las combinaciones. En este caso, la plataforma debe ser extendida para soportar individuos con múltiples cromosomas.

Se espera que la incorporación de esta información adicional del problema en la evolución, reduzca el sesgo en la búsqueda genética, lo que permitiría explotar aún mas

las potencialidades de una estructuración bajo un enfoque celular, conduciendo a resultados más concluyentes.

Capítulo 7 REFERENCIAS BIBLIOGRÁFICAS

Affenzeller, M. (2004). THE INFLUENCE OF POPULATION GENETICS FOR THE REDESIGN OF GENETIC ALGORITHMS. Inst. Syst. Theory Simul. Johannes Kepler Univ. Altenbergerstrasse 69 - 4040 Linz Austria.

Alba, E., and Dorronsoro, B. (2008). Cellular Genetic Algorithms.

Alba, E., and Troya, J.M. (2000). Cellular evolutionary algorithms: Evaluating the influence of ratio. Int. Con- Ference Parallel Probl. Solving Nat. VI PPSN-VI *1917*, 29–38.

Alba, E., Giacobini, M., Tomassini, M., and Romero, S. (2002). Comparing synchronous and asynchronous cellular genetic algorithms. Int. Conf. Parallel Probl. Solving Nat. VII PPSN-VII *2439*.

Álvarez-Valdés, R., Parajón, A., and Tamarit, J.M. (2002). A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. Comput. Oper. Res. 925}947.

Álvarez-Valdés, R., Parreño, F., and Tamarit, J.M. (2005). A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. YUniversity Valencia Dep. Stat. Oper. Res. 46100 Burjassot Valencia Spain.

Beasley, J.E. (2000). A population heuristic for constrained two-dimensional non-guillotine cutting. Eur. J. Oper. Res. 601–627.

Beraudo, V., Alfonso, H., Minetti, G., and Salto, C. (2004). Constrained Two-Dimensional Non-Guillotine Cutting Problem: an Evolutionary Approach. Comput. Soc.

Bethke, A. (1976). Comparison of genetic algorithms and gradient based optimizers on parallel processors: Efficiency of use of precessing capacity. Tech. Rep. 197 Ann Arbor Univ. Mich. *197*.

Bortfeldt, A., and Winter, T. (2008). A Genetic Algorithm for the Two-Dimensional Knapsack Problem with Rectangular Pieces. Diskussionsbeitrag.

Caprara, A., and Monaci, M. (2003). On the two-dimensional Knapsack Problem. Oper. Res. 5–14.

Chen, Y. (2007). A recursive algorithm for constrained two-dimensional cutting problems. Springer Sci. Media.

Christofides, N., and Whitlock, C. (1977). An Algorithm for Two-Dimensional Cutting Problems. Oper. Res. *1*.

Clautiaux, F., Jouglet, A., and Moukrim, A. (2013). A New Graph-Theoretical Model for the Guillotine-Cutting Problem. Inf. J. Comput. 72–86.

Collins, R.J., and Jefferson, D.R. (1991). Selection in massively parallel genetic algo- rithms. Fourth Int. Conf. Genet. Algorithms ICGA 249–256.

Cui, Y. (2007). An exact algorithm for generating homogenous T-shape cutting patterns. Comput. Oper. Res. *34*, 1107–1120.

Cui, Y., and Chen, Q. (2012). Simple heuristic for the constrained two-dimensional cutting problem. Proc. Inst. Mech. Eng. PART B-J. Eng. Manuf. *226*, 565–572.

Cui, Y., and Huang, B. (2012). Heuristic for constrained T-shape cutting patterns of rectangular pieces. Comput. Oper. Res. 3031–3039.

Cung, V.-D., Hifi, M., and Le Cun, B. (2000). Constrained two-dimensional cutting stock problems a best-First branch-and-bound algorithm. Int. Trans. IN 185–210.

Dolatabadi, M., Lodi, A., and Monaci, M. (2012). Exact algorithmsforthetwo-dimensionalguillotineknapsack. Comput. Oper. 48–53.

Dorronsoro, B., and Alba, E. (2005). The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. IEEE Trans. Evol. Comput. *9*, 126–142.

Dorronsoro, B., and Alba, E. (2007). Diseño e Implementación de Algoritmos Genéticos Celulares para Problemas Complejos. TESIS Dr. Univ. Màlaga.

Dyckhoff, H. (1990). A typology of cutting and packing problems. Eur. J. Oper. Res. 145–159.

Eiben, A.E., and Smith, J.E. (2003). Introduction to Evolutionary Computing. pp. 41–69.

Fayard, D., Hifi, M., and Zissimopoulos, V. (1998). An efficient approach for large-scale two-dimensional guillotine cutting stock problems. J. Oper. Res. Soc. *49*, 1270–1277.

Flores, Á. (2012). UN ALGORITMO GENETICO COOPERATIVO PARA PROBLEMAS DE CORTE Y EMPAQUE.

Garey, M.R., and Johnson, D.S. (1979). Computers and Intractability: A Guide to the Theory od NP-Completeness.

Giacobini, M., Tomassini, M., and Tettamanzi, A. (2003). Modelling selection inten- sity for linear cellular evolutionary algorithms. Proc Int. Conf. Artificial Evol. *2936*, 345–356.

Gonçalves, J.F., and Resende, M.G.C. (2011). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. Springer Sci. Media 180–201.

Gordon, V., Mathias, K., and Whitley, D. (1994). Cellular genetic algorithms as function optimizers: Locality effects. ACM Symp. Appl. Comput. SAC 237–241.

Gorges-Schleuter, M. (1999). An analysis of local selection in evolution strategies. Proc Genet. Evol. Comput. Conf. GECCO *1*, 847–854.

Hifi, M. (1997). An improvement of Viswanathan and Bagchi's exact algorithm for constrained two-dimensional cutting stock. Comput. Oper. Res. *24*, 41–52.

Hifi, M. (2004). Dynamic Programming and Hill-Climbing Techniques for Constrained Two-Dimensional Cutting Stock Problems. J. Comb. Optim. 65–84.

Hifi, M., and M'Halla, R. (2003). An Exact Algorithm for Constrained Two-Dimensional Two-Staged Cutting Problems. Oper. Res. *53*, 140–150.

Hifi, M., and Zissimopoulos, V. (1997). A recursive exact algorithm for weighted two-dimensional cutting. Eur. J. Oper. Res.

Hopper, E. (2000). Two-dimensional Packing utilising Evolutionary Algorithms and other Meta-Heuristic Methods.

Hopper, E., and Turton, B.C.H. (2000). An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. Eur. J. Oper. Res. *128*, 34–57.

Hutter, F., Hoos, H.H., and Leyton-Brown, K. (2009). ParamILS: An Automatic Algorithm Configuration Framework. J. Artif. Intell. Res. 267–283.

Leung, T.W., Chan, C.K., and Troutt, M.D. (2003). Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. Eur. J. Oper. Res. 530–542.

Manderick, B., and Spiessens, P. (1991). A massively parallel genetic algorithm: Imple-mentation and first analysis. Fourth Int. Conf. Genet. Algorithms ICGA 279–286.

Morabito, R., and Arenales, M. (1992). An and-or-graph approach for two-dimensional cutting problems. *58*, 263–271.

Morabito, R., and Arenales, M. (1996). Staged and constrained two-dimensional guillotine cutting problems: An and/orgraph approach. Eur. J. Oper. Res. *94*, 548–560.

Morabito, R., and Pureza, V. (2008). A heuristic approach based on dynamic programming and and/or-graph search for the constrained two-dimensional guillotine cutting problem. Ann Oper Res *179*, 297–315.

Muhlenbein, H., Gorges-Schleuter, M., and Kramer, O. (1988). Evolution algorithms in combinatorial optimization. Parallel Comput. 65–88.

Oliveira, J.F., and Ferreira, J.S. (1990). An improved version of Wang's algorithm for two-dimensional cutting problems. *44*, 256–266.

Ono, T., and Ikeda, T. (1998). Optimizing Two-dimensional Guillotine Cut by Genetic Algorithms. Dept Comput. Sci. Eng. Fukoka Inst. Technol. 3-30-1 Wajiro-Higashi Higashi-Ku Fukoka 811-0295 Jpn.

Robertson, C. (1987). Parallel implementation of genetic algorithms in a classifier system. Proc Second Int. Conf. Genet. Algorithms ICGA 140–147.

Romero, F. (2003). Un Algoritmo Genético Paralelo para Resolver el Problema de Corte de Piezas Guillotina Bidimensional Restricto.

Sarma, J., and De Jong, K.A. (1995). On decentralizing selection algorithms. Sixth Int. Conf. Genet. Algorithms ICGA 17–23.

Sarma, J., and De Jong, K.A. (1996). An analysis of the effect of the neighborhood size and shape on local selection algorithms. HM Voigt W Ebeling Rechenberg HP Schwefel Ed. Proc Int. Confer- Ence Parallel Probl. Solving Nat. IV PPSN-IV *1141 of Lecture Notes in Computer Science (LNCS)*, 236–244.

Simoncini, D., Verel, S., Collard, P., and Clergue, M. (2006). Anisotropic selection in cellular genetic algorithms. Genet. Evol. Compu- Tation Conf. GECCO 559–566.

Sprave, J. (1999). A unified model of non-panmictic population structures in evolu- tionary algorithms. IEEE Int. Conf. Evol. Comput. CEC *2*.

Tschoke, S., and Holthofer, N. (1995). A new parallel approach to the constrained two-dimensional cutting stock problem. Proc. Second Int. Workshop 285–300.

Vasko, F.J. (1989). A computational improvement to Wang's two-dimensional cutting stock algorithm. Comput. Ind. Eng. *16*, 109–115.

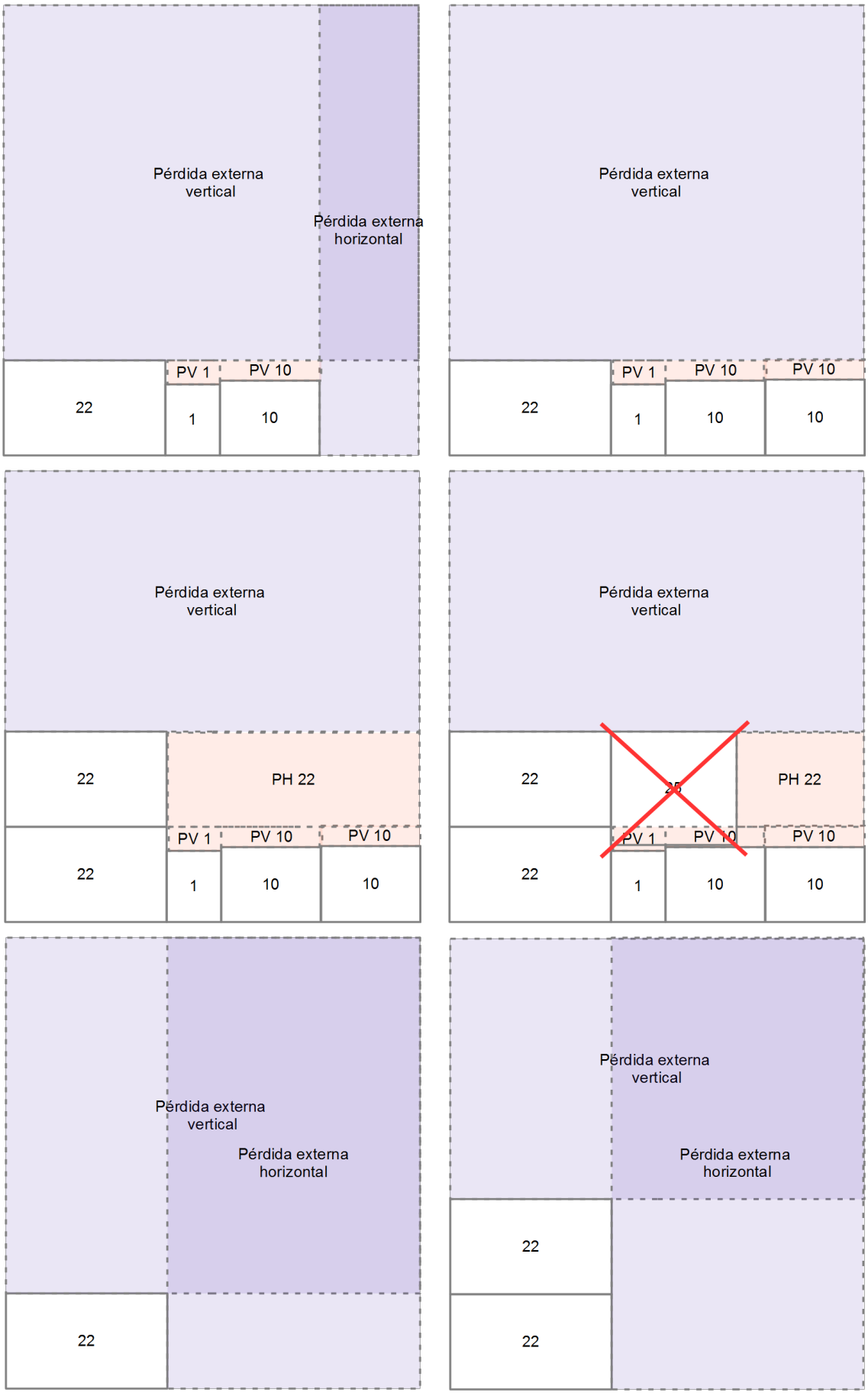
Viswanathan, K.V., and Bagchi, A. (1993). An Exact-Best-First Search Procedure for the Constrained Rectangular Guillotine Knapsack Problems. Proc. Am. Assoc. Artif. Intell. 145–149.

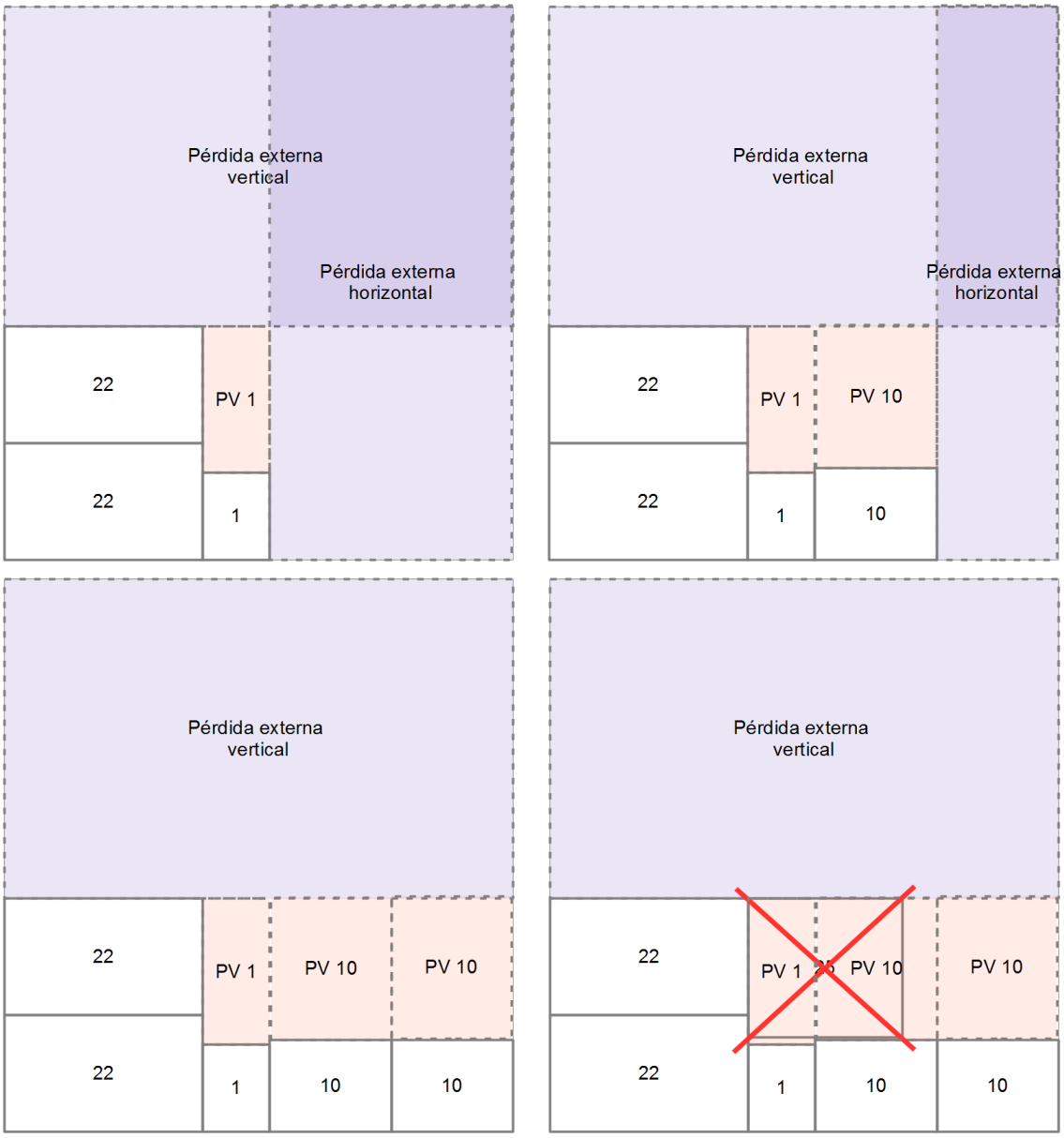
Wang, P.Y. (1983). Two Algorithms for Constrained Two-Dimensional Cutting Stock Problems. Oper. Res.

Wascher, G., Haubner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. Eur. J. Oper. Res.

Whitley, D. (1993). Cellular genetic algorithms. Forrest Ed. Proc Fifth Int. Conf. Genet. Algorithms ICGA 658.

Yoon, K., Ahn, S., and Kang, M. (2013). An improved best-first branch-and-bound algorithm for constrained two-dimensional guillotine cutting problems. Int. J. Prod. Res. *226*, 565–572.


Pérdida externa vertical
Pérdida externa horizontal
22
PV 1
PV 10
1
10
Pérdida externa vertical
22
PV 1
PV 10
PV 10
1
10
10
Pérdida externa vertical
22
PH 22
22
PV 1
PV 10
PV 10
1
10
10
Pérdida externa vertical
22
PH 22
22
PV 1
PV 10
PV 10
1
10
10
Pérdida externa vertical
Pérdida externa horizontal
22
Pérdida externa vertical
Pérdida externa horizontal
22
22


Pérdida externa vertical
Pérdida externa horizontal
22
22
PV 1
1
Pérdida externa vertical
Pérdida externa horizontal
22
22
PV 1
PV 10
1
10
Pérdida externa vertical
22
22
PV 1
PV 10
PV 10
1
10
10
Pérdida externa vertical
22
22
PV 1
PV 10
PV 10
1
10
10