# INTRODUCCIÓN

## 1.1 ANTECEDENTES Y MOTIVACIÓN

## 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

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.* Ambos problemas son 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 item, 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 items equivale a su área, la notación de Beasley es suficiente para describir el problema. Estos casos, correspondientes a las versiones no ponderadas, aparecen tipicamente 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 ejemplo, es posible formularlo como un problema de programación lineal extendiendo la formulación del bien conocido 1CP. 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.

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

### 1.3.2 Propósito de la solución

## 1.4 OBJETIVOS Y ALCANCES DEL PROYECTO

### 1.4.1 Objetivo general

### 1.4.2 Objetivos específicos

### 1.4.3 Alcances

## 1.5 METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS

### 1.5.1 Metodología a usar

### 1.5.2 Herramientas de desarrollo

### 1.5.3 Ambiente de desarrollo

## 1.6 ORGANIZACIÓN DEL DOCUMENTO

## 1.7 GLOSARIO

# 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.

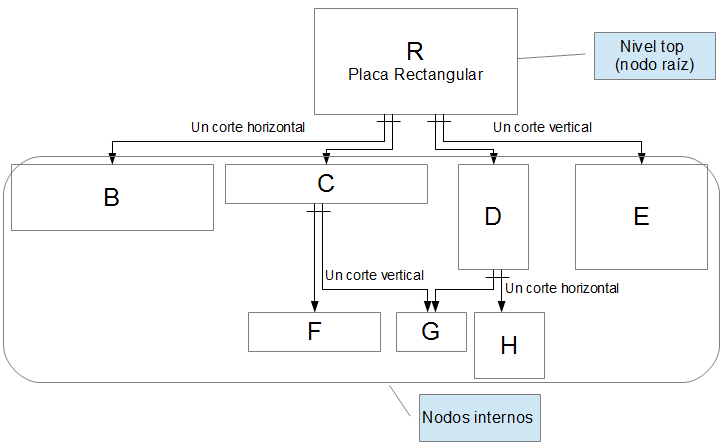


FIGURA 2.1 *Enfoque top-down: Proceso de búsqueda en el grafo dirigido*

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.

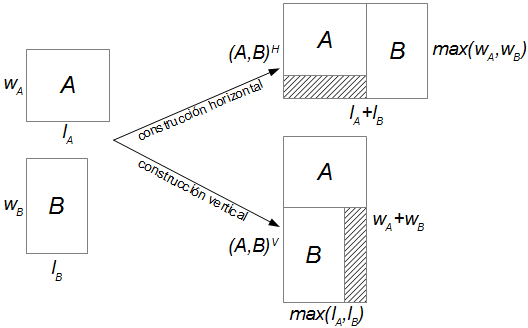


FIGURA 2.2 *Enfoque bottom-up: Construcción horizontal o vertical de patrones*

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, como se muestra en la 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) propone 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* (sin etapas). 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 media o larga escala. A menudo se usan heurísticas para tratar instancias de mayor dureza. Dentro de los métodos heurísticos, (Wang, 1983) Propuso un algoritmo heurístico 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 de 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.

A continuación se describen los avances desde...es el enfoque usado en este trabajo...

Dentro del enfoque metaheurístico de los *AG...*

Por último, es interesante mencionar un exponente basado en el enfoque metaheurístico de los algoritmos genéticos, de modo de establecer un referente en el modelamiento del problema mediante conceptos genéticos, así como su capacidad para resolver este tipo de problemas. En particular

Aqui falta agregar los trabajos de parada...

(Ono and Ikeda, 1998) proponen un método que usa *AG* para optimizar el layout de piezas rectangulares en el problema de corte con restricción de guillotina, con el fin de lograr una mínima cantidad de pérdida. En el AG propuesto, una solución está dada por una ecuación para determinar la disposición de piezas en la placa. De acuerdo a la ecuación, todas las piezas son integradas recursivamente mediante un algoritmo de layout, utilizando el enfoque *bottom-up*. La representación utiliza una parte binaria, representando los operadores binarios V o H, correspondientes a combinaciones verticales u horizontales respectivamente, y una parte entera, representando las piezas. Así, el cromosoma es interpretado como un árbol binario, leído en notación post-fija (ver )



2.7

(Beasley, 2000) presenta un enfoque evolutivo, basado en una formulación no-lineal del *CTDC* sin la restricción de guillotina. Utiliza una representación binaria/real. 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. Para la reproducción considera el apareamiento de dos padres con un solo hijo. Los padres son seleccionados aleatoriamente de la población, y como operador de cruzamiento utilizan *cruzamiento uniforme*. Para la mutación considera que uno de cada diez hijos puede mutar, un gen correspondiente a la parte binaria es seleccionado aleatoriamente y si su valor es 1, es volteado a 0. 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. Se utiliza un modelo *steady-state* para reemplazar la población, evaluando ambas medidas de *fitness* para determinar los individuos que deben ser reemplazados. El enfoque presentado pudo resolver con eficacia problemas tomados de la literatura en cuestión de segundos.

(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. Basándose en ésta métrica, clasifican a los individuos en tres categorías: en los que todas las piezas pudieron ser cortadas, en los que el número de piezas cortadas es mayor al número de piezas que no pudieron ser cortadas, y en los que el número de piezas cortadas es menor al número de piezas que no pudieron ser cortadas. Se proponen tres heurísticas de colocación para interpretar el cromosoma y evaluar su calidad. Utilizan un algoritmo *steady-state*, cuyo objetivo es maximizar la diferencia antes mencionada. Usan *cruzamiento uniforme* y un operador de mutación *big-creep* sobre la parte binaria. El individuo a ser reemplazado es seleccionado aleatoriamente de los 10 peores individuos. Para seleccionar los padres, usaron una adaptación del torneo-binario, en el cual el ganador pertenece a la primera categoría, si dos individuos pertenecen a la misma categoría, gana el que tenga mejor *fitness*. Fijaron las probabilidades de cruzamiento y mutación a 0.6 y 0.1 respectivamente, y fijaron el tamaño de población a 120 individuos. Analizan la composición de la población inicial y final de acuerdo a las 3 categorías de individuos, y concluyen que la evolución es capaz de hacer emerger las mejores soluciones, en donde la mayoria de las piezas pueden ser cortadas. Además la mejor heurística de colocación de las evaluadas fué aquella que preordena las piezas en un orden decreciente de sus valores.

(Leung et al., 2003) aplicaron un algoritmo genético simple (*GA*) y un algoritmo genético mixto con simmulated annealing (*SAGA*) al 2D-KP, los cuales fueron comparados. 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. Usaron un esquema genotipo/fenotipo, en donde una representación de permutación, correspondiente a la secuencia de piezas que debe ser empacada/cortada, constituye el genotipo, en conjunto con una estrategia de colocación (*DP*) como algoritmo decodificador, para así obtener un *layout* válido, constituyendo así el fenotipo. En sus experimentos ellos variaron el tamaño de población de 10 a 100 individuos. Utilizaron selección por *ruleta*, *swap-mutation*, la cual intercambia dos genes aleatorios de un individuo. Para el cruzamiento usaron dos operadores, con la idea de preservar subsecuencias con buenos órdenes. 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 algún momento maduro de la evolución (ver figura 2.7).



FIGURA 2.8 *GA vs SAGA Pérdida promedio en la población versus Iteraciones*

(Bortfeldt and Winter, 2008) introducen un algoritmo genético llamado Strip Packing Genetic Algorithm Layer (*SPGAL*) para distintas variantes del *2D-KP*, el caso con guillotina así como el caso sin esta restricción. Así también, tanto el caso con restricción de orientación como sin esta restricción son considerados. El *AG* propuesto no usa una codificación especial para las soluciones. En lugar de ello, la búsqueda tiene lugar directamente en el espacio fenotípico, dado por planes de empaque. 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*

El *AG* usa reemplazo generacional sin duplicados. Para generar la población inicial utilizan la heurística Best Fit Decreasing Height (*BFDH*) con algunas mejoras. Utilizan una estrategia elitista, y usan dos tipos de mutación, una estándar y una mutación por fusión. La mutación estándar es usada de manera concurrente con el operador de cruzamiento, cuyas probabilidades son complementarias (pcruzam.+pmutac.=1). Utilizan selección por ranking para seleccionar el primer padre, mientras que el segundo padre es seleccionado aleatoriamente. Para el cruzamiento, las capas con mejores *fr* son extraídas de los padres para combinar las buenas características de estos. Para la mutación estándar, el número y la selección de capas que son modificadas se determinan al azar. Para la mutación por fusión

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. Aplican un enfoque de diversificación, en donde la búsqueda genética es repetida varias veces bajo distintas condiciones y la mejor solución obtenida es emitida al final. El *AG* propuesto fue comparado con varias heurísticas recientes de esa época, sus resultados mostraron que éste, generalmente mostraba un comportamiento medio para el caso con restricción de guillotina, lo cual atribuyeron a la estructura adicional de capas de los planes de empaque. Sin embargo, para instancias de larga escala sin la restricción de guillotina el AG propuesto alcanza la calidad de las mejores heurísticas conocidas.

(Wang, 2010)

(Gonçalves and Resende, 2011)

## 2.2. ALGORITMOS GENÉTICOS CELULARES

En esta sección se presentan algunos antecedentes y evidencias de los AG celulares...

# MATERIALES Y MÉTODOS

## 3.1 ALGORITMO GENÉTICO CELULAR

## 3.2 MODELAMIENTO DEL PROBLEMA

### 3.2.1 Esquema general

Al utilizar un enfoque evolutivo, la personalización de la metaheurística se centra en dos tareas, por una parte, decidir la representación genética de una solución candidata para el problema, y por otra, decidir los operadores genéticos apropiados para el problema en cuestión. Para la representación genética se utiliza el esquema genotipo-fenotipo, el cual involucra la definición del genotipo y el mapeo al fenotipo mediante una función constructora. El uso de una codificación especial para las soluciones no es un requisito, en lugar de ello la búsqueda genética puede llevarse a cabo directamente en el espacio fenotípico (tómese como ejemplo el AG de Bortfeldt 2008). El propósito del esquema genotipo-fenotipo es proveer un marco de trabajo para el diseño de algoritmos evolutivos, el cual permite al proceso evolutivo abstraerse del problema particular que se está abordando, enfocándose en aspectos propios de la evolución, como la diversidad genética o la presión de selección.

Los enfoques metaheurísticos que abordan problemas de *C&P*, a menudo utilizan heurísticas de colocación como algoritmos decodificadores de una solución particular; Las heurísticas de colocación son algoritmos que reciben una secuencia de piezas y sistemáticamente van determinando su disposición en la placa. La idea es que la metaheurística se encargue de realizar la búsqueda en el espacio de soluciones, mientras que la heurística de colocación se encarga de realizar la traducción a una solución válida para el problema. El esquema general es descrito en la ilustración xx, 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 3.1 *Esquema general: Función constructora + heurística colocación*

### 3.2.2 Representación

La representación utilizada está basada en la propuesta por A. Flores 2012, la cual utiliza una codificación binaria para representar todas las posibles secuencias de piezas[[2]](#footnote-2). 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 x.x.a, 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.

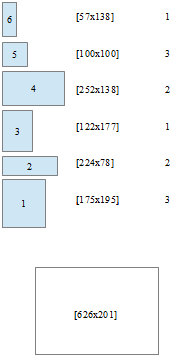
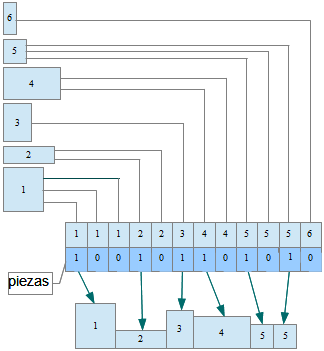


FIGURA 3.2.b *Cromosoma de piezas e interpretación*

FIGURA 3.2.a *Instancia de ejemplo problema*

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 x.x.b 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 x.x.a. 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 figura x.x.b. 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[[3]](#footnote-3). 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 3.4.a *Cromosoma completo y su interpretación* |
| FIGURA 3.4.b *Interpretación conceptual 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.

En la figura x.x 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 la sucesión de saltos en una posición, este mecanismo permite visitar las piezas que no han sido visitadas hasta el ciclo de saltos actual.

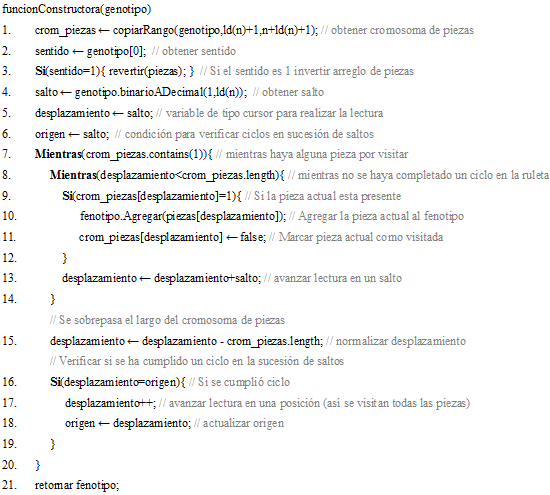


FIGURA 3.5 *Pseudocódigo función constructora*

Para ver el funcionamiento de la función constructora se incluye la traza para la solución candidata de ejemplo. Inicialmente se realiza la descomposición del cromosoma, extrayendo la ruleta de 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 los saltos.

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

FIGURA 3.6 *Traza función constructora; obtención de ruleta de piezas, parámetros de lectura y tres primeras iteraciones de lectura de piezas*

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

FIGURA 3.7 *Traza función constructora, resto de iteraciones de lectura de piezas*

## 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 Ilustración x.x.a. 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 Ilustración x.x.b.

|  |  |
| --- | --- |
| FIGURA 3.8.a *Combinación horizontal de las piezas A y B* | FIGURA 3.8.b *Combinación horizontal de las piezas A y B* |

Como se observa en las figuras, cada combinación tiene asociada una pérdida interna, correspondiente al área rectangular que contiene las piezas y que queda inutilizada. A su vez, el patrón resultante también tiene asociada una pérdida externa, correspondiente al área de la placa que queda inutilizada. En este trabajo se extiende el concepto de combinación vertical-horizontal, de manera que las pérdidas internas son pérdidas, también verticales u horizontales, asociadas a una pieza en particular. Del mismo modo, las pérdidas externas son pérdidas verticales u horizontales asociadas al patrón de corte. Bajo este concepto, en la figura x.x.a 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 x.x.b la pérdida interna es una pérdida vertical asociada a la pieza B, la cual no tiene pérdida horizontal asociada.

En la figura x.x se muestra el pseudocódigo de la heurística de colocación, que recibe como parámetro el fenotipo, (dado por una secuencia de piezas), y retorna un layout (consistente en un patrón guillotinable). Además existe una estructura global *“listaPerdidasInternas”*, que almacena las pérdidas asociadas a cada pieza. La heurística recorre la secuencia del inicio hasta el final, y para cada pieza evalúa cada uno de los siguientes casos:

* 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[[4]](#footnote-4). Luego se actualizan las pérdidas internas.
  + Si no es posible insertar la pieza en alguna pérdida interna, se busca 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.

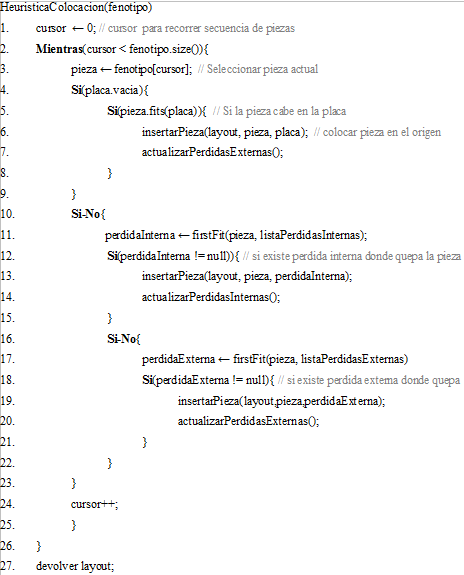


Figura 3.9 *Pseudocódigo heurística colocación*

Para ver el funcionamiento de la heurística se incluye la traza para la solución candidata de ejemplo. 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 (figura x.x.2). 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 (figura x.x.3). 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 (figura x.x.4). 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 1.10 *Traza heurística colocación 1a parte*

|  |
| --- |
|  |
|  |

FIGURA 3.11 *Traza heurística colocación 2a parte*

### 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.

...

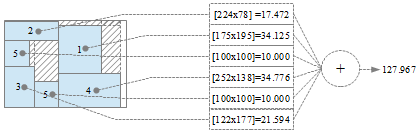


FIGURA 3.12 *Función objetivo y fitness de la solución candidata*

### 3.2.6 Operadores genéticos

Los operadores genéticos constituyen el motor del ciclo reproductivo y así el núcleo evolutivo del AG. Inspirados en procesos biológicos como la selección natural, la reproducción sexual, y la mutación, a continuación se explicarán los operadores genéticos más comunes para una representación binaria, la cual ha sido utilizada en este modelamiento.

**6.1 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.

**6.1.1 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.

**6.1.2 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:

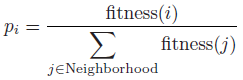


Figura 3.14 *Probabilidad de seleccionar individuo i mediante ruleta*

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.

**6.1.3 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 permite 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

La selección

**6.2 Cruzamiento**

Tal como en la reproducción sexual, el cruzamiento considera el apareamiento entre dos individuos[[5]](#footnote-5), 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.

**6.2.1 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.

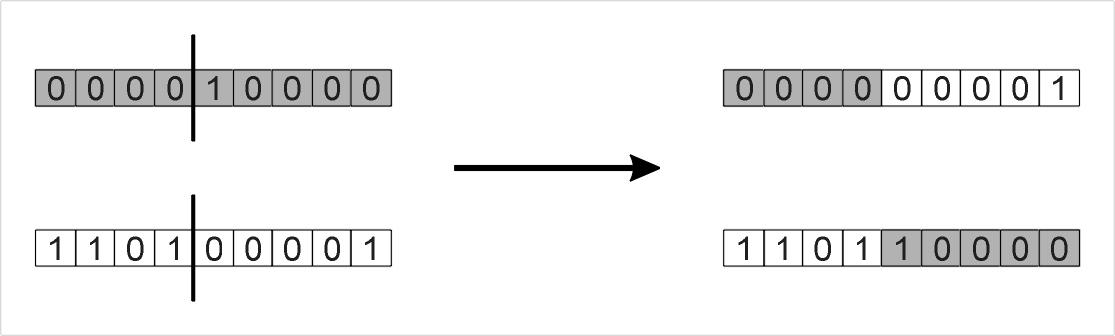


Figura 3.15 *Cruzamiento de 1 punto*

**6.2.1 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.

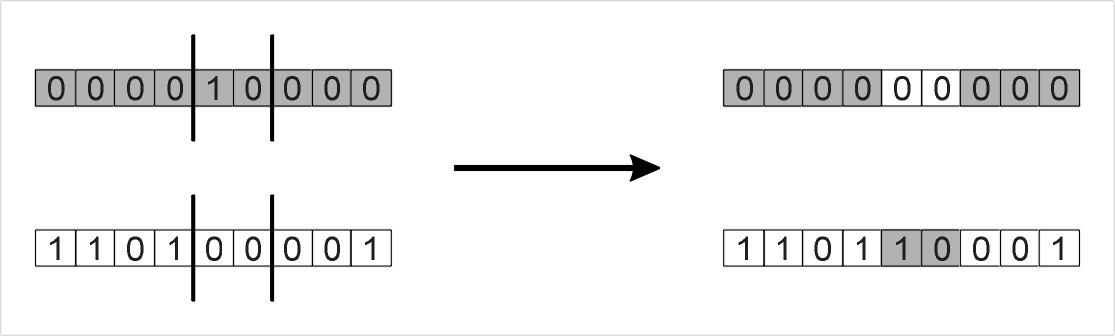


Figura 3.16 *Cruzamiento de n puntos; n=2*

Una característica importante del 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.

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

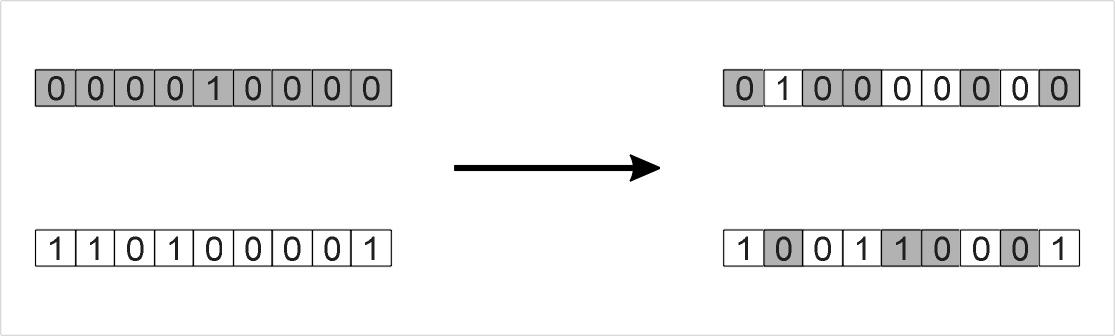


Figura 3.17 *Cruzamiento uniforme*

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.

**6.3 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).

**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

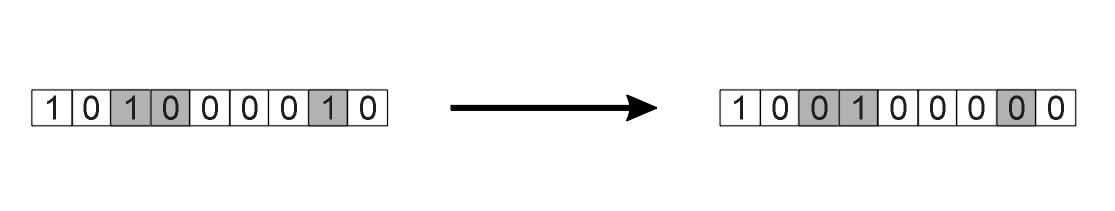


Figura 3.18 *Mutación binaria*

lalalala

lalalala

## 3.2 DATOS DE PRUEBA

## 3.4 CONFIGURACIÓN DE ALGORITMOS

### 3.4.1 Motivación

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:

* **Desarrollo de algoritmos complejos:** El ajuste de parámetros de un algoritmo heurístico es una tarea muy laboriosa, y de hecho puede consumir una gran parte del tiempo del desarrollo general. El uso de métodos automatizados de configuración de algoritmos puede conducir a importantes ahorros de tiempo y potencialmente lograr mejores resultados que métodos manuales ad-hoc
* **Estudios empíricos, evaluaciones y comparaciones de algoritmos**: Una cuestión central en la comparación de algoritmos heurísticos es si un algoritmo supera a otro porque es fundamentalmente superior, o porque sus desarrolladores fueron más exitosos optimizando sus parámetros. Los métodos automatizados de configuración pueden mitigar este problema de comparaciones injustas y así facilitar estudios comparativos más significativos
* **Uso práctico de los algoritmos:** La habilidad de algoritmos heurísticos complejos para resolver instancias grandes y difíciles a menudo depende en forma crítica de un adecuado ajuste de parámetros. Los usuarios finales usualmente tienen poco o ningún conocimiento del impacto del ajuste de los parámetros en el rendimiento del algoritmo, y así usan simplemente el ajuste por defecto. Aún cuando haya sido cuidadosamente optimizado con un benchmark estándar, tal configuración puede no funcionar bien en la instancias particulares encontradas por un usuario. Los métodos automatizados de configuración pueden ser usados para mejorar el rendimiento con principios y de manera conveniente

El problema de la 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 x.x 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*

Teniendo claros los elementos que intervienen en el problema de configuración, resulta fácil identificar cada uno de ellos. En nuestro caso tenemos varios algoritmos objetivo correspondientes a cada uno de los AG propuestos para la comparación, por lo que debe resolverse un problema de configuración por cada uno de ellos. Por su 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 será descrita a continuación.

### 3.4.2 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

lalalalala

aqui agregar marco teórico

### 3.4.3 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, específicos para dicho algoritmo.

La figura x.x 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 las cardinalidades de los espacios de configuración, esto permite estimar los tiempos de espera para configurar cada algoritmo.

|  |  |  |  |
| --- | --- | --- | --- |
| *algoritmo parámetro* | *AG generacional* | *AG distribuido* | *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 | ✔ | ✔ |
| *cantidad islas* | ─ | 4; 5; 6; 7; 8; 9; 10 | ─ |
| *tamaño islas* | ─ | ✔ | ─ |
| *frecuencia migración* | ─ | 5000; 10000; 15000;  20000; 25000 | ─ |
| *enrejado* | ─ | ─ | 20x20; 10x40; 5x80; 4x100; 2x200; 1x400 |
| *vecindad* | ─ | ─ | Linear5; Compact9; Compact13 |
| *política actualización* | ─ | ─ | ls; frs; nrs; uc; ss; synchronous |
| *# total configuraciones* | 27000 | 35 | 108 |

Tabla 3.1 *Valores posibles para cada parámetro*

*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. A continuación se especifican los principales parámetros de un *escenario de sintonización,* y cómo fueron definidos para la configuración de los algoritmos objetivo:

|  |  |
| --- | --- |
| ***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 3.2 *Parámetros escenario sintonización paramILS*

La figura x.x muestra el escenario de sintonización para el AG-generacional

alalala

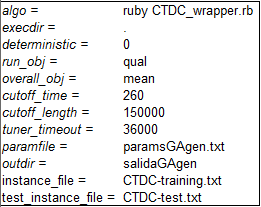


figura x.x.- Ejemplo de escenario de sintonización para el *AG-Generacional*

|  |  |  |  |
| --- | --- | --- | --- |
| *algoritmo parámetro* | *AG generacional* | *AG distribuido* | *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** | ✔ | ✔ |
| *cantidad islas* | ─ | 4; **5**; 6; 7; 8; 9; 10 | ─ |
| *tamaño islas* | ─ | ✔ | ─ |
| *frecuencia migración* | ─ | 5000; 10000; **15000**;  20000; 25000 | ─ |
| *enrejado* | ─ | ─ | 20x20; 10x40; 5x80; 4x100 ;2x200; **1x400** |
| *vecindad* | ─ | ─ | Linear5;**Compact9**; Compact13 |
| *política actualización* | ─ | ─ | ls; frs; nrs; **uc**; ss; synchronous |

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

## 3.4 DISEÑO EXPERIMENTAL

## 3.5 HARDWARE Y SOFTWARE UTILIZADO

# RESULTADOS

# ANÁLISIS DE RESULTADOS

# CONCLUSIONES

# REFERENCIAS BIBLIOGRÁFICAS

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. Es fácilmente demostrable que la representación propuesta no es capaz de generar todas las secuencias posibles. Sin embargo se asume que es capaz de generar una diversidad suficiente [↑](#footnote-ref-2)
3. 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-3)
4. 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-4)
5. En un contexto computacional, donde no existen las limitaciones biológicas [↑](#footnote-ref-5)
6. Estructura geométrica típica en AG celulares. Sin embargo existen otras... [↑](#footnote-ref-6)