Contenido

[Capítulo 1 INTRODUCCIÓN 1](#_Toc376970028)

[1.1 ANTECEDENTES Y MOTIVACIÓN 1](#_Toc376970029)

[1.2 DESCRIPCIÓN DEL PROBLEMA 3](#_Toc376970030)

[1.3 SOLUCIÓN PROPUESTA 12](#_Toc376970031)

[1.3.1 Características de la solución 12](#_Toc376970032)

[1.3.2 Propósito de la solución 13](#_Toc376970033)

[1.4 OBJETIVOS Y ALCANCES DEL PROYECTO 13](#_Toc376970034)

[1.4.1 Objetivo general 13](#_Toc376970035)

[1.4.2 Objetivos específicos 13](#_Toc376970037)

[1.4.3 Alcances 14](#_Toc376970045)

[1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS 14](#_Toc376970047)

[1.5.1 Metodología y herramientas a utilizar 14](#_Toc376970048)

[1.5.3 Ambiente de desarrollo 15](#_Toc376970049)

[1.6 ORGANIZACIÓN DEL DOCUMENTO 15](#_Toc376970050)

[Capítulo 2 ESTADO DEL ARTE 17](#_Toc376970051)

[2.1. EL PROBLEMA DE CORTE DE PIEZAS GUILLOTINABLE BIDIMENSIONAL RESTRICTO (CTDC) 17](#_Toc376970052)

[2.2. ALGORITMOS GENËTICOS APLICADOS AL PROBLEMA DE CORTE DE PIEZAS GUILLOTINABLE BIDIMENSIONAL RESTRICTO 25](#_Toc376970053)

[2.3. ALGORITMOS GENÉTICOS CELULARES 33](#_Toc376970054)

[2.4 RESUMEN 38](#_Toc376970055)

[Capítulo 3 MATERIALES Y MÉTODOS 39](#_Toc376970056)

[3.1 ALGORITMOS GENÉTICOS 39](#_Toc376970057)

[3.1.1 Representación de individuos 40](#_Toc376970058)

[3.1.2 Operadores genéticos 41](#_Toc376970059)

[3.2 ALGORITMOS GENÉTICOS CELULARES 46](#_Toc376970060)

[3.3 MODELAMIENTO DEL PROBLEMA 46](#_Toc376970061)

[3.2.1 Genotipo-fenotipo 46](#_Toc376970062)

[3.2.2 Representación 47](#_Toc376970063)

[3.2.3 Función Constructora 50](#_Toc376970064)

[3.2.4. Heurística de colocación 53](#_Toc376970065)

[3.2.5 Función objetivo 57](#_Toc376970066)

[3.2.6 Efectos de los operadores genéticos sobre la representación adoptada 57](#_Toc376970067)

[3.2 DATOS DE PRUEBA 61](#_Toc376970068)

[3.4 CONFIGURACIÓN DE ALGORITMOS 61](#_Toc376970069)

[3.4.1 ParamILS 62](#_Toc376970070)

[3.4.2 Plan de configuración y diseño de escenarios de sintonización 63](#_Toc376970071)

[3.4 DISEÑO EXPERIMENTAL 66](#_Toc376970072)

[3.5 HARDWARE Y SOFTWARE UTILIZADO 67](#_Toc376970073)

[Capítulo 4 RESULTADOS COMPUTACIONALES Y ANÁLISIS 67](#_Toc376970074)

[Capítulo 5 CONCLUSIONES 81](#_Toc376970077)

[Capítulo 6 REFERENCIAS BIBLIOGRÁFICAS 81](#_Toc376970078)

# INTRODUCCIÓN

## 1.1 ANTECEDENTES Y MOTIVACIÓN

El problema de corte en 2D restricto pertenece a la clase NP-Hard. Este es un problema muy interesante desde un punto de vista tanto teórico como práctico, que ha sido estudiado durante varios años, por lo que mucho conocimiento se ha generado en torno a él y muchas formas de abordarlo 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.

Las capacidad de los AG celulares...

Existe evidencia de que un enfoque celular supera enfoques no estructurados en varios problemas académicos, como . Sin embargo no se han encontrado evaluaciones en problemas de carácter geométrico los cuales poseen una gran dureza, ampliamente reconocida ...donde las capacidades de ...Por otra parte existe evidencia de un enfoque distribuido supera el comportamiento de un AG secuencial, sin embargo comparaciones orientadas a ...

Sin embargo, hasta ahora

Por otra parte, en la literatura usualmente conocimiento específico del problema es utilizado para generar soluciones competitivas. Es interesante preguntarse si los conceptos celulares por si solos, permiten a la evolución genética mejorar el aprendizaje de reglas que permitan proveer resultados comparables a los obtenidos mediante dicho conocimiento específico. Lo cual podría ser observable en las características de las mejores soluciones evolucionadas.

Sin embargo para varias instancias simples ninguno de los AG fue capaz de alcanzar soluciones óptimas. Lo que lleva a pensar que un sesgo en la representación, incapaz de generar ciertas secuencias de piezas, impide que se visiten ciertas regiones del espacio, lo que al mismo tiempo no haya permitido al AG, y en especial al AG celular, alcanzar mejores soluciones.

Por otra parte, distintos strings binarios eran interpretados como las mismas secuencias de piezas por lo que esto también tiene efectos negativos en la diversidad de población, que usar directamente una representación de permutación para las secuencias de piezas.

## 1.2 DESCRIPCIÓN DEL PROBLEMA

Los problemas de corte y empaque pertenecen a una antigua y bien conocida familia, llamada *C&P* (cutting and 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 and 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, (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 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 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 , 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*.



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

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.

Dentro de los problemas de corte en *2D*, una variante clásica es el *TDC* (two-dimensional cutting) (Hifi and 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 . Según esta tipología, el *TDC* cae dentro del tipo de problema de *maximización de output* (Gonçalves and 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. Por otra parte, 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:

1. el problema de cortar piezas más pequeñas de una placa rectangular
2. 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 and Monaci, 2003), el cual es fuertemente *NP-duro* (Garey and 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 pi para el ítem i. Un patrón de corte puede por lo tanto, contener como máximo pi ítems de tipo i.
* Doblemente restricto (Doubly Constrained) (*DCTDC*): Se fija un límite superior pi y un límite inferior qj. Un patrón de corte puede contener como máximo pi y como mínimo qj 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 bi, 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 and Huang, 2012). De hecho, a menudo es usado como un problema auxiliar del C*TDC* (Hifi and 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 actual.

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. Por su parte, 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).

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.



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

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 and 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*](http://paginas.fe.up.pt/~esicup/tiki-index.php).

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 a 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 and 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 and 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, por el contrario patrones más complejos conducen a una mejor utilización del material.

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 1.8 *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 lineal 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 con patrones de guillotina generales, sin una restricción de k-etapas.

La variante que se va a abordar en este trabajo de titulación es el *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). (K. Yoon 2012) 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 *li* *x* *wi*, valor vi (equivalente a su área *si=li\*wi*), y un límite superior de demanda *bi*, *i=1,...,m*. El objetivo es cortar la placa en *xi* piezas del tipo *i*, de manera que *0<=xi<=bi, i=1,...,m* y la utilidad total 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*, *li*, *wi*, y *bi*, i=1,...,m, son enteros no negativos

El vector (x1,...,xn) corresponde a un patrón de corte, si es posible producir xi 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 de corte de piezas mediante un enfoque evolutivo. Existen varias formas de hacer esto, sin embargo, dado que se desea evaluar el comportamiento en la búsqueda sobre el espacio de soluciones, el modelo debe estar pensado para permitir que el algoritmo tenga la capacidad de explorar cualquier región del espacio de búsqueda, guiándose solamente por la evolución, tanto en un contexto de no estructuración como estructuración de población. Para promover la reducción del sesgo en la búsqueda de soluciones, 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.

Para determinar si la estructuración de la población mediante un enfoque celular permite a un AG obtener resultados más competitivos versus el uso de otros enfoques conocidos, se han seleccionado distintas familias de AG canónicos: dos familias con poblaciones no estructuradas: AG generacional y AG steady-state y una familia de AG í. Luego se debe diseñar cada uno de los AG propuestos, seleccionando los operadores genéticos más apropiados para el problema en cuestión dada la representación adoptada.

### 1.3.2 Propósito de la solución

El propósito de este trabajo es evaluar el desempeño computacional de un algoritmo genético cuya población se estructura bajo conceptos celulares, y de esa manera, determinar si este enfoque 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

Estudiar 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

1. Realizar un estado del arte sobre algoritmos genéticos celulares
2. Realizar un estado del arte sobre algoritmos genéticos aplicados al problema en estudio o similares
3. Diseñar e implementar las representaciones y funciones constructoras para el problema en estudio
4. Diseñar e implementar un algoritmo genético y unalgoritmo genético celular que resuelvan el problema en estudio
5. Identificar conjuntos de instancias de prueba
6. Diseñar un experimento computacional que permita evaluar los algoritmos propuestos
7. Analizar los resultados obtenidos

### 1.4.3 Alcances

En este trabajo, dos modelos de población en *AG* son comparados, analizando tanto su comportamiento numérico como en la búsqueda de soluciones. Para ello, se diseña cada uno de los *AG* a evaluar. Además, se analizan los efectos de la estructuración de la población bajo un enfoque celular, sobre la búsqueda que realiza el *AG*.

## 1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS

### 1.5.1 Metodología y herramientas a utilizar

En este trabajo se utiliza una plataforma computacional, *JCELL*..., para evolucionar individuos (soluciones) 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 alcanzadas por cada enfoque, en términos de precisión o proximidad a la solución óptima. Los resultados obtenidos mediante el experimento computacional son analizados. Para sustentar las hipótesis de trabajo se realizan pruebas estadísticas apropiadas a la distribución de los datos. Los test estadísticos son realizados mediante la suite estadística *IBM SPSS*.

### 1.5.3 Ambiente de desarrollo

## 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 presenta... En él, quedan al descubierto las respectivas brechas del conocimiento que justifican la realización de este trabajo. El Capítulo 3, describe el procedimiento experimental realizado. En éste se enuncian los conceptos de algoritmos genéticos, operadores y estructuración de población, centrándose en el enfoque celular. En el Capítulo 4, se detallan los resultados obtenidos para determinar un tamaño mínimo de población y se analiza la eficiencia y calidad del proceso con respecto a las principales variables de interés. El Capítulo 5, presenta las respuestas a las dos preguntas de investigación, mediante el análisis de los resultados obtenidos al usar distintos conjuntos de entrenamiento o casos de fitness y la robustez de los algoritmos frente a instancias grandes. Finalmente, el Capítulo 6 presenta las conclusiones del estudio, donde se resumen los principales hallazgos y se enumeran las consideraciones para trabajos futuros.

# ESTADO DEL ARTE

## 2.1. EL PROBLEMA DE CORTE DE PIEZAS GUILLOTINABLE BIDIMENSIONAL RESTRICTO (CTDC)

El estado del arte del *TDC* es muy extenso y variado; en la literatura existen 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 varios estudios se han enfocado en clases especiales de patrones de corte, tales como los: *p-group*, *k-staged* (Scheithauer 2004, Alves 2010), *t-shaped*. Sin embargo, pocos estudios han considerado algoritmos exactos para el *CTDC* (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.



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

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

El *CTDC* ha sido resuelto de manera exacta mediante el uso de ambos enfoques. (Christofides and 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 and Zissimopoulos, 1997) propusieron un algoritmo mejorado, el cual utilizaba *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. El enfoque *bottom-up*, propuesto originalmente por (Wang, 1983), fue generalizado por (Viswanathan and 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 prometedor, utilizando para ello, una estimación del *upper-bound* relativo a *P[[1]](#footnote-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-fist* requiere más memoria, pero menor tiempo de cómputo, 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 *upper-bounds* más estrictos. Además obtiene soluciones óptimas para instancias de larga escala, previamente desconocidas.



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

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*

El modelo es embebido en un esquema de *Constraint Programming*. El cual busca un conjunto adecuado de arcos. *Constraint Programming* es un paradigma orientado a problemas combinatoriales que pueden ser descritos por un conjunto de variables, un conjunto de valores para cada variable, y un conjunto de restricciones entre las variables. Su potencial radica en un proceso denominado “propagación de restricción”, donde se realizan ciertas deducciones que reducen el esfuerzo computacional. El algoritmo es evaluado en el problema de strip cutting, en el cual el ancho es fijo..................

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. 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 *Tp<βWH*, donde *Tp* 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 and 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 and Arenales, 1992) presentan un enfoque basado en el modelo *And/ord-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 cortesy estrategias *hill-climbing* para reducir la búsqueda. En este trabajo resuelven instancias de larga escala para el *UTDC*. Unos años más tarde, (Morabito and 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 and 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 and 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* (ver ), 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 GENËTICOS APLICADOS AL PROBLEMA DE CORTE DE PIEZAS GUILLOTINABLE BIDIMENSIONAL RESTRICTO

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.

Según (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); aplicaron un algoritmo genético simple (*GA*) al 2D-KP, usando un enfoque en dos etapas, en 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 *layout* válido, o fenotipo. Este trabajo, también constituye una 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*. 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 un momento maduro 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 and 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: algortimo 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 layout 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 layout dentro de la estructura de datos del AG, aunque se necesitan ciertas reglas adicionales para fijar la posición en el layout. Por ejemplo, (Ono and Ikeda, 1998) proponen un GA que utiliza una parte binaria, representando los operadores binarios V o H, correspondientes a combinaciones verticales u horizontales respectivamente, y una parte numérica, representando las piezas. Así, el cromosoma es interpretado como un árbol binario, leído en notación post-fija (ver figura 2.7). 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 layout, utilizando el enfoque *bottom-up*



FIGURA 2.8 *Cromosoma y su interpretación como layout guillotinable*

Otra representación usando este enfoque se encuentra en (Beasley, 2000), donde se 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 layout 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 enfoques, un tercer enfoque traslada el proceso de búsqueda genética al dominio de layouts. Debido a que las operaciones genéticas son realizadas directamente en los layouts, 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 enfoque la representación debe hacerse cargo de las restricciones. Usualmente alguno de los siguiente mecanismos son usados para ello: rechazo, reparación o penalización. Un ejemplo de este enfoque se encuentra en (Bortfeldt and 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 2.8). 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 tiene 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.

(Wang, 2010) propone un AG adaptivo para el *2D-KP*. Su enfoque es distinto al resto, ya que formula el problema como un problema de *factores atractivos dinámicos*, donde, dadas *n* piezas que están esperando en el layout, el objetivo es poner tantos bloques rectangulares como sea posible. El requisito es que la tasa de utilización del área sea la mayor posible y que los bloques rectangulares no se superpongan. Se define una función de posicionamiento de bloques, cuyos diversos parámetros, determinan diferentes ubicaciones de los bloques rectangulares, lo cual genera diferentes layouts. Así, el problema es transformado a un problema de optimización de parámetros. Utiliza probabilidades de cruzamiento y mutación adaptivas. La selección usa elitismo, y como operador usa selección por ruleta. Para el cruzamiento, en las primeras generaciones individuos al azar son apareados. Posteriormente, solo se aparean los individuos que sean suficientemente diferentes entre si. Para ello se usa distancia de hamming. Para la mutación, la probabilidad de mutaciòn es en funciòn del fitness del individuo y los fitness maximo y promedio de la población, de manera que individuos superiores al promedio experimentan...

Desde una perspectiva actual, una de las aproximaciones evolutivas más recientes es (Gonçalves and Resende, 2011), donde proponen 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 rectángulo. 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 distribuídas; si están concentradas en una zona, será mas 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 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.



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

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 heuristica 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 de la busqueda realizada por un algoritmo genético, para ciertos escenarios y tipos de problemas.

Los algoritmos evolutivos celulares fueron diseñados inicialmente para funcionar en máquinas masívamente 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 and 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 and 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. Mas 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. Mas 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.

(Giacobini et al., 2005) propusieron algunos modelos matemáticos para aproximar las curvas de crecimiento de cGAs con poblaciones donde la topología es definida como un grafo aleatorio, donde la distancia entre dos individuos es en general mucho menor que en el caso de las

(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 and 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 and 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. Mas 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 linea, (Dorronsoro and 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 and 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 and 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 and 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.

A pesar de que existen diversas extensiones y mejoras al modelo canónico de cGAs, aqui solo se presentan los relacionados al caso simple. Sin embargo la gihu

## 2.4 RESUMEN

Del estado del arte se concluye 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 a menudo exhiben propiedades geométricas que se traducen en buenas soluciones, como por ejemplo los patrones de bloque.... También, a menudo se usan heurísticas basadas en el preordenamiento de las piezas en función de sus características geométricas, como su área, ancho, largo, etc. Por lo que una primera pregunta que podriamos formularnos, es si en un escenario en que no se cuente con estas nociones, en este caso de la geometría del problema, ¿podría la evolución por si sola revelarnos este conocimiento? ¿podrán los conceptos celulares promover la confiabilidad en la extracción de dicho conocimiento?

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

Por otra parte, se ha reconocido la existencia de problemáticas en el comportamiento de la búsqueda respecto a la convergencia del algoritmo, y han habido diferentes formas 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 de manera exitosa. Los problemas quizá más relacionados son *TSP* y *VRP*. Aunque para estos problemas se usaron enfoques híbridos, combinandolo con operadores de búsqueda local. Una característica del enfoque celular, es que gracias a permite facilmente modificar el comportamiento en la búsqueda del algoritmo,

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

# MATERIALES Y MÉTODOS

Este capítulo detalla la metodología utilizada, así como los pasos experimentales que permiten asegurar la replicabilidad de la experimentación computacional realizada en este trabajo.

## 3.1 ALGORITMOS GENÉTICOS

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 *AG* 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 and 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 cuestión. 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).

Como se mencionó anteriormente, las 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 la representación genética de una solución candidata para el problema, 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 and Smith, 2003).

#### Representación binaria

Históricamente, esta es una de las primeras representaciones, y muchos AG la han adoptado, erróneamente, de manera independiente al problema que han tratado de resolver. Aqui, el genotipo consiste en un string binario, en donde, para una aplicación particular, se debe decidir cómo 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, lo cual usualmente es un error, y mejores resultados se pueden obtener mediante el uso de otros tipos de representación.

#### 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.

1. 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. En el caso lineal la probabilidad de seleccionar un individuo *i* es:



Figura 3.13 *Probabilidad de seleccionar individuo i mediante 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.

1. 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 probabilidad de seleccionar un individuo es proporcional a su *fitness*:

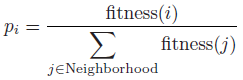


Figura 3.14 *Probabilidad de seleccionar individuo i mediante ruleta*

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 *pi*supera a *r*, entonces es seleccionado.

1. 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]](#footnote-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.

1. 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.

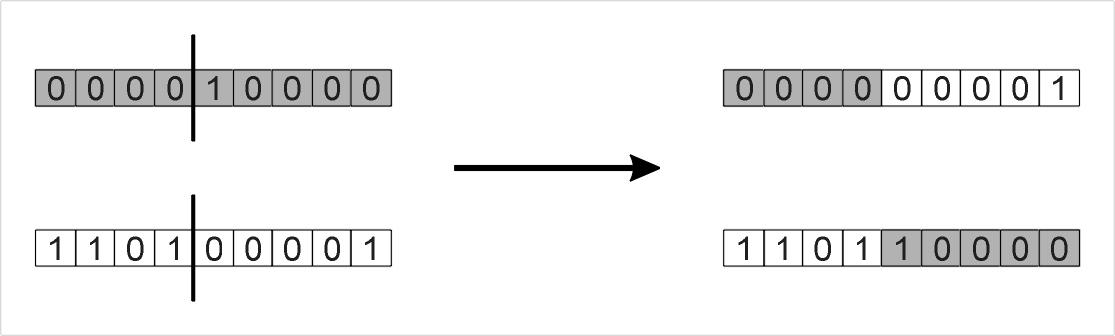


Figura 3.15 *Cruzamiento de 1 punto*

1. 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.

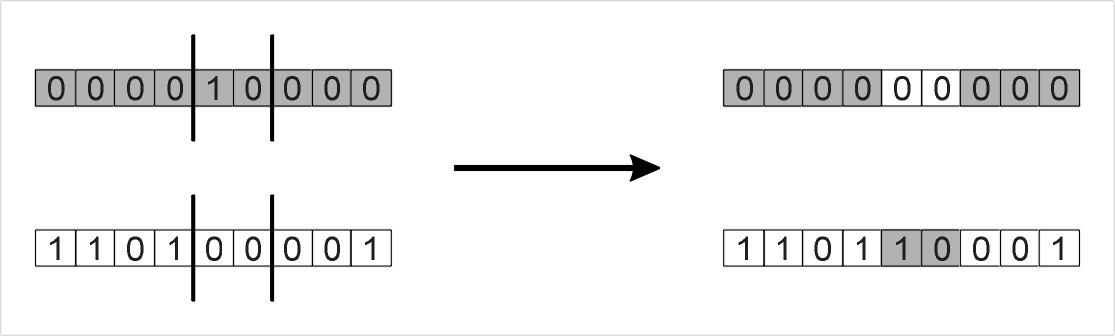


Figura 3.16 *Cruzamiento de n puntos; n=2*

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.

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

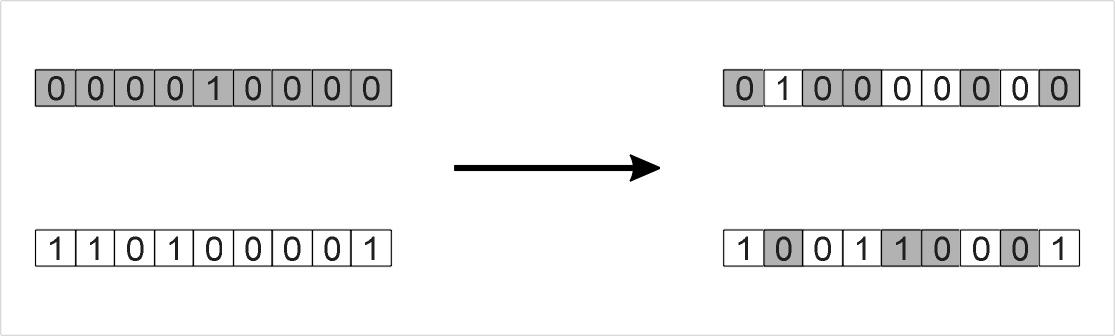


Figura 3.17 *Cruzamiento uniforme*

A diferencia del cruzamiento de n puntos, 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).

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

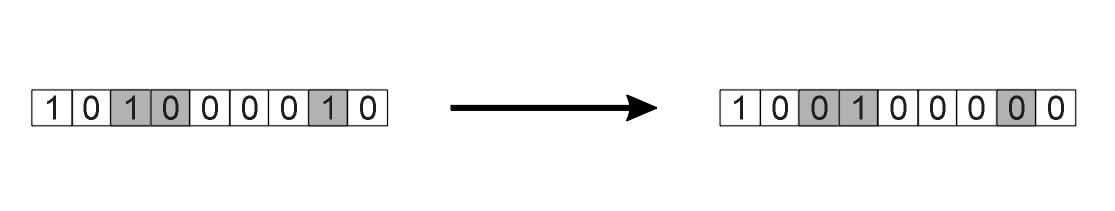


Figura 3.18 *Mutación binaria*

Los operadores genéticos explicados en esta sección, corresponden a los más generales y pueden ser usados para cualquier tipo de representación. Sin embargo, también existen operadores específicos, más apropiados para cada una de ellas. Una revisión de estos puede ser encontrada en (Eiben and Smith, 2003).

## 3.2 ALGORITMOS GENÉTICOS CELULARES

## 3.3 MODELAMIENTO DEL PROBLEMA

### 3.2.1 Genotipo-fenotipo

Los *AG* que abordan problemas de *C&P*, a menudo utilizan un enfoque en dos etapas 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 and Turton, 2000). Más allá de esto, la principal razón por la cual se 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 el enfoque en dos etapas, 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 para el problema. El esquema general es descrito en la figura 3.2, 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.



FIGURA 3.2 *Esquema general: Función constructora + heurística colocación*

### 3.2.2 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]](#footnote-3). Para ello, el cromosoma se divide en dos partes: un cromosoma de piezas, que decide cuáles piezas estarán presentes en la secuencia, y un cromosoma de ordenamiento, el cual codifica qué orden tienen las piezas en la secuencia.

Para el cromosoma de piezas 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 el largo del cromosoma de piezas 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 3.3, 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.

|  |  |
| --- | --- |
| FIGURA 3.3 *Instancia de ejemplo problema* | FIGURA 3.4 *Cromosoma de piezas e interpretación* |

Las piezas son serializadas en un arreglo que mantiene cada una de las copias, y que es usado como referencia para interpretar cada gen del cromosoma. En la figura 3.4 se ilustra la interpretación de un cromosoma de ejemplo, según el cual: la 1a copia de la pieza de tipo 1 está presente en la secuencia, la 2a y 3a copias no lo están, la 1a copia de tipo 2 si lo está, etc.

Los genes de ordenamiento determinan el orden de las piezas mediante una lectura del cromosoma de piezas. 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 3.5. 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 del cromosoma de 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 . 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]](#footnote-4). La interpretación completa del genotipo se describe mediante el cromosoma del ejemplo: Las 1as copias de las piezas de tipo 1, 2, 3, 4, y la 1a y 3a 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 3.5 *Cromosoma completo y su composición* | FIGURA 3.6 *Interpretación conceptual del cromosoma como "Ruleta de piezas"* |

### 3.2.3 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 obtiene el cromosoma de piezas y los parámetros necesarios para a continuación proceder con su lectura.

A continuación se muestra el pseudocódigo de la función constructora, que recibe como parámetro el genotipo, además existe una estructura global “piezas” que 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 el ciclo de saltos actual.

funcionConstructora(genotipo)

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;

Para explicar el funcionamiento de la función constructora, se incluye la traza para la solución candidata de ejemplo, la cual es ilustrada en la figura 3.7. 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, la 1a pieza leída es la 3, la cual está presente, por lo tanto se agrega a la secuencia ordenada, 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.

FIGURA 3.7 *Traza función constructora para solución candidata de ejemplo*

|  |  |  |  |
| --- | --- | --- | --- |
|  | | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  | |  | |
|  | | | |

## 3.2.4. 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 . 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 3.9.

|  |  |
| --- | --- |
| FIGURA 3.8 *Combinación vertical de las piezas A y B* | FIGURA 3.9 *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 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 3.9 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 la . 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 . 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 3.10 *Pérdidas secantes asociadas a la pieza A* | FIGURA 3.11 *Reducción de pérdida horizontal A, producto de una inserción* |

La heurística de colocación recibe como parámetro el fenotipo, (dado por una secuencia de piezas), y retorna un layout (consistente en un patrón guillotinable). El layout 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, 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]](#footnote-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.

El siguiente es el seudocódigo de la heurística de colocación:

heuristicaColocacion(fenotipo)

1. cursor ← 0

2. **Mientras**(cursor < fenotipo.size()) **hacer**

3. pieza ← fenotipo[cursor];

4. **Si**(placa.vacia)

5. **Si**(pieza.cabe(placa))

6. insertarPieza(layout,pieza,placa);

7. actualizarPerdidasExternas();

8. **fin-si**

9. **Si-no**

10. perdidaInterna ← firstFit(pieza,listaPerdidasInternas);

11. **Si**(perdidaInterna != null)

12. insertarPieza(layout,pieza,perdidaInterna);

13. actualizarPerdidasInternas();

14. **Si-no**

15. perdidaExterna ← firstFit(pieza,listaPerdidasExternas)

16. **Si**(perdidaExterna != null)

17. insertarPieza(layout,pieza,perdidaExterna);

18. actualizarPerdidasExternas();

19. **Fin-Si**

20. **Fin-Si**

21. **Fin-Si**

22. cursor ← cursor + 1;

23. **Fin-Mientras**

24. devolver layout;

Para explicar el funcionamiento de la heurística, se incluye la traza para la solución candidata de ejemplo, la cual es ilustrada en la figura 3.10. 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.

FIGURA 3.12 *Traza heurística colocación para solución candidata de ejemplo*

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 3.2.5 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 3.13 *Función objetivo y fitness de la solución candidata*

### 3.2.6 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 ejemplo de esta dificultad, es que generalmente, solo los primeros genes del cromosoma son 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 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 3.4 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 3.14 *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 3.5.



FIGURA 3.15 *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 genes, 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 3.6, 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 3.16 *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 3.7, 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 prometedoras.



FIGURA 3.17 *Efecto de la mutación en un gen de ordenamiento para la representación adoptada*

## 3.2 DATOS DE PRUEBA

## 3.4 CONFIGURACIÓN DE ALGORITMOS

Muchos algoritmos de alto rendimiento poseen parámetros cuyos ajustes controlan importantes aspectos de su comportamiento. Este es el caso de los procedimientos heurísticos, usados para resolver problemas computacionalmente difíciles. Hay muchos reconocimientos en la literatura de que la búsqueda de configuraciones que optimicen el rendimiento de estos algoritmos a menudo requiere un esfuerzo considerable. En muchos casos, esta tediosa tarea es realizada manualmente de manera ad-hoc, por lo tanto su automatización es de gran importancia práctica en varios contextos (Hutter et al., 2009).

El problema de configuración de algoritmos puede ser enunciado informalmente como: Dado un algoritmo, un conjunto de parámetros para el algoritmo y un conjunto de datos de entrada, encontrar los valores para los parámetros bajo los cuales el algoritmo logre el mejor rendimiento posible con los datos de entrada. Para diferenciar los algoritmos cuyo rendimiento debe ser optimizado de los algoritmos utilizados para esa tarea, los primeros son denominados algoritmos objetivo, mientras que los últimos son llamados configuradores.

La figura 3.16 ilustra el problema de configuración. 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 3.18 *Problema de configuración de algoritmos*

Para este experimento se tienen varios algoritmos objetivo, correspondientes a cada uno de los AG propuestos para la evaluación, por lo que debe resolverse un problema de configuración por cada uno de ellos. Por otra parte, las instancias corresponden al conjunto de instancias seleccionado, las cuales son descritas en la sección x.x. El configurador corresponde a la herramienta de optimización de parámetros a utilizar, la cual es descrita a continuación.

### 3.4.1 ParamILS

*ParamILS* es un framework para la optimización de parámetros, el cual utiliza la metaheurística basada en trayectoria, Iterated Local Search (de ahí su nombre). 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. Para un… referencias

### 3.4.2 Plan de configuración y diseño de escenarios de sintonización

Distintos algoritmos objetivo, correspondientes a cada AG propuesto, 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 algoritmo. Para mantener la consistencia en dichas comparaciones se define un plan de configuración, el cual prioriza los algoritmos con una menor cantidad de parámetros, comunes para todos ellos. De este modo, la configuración de un algoritmo posterior asumirá el ajuste de los parámetros previamente obtenidos, y solo requerirá la obtención de parámetros adicionales y específicos para dicho algoritmo.

La tabla 3.1 muestra los valores posibles que se consideraron 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 los AG no estructurados poseen parámetros comunes a todos, uno de ellos, el AG-generacional, será el primero en ser configurado, de cuyo resultado se obtendrá la totalidad de parámetros para el AG steady-state y parte de los parámetros para los AG estructurados. En particular, para el AG-distribuido, una previa obtención del tamaño de la población permite fijar el tamaño de las islas, el cual se obtiene dividiendo el tamaño de la población por la cantidad de islas. En el caso del AG-celular, una previa obtención del tamaño de población permite fijar el área de la cuadrícula bidimensional o enrejado[[6]](#footnote-6), de manera que los valores posibles para este parámetro son las dimensiones del enrejado. Además se añaden la cardinalidad de los espacios de configuración, esto permite estimar los tiempos de espera para configurar cada algoritmo.

TABLA 3.1 *Valores posibles para cada parámetro de cada algoritmo objetivo*

|  |  |  |
| --- | --- | --- |
|  | *AG generacional* | *AG celular* |
| *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 | ✔ |
| *Enrejado* | ─ | 20x20; 10x40; 5x80; 4x100; 2x200; 1x400 |
| *Vecindad* | ─ | Linear5; Compact9; Compact13 |
| *política actualización* | ─ | ls; frs; nrs; uc; ss; synchronous |
| *# total configuraciones* | 4500 | 108 |

*ParamILS* no restringe los valores de los parámetros a dominios únicamente numéricos, por el contrario, asume que todos los parámetros son categóricos. Gracias a esta característica se podria definir por ejemplo un parámetro para seleccionar entre heurísticas de búsqueda. Para los algoritmos objetivo se puede definir varios parámetros categóricos, como los distintos operadores genéticos, y parámetros específicos del AG-celular. Se aprovechó el hecho de que varios de estos se encuentran implementados en la plataforma evolutiva utilizada (*JCell*), y fueron incluidos en el escenario de sintonización.

*ParamILS* obtiene desde un archivo toda la información necesaria para realizar la configuración de parámetros, este archivo es llamado *escenario de* sintonización. En la se especifican los principales parámetros de un *escenario de sintonización,* y cómo fueron definidos para la configuración de los algoritmos objetivo.

TABLA 3.2 *Definición de* *parámetros del escenario sintonización paramILS*

|  |  |
| --- | --- |
| *Parámetro* | *Descripción* |
| *algo* | Un algoritmo ejecutable o una llamada a un script de envoltura en torno al algoritmo, que cumpla con el formato de entrada/salida de ParamILS. Para este trabajo se desarrolló un script de envoltura, el cual entre otras cosas 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 empíricamente 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 empíricamente 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. |

TABLA 3.3 *Valores obtenidos por ParamILS para cada algoritmo objetivo*

|  |  |  |
| --- | --- | --- |
|  | *AG generacional* | *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 | ✔ |
| *enrejado* | ─ | 1x400 |
| *vecindad* | ─ | Compact9 |
| *política actualización* | ─ | ls; frs; nrs; uc; ss; synchronous |

## 3.4 DISEÑO EXPERIMENTAL



## 3.5 HARDWARE Y SOFTWARE UTILIZADO

# RESULTADOS COMPUTACIONALES Y ANÁLISIS

De acuerdo con los objetivos del estudio, este capítulo compara 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 4.1)
* 6 instancias difíciles (tabla 4.2)
* 10 instancias de tamaño grande (tabla 4.3)

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 4.1 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 todas los casos, *AGc* presenta un *Best* superior a *AG*, a excepción de 2 instancias, 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 4.1  *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 | \*13040 | \*12796 | 1,46 |
| CHL1s | 132 | 100 | 30 | 13099 | 12846 | 12807 | 2,23 | \*13014 | \*12839 | 1,99 |
| CHL2s | 62 | 55 | 10 | 3279 | 3280 | 3280 | 0,00 | \*3284 | 3280 | 0,00 |
| CHL3s | 157 | 121 | 15 | 7402 | \*13015 | 12487 | 0,00 | 12966 | \*12493 | 0,00 |
| CHL4s | 207 | 231 | 15 | 13932 | 24790 | 25088 | 0,00 | \*25978 | \*25216 | 0,00 |
| CHL5 | 20 | 20 | 10 | 390 | 391 | 390 | 0,00 | 391 | \*391 | 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 | \*247392 | \*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 o superan los mejores valores conocidos. Más precisamente, 4 de estas instancias superan los mejores conocidos. En particular, para las instancias *CHL3s* y *CHL4s* ambos algoritmos superan ampliamente el mejor valor conocido, al extremo de duplicarlo. Estos casos, aparentemente anómalos, pueden deberse a errores de tipeo en el óptimo o en alguna de las dimensiones de la placa, e introducen ruido para el análisis estadístico. Es por esto que, para ellos, se asumió un %*GAP* igual a 0. Las dos instancias restantes, CHL2s y CHL5, superan ligeramente la mejor conocida. *AGc* supera la mejor conocida para una instancia adicional (*A5*), en la que *AG* no lo hace. En general todos los casos donde se alcanzó el óptimo, el *Best* es muy cercano o igual al *Mean*, por lo que son instancias en donde los *AG* 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 9 instancias iguales, y para las restantes 26 inferiores. La diferencia en el *GAP* alcanza un valor máximo de 1,98% para la instancia *CU2*, siendo en promedio 0,47%. En promedio el *GAP* obtenido por *AG* equivale a 1,72% y el obtenido por *AGc* equivale a un 1,25%. Por lo tanto *AGc* es capaz de obtener resultados en promedio un 0,47% más precisos que *AG* para este grupo.

La tabla 4.2 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 casi idénticas, siendo *AG* superior en 1 unidad. La diferencia en el *Mean* alcanza un valor máximo para la instancia *Hchl6s¸* donde *AGc* supera por 746 unidades de ganancia a *AG*, siendo en promedio para este grupo, 371 unidades superior.

Para 4 de las 6 instancias *AGc* presenta un *Best* superior a *AG*. Para las 2 restantes los *Best* alcanzados fueron equivalentes. La diferencia en el *Best* alcanza su valor máximo para la instancia *Hchl5s*, donde *AGc* supera por 615 unidades de ganancia a *AG*, siendo en promedio para este grupo 244 unidades superior.

Para 2 de las 6 instancias ambos algoritmos superan los mejores valores conocidos. *AGc* supera el mejor valor conocido para una instancia adicional, en la que *AG* no lo hace.

TABLA 4.2 *Resultados computacionales para las instancias difíciles*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instancia | | | | | *AG* | | | *AGc* | | |
|  | *W* | *L* | *NP* | *Optimo* | *Best* | *Mean* | *%GAP* | *Best* | *Mean* | *%GAP* |
| Hchl3s | 127 | 98 | 10 | 12215 | 12059 | 11994 | 1,81 | \*12246 | \*12088 | 1,04 |
| Hchl4s | 127 | 98 | 10 | 12006 | 12132 | 11903 | 0,85 | 12132 | \*12083 | 0,00 |
| Hchl5s | 205 | 223 | 25 | 45410 | 43979 | 43451 | 4,31 | \*44594 | \*43964 | 3,18 |
| Hchl6s | 253 | 244 | 22 | 61040 | 59184 | 58405 | 4,32 | \*59668 | \*59151 | 3,09 |
| Hchl7s | 263 | 241 | 40 | 63112 | 62088 | 61131 | 3,14 | 62088 | \*61640 | 2,33 |
| Hchl8s | 49 | 20 | 10 | 911 | 939 | \*921 | 0,00 | \*946 | 920 | 0,00 |

Para todas las instancias de este grupo, los %*GAP* obtenidos por *AGc* son inferiores o iguales a los obtenidos por *AG*; siendo para 1 instancia iguales y equivalentes a 0, y para las restantes 5 inferiores. *AGc* obtiene otro %*GAP* equivalente 0, pata la instancia Hchl4s, donde el Mean supera ampliamente el mejor valor conocido. La diferencia en el %*GAP*, alcanza un valor máximo de 1,22% para la instancia *Hchl6s*, siendo el promedio de un 0,8%. En promedio el %*GAP* obtenido por AG equivale a 2,47% y el obtenido por *AGc* equivale a un 1,81%. Por lo tanto *AGc* es capaz de obtener resultados en promedio un 0,8% más precisos que AG para este grupo.

La tabla 4.3 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 uno *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 4.3 *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 el 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*=0,47%) para el grupo1, donde las instancias son pequeñas y medianas, mayores (*Δ%GAP*=0,57%) para el grupo3, correspondientes a instancias grandes, y mayores (*Δ%GAP*=0,8%) para el grupo2, correspondientes a instancias difíciles. En general, numéricamente las diferencias parecen ser poco significativas, de hecho en promedio el *Δ%GAP* para el total de instancias es de 0,5% 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 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 y 4.2 evidencian la no normalidad de los datos, donde existe una alta frecuencia de *%GAP* concentrados en los valores pequeños.

TABLA 4.4 *Resultados test de normalidad para el %GAP exhibido por cada algoritmo*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *algoritmo* | *Kolmogorov-Smirnova* | | | *Shapiro-Wilk* | | |
|  | *Estadístico* | *gl* | *Sig.* | *Estadístico* | *Gl* | *Sig.* |
| *%GAP* | *AG* | ,144 | 49 | ,012 | ,929 | 49 | ,006 |
| *AGc* | ,167 | 49 | ,001 | ,915 | 49 | ,002 |

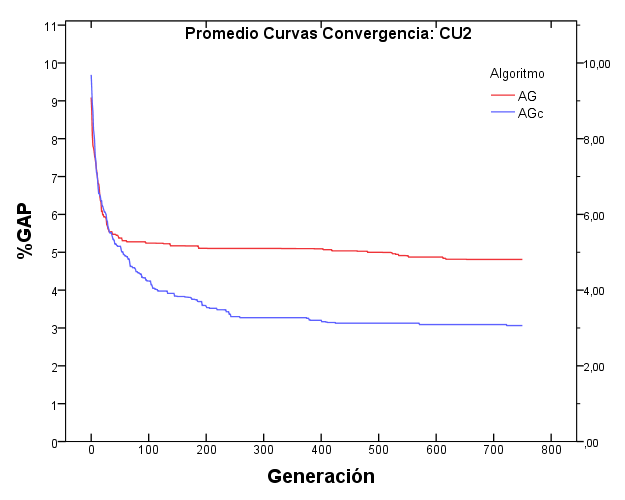
|  |  |
| --- | --- |
| FIGURA 4.1 *Histograma para el %GAP en AG* | FIGURA 4.2 *Histograma para el %GAP en AGc* |

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 3.1. Como se puede observar, el nivel de significación es de 0,037 < 0,05. Luego se rechaza la hipótesis nula y es posible concluir que existen diferencias estadísticamente significativas entre los *%GAP* de los grupos comparados, que pueden ser explicadas por el algoritmo utilizado.

TABLA 4.5 *Resultados prueba U de Mann-Whitney para el %GAP*

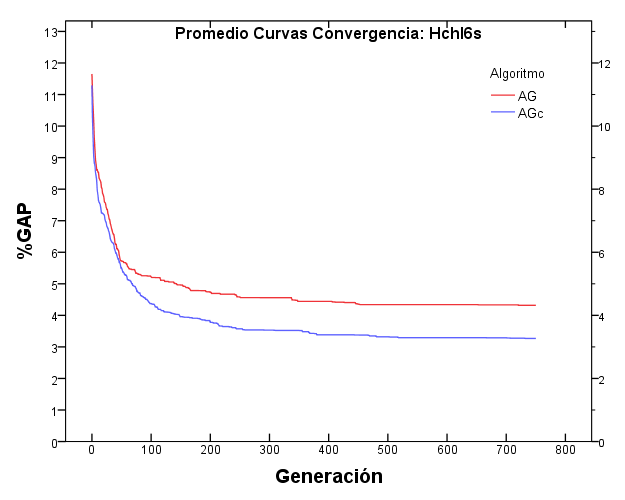
|  |  |
| --- | --- |
|  | *%GAP* |
| *U de Mann-Whitney* | 908,000 |
| *W de Wilcoxon* | 2133,000 |
| *Z* | -2,087 |
| *Sig. asintót. (bilateral)* | ,037 |

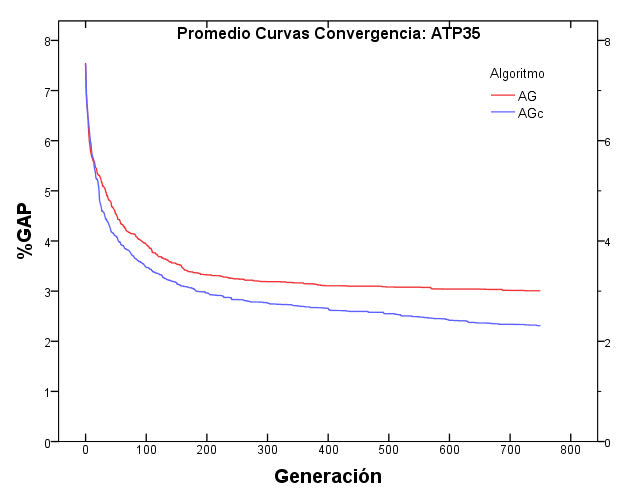
Otro aspecto importante a analizar, es el comportamiento en la convergencia de los algoritmos, lo que permite deducir cuál es la influencia en la presión de selección. Se ha seleccionado una instancia de cada grupo, donde se grafica el promedio de las curvas de convergencia del *%GAP* del mejor individuo, 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 layout), 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 delmejor individuo en las 30 ejecuciones en cada generación, dada por el eje *x.*

FIGURA 4.3 *Promedio de curvas de convergencia de AG y AGc para la instancia CU2*

La figura 4.3 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 avanza 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 4.4 muestra las curvas de convergencia para la instancia *Hchl6s*, perteneciente al segundo grupo. Similar al 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 4.4 *Promedio de curvas de convergencia de AG y AGc para la instancia Hchl6s*

FIGURA 4.5 *Promedio de curvas de convergencia de AG y AGc para la instancia ATP35*

La figura 4.5 muestra las curvas de convergencia para la instancia *ATP35*, perteneciente al tercer grupo. Una vez más, *AGc* rápidamente supera a *AG*, obteniendo soluciones de mayor precisión. Esta superioridad aumenta de forma monótona a medida que avanza la evolución, ya que la brecha entre las curvas tiende a crecer ligeramente. 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 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, por lo que es más difícil que se atasque en óptimos locales en etapas tempranas de la evolución.

En general, 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 prometedoras), 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 prometedoras), los cuales son superados en etapas maduras. En este caso, dicha superación ocurre 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.

Aún cuando la prueba estadística realizada sugiere diferencias significativas entre 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 (0,037), 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 sub-optimales, 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 simplemente consistió en sustituir la representación binaria por una representación de permutación, en conjunto con una función constructora apropiada para la nueva representación, manteniendo el resto de los ajustes fijos. Los resultados del experimento adicional se incluyen en el anexo x.x. En resumen, el algoritmo es capaz de alcanzar valores óptimos para 4 instancias adicionales y aumentar la precisión del *%GAP* en un 0,28%. Por lo tanto, el uso de una representación más apropiada tiene un importante efecto, disminuyendo el sesgo en la búsqueda genética, y mejorando 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. En resumen, la heurística no es capaz de construir el patrón óptimo debido a que, efectivamente exhibe un sesgo, al no considerar un caso de inserción de piezas en el patrón. Este tipo de inserción coincide con el encaje efectuado por la heurística *bottom-left*, la cual aparentemente no viola la restricción de guillotina.

Al eliminar estas fuentes de sesgo en la búsqueda genética, existen dos escenarios posibles; El primer escenario es que, debido a que el sesgo radica en la capa común de modelamiento, ambos algoritmos mejoren su desempeño, aumentando su precisión en la misma medida, por lo que las diferencias tenderían a mantenerse. En el segundo escenario, la eliminación del sesgo en la búsqueda, y por consiguiente, la mejora en la capacidad de exploración del algoritmo, permite explotar de mejor manera las cualidades de la estructuración bajo un enfoque celular, permitiendo que *AGc* mejore sus resultados en una proporción mayor que *AG*, obteniendo así, una mayor diferencia en las soluciones obtenidas 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, 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 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 and Winter, 2008), un algoritmo genético específico para el problema, el cual realiza la búsqueda genética directamente en el espacio de patrones. Para los dos grupos restantes, correspondientes a instancias difíciles y 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 muestra las comparaciones para el primer grupo. No existe 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 solo para 2 de las 11 instancias *AGc2* supera al resto. Al comparar ambos enfoques evolutivos, *AGc2* supera a *SPGAL* para 4 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, entre otros. *GRASP* encuentra valores óptimos para 5 de las 11 instancias, *SPGAL* lo hace para 4 de las 11 instancias, y *AGc2* para 3 de. En este sentido, *GRASP* muestra ser el superior de los 3 algoritmos comparados para este grupo.

TABLA 4.6 *Comparación entre GRASP, SPGAL, y AGc2 para instancias medianas*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Instancia* | | | | | *GRASP* | *SPGAL* | *AGc2* |
|  | *W* | *L* | *NP* | *Optimo* |
| CU1 | 100 | 125 | 25 | 12330 | 12312 | \*12330 | 12312 |
| CU2 | 150 | 175 | 35 | 26100 | \*26100 | \*26100 | 25806 |
| CU3 | 134 | 125 | 45 | 16679 | \*16652 | 16598 | 16625 |
| CU4 | 285 | 354 | 45 | 99366 | \*99264 | 98764 | 99039 |
| CU5 | 456 | 385 | 50 | 173364 | \*173364 | 171935 | 171935 |
| CU6 | 356 | 447 | 45 | 158572 | 158572 | 158572 | 158572 |
| CU7 | 563 | 458 | 25 | 247150 | 247150 | 246143 | \*247392 |
| 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 muestra las comparaciones para el segundo grupo. Similar al grupo anterior, no se aprecia una superioridad marcada de un algoritmo por sobre el resto, aunque en este caso, *HCEB* y *AGc2* son más competitivos que *GRASP*. Para 3 de las 6 instancias *HCEB* es superior al resto, presentando para dos casos, valores óptimos. AGc2 es superior al resto para otras 3 instancias, presentando valores superiores al mejor conocido.

TABLA 4.7 *Comparación entre GRASP, HCEB, y AGc para instancias difíciles*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Instancia* | | | | | *GRASP* | *HCEB* | *AGc2* |
|  | *W* | *L* | *NP* | *Optimo* |
| Hchl3s | 127 | 98 | 10 | 12215 | 12159 | 12214 | \*12297 |
| Hchl4s | 127 | 98 | 10 | 12006 | 11621 | 11964 | \*12146 |
| Hchl5s | 205 | 223 | 25 | 45410 | 44346 | \*45410 | 44869 |
| Hchl6s | 253 | 244 | 22 | 61040 | 60403 | \*61040 | 60203 |
| Hchl7s | 263 | 241 | 40 | 63112 | 62547 | \*63102 | 62294 |
| Hchl8s | 49 | 20 | 10 | 911 | 904 | 876 | \*966 |

La 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 de las 10 instancias. A pesar de lo anterior, en general *AGc2* logra resultados de una precisión razonable para la mayoría de las instancias, mostrando márgenes relativamente pequeños respecto a los valores óptimos.

TABLA 4.8 *Comparación entre GRASP, HCEB, AGc 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 | 814931 |
| ATP32 | 307 | 124 | 56 | 38068 | 37786 | \*38068 | 37655 |
| ATP33 | 241 | 983 | 44 | 236611 | 231178 | \*236611 | 234935 |
| ATP34 | 795 | 456 | 27 | 361398 | 353822 | \*361357 | 358674 |
| ATP35 | 960 | 649 | 29 | 621021 | 607864 | 621021 | 612654 |
| ATP36 | 537 | 244 | 28 | 130744 | 129634 | 130744 | 128770 |
| ATP37 | 440 | 881 | 43 | 387276 | 379329 | 387276 | 382031 |
| ATP38 | 731 | 358 | 40 | 261395 | 252605 | 261395 | 257579 |
| ATP39 | 538 | 501 | 33 | 268750 | 263076 | 268750 | 267593 |

Para poder apreciar la precisión alcanzada por *AGc2* en términos cualitativos, a continuación de incluyen layouts óptimos para algunas instancias, obtenidos de (Yoon et al., 2013), en conjunto con los mejores layouts obtenidos por el algoritmo para cada una de ellas.

|  |
| --- |
| FIGURA 4.6 *Un layout óptimo para la instancia ATP34, con W=795*, *L=456*, *NP=27 y ganancia igual a 361368* |
| layoutATP34-2.png  FIGURA 4.7 *Mejor layout obtenido por AGc2 para la instancia ATP34, con W=795, L=456, NP=27, ganancia igual a 358674 y %GAP=0,75%* |
|  |

|  |  |
| --- | --- |
| FIGURA 4.8 *Un layout óptimo para la instancia Hchl5s, con W=205*, *L=223*, *NP=25 y ganancia igual a 45410* | **layoutHchl5s-2.png**  FIGURA 4.9 *Mejor layout obtenido por AGc2 para la instancia Hchl5s, con W=205, L=223, NP=25, ganancia igual a 44594 y %GAP=1,19%* |

|  |  |
| --- | --- |
| FIGURA 4.10 *layout de la mejor solución conocida para la instancia Hchl4s, con W=127*, *L=98*, *NP=10 y ganancia igual a 12006* | layoutHchl4s-2.png  FIGURA 4.11 *Mejor layout obtenido por AG2c para la instancia Hchl4s, con W=127, L=98, NP=10, ganancia igual a 44594 y %GAP=0%* |

# CONCLUSIONES

# REFERENCIAS BIBLIOGRÁFICAS

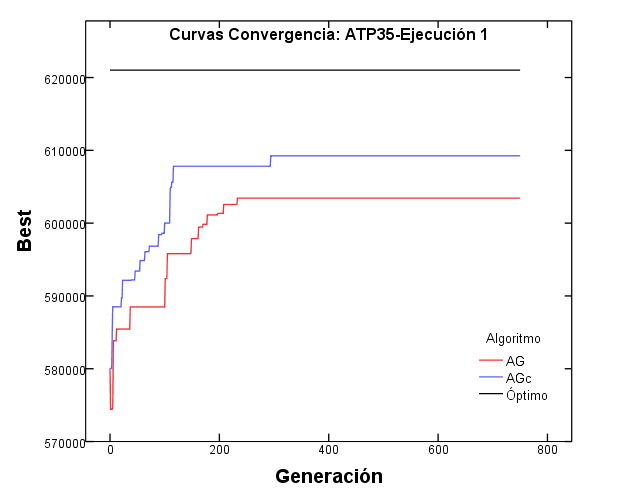


FIGURA 6.1 *Curvas de convergencia de AG y AGc para la instancia ATP35 en la ejecución 1*

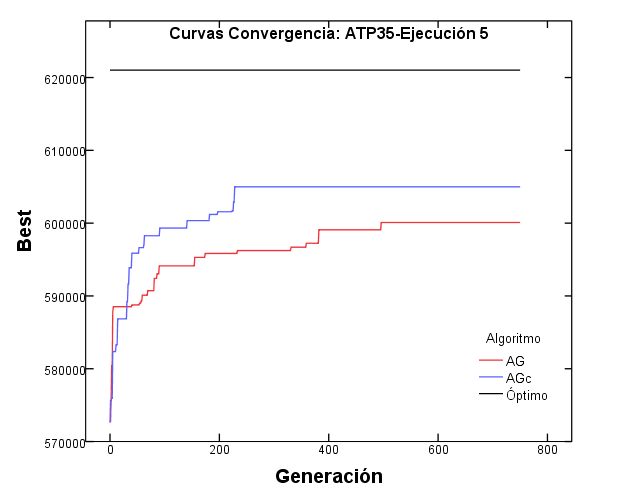


FIGURA 6.2 *Curvas de convergencia de AG y AGc para la instancia ATP35 en la ejecución 5*

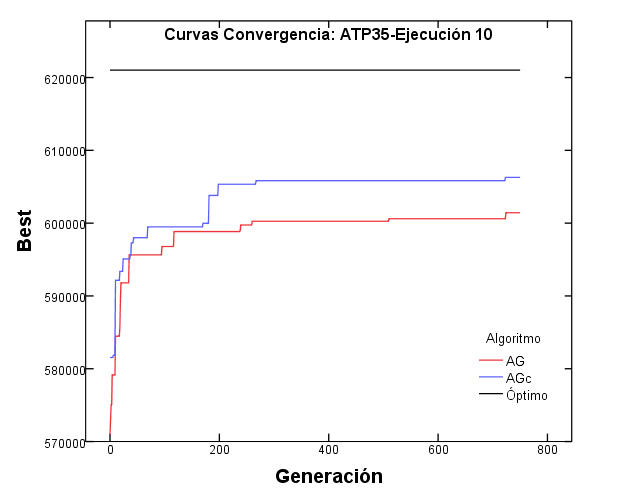


FIGURA 6.3 *Curvas de convergencia de AG y AGc para la instancia ATP35 en la ejecución 10*

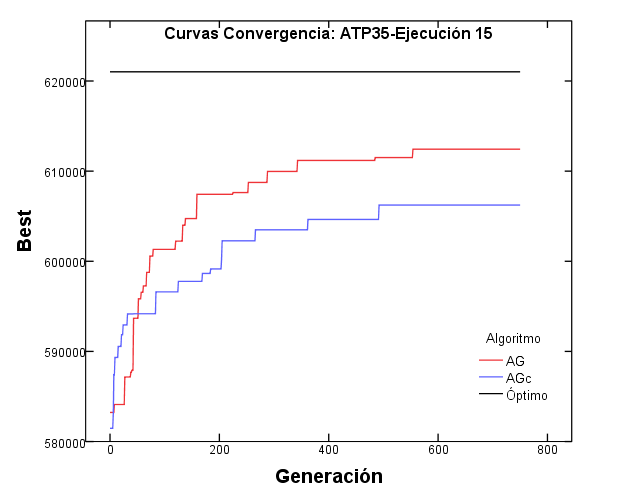


FIGURA 6.4 *Curvas de convergencia de AG y AGc para la instancia ATP35 en la ejecución 15*

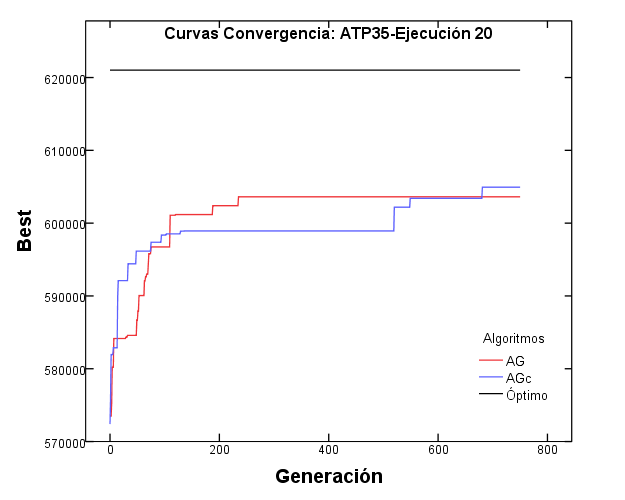


FIGURA 6.5 *Curvas de convergencia de AG y AGc para la instancia ATP35 en la ejecución 20*

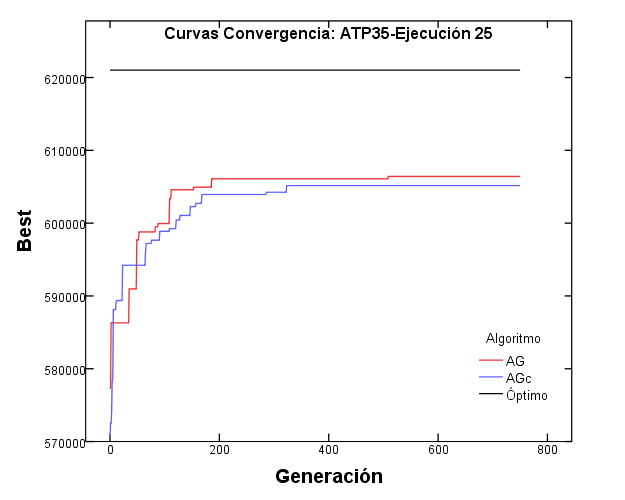


FIGURA 6.6 *Curvas de convergencia de AG y AGc para la instancia ATP35 en la ejecución 20*

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

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. [↑](#footnote-ref-1)
2. En un contexto computacional, donde no existen las limitaciones biológicas, esto podría ser fácilmente generalizado a *n* padres y *n* hijos [↑](#footnote-ref-2)
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 razonables [↑](#footnote-ref-3)
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. [↑](#footnote-ref-4)
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. [↑](#footnote-ref-5)
6. Estructura geométrica típica en AG celulares. Sin embargo existen otras... [↑](#footnote-ref-6)