Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación de Algoritmos Genéticos para la optimización del corte de material

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Iván Sánchez Rodríguez

Tutor: Federico Barber Sanchís

2015/2016

Resumen

El problema de corte de material consiste en, dada una pieza de material de tamaño estándar, obtener las diferentes piezas deseadas realizando cortes sucesivos, minimizando el material inservible que se desecha, conocido como merma, o maximizando el número de piezas que se pueden obtener a partir de la pieza original. Se trata de un problema realista de optimización combinatoria, en ocasiones optimización multicriterio, que pertenece al conjunto de problemas conocido como "problemas de corte y empaquetado", y cuya solución optimizada requiere a menudo la aplicación de técnicas metaheurísticas.

En el presente proyecto se abarca la resolución del problema mediante el diseño, desarrollo, implementación y evaluación de un algoritmo genético de carácter *anytime*, capaz de obtener y mejorar sucesivamente la solución alcanzada hasta que sea viable.

Palabras clave: optimización, algoritmos, genéticos, corte, material.

Abstract

The cutting stock problem consists in, given a standard-sized sheet of material, obtain all of the desired pieces making successive cuts, minimizing wasted material, known as scrap, waste or trim loss; or maximizing the number of pieces that can be obtained from the original sheet. It is a realistic combinatorial optimization problem, sometimes multi-objective optimization that belongs to the set of problems known as “cutting and packing problems” and which optimized solution often requires the use of metaheuristic techniques.

This project covers the resolution of the problem by designing, developing, implementing and evaluating a genetic anytime algorithm able to obtain and improve successively the solution until it is viable.

Keywords : optimization, genetic, algorithm, cutting, stock.

Tabla de contenidos

1. Introducción 8
   1.1. Clasificación del problema a resolver 9
2. Planteamiento del problema 14
   2.1. Formalización 15
      2.1.1. Objetivos 15
      2.1.2. Restricciones 16
3. Métodos alternativos 17
   3.1. Aproximaciones heurísticas 17
   3.2. Búsqueda con grafos *AND*/*OR* 18
   3.3. Algoritmo de aproximación O(m^3) 19
   3.4. Algoritmo de Wang 20
   3.5. Otros métodos 21
4. Diseño del algoritmo genético 22
   4.1. Introducción a los algoritmos genéticos 22
   4.2. Funcionamiento de un algoritmo genético 22
   4.3. Diseño del algoritmo genético 23
      4.3.1. Parámetros básicos de la población 23
      4.3.2. Codificación de las soluciones 23
      4.3.3. Decodificación de las soluciones 27
      4.3.4. Función de aptitud 28
      4.3.5. Método de selección 29
      4.3.6. Método de cruce 30
      4.3.7. Método de mutación 32
      4.3.8. Método de reemplazo 33
5. Implementación 33
   5.1. Software utilizado 34
   5.2. Estructura de la aplicación 34
   5.3. Implementación del algoritmo genético 36
      5.3.1. Codificación de las soluciones 37
      5.3.2. Decodificación de las soluciones 37
      5.3.3. Función de aptitud 39
      5.3.4. Método de selección 40

5.3.5. Método de cruce 40
5.3.6. Método de mutación 41
5.3.7. Transición entre generaciones 42
5.4. Interfaz gráfica de usuario 42
6. Evaluación del sistema 45
7. Conclusiones 58
8. Bibliografía 59
9. Anexo 62

Índice de figuras

Figura 1: Ejemplo de disposición de piezas cortadas con corte de guillotina 12
Figura 2: Ejemplo de disposición de piezas cortadas sin corte de guillotina 12
Figura 3: Disposición de las piezas de una instancia resuelta con un algoritmo heurístico 18
Figura 4: Ejemplo del algoritmo de aproximación 19
Figura 5: Método de disposición de piezas en el algoritmo de Wang 20
Figura 6: Fenotipo del cromosoma "1 2 H" 24
Figura 7: Fenotipo del cromosoma "1 2 V" 24
Figura 8: Individuo ejemplo 25
Figura 9: Representación en forma de árbol de los cortes a realizar en la disposición del ejemplo 26
Figura 10: Interpretación del cromosoma "1 5 H 7 6 V 2 H 4 ..." 28
Figura 11: Separación de los genes 30
Figura 12: Ejemplo de construcción del árbol a partir de una operación de tipo V 38
Figura 13: Interfaz gráfica de la primera ventana de la aplicación. Ejemplo de uso 43
Figura 14 Interfaz gráfica de la primera ventana de la aplicación. Ejemplo de resultado 44
Figura 15: Una solución óptima al problema utilizado para realizar las pruebas. 45
Figura 16: Fenotipo de la mejor solución obtenida en la prueba número uno 47
Figura 17: Fenotipo de la mejor solución obtenida en la prueba número dos 48
Figura 18: Fenotipo de la mejor solución obtenida en la prueba número tres 49
Figura 19: Fenotipo de la mejor solución obtenida en la prueba número cuatro 50
Figura 20: Fenotipo de la mejor solución obtenida en la prueba número cinco 51
Figura 21: Fenotipo de la mejor solución obtenida en la prueba número seis 52
Figura 22: Fenotipo de la mejor solución obtenida en la prueba número siete 53
Figura 23: Fenotipo de la mejor solución obtenida en la prueba número ocho 54

1. Introducción

Muchos materiales utilizados en la industria y en la construcción se encuentran inicialmente en forma de unidades enteras de tamaño estándar (hojas de vidrio, rollos de hojalata, madera contrachapada, planchas de metal, etc.). Para utilizar estos materiales directamente u obtener productos semielaborados, es necesario dividirlos en partes que tengan las dimensiones requeridas. Al hacer esto, normalmente se forman restos y el material realmente utilizado constituye únicamente un porcentaje de la cantidad total, mientras que el resto se desecha. Es cierto que en algunos casos se puede encontrar alguna aplicación para los restos, pero su utilización o bien requiere gastos adicionales (el material residual se tiene que refundir, soldar, etc.) y por lo tanto está asociado con pérdidas, o bien se utiliza como producto final con mucho menos valor que los productos originales obtenidos (los restos de madera de construcción se utilizan como combustible, por ejemplo). Por lo tanto, la minimización de los restos es claramente un problema importante y realista, ya que permite reducir la pérdida de materiales valiosos.

Este problema es conocido como el problema de corte de material (*cutting stock problem*). Se trata de un problema de optimización combinatoria de complejidad *NP-hard*, en concreto, NP-Completo [9], dado que de todas las posibles disposiciones posibles de las piezas que se desean obtener, se requiere aquella que maximice o minimice ciertos objetivos.

Dependiendo de las características restrictivas y del número de dimensiones del material a recortar, se puede diferenciar en distintos tipos. Al ser recorte de material, normalmente el problema es unidimensional o bidimensional, aunque existen casos de corte tridimensional, como por ejemplo obtención de piedras preciosas, como por ejemplo diamantes a partir del diamante en bruto [4] [6]. Independientemente del número de dimensiones, el problema puede diferenciarse en tres tipos, según [7], dependiendo el objetivo final:

- Organizar[1] tantas piezas deseadas en la pieza original[2] como sea posible (maximización).
- Organizar todas las piezas en una pieza original de mínimo tamaño o en el menor número de piezas originales posible (minimización).
- Cortar las piezas deseadas de una o más piezas originales minimizando el material residual.

[1] Mediante la organización se consigue una disposición de las piezas que se utiliza posteriormente para realizar el corte.

[2] También conocida como contenedor, ya que este problema se encuentra estrechamente relacionado con el problema de la mochila (*knapsack problem*). Entre los dos problemas se puede establecer la siguiente analogía: la pieza original es el contenedor o mochila, y las piezas que se desea obtener son los objetos a introducir en el contenedor, de manera que se busca una disposición de las piezas que cumpla las restricciones requeridas.

La primera formulación conocida de tan solo la versión unidimensional de este problema fue realizada por el economista ruso L. V. Kantorovich en 1939 [1]. En este estudio, Kantorovich utilizó métodos de programación lineal para abordar cuestiones relacionadas con la optimización del uso de diferentes máquinas en una fábrica, entre otros.

Más tarde, se logró un significativo avance en la resolución de problemas de corte gracias al trabajo de Gilmore y Gomory en 1961 [2] y 1963 [3], en el que describieron su técnica de generación de patrones para resolver el problema de minimizar la pérdida del material en mediante programación lineal.

A partir de entonces, el interés en este tema ha aumentado significativamente, hasta el punto en que se han registrado más de 500 artículos que tratan sobre aproximaciones en su resolución y diferentes aplicaciones [5].

1.1. Clasificación del problema a resolver

La gran variedad de aplicaciones mostrada en toda la bibliografía existente que trata sobre los problemas relacionados con el corte y empaquetado, llevó a Dyckhoff [6] a desarrollar un esquema formal de clasificación que diferenciara entre los diferentes tipos que hay en esta clase de problemas. Este esquema clasifica los problemas dependiendo de las siguientes características:

1. *Dimensionalidad*

   (N) Número de dimensiones

2. Tipo de asignación

   (B) Todos los objetos[3] y una selección de ítems[4]

   (V) Una selección de objetos y todos los ítems

3. Variedad de los objetos grandes (objetos)

   (O) Un objeto

   (I) Objetos idénticos

   (D) Objetos distintos

4. Variedad de los objetos pequeños (ítems)

   (F) Pocos objetos de diferente forma

[3] "Objeto" designa de manera general el concepto de pieza de material original en un problema de corte; o el contenedor, en un problema de empaquetado.

[4] "Ítem" designa de manera general el concepto de pieza a obtener recortando en un problema de corte; o el objeto a ordenar dentro del contenedor, en un problema de empaquetado.

(M) Muchos objetos de diferente forma

(R) Muchos objetos de relativamente pocas formas distintas (no congruentes)

(C) Formas congruentes

De esta forma se obtiene una clasificación efectiva de los distintos problemas que puede haber en la clase de problemas de corte y empaquetado.

Noción	Pertenece al tipo
Problema clásico de la mochila	1/B/O/
Problema de carga de palés	2/B/O/C
Problema de la mochila N-dimensional	/B/O/
Problema de *Bin Packing* dual	1/B/O/M
Problema de carga de vehículos	1/V/I/F, o 1/V/I/M
Problema de carga de contenedores	3/V/I/ , o 3/B/O/
Problema clásico de *Bin Packing*	1/V/I/M
Problema clásico de corte de material	1/V/I/R
Problema de *Bin Packing* bidimensional	2/V/D/M
Problema usual de corte bidimensional de material	2/V/I/R
Problema general de corte de material	1/ / / / , 2/ / / / , o 3/ / / /
Problema de balanceo de líneas de ensambladura	1/V/I/M
Problema de planificación de multiprocesadores	1/V/I/M
Problema de asignación de memoria	1/V/I/M
Problema de cambio de monedas	1/B/O/R
Problema de *multi-period capital budgeting*	*n*/B/O/

Tabla 1. Diferentes problemas de corte y empaquetado y su tipo

El trabajo realizado y el presente documento se centran en concreto en la resolución del problema de corte bidimensional rectangular de material, en el que los objetos son piezas rectangulares como podrían ser planchas de metal, lonas de tejido, piezas de mármol etc., y los ítems son piezas rectangulares de menor tamaño que se obtienen a partir de la original, realizando cortes sucesivos. Por lo tanto, el problema a resolver se cataloga según Dyckhoff como 2/V/O/R, ya que:

- El número de dimensiones *N* es 2
- El tipo de asignación es V, porque algunos objetos (e.g. lámina de metal) son seleccionados y a partir de ellos, se deben obtener una serie de ítems (e.g. piezas de metal) específicos.
- La variedad de los objetos grandes es de tipo I, porque todas las láminas originales son de tamaño estándar.
- La variedad de los ítems es de tipo R, porque el número de piezas a obtener es relativamente alto, pero las distintas formas que pueden tener son pocas, dado que normalmente se desea obtener un número determinado de piezas de las mismas características (congruentes).

Sin embargo, la clasificación que es capaz de realizar el esquema de Dyckhoff no es suficientemente específica como para determinar con exactitud y precisión el problema a tratar. El problema de corte bidimensional de material, en concreto el de obtener piezas rectangulares a partir de una pieza original rectangular de mayor tamaño, cuenta con dos restricciones que pueden darse o no, y que condicionan en gran medida la manera en que hay que abordarlo. Estas restricciones adicionales se agrupan según [8] en las siguientes:

- (C1) Restricción de rotación:
  Aunque girar las piezas 90º en la disposición es usualmente factible, existen casos en los que las piezas deben ser cortadas en una dirección, por ejemplo, casos en los que el material tiene características unidireccionales, como algunos tipos de tela, madera con vetas, etc. Por lo tanto, si una pieza a obtener no tiene el mismo ancho que largo, su disposición es invariable y la restricción debe ser cumplida.

- (C2) Restricción de corte de guillotina:
  Se requiere que todos los cortes realizados sean cortes de guillotina. Un corte de guillotina es un corte sobre la pieza rectangular de material que va de extremo a extremo, siendo la dirección del corte paralela a los lados superior e inferior de la pieza cortada; en otras palabras: un corte de guillotina es aquél que realizado en una pieza rectangular, da lugar a dos piezas rectangulares.
  Debido al tipo de máquinas que usualmente realizan los cortes en el material, en ocasiones sólo se pueden realizar cortes que sean de guillotina, por lo tanto, si no se dispone de una máquina capaz de hacer cortes que no vayan de extremo a extremo, esta restricción debe ser cumplida para que la organización de las piezas resultante sea factible para realizarse con los instrumentos que se poseen.

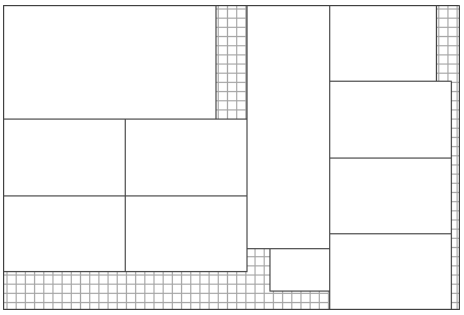


Figura 1: Ejemplo de disposición de piezas cortadas con corte de guillotina

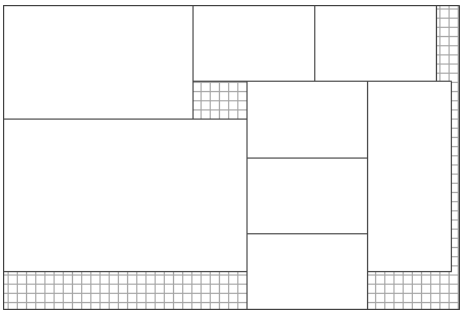


Figura 2: Ejemplo de disposición de piezas cortadas sin corte de guillotina

Dependiendo de la necesidad de cumplir estas restricciones, se pueden obtener cuatro problemas distintos:

- RF: Las piezas se pueden rotar 90º (R) y no se requiere que todos los cortes sean de guillotina (F);
- RG: Las piezas se pueden rotar 90º (R) y se requiere que todos los cortes sean de guillotina (G);
- OF: La orientación de las piezas es fija (O) y no se requiere que todos los cortes sean de guillotina (F);
- OG: La orientación de las piezas es fija (O) y se requiere que todos los cortes sean de guillotina (G);

Teniendo la diferenciación anterior entre los problemas a tratar, si tenemos en cuenta que la única diferencia entre los problemas son las posibles restricciones que pueden tener, podemos establecer un orden parcial de complejidad entre los cuatro tipos diferentes, de manera que una clase de problemas esté contenida en otra clase más restrictiva (i.e., una clase de problemas es un subconjunto de otra):

$$RF \subset \{RG, OF, OG\} \tag{1}$$

$$RG \subset OG \tag{2}$$

$$OF \subset OG \tag{3}$$

Obviamente, una solución que es factible con respecto al tipo *OG*, es también factible con respecto a los otros tres tipos. De ese mismo modo, una solución que es factible con respecto al tipo *OF* o al tipo *RG* es también factible con respecto al tipo *RF*, debido a que al eliminar restricciones no es necesario cambiar la solución para que siga siendo factible.

Por esa misma razón se ha decidido centrar este trabajo en el problema de tipo *OG*, ya que dada una solución a un problema de tipo *OG*, es también válida para los otros tipos, consiguiendo así cubrir una mayor casuística y obteniendo un resultado con una aplicación más general.

2. Planteamiento del problema

Una vez se ha determinado con cierta exactitud el problema a resolver, es necesario formalizarlo como un problema de optimización para poder proceder a su implementación.

Los objetivos[5] de este problema de optimización son dos:

1. **Minimizar el área ocupada por el rectángulo mínimo** posible que abarque todas las piezas según la disposición efectuada. La finalidad de este objetivo es minimizar el área que quede entre las piezas una vez dispuestas, ya que se considera como material inútil al ser de pequeño tamaño, mientras que se considera material útil el área resultante a los lados de este rectángulo.
2. **Maximizar la proporción entre el alto y el ancho del rectángulo mínimo** posible que abarque todas las piezas según la disposición efectuada.
   En algunos casos, la pieza original tiene un ancho determinado y lo único que se busca es minimizar la longitud a gastar al obtener un conjunto de piezas. Para generalizar el problema en este aspecto, se ha decidido acotar la pieza original con una anchura y una longitud predeterminadas y tratar de obtener una disposición lo más cuadrada posible, para evitar distribuciones demasiado verticales o demasiado horizontales.

A modo de resumen, el problema de corte bidimensional de material en el que se centra este trabajo es aquel que cumple los siguientes requisitos:

- Solo hay una pieza original, de la que se obtienen las demás piezas;
- La pieza original es de forma rectangular;
- Las piezas que se desean obtener son de forma rectangular;
- Las piezas que se desean obtener tienen un tamaño menor a la pieza original;
- No se designa una cantidad de piezas a obtener: si se desean dos piezas de las mismas características, se indica con dos piezas distintas con los mismos parámetros;
- El material perdido es aquél que queda entre dos o más piezas recortadas;
- Las piezas no pueden ser rotadas;
- La distribución final de las piezas debe permitir que sea obtenida utilizando únicamente cortes de guillotina;
- El posicionamiento de las piezas se realiza de manera relativa con la esquina superior izquierda de la pieza original, de manera que el punto de la esquina superior izquierda de la primera pieza dispuesta coincide con el punto de la esquina superior izquierda de pieza original.

[5] Como más adelante se mostrará, el aspecto multicriterio que caracteriza el problema no tiene por qué contar con unos objetivos con el mismo peso.

- Para ganar generalidad en el método, los tamaños, determinados por la anchura y la longitud, se cuantifican en unidades (u). Cualquier extensión que requiera datos en un sistema real de medición, podrá ser calculado de manera trivial una vez se obtenga la solución.

2.1. Formalización

Sean:

- $\boldsymbol{P}$ la pieza original de material;
- $\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_n}$ las piezas a obtener;
- $\boldsymbol{r}$ la designación de un espacio definido en el plano, en forma de un rectángulo;
- $\boldsymbol{w_i} \in \mathbb{R}$, el ancho de la pieza i o del rectángulo r;
- $\boldsymbol{l_i} \in \mathbb{R}$, el largo de la pieza i o del rectángulo r;
- $\boldsymbol{x_i}$ la posición en el eje de abscisas de la pieza i o del rectángulo r con respecto al punto inicial[6];
- $\boldsymbol{y_i}$ la posición en el eje de ordenadas de la pieza i o del rectángulo r con respecto al punto inicial;
- $\boldsymbol{SPC}(p_1, p_2, \ldots, p_k)$ la función que obtiene el rectángulo r de menor tamaño posible que contiene a $p_1, p_2, \ldots, p_k$;

2.1.1. Objetivos

El objetivo número uno del problema es minimizar el material desechado, siendo éste el material que está dentro del rectángulo mínimo que cubre todas las piezas pero que no forma parte de ninguna pieza. Este objetivo se puede expresar como:

$$\underset{(x_1,x_2,\ldots,x_n),(y_1,y_2,\ldots,y_n)}{argmax} f(p_1, p_2, \ldots, p_n) = \frac{\sum_{i=1}^{n} w_i l_i}{w_{SPC(p_1,p_2,\ldots,p_n)} l_{SPC(p_1,p_2,\ldots,p_n))}} \quad (4)$$

[6] Punto (0, 0). Corresponde a la esquina superior izquierda de P

Y el objetivo número dos del problema indica que se quiere maximizar la proporción regular del rectángulo *r* de menor tamaño posible que contiene a todas las piezas que se desean obtener, haciendo que éste tenga una forma lo más cuadrada posible, por tanto se define como:

$$\underset{(x_1,x_2,\ldots,x_n),(y_1,y_2,\ldots,y_n)}{argmax} \; g(p_1,p_2,\ldots,p_n) = \frac{\min(w_{SPC(p_1,p_2,\ldots,p_n)}, l_{SPC(p_1,p_2,\ldots,p_n)})}{\max(w_{SPC(p_1,p_2,\ldots,p_n)}, l_{SPC(p_1,p_2,\ldots,p_n)})} \quad (5)^7$$

2.1.2. Restricciones

Ambos objetivos están sujetos tanto a la restricción número uno (*R*1), que obliga a que el rectángulo *r* de menor tamaño posible que contiene a todas las piezas que se desean obtener esté contenido en los límites de la pieza original *P*. Esta restricción se denota como:

$$w_{SPC(p_1,p_2,\ldots,p_n)} \leq w_P \;\wedge\; l_{SPC(p_1,p_2,\ldots,p_n)} \leq l_P \quad (6)$$

Y a la restricción número dos (*R*2), que asegura que no existe ningún solapamiento de piezas en la disposición:

$$\begin{gathered} \nexists\, p_i,\, p_j \mid \\ x_i < x_j + w_j \;\wedge \\ x_i + w_i > x_j \;\wedge \\ y_i > y_j - l_j \wedge \\ y_i - l_i < y_j \end{gathered} \quad (7)$$

Una vez el problema ha quedado formalizado y especificado con totalidad, es posible iniciar el proceso de diseño y desarrollo de la herramienta que lo resolverá.

[7] Aquí, *f* puede tomar como valor máximo el valor 1, que significa que el rectángulo es un cuadrado perfecto.

3. Métodos alternativos

Existen una gran cantidad de aproximaciones para resolver variantes del problema de corte de material que han sido ideadas a lo largo de los años por distintos autores.

La primera aproximación importante para resolver este problema fue la de Gilmore y Gomory en 1961 [2], en la que se plantea la cuestión como un problema de programación lineal a resolver mediante el algoritmo *Simplex*. Sin embargo, el proceso de seleccionar una nueva columna para mejorar la solución existente en el algoritmo *Simplex* es muy costoso en un problema con tantas restricciones combinatorias y actividades[8] como el que se trataba de resolver, pero se obtuvo una variante del algoritmo *Simplex* que hallaba una nueva columna que mejoraba la solución resolviendo un problema auxiliar de menor coste computacional. Sin embargo, tal y como indica Riehme *et al*. [17], éste método en general no es apropiado para instancias del problema con demandas pequeñas. Además, para instancias muy grandes, el coste temporal de cálculo todavía es inadmisible.

Desde entonces hasta el día de hoy, muchos investigadores han estado estudiando diversos métodos y algoritmos para poder resolver este problema complejo en un tiempo computacional razonable. Entre otros métodos, podemos encontrar:

- Aproximaciones heurísticas [10], [11], [16]
- Búsqueda con grafos *AND/OR* [12], [19]
- Algoritmo de aproximación O(m^3) [13]
- Algoritmo de Wang [15], [24]

3.1. Aproximaciones heurísticas

Los métodos basados en heurística que se encuentran en la bibliografía se centran principalmente en el problema de corte bidimensional de material por etapas, en el que la pieza original debe ser cortada siguiendo la restricción de que primero únicamente se pueden realizar cortes longitudinales, y sobre las piezas resultantes, llamadas tiras (*strips* o *shelves* en inglés, dependiendo del autor) se aplican los cortes transversales para obtener las piezas deseadas. Un algoritmo heurístico popular para esta aplicación sigue los siguientes pasos [18]:

1. Obtener los datos de entrada: Dimensiones del *stock*, piezas requeridas y la cantidad requerida de cada tipo.
2. Calcular el área total de las piezas requeridas

[8] En el algoritmo Simplex, una actividad se denota con una variable o un conjunto de variables que pueden tomar un valor determinado

3. Asignar prioridad a las piezas que se deben obtener. Mayor anchura implica mayor prioridad. Si dos piezas tienen la misma anchura, la que tenga la mayor longitud debe tener prioridad a la hora de ser elegida.
4. Para asignar la primera pieza, que coincide con la más ancha, se crea una tira del mismo ancho y se coloca la pieza, cubriendo la anchura de la tira totalmente y la longitud hasta el punto en el que la pieza termina.
5. Si la siguiente pieza (que será igual o menos ancha) tiene una longitud que cabe en el resto de la tira, se coloca en dicha tira, si no, se crea una tira nueva con su misma anchura y se coloca siguiendo el paso cuatro.
6. Colocar el resto de piezas siguiendo los pasos cuatro y cinco hasta que se hayan organizado todas

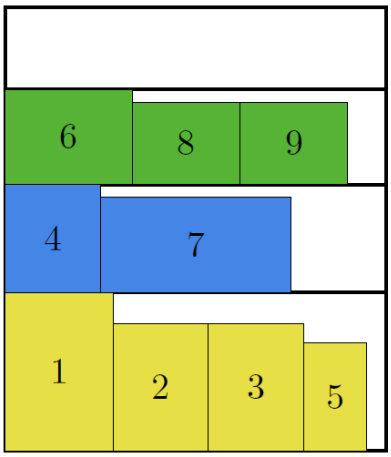


Figura 3: Disposición de las piezas de una instancia resuelta con un algoritmo heurístico

3.2. Búsqueda con grafos *AND/OR*

El método de búsqueda con grafos *AND/OR* se basa en representar la pieza original como un nodo inicial de un árbol binario. Dicho nodo inicial se ramifica en dos nodos que corresponden a la pieza inicial cortada de dos maneras diferentes: el nodo hijo izquierdo representa la pieza original cortada a lo largo, y el nodo hijo derecho representa la pieza original cortada a lo ancho. La cuestión es, una vez obtenido el árbol que comprende todas las posibles soluciones realizando sucesivos cortes en la pieza original, buscar aquél camino que satisfaga las necesidades buscadas, i.e., que sea suficientemente aceptable.

3.3. Algoritmo de aproximación O(m^3)

Éste método se basa en colocar las piezas al lado de piezas que ya estén colocadas asegurando que se cumple la restricción de que el corte es de guillotina. Cuando se quiere disponer una pieza, este algoritmo escoge todas las posibles piezas que tengan un lado libre, i.e., un lado que no toque ninguna otra pieza y genera un área horizontal, que es el área crítica donde la pieza puede ser dispuesta. Posteriormente, comprueba en qué lugares de dicha área puede colocar la pieza sin incumplir la restricción de corte de guillotina y la coloca en esa posición.

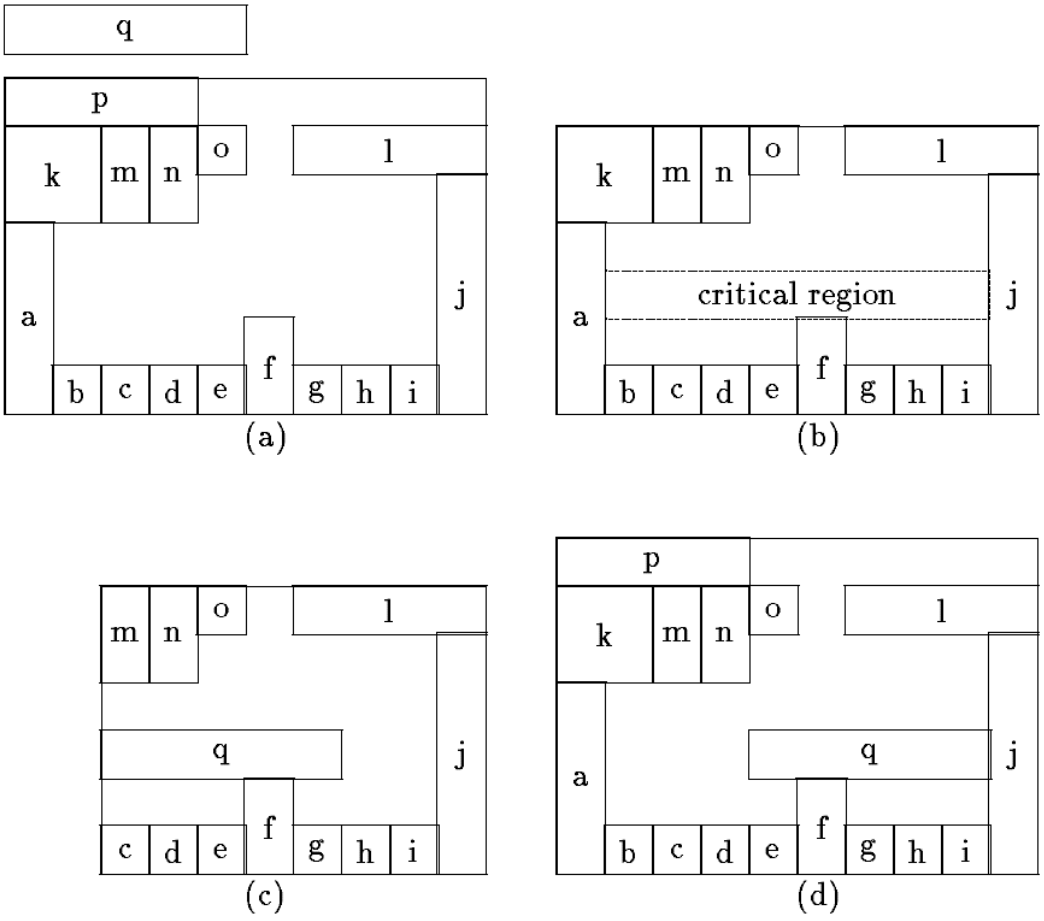


Figura 4: Ejemplo del algoritmo de aproximación

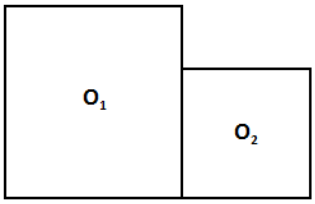
Figura 4a. Se desea determinar si la pieza q cabe en la ordenación de piezas actual de manera que no se incumple la restricción de corte de guillotina.

1. Se determinan todos los bordes expuestos de piezas ya colocadas. En este caso, son: b_{top}, c_{top}, d_{top}, e_{top}, f_{top}, g_{top}, h_{top}, i_{top}, l_{left}, l_{top}, o_{bottom}, o_{right}, o_{top}, p_{right}, n_{bottom}, y m_{bottom}.

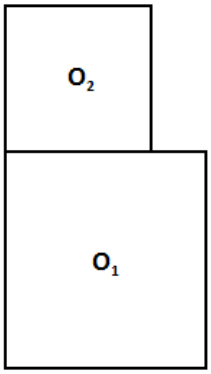
2. Para determinar si el rectángulo *q* cabe en la ordenación actual, todos los bordes expuestos deben ser considerados.
3. Para continuar con el ejemplo, se considera la colocación de *q* encima de *f*, en la frontera expuesta f_{top}. Una región crítica (*critical region*) se define con respecto al borde expuesto, con la longitud de la pieza a insertar.
4. El algoritmo busca una posición factible dentro de la región crítica en la que colocar la pieza.

3.4. Algoritmo de Wang

Wang [24] desarrolló una aproximación alternativa para generar patrones de corte generales que satisfacen la restricción de corte de guillotina. En este algoritmo, Los rectángulos se combinan en un proceso de construcción horizontal y vertical, en el que O_i es un rectángulo de ancho W_i y longitud L_i como se muestra en la figura 5.



(a) Composición horizontal de O_1 y O_2



(b) Composición vertical de O_1 y O_2

Figura 5: Método de disposición de piezas en el algoritmo de Wang

Una vez se especifica el valor B que indica el valor aceptable de pérdida de material, el algoritmo actúa de la siguiente manera [25]:

1. Definir $L^{(0)} = F^{(0)} = \{o_1, o_2, \ldots, o_n\}$, y establecer $K = 1$.
2.
    a. Calcular $F^{(K)}$, que es el conjunto de todos los rectángulos T que satisfacen:
        i. T está formado por composiciones horizontales o verticales de dos rectángulos de $L^{(K-1)}$,
        ii. la cantidad de material desechado en T no excede B, y
        iii. aquellos rectángulos o_1 que aparecen en T no violan las restricciones del número de veces que puede aparecer un determinado tamaño en el patrón
    b. Establecer $L^{(K)} = L^{(K-1)} \cup F^{(K)}$. Eliminar patrones equivalentes que haya en $L^{(K)}$.
3. Si $F^{(K)}$ no es vacío, establecer $K = K + 1$ y volver al paso dos. Si no, ir al paso cuatro.
4.
    a. Establecer $M = K - 1$
    b. Elegir el rectángulo en $L^{(M)}$ tal que tenga el mejor desecho de material al ser colocado en el rectángulo de la pieza original.

3.5. Otros métodos

Existen otros algoritmos para resolver distintas variantes del problema de corte de material. Muchos algoritmos son ya conocidos y aplicados en un gran número de problemas de toda índole, y son aplicados también para el problema de corte de material realizando diversas transformaciones o codificaciones para que sean manejables por los algoritmos. Entre otros, se pueden encontrar algoritmos de búsqueda en árboles (*tree-search algorithms*) [14], [20]; algoritmos de programación dinámica [21]; algoritmos de búsqueda tabú [22]; algoritmos GRASP [23]; entre otros.

4. Diseño del algoritmo genético

Tal y como se ha indicado anteriormente, se ha empleado un algoritmo genético para resolver el problema una vez formalizado.

4.1. Introducción a los algoritmos genéticos

Un algoritmo genético es un procedimiento computacional heurístico dedicado a la búsqueda estocástica (que no aleatoria), normalmente utilizado para resolver problemas de optimización o aprendizaje automático (*machine learning*). Este método mimetiza a la naturaleza, de manera que la evolución que tienen los seres vivos se aplica a las soluciones de un problema, representadas como cromosomas, para que estas evolucionen y sean cada vez mejores (más aptas).

4.2. Funcionamiento de un algoritmo genético

Un algoritmo genético tipo cuenta con un bucle principal, en el que se realizan las acciones características más importantes. La siguiente enumeración representa ordenadamente cómo actúan:

1. Se genera una población inicial de soluciones aleatoriamente obtenidas.
2. Se evalúan las soluciones con la función de aptitud o evaluación.
3. Se comprueba si se ha obtenido una solución óptima (o suficientemente buena) o si se cumple alguna otra condición de parada. En caso afirmativo, termina. En caso negativo, continúa en el paso cuatro.
4. Se seleccionan *n* individuos (dependiendo del método de selección)
5. Se cruzan los individuos seleccionados y se obtienen nuevos individuos.
    a. Los nuevos individuos sufren o no mutación de manera aleatoria.
6. Se reemplaza la población con los nuevos individuos obtenidos y se vuelve al paso número dos.

De esta manera, aunque depende del método de selección, lo deseable es que se elija con mayor probabilidad a los individuos más aptos, de manera que al combinar las soluciones en la fase de cruce, se obtengan soluciones que probablemente sean igual de aptas, o incluso más aptas [26], [27], [28].

4.3. Diseño del algoritmo genético

A la hora de resolver un problema aplicando un algoritmo genético, son diversas las cuestiones que hay que tener en cuenta. Existen diferentes técnicas en cada aspecto de los operadores genéticos (cruce, mutación y selección), lo que logra que estos algoritmos sean normalmente flexibles y aptos para ser utilizados en un gran número de problemas, pero la representación de la solución del problema a resolver como individuo (cromosoma) no suele ser trivial, y es un aspecto notablemente importante ya que puede impactar severamente en el éxito o no del algoritmo.

Normalmente, el genotipo[9] de un individuo se representa como una lista o cadena de operadores binarios u operadores formados por caracteres, aunque para algunas aplicaciones, también se utilizan codificaciones en forma de árbol [29].

4.3.1. Parámetros básicos de la población

La población de individuos cuenta con *n* individuos, que son cambiados cada generación para avanzar en la resolución del problema. El valor de *n* se determina empíricamente y a no ser que sea un valor extremadamente bajos, se tiende a alcanzar resultados similares en cada ejecución del algoritmo. Para valores demasiado altos de *n*, el algoritmo resulta ser más lento y no logra ninguna mejora aparente con respecto a la solución obtenida. Es por ello por lo que es necesario establecer un valor a *n* que no sea demasiado bajo como para que comprometa a la población a tener poca diversidad, y que ocurra una convergencia prematura o temprana[10]; ni que sea demasiado alto y requiera más recursos en vano.

Generalmente, un tamaño aceptable de la población se encuentra en un rango de entre 30 y 100 individuos, aunque depende del problema al que se aplique. En este caso, se han realizado pruebas con distintos tamaños de población comprendidos entre el rango mencionado.

4.3.2. Codificación de las soluciones

A la hora de codificar las soluciones para aplicar el algoritmo genético, hay que tener claro qué estructuras de datos se requiere utilizar para manejar las soluciones del problema y cómo se van a representar estas soluciones, es decir, hay que diseñar una

[9] Información genética
[10] Convergencia es el término que indica que todos o casi todos los individuos de la población tienen el mismo material genético. Significa que se ha alcanzado un óptimo, que puede ser local o global. Cuando la convergencia es temprana, es mucho más probable que dicho óptimo sea local, cosa que no es deseable.

función que transforme el genotipo de un individuo en el fenotipo[11] correspondiente, para poder visualizar y entender la solución que representa un individuo dado.

Para ello, primeramente hay que determinar la forma de representar los individuos, *i.e.*, las soluciones del problema. Para hacer que los genes del cromosoma representen una disposición de ítems en la pieza original sin ambigüedad, se ha adoptado una notación postfija (también conocida como notación polaca inversa) para construir los cromosomas. En estos cromosomas, se utilizan dos operadores: el operador H y el operador V. El operador H toma dos rectángulos (*i.e.*, pieza, ítem) como argumentos y produce el mínimo rectángulo capaz de cubrir los dos tomados, dispuestos horizontalmente uno al lado del otro. El operador V realiza la misma acción, pero la diferencia radica en que los rectángulos tomados como operando se disponen verticalmente. Por ejemplo, el cromosoma "1 2 H" indica que el rectángulo número dos se dispone horizontalmente, a la derecha del rectángulo 1 de la siguiente manera:

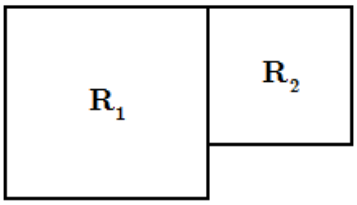


Figura 6: Fenotipo del cromosoma "1 2 H"

Del mismo modo, el cromosoma "1 2 V" indica que el rectángulo número dos se dispone verticalmente, debajo del rectángulo 1 de la siguiente manera:

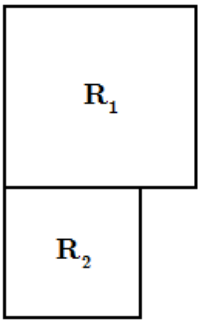


Figura 7: Fenotipo del cromosoma "1 2 V"

Siguiendo esta forma de construir las soluciones, la longitud (tamaño vertical) L_h y la anchura (tamaño horizontal) W_h de las piezas integradas[12] **con un operador H** se puede determinar de la siguiente manera:

[11] Manifestación que realiza el individuo debido a su genotipo, normalmente de manera física. En este caso, el fenotipo es la solución del problema que representa el genotipo de un individuo, es decir, una disposición de las piezas que se desean obtener

$L_h = L_1 + L_2$, $W_h = \max(W_1, W_2)$,

donde L_1 y L_2 son las longitudes de las piezas uno y dos, y *max* es una función que obtiene como entrada dos parámetros y devuelve aquél que sea mayor. En el caso de que las piezas estén integradas con un operador V, la longitud y la anchura de la pieza integrada se obtiene realizando las mismas operaciones pero al revés:

$L_h = \max(L_1, L_2)$, $W_h = W_1 + W_2$

A continuación se muestra un ejemplo más completo de lo que podría ser un caso real propuesto. Sea el cromosoma de un individuo "1 5 *H* 7 6 *V* 2 *H* 4 *H V* 9 8 *V* 3 *V H*", éste representa la siguiente solución para unos tamaños dados de las piezas demandadas:

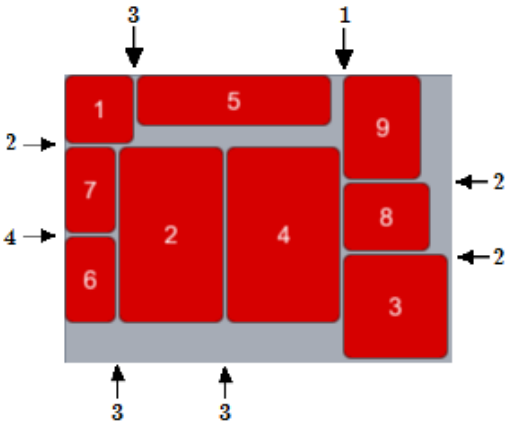


Figura 8: Individuo ejemplo

Las flechas indican el orden en el que los cortes deben ser realizados. De este modo se comprueba que efectivamente, el corte de guillotina se puede realizar para obtener las piezas a partir de ese patrón. Si se construye la representación arbórea derivada, se puede determinar la serie de cortes que hay que realizar para obtener una pieza en concreto.

[12] De aquí en adelante, los rectángulos o piezas que se obtengan a partir de dos piezas o piezas integradas realizando una operación H o V, se denominará "pieza integrada".

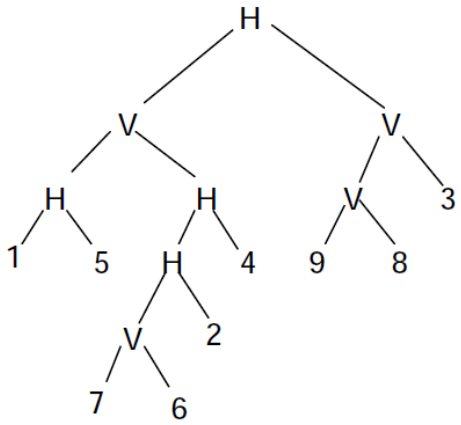


Figura 9: Representación en forma de árbol de los cortes a realizar en la disposición del ejemplo

Para obtener la serie de cortes necesarios para obtener una pieza, tan sólo hay que trazar un camino desde la raíz del árbol hasta el nodo hoja en el que se encuentre el número de la pieza deseada y realizar los cortes que muestre dicho camino.

La formulación seguida para generar los cromosomas, al estar en formato de notación postfija, debe seguir una serie de restricciones para que el cromosoma sea efectivamente válido. Además, el hecho de seguir una notación no ambigua implica que todos los cromosomas tienen la misma longitud para un número de piezas n dado, que es un aspecto deseable en un algoritmo genético para facilitar la lectura y tratamiento de los individuos (cromosomas).

Sea N_p la cantidad de números de pieza y N_o la cantidad de operadores que tiene un cromosoma, en primer lugar las siguiente restricción se debe cumplir, si no, el cromosoma no da lugar a una solución válida:

$$N_o = N_p - 1 \tag{7}$$

En segundo lugar, nótese que el hecho de adoptar la notación postfija hace que las expresiones de los cromosomas estén libres de ambigüedad, por lo que no es necesaria la utilización de paréntesis. Teniendo eso en cuenta y la restricción dada por la ecuación (7), se puede obtener la longitud del cromosoma N_g para un número determinado de piezas:

$$N_g = N_p + N_o = 2N_p - 1 \tag{8}$$

Y a partir de esas dos ecuaciones, se puede obtener tanto N_p como N_o como N_g únicamente si se conoce el valor de uno de ellos.

Otro aspecto a considerar es una restricción en las posiciones relativas de los números de pieza y de los operadores. Tomando un operador cualquiera, sea n_p el número de números de pieza que existen en la parte izquierda de este operador y sea n_o el número de operadores que existen en la parte izquierda de este operador, teniendo en cuenta ese mismo operador también, la siguiente relación debe cumplirse:

$$1 \leq n_o \leq n_p - 1 \tag{9}$$

Dado que las relaciones descritas son irrelevantes para el tipo de operadores, todas las disposiciones posibles se pueden obtener seleccionando los operadores pertinentes (H o V) y cambiando tanto las posiciones de éstos como las posiciones de los números de pieza en el cromosoma.

4.3.3. Decodificación de las soluciones

Decodificar un individuo significa obtener el fenotipo a partir del genotipo, es decir, obtener la posición de las piezas y en general la disposición de todos los ítems a partir de un cromosoma válido.

Como la información genética se encuentra en una sentencia en notación postfija, la idea principal de la decodificación del cromosoma se basa en recorrerlo e ir operando según indique el gen en el que se encuentra, mediante una pila, de manera que si el gen corresponde a una pieza, se apila la pieza, y si corresponde a un operador, se extraen de la pila las dos últimas piezas, se opera con ellas y la pieza integrada obtenida resultante de la operación se apila. Al final del recorrido del cromosoma, en la pila únicamente queda un elemento que corresponde al rectángulo integrado compuesto por todas las piezas. En el espacio dedicado a la implementación de las ideas descritas en el diseño del algoritmo genético se tratará la decodificación de las soluciones con más precisión.

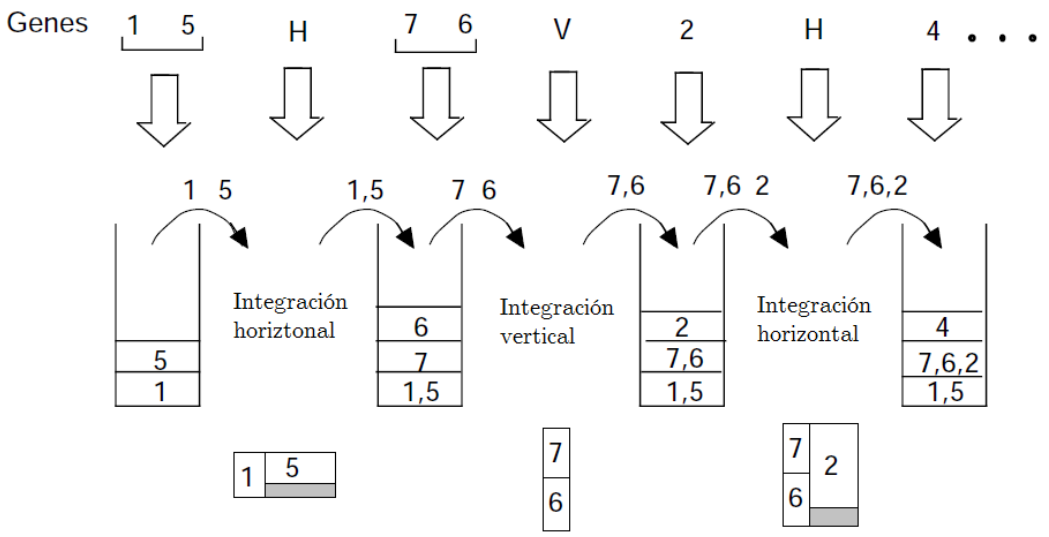


Figura 10: Interpretación del cromosoma "1 5 H 7 6 V 2 H 4 ..."

4.3.4. Función de aptitud

Dados los objetivos descritos en el punto 2.1.1, la función de aptitud es tan simple como agrupar en una misma función el valor de la aptitud que tiene una solución dada para cada uno de los objetivos. Ésta agrupación de objetivos se computa con un peso diferente para cada uno, de manera que un objetivo tiene más importancia que el otro. La diferencia de los pesos puede implicar un gran cambio a la hora de obtener resultados en el algoritmo, pues la función de aptitud es el elemento que simula los factores ambientales que condicionan a los individuos para que sobrevivan o no.

De esa manera, sea $S = (p_1, p_2, \ldots, p_n)$ una solución factible, la función de aptitud viene dada por:

$$fitness(S) = w_f \times \boldsymbol{f}(S) + w_g \times \boldsymbol{g}(S) \quad (10)$$

Donde:

- w_f es el peso asociado al objetivo uno;
- $\boldsymbol{f}(S)$ es la función que calcula el valor de aptitud del individuo teniendo en cuenta la satisfacción del objetivo uno, i.e., la función (4);
- w_g es el peso asociado al objetivo dos;
- $\boldsymbol{g}(S)$ es la función que calcula el valor de aptitud del individuo teniendo en cuenta la satisfacción del objetivo dos, i.e., la función (5).

El valor máximo de la aptitud para un individuo dado, siempre es la suma de w_f y w_g, ya que las funciones $\boldsymbol{f}$ y $\boldsymbol{g}$ devuelven como máximo el valor $\mathbf{1}$ si se cumple completamente el objetivo. Por ejemplo, si el peso esablecido al objetivo número uno es tres (w_f = 3) y el peso establecido al objetivo número dos es uno (w_g = 1), la función *fitness* como máximo puede valer cuatro (*fitness* = 4).

4.3.5. Método de selección

El método de selección utilizado es el de selección por rueda de la ruleta (*roulette wheel selection*). Este método de selección establece una probabilidad de selección que es equivalente a la proporción de aptitud que un individuo tiene con respecto a la aptitud de la población total. La siguiente fórmula determina la probabilidad de un individuo dado a ser elegido:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{11}$$

Donde:

- p_i es la probabilidad de que el individuo sea seleccionado;
- f_i es el valor de aptitud del individuo i;

Existen dos inconvenientes que pueden dar problemas al utilizar este método de selección, que son:

- Peligro de Convergencia Prematura (*premature convergence*) a un sub-óptimo debido a que existen individuos que son mucho más aptos que los demás y dominan a la población completa muy rápidamente.
- Baja Presión Selectiva (*selection pressure*) cuando los valores de aptitud son muy cercanos entre sí. Se produce estancamiento en el proceso de búsqueda.

Sin embargo, estos inconvenientes dependen de la distribución de la aptitud en toda la población. Como la función de aptitud definida depende de los pesos w_f y w_g, se puede distribuir la aptitud de manera que la diferencia no pueda ser ni muy alta ni muy baja y evitar así los inconvenientes que tiene este método de selección.

Por otra parte, se establece también una selección elitista para los dos mejores individuos, de manera que en el transcurso de las generaciones, siempre sobreviven los dos mejores individuos, sin importar si son elegidos o no. Esta selección elitista asegura que la mejor solución encontrada no se pierda y que se mantenga hasta que se encuentre una mejor.

4.3.6. Método de cruce

El método de cruce involucra a dos progenitores seleccionados mediante la técnica de la rueda de la ruleta para formar dos descendientes. Tanto el número de progenitores seleccionados por cada cruce como el método de selección son valores fijos del algoritmo y no pueden ser cambiados.

Al ser cromosomas compuestos por genes de dos tipos distintos (operadores y números de pieza), el cruce debe ser especial. Dado que los genes que indican números de pieza y los genes que indican operadores no comparten las mismas características y restricciones en el cromosoma, éstos se separan de los individuos antes de realizar el cruce, de manera que con la ayuda de una máscara de bits, se seleccionan todos los operadores por un lado y todos los números de pieza por otro para combinarlos con los de otro individuo y después unirlos para formar la descendencia.

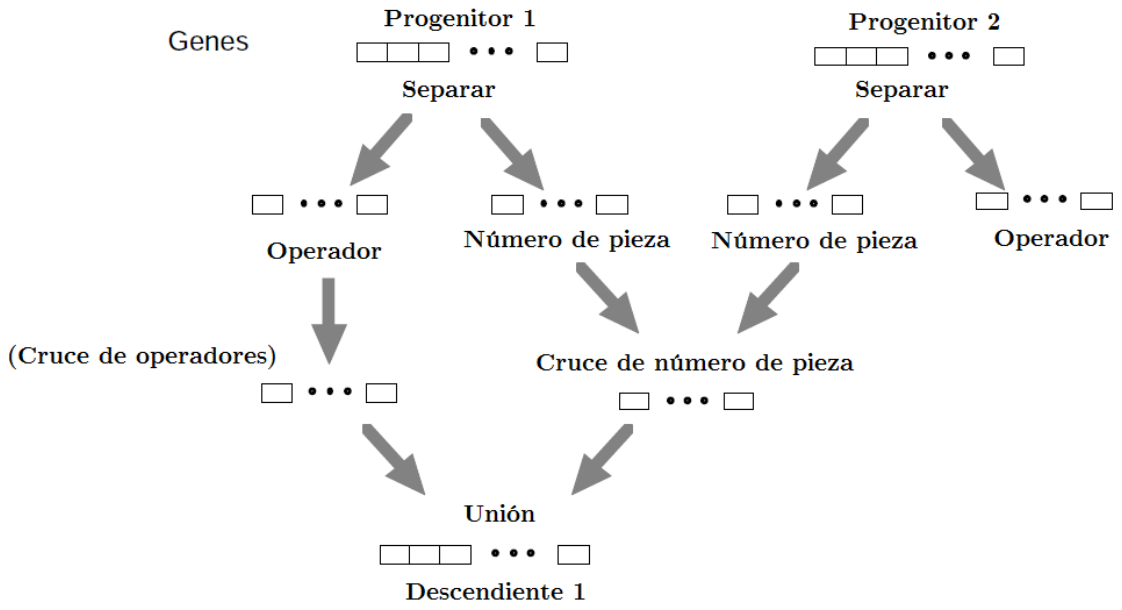


Figura 11: Separación de los genes

En cuanto al cruce, en primer lugar, dada la naturaleza ordinal de los genes del cromosoma que contienen números de pieza, no se puede emplear cualquier método de cruce, como por ejemplo el cruce por intercambio directo, pues los cromosomas resultantes no serían válidos, porque tendrían números de pieza repetidos y otros faltantes. Es por eso por lo que hay que emplear métodos para recombinar **cromosomas ordenados**. Algunos ejemplos de este tipo de cruce son el *PMX* (*Partially-mapped crossover*), el *OX* (*Order crossover*) o el *CX* (*Cyclic Crossover*) [30]. Según [31], el método de cruce de los anteriormente mencionados que mejor se adapta a la codificación elegida de los individuos es el *PMX*, ya que es el método con el

que se obtienen los mejores resultados de convergencia con el transcurso de las generaciones.

En segundo lugar, para los operadores, como éstos no tienen que seguir ningún orden y pueden estar repetidos o no, no es necesario realizar un cruce especial, por lo que se intercambian de manera directa y se colocan en las mismas posiciones que uno de los cromosomas de los padres para asegurar que el cromosoma es válido.

El siguiente ejemplo muestra un cruce completo dados los cromosomas de dos individuos:

Progenitor 1	1	3	H	2	4	V	H	5	6	V	V

Progenitor 2	2	3	V	4	V	1	H	5	V	6	H

Números de pieza del progenitor 1	1	3		2	4			5	6		

Compacto 1	1	3	2	4	5	6

Operadores del progenitor 1			H			V	H			V	V

Números de pieza del progenitor 2	2	3		4		1		5		6	

Compacto 2	2	3	4	1	5	6

Operadores del progenitor 2			V		V		H		V		H

Se aplica PMX a “Compacto 1” y “Compacto 2”:

1. Se realiza intercambio directo de los genes comprendidos en el segmento (aleatorio) [3, 4] (ambos inclusive), de las listas de números de pieza compactas y se obtiene:

Compacto 1 post-intercambio	1	3	4	1	5	6

Compacto 2 post-intercambio	2	3	2	4	5	6

2. Se establece la relación de los números comprendidos en el segmento:

$$4 \leftrightarrow 2 \leftrightarrow 1$$

3. Se intercambian los valores comprometidos que estén fuera del segmento por uno válido de la relación:

Compacto 1 validado

2	3	4	1	5	6

Compacto 2 validado

1	3	2	1	5	6

4. Se intercambian los operadores que se encuentren dentro del segmento delimitado por los números de pieza elegidos aleatoriamente en el paso 1. En este caso, como el cromosoma del progenitor no tiene operadores entre el gen que indica el número de pieza 2 y el número de pieza 4, no se realiza cruce de operadores.
5. Las listas de operadores se combinan con las listas compactas de los progenitores para obtener a los dos descendientes:

Descendiente 1

2	3	H	4	1	V	H	5	6	V	V

Descendiente 2

1	3	V	2	V	1	H	5	V	6	H

4.3.7. Método de mutación

La mutación adoptada consiste en el intercambio de los elementos de dos genes elegidos aleatoriamente, sean genes que contengan un operador o un número de pieza. La probabilidad de mutar se aplica a todos los individuos excepto a los individuos élite una vez se ha formado la nueva generación, constituida íntegramente por la descendencia obtenida en los cruces como se indica en el punto 4.3.8.

Nótese que si se tiene en cuenta que la ecuación (9) debe cumplirse para que un cromosoma represente una solución válida, no todos los intercambios son aceptables.

Por eso mismo, sean p_1 y p_2 dos elementos cualesquiera del cromosoma y p_1 esté localizado a la izquierda de p_2, el intercambio mutuo es posible si:

- p_1 y p_2 son números de pieza,
- p_1 es un operador, independientemente del tipo que sea p_2,
- p_1 es un número de pieza, p_2 es un operador y después de ser intercambiados, se cumple (9).
  Sin embargo, como es deseable saber si el resultado es válido **antes** de intercambiar el contenido de los genes, se puede generalizar y expresar de la siguiente manera:

  p_1 es un número de pieza, p_2 es un operador y todos los operadores que existan entre p_1 y p_2 deben cumplir que:

$$n_o \leq n_p - 3 \tag{12}$$

4.3.8. Método de reemplazo

El método de reemplazo consiste en que toda la población (excepto los n individuos élites) es reemplazada por la descendencia generada, de manera que entre generaciones únicamente sobreviven los individuos élite. Por ejemplo, si el tamaño de la población es de 50 y el número de élites designados es 2, para pasar de la generación k a la generación k+1, se deben realizar 24 cruces de 48 progenitores (obviamente pueden repetirse progenitores en distintos cruces) que dan lugar a 48 individuos nuevos, que son los que forman la población de la generación k+1.

5. Implementación

Para resolver el problema y poner en práctica las diversas ideas que se han redactado anteriormente en este documento, se ha desarrollado una aplicación completamente desde cero que brinda al usuario la capacidad de seleccionar el tamaño y el número de piezas que desea obtener de la pieza original, además de visualizar el mejor individuo de cada generación y avanzar paso por paso o de golpe hasta que el mejor individuo cambie, todo ello a efecto de demostrar la capacidad y corrección del algoritmo.

5.1. Software utilizado

Para la implementación de la aplicación se han utilizado dos herramientas principalmente, que son Unity3D[13] para el desarrollo de la interfaz gráfica y Microsoft Visual Studio[14] para el desarrollo de los *scripts* que se encargan de la funcionalidad. El lenguaje en el que se han desarrollado los *scripts* es Visual C#. La razón de que estas herramientas hayan sido elegidas es, entre otras, la versatilidad que proporciona la integración de aplicaciones escritas en C# con Unity3D, además de que C# es un lenguaje flexible, aunque fuertemente *tipado*, cosa que facilita la depuración y mantener la estructura del programa.

5.2. Estructura de la aplicación

La aplicación está formada por dos *scripts* diferentes. El primero de ellos se encarga mayormente de las acciones que el usuario puede realizar en la ventana principal de la aplicación y al funcionamiento en general de ésta, que corresponde con la especificación del problema, mientras que el segundo se encarga principalmente de las acciones ocurridas en la segunda ventana de la aplicación, como procesar el algoritmo genético y mostrar los resultados. En la segunda ventana, el usuario únicamente es capaz de avanzar generaciones en el algoritmo genético.

Al iniciar la aplicación se crean las variables y estructuras de datos vacías que son necesarias para la ejecución de las diversas acciones que debe realizar el primer *script*. No ocurre así con las variables y estructuras de datos del segundo, ya que éstas son creadas cuando se ha terminado de confeccionar el problema en la primera ventana y el usuario pulsa el botón "*Start*". Las variables y estructuras de datos más importantes que se generan en el primer *script* son:

- Variables que contienen las direcciones a los elementos gráficos de *Unity*, como son los *buttons*, *sliders*, *inputFields*, *texts* y demás.
- Una lista genérica de *GameObjects*[15], que es la que contiene en el momento de la ejecución la lista de las piezas seleccionada por el usuario.
- Un nodo inicial del árbol binario que se utiliza en el momento de la ejecución para dividir el espacio de la pieza original conforme se coloquen las piezas. Es utilizado por un algoritmo voraz que dispone las piezas. Este algoritmo se ha implementado para ofrecer al usuario una aproximación visual de las piezas que ha seleccionado, puestas en la pieza original.

[13] Unity Versión 5.3.1p3 Personal
[14] Microsoft Visual Studio Community 2015. Versión 14.0.24720.00 *Update* 1
[15] Un *GameObject* en *Unity* es un objeto genérico que puede representar cualquier entidad y tiene las características básicas de un objeto en *Unity*, como es la posición, el tamaño, y la jerarquía de objetos a la que pertenece.

La función principal de la primera ventana es que el usuario especifique las piezas que desea y el tamaño de éstas. Cuando el usuario indica el tamaño que desea de una pieza en concreto y pulsa el botón "*Add Piece*", la aplicación instancia el *Panel* que representa la pre-visualización de la pieza, con lo que se consigue una pieza idéntica para colocar en el panel utilizado como lienzo (*canvas*) que representa la pieza original.

Dado que la pieza instanciada es idéntica, lo único que hay que realizar después de instanciarla es cambiar la posición para que se disponga en el *Panel* que representa la pieza original. Para realizar dicha acción de manera sencilla, a la pieza nueva se leestablece como padre (*Parent*) en la jerarquía de objetos el *Panel* de la pieza original, para que así su posición relativa (relativa al padre) equivalga con la posición real que el objeto tendría en la disposición. De esa manera, la primera pieza tiene siempre la posición relativa (0, 0) con respecto al *Panel* que actúa como pieza original. Además, el script se encarga de mantener la lista de *GameObjects*, de manera que el usuario puede borrarla y empezar de nuevo con el botón "*Clear*".

Sin embargo, en ese proceso no se calcula la primera posición que tiene la pieza para que el usuario la visualice. Ese proceso sólo se encarga de instanciar la pieza de la pre-visualización y colocarla de manera relativa a la pieza original dada una posición. La posición de la pieza a priori la determina el algoritmo voraz que utiliza el árbol binario para separar el espacio de la pieza original. Para que el algoritmo sea capaz de obtener los datos y manejarlos de una manera sencilla, se han *virtualizado* los paneles en forma de objetos de la clase *Rectangle*. Así, un panel (i.e., pieza) es representado por un objeto que sólo tiene posición, longitud y anchura.

Una vez el usuario ya ha seleccionado las piezas que desea, la lista genérica de piezas está construida y se da paso al proceso genético.

5.3. Implementación del algoritmo genético

El segundo *script* se encarga de todo el proceso del algoritmo genético: generación de la primera población de manera aleatoria, selección, mutación, cruce, etc. A través del método *GetComponent()*, se puede obtener cualquier componente de un objeto en *Unity*. En esta aplicación, los dos scripts se encuentran como componentes de un objeto vacío llamado *Script_Holder*. A través de *GetComponent()*, es posible obtener las variables de un *script* en otro, y es así cómo se transfiere la lista de piezas de un script a otro.

Antes de proceder con la explicación de la implementación de los diferentes aspectos del algoritmo genético, cabe destacar que se ha desarrollado una clase llamada *Utilities* con diferentes métodos que sirven de apoyo en muchas ocasiones en el resto del código.

Los métodos más importantes son:

- *Unfold_relationships(…)*: Recibe de entrada un diccionario cuyas claves son cadenas (*Strings*) y los valores de las claves son listas de cadenas. El método no devuelve nada, pues los cambios realizados en el diccionario se mantienen, ya que el paso de diccionarios en C# es por referencia.
  La finalidad de este método es desplegar las relaciones existentes en el diccionario, de manera que si para la clave "1" el valor es "2" y para la clave "2" el valor es "3, 1", el valor de la clave 1 pasará a ser "1, 2, 3" y el valor de la clave "2" pasará a ser "1, 2, 3". Éste método se reduce a aplicar la transitividad para no sea necesario realizar un proceso de búsqueda e implementar la detección de ciclos cuando se quiera obtener todos los valores con los que está relacionada una clave en concreto.
  Éste método se utiliza en el apartado de cruce, y sirve para construir el diccionario que especifica las relaciones entre los números de pieza de los cromosomas (véase paso 2 del *PMX más atrás*).

- *is_a_number(…)*: Éste método está sobrecargado de manera que puede ser llamado con un parámetro o dos. El único parámetro en el primer caso es una cadena de caracteres, y de lo que se encarga el método es de intentar realizar un análisis léxico a la cadena y determinar si ésta es un número o una letra[16]. En el segundo caso, el primer parámetro es también una cadena de caracteres y el segundo parámetro es una variable de tipo entero pasada por referencia mediante "*out*" para que el método devuelva en ella el número que se ha leído, si era un número. El método en sí devuelve *true* o *false* dependiendo de si la cadena contiene un número o no.
  Éste método se utiliza en todas las ocasiones en las que se requiere realizar una distinción en el contenido de un gen en particular de un cromosoma, es decir, cuando se desea saber si un gen contiene información sobre un número de pieza o sobre un operador.

[16] Se supone que siempre se obtiene un carácter a pesar de que el tipo de dato sea de cadena.

- *get_random_double_between(...)*: Devuelve un número real entre los dos especificados como parámetros. El tercer parámetro es un objeto de tipo *System.Random* que es necesario para evitar que los números creados en un intervalo de tiempo corto obtengan la misma semilla y por lo tanto sean iguales.

5.3.1. Codificación de las soluciones

Las soluciones codificadas toman forma de lista de cadenas (lista de *strings*) para representar los cromosomas. El hecho de que sean cadenas es debido a la mayor facilidad de manejo con respecto a caracteres (tipo *char*), y siempre y cuando se cuide la creación y modificación de los cromosomas, éstas listas contendrán únicamente elementos con un carácter, por lo que no debe inducir a error.

Por tanto, siguiendo la representación del genotipo de los individuos expuesta en el apartado de diseño, se puede establecer la siguiente relación para llevarlo a la práctica:

- El cromosoma se representa mediante una lista de cadenas (*List<string>*);
- Cada gen del cromosoma es un elemento de la lista;

Aunque la lista tiene un tamaño dinámico, dado un número de piezas n_p, el tamaño del cromosoma (número de genes, n_g) siempre es:

$$n_g = 2n_p - 1 \tag{13}$$

Nótese que no es necesario implementar ningún algoritmo que codifique las soluciones, ya que inicialmente, las soluciones que se generan aleatoriamente ya están codificadas, y cuando se crean nuevas a partir del cruce, también están codificadas. Este aspecto unidireccional en el ámbito de la codificación y decodificación hace prescindible cualquier método para codificar las soluciones.

5.3.2. Decodificación de las soluciones

El proceso de decodificación de las soluciones es el encargado de obtener una representación visual de la solución a partir del genotipo, en este caso, de la lista de cadenas que representa el cromosoma. Este proceso es relativamente largo y existen varias fases en las que se ha centrado la implementación.

La idea principal es utilizar una pila de árboles binarios, de manera que la pila sigue el esquema de construcción de la figura 10, pero los elementos que se apilan y quitan de la pila para ser integrados y apilados de nuevo son árboles binarios, que establecen una

jerarquía de construcción. Las piezas se representan con objetos de la clase *Rectangle_Node*, que son árboles binarios que contienen objetos que designan rectángulos (i.e., tienen hijo derecho, hijo izquierdo, y un objeto de la clase *Rectangle*, con su posición, longitud y ancho). Cuando dos piezas son sacadas de la pila para ser integradas, lo que el algoritmo realiza es una construcción de un árbol binario, en el que el nodo raíz representa el rectángulo integrado por los rectángulos que representan los dos nodos descendientes.

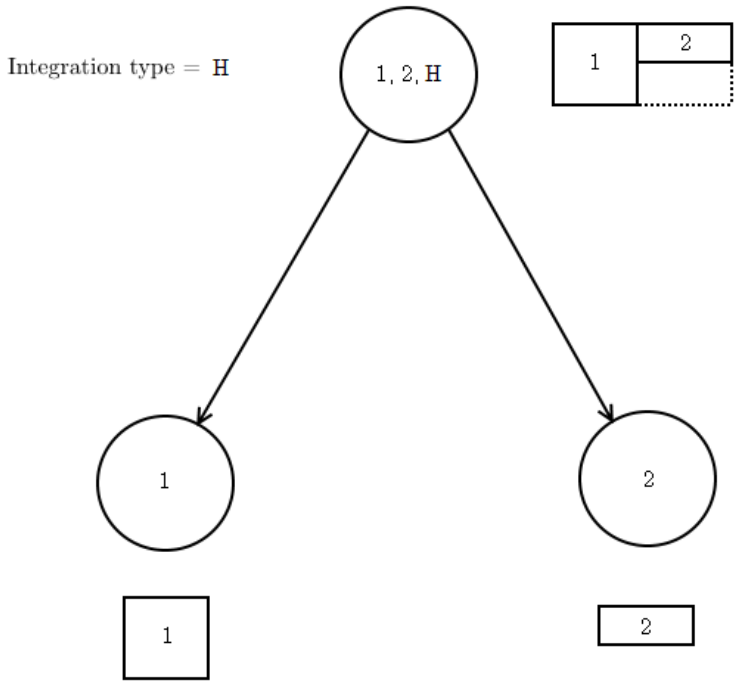


Figura 12: Ejemplo de construcción del árbol a partir de una operación de tipo H

De esta manera, cuando se ha apilado el último objeto, en la pila queda únicamente un árbol, cuyo nodo raíz contiene el rectángulo que integra todas las piezas en sus hijos. Todos los nodos del árbol que no sean hojas, corresponden a rectángulos integrados, mientras que todos los nodos hoja representan piezas en sí mismas.

Sin embargo, este proceso sólo es para construir la jerarquía de piezas integradas que forman las piezas deseadas y otras piezas integradas entre sí, además de calcular sus tamaños, pero las posiciones relativas no se establecen. El método implementado *calculate_positions* recorre el árbol de manera recursiva para establecer las posiciones de las piezas (sean piezas en sí o piezas integradas) relativas con la posición de sus nodos padre. En principio, el tipo de recorrido no importa, por lo que se adopta un recorrido en *Post Orden*. A continuación se presenta el algoritmo en pseudocódigo:

```
Calcular_posiciones(nodo)
        SI el nodo actual es una hoja:
                no hacer nada
```

```
SI NO:
        Calcular_posiciones(nodo.hijo_izquierdo)
        Calcular_posiciones(nodo.hijo_derecho)
        Establecer posición relativa del hijo izquierdo a (0,0)
        SI la integración es de tipo V:
                y = - nodo.hijo_izquierdo.longitud
                Establecer posición relativa del hijo derecho a (0,y)
        SI NO:
                x = nodo.hijo_izquierdo.ancho
                Establecer posición relativa del hijo derecho a (x,0)
```

De esta manera, se calculan las posiciones relativas de las piezas con respecto a sus nodos padre. Nótese que el hijo izquierdo siempre es la primera pieza colocada en la pieza integrada, y es por esa razón por la que su posición relativa siempre es (0,0). Sin embargo, a no ser que ese nodo sea el nodo hoja al que se llega si siempre se toma la rama de la izquierda, llega un momento en el que la ramificación es hacia la derecha y es ahí cuando de manera relativa se posiciona la pieza integrada o pieza normal a la derecha o abajo (según la operación) de otra pieza integrada o pieza normal, condicionando con ello las posiciones de todos los nodos descendientes que haya a partir de ahí.

Cuando se han calculado todas las posiciones relativas respecto a los padres en el árbol, el nodo raíz representa la solución sin ambigüedad, y es utilizado por el método *draw_pieces* que, mediante un concepto similar de jerarquía y posiciones relativas, construye la solución para que pueda ser visualizada. Nótese que tanto la información del tamaño de las piezas integradas como la posición de éstas se encuentran en el *Rectangle_Node* correspondiente dentro del árbol.

5.3.3. Función de aptitud

La implementación del cálculo de la función de aptitud es trivial, ya que tan sólo es necesario tomar los valores de longitud y ancho del nodo raíz del árbol una vez está construido y aplicar las operaciones indicadas en las ecuaciones de los objetivos, i.e. (4) y (5).

5.3.4. Método de selección

La implementación del método de la ruleta es relativamente sencilla. El proceso puede ser resumido en los siguientes pasos:

1. Suma. Se calcula la suma de la aptitud de todos los individuos de la población y se almacena en *S*.
2. Selección. Se genera un número aleatorio en el intervalo (0, S) y se almacena en *R*.
3. Bucle. Se recorre la población y se mantiene una suma parcial, *sum,* que empieza en 0 y se suma el valor de la aptitud de cada individuo que se recorre. En el momento en el que la suma parcial *sum* sobrepasa *S,* el último individuo que ha sumado su aptitud en *sum* es el seleccionado.

Nótese que cuanto mayor es la aptitud del individuo, si el proceso aleatorio que obtiene *R* sigue una distribución uniforme (que en teoría, lo es) mayor será la probabilidad de que sea escogido.

5.3.5. Método de cruce

La implementación del método de cruce sigue el esquema mostrado en el ejemplo del apartado de diseño del cruce. En primer lugar, se crean las listas que contienen los genes, los números de pieza y los operadores de ambos padres. En las listas números de pieza no existen huecos, mientras que en las de los operadores, sí. Posteriormente se crea un diccionario vacío cuyas claves son cadenas y los valores son listas de cadenas.

Después, se obtiene de manera aleatoria dos puntos cualesquiera de las listas compactadas (las listas de números de pieza) para concretar los puntos de corte. Una vez se conocen estos puntos de corte, se procede a realizar un recorrido por la lista de operadores del progenitor número uno y se determina qué operadores deben ser cambiados en el cruce. Una vez se conoce qué operadores deben ser cambiados, se recorren las dos listas de operadores (las correspondientes a ambos progenitores) y se cambian los que estén marcados, de manera ordenada (i.e., el tercer operador del cromosoma del progenitor uno sólo puede ser cambiado por el tercer operador del cromosoma del progenitor dos).

Para la parte de números de pieza, en primer lugar se realiza el intercambio directo de los genes que se encuentran dentro del segmento delimitado por los puntos de corte aleatoriamente obtenidos. En segundo lugar, se introducen en el diccionario las distintas relaciones que tienen los genes y se llama a la función *unfold_relationships* para desplegar las relaciones en el diccionario. En tercer lugar, se realiza el recorrido por las listas que determina si faltan números de pieza o existen repetidos y se modifica utilizando las relaciones del diccionario para legalizar los cromosomas. A continuación se muestra en pseudocódigo el funcionamiento del recorrido:

```
Desde i=0 hasta primer_punto_corte – 1
y desde  i= segundo_punto_corte + 1 hasta FIN:

    Desde j=primer_punto_corte hasta segundo_punto_corte :
        Si lista_progenitor1[i] y lista_progenitor1[j] son iguales:
            Para cada elemento relacionado con el elemento lista_progenitor1[j]
                Si ese elemento no está en el gen lista_progenitor1:
                    Intercambiar lista_progenitor1[i] por el elemento
                    Encontrar el elemento en la parte externa[17] de lista_progenitor2 e
                    intercambiar por el elemento que había antes en lista_progenitor1[i]
```

De esta manera, recorriendo únicamente la primera lista de número de piezas, se consigue legalizar ambos cromosomas, ya que los cambios en el primero se pueden aprovechar porque son los cambios inversos que se deben realizar en el segundo.

5.3.6. Método de mutación

La implementación del método de mutación sigue la línea descrita en el diseño del método. El método implementado *mutation*(...) devuelve un valor booleano que indica si la mutación se ha podido efectuar o no. La mutación no se puede efectuar si para todos los genes que se encuentran entre los dos genes seleccionados, ambos inclusive, no se cumple la restricción señalada por (12).

A continuación se muestra en pseudocódigo el funcionamiento del método:

```
Seleccionar dos genes de manera aleatoria (p1 y p2)
Si los dos genes contienen números de pieza:
Intercambiar los genes
    Devolver True
Si el primer gen contiene un operador:
    Intercambiar los genes
    Devolver True
Si el primer gen contiene un número de pieza y el segundo contiene un operador:
    Si cada gen entre p1 y p2, ambos inclusive, cumple que No ≤ Np – 3 :
    Intercambiar los genes
```

[17] Segmento o segmentos que no corresponden con el segmento seleccionado para el intercambio de genes. Se encuentran a la izquierda y a la derecha del segmento de genes seleccionado.

```
    Devolver True
Si no:
    Devolver False
```

De esa manera, se obtiene un método que intercambia dos genes aleatoriamente en el caso de que se pueda. Para asegurar la mutación para un individuo dado, es necesario realizar un bucle que lance este método hasta que devuelva *True*, que significa que la mutación se ha efectuado con éxito.

5.3.7. Transición entre generaciones

Una vez implementados todos los operadores genéticos, deben ser aglomerados en un mismo método que, dada una generación de individuos, aplique las operaciones genéticas en el orden debido para obtener la generación siguiente. De esta manera, el método implementado de nombre *compute_one_generation*(...), toma la lista de población, el tamaño de la población, la probabilidad de mutación, el número de élites que debe salvar en cada transición y calcula una generación aplicando los siguientes pasos:

1. Se seleccionan los *n* individuos élite y los añade a la nueva generación;
2. Se calcula la suma *S* de la aptitud de todos los individuos;
3. Hasta que la nueva población no esté completa:
    a. Se seleccionan dos progenitores;
    b. Se llama a la función *crossover*() y se obtienen los dos descendientes;
    c. Se aplica la mutación probable a los nuevos individuos generados;
    d. Se añaden los individuos a la población correspondiente a la nueva generación
4. Se sustituye la población actual por la población de la nueva generación.

5.4. Interfaz gráfica de usuario

La aplicación cuenta con una interfaz gráfica de usuario relativamente sencilla, con únicamente dos ventanas tal y como se ha introducido anteriormente. La primera ventana proporciona al usuario utilidades para que establezca el problema que desea resolver y los parámetros del algoritmo, mientras que la segunda ventana permite al usuario visualizar cómo mejora la solución del problema a lo largo de las generaciones. Además, el usuario tiene la capacidad de indicar que transcurra una generación, mil

generaciones o hasta que se mejore la solución. Sin embargo, dada la selección elitista y que el algoritmo puede quedarse en un óptimo local, es posible que al seleccionar la opción de calcular generaciones hasta que mejore, dure demasiado tiempo o incluso se quede en un bucle, por eso se ha implementado de manera que si transcurren diez mil generaciones al seleccionar esta opción, pare aunque no encuentre una mejor solución.

Ambas ventanas tienen un tamaño predefinido, que es 1210x658. Además, la aplicación funciona en modo ventana, y no se puede maximizar.

A continuación se muestran dos capturas de pantalla, una de cada ventana de la interfaz gráfica de la aplicación:

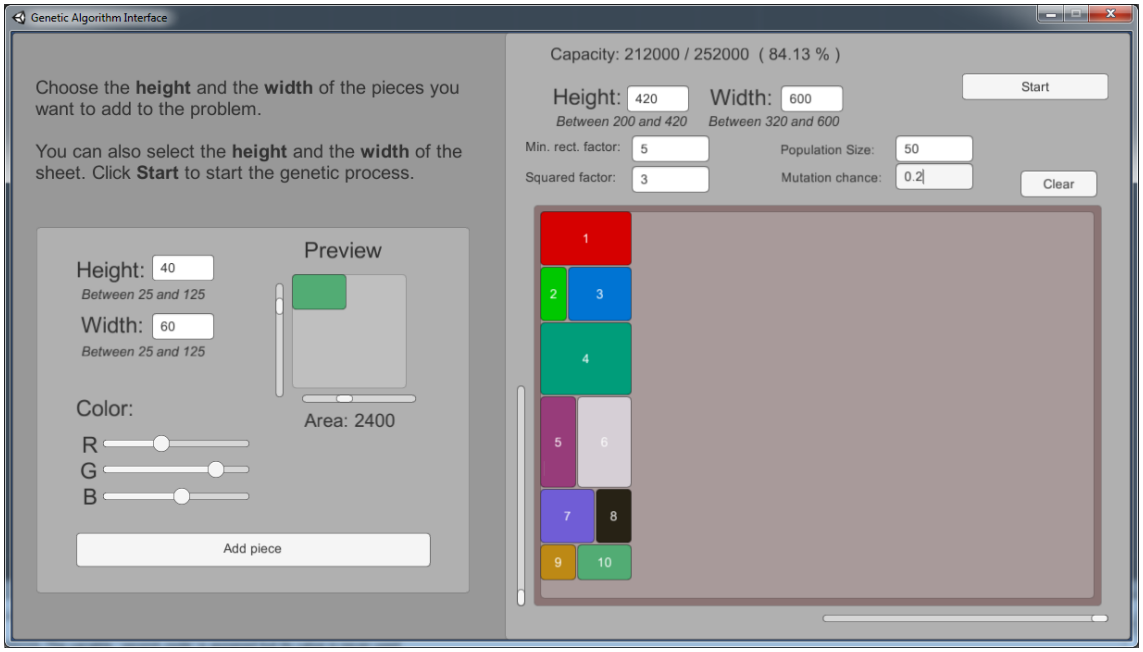


Figura 13: Interfaz gráfica de la primera ventana de la aplicación. Ejemplo de uso

Figura 14 Interfaz gráfica de la primera ventana de la aplicación. Ejemplo de resultado

6. Evaluación del sistema

Una vez se ha desarrollado la aplicación, es necesario probarla para, en primer lugar, comprobar que se obtienen resultados relativamente buenos y en segundo lugar, determinar que el tiempo que tarda en obtener esos resultados es razonable. Asimismo, como la ejecución depende de distintos parámetros, también es aconsejable probar diferentes combinaciones para evaluar el impacto de éstos sobre el resultado final obtenido. Por eso mismo, las pruebas diseñadas consisten en un problema relativamente pequeño (diez piezas), cuya solución óptima se conoce y se pretende evaluar la diferencia existente entre los resultados obtenidos y la solución óptima conocida. Además, también se desea evaluar el tiempo de cálculo, en este caso, tiempo requerido para calcular mil generaciones y el número medio de generaciones hasta obtener una solución aceptable.

Las pruebas que se han realizado consisten en resolver el problema de obtener las siguientes piezas y que se satisfagan los objetivos descritos:

- Pieza #1, de 100x60,
- pieza #2, de 30x60,
- pieza #3, de 70x60,
- pieza #4, de 100x80,
- pieza #5, de 60x60,
- pieza #6, de 40x60,
- pieza #7, de 40x40,
- pieza #8, de 60x40,
- pieza #9, de 60x100 y
- pieza #10, de 40x100.

Cuya solución óptima se conoce[18]:

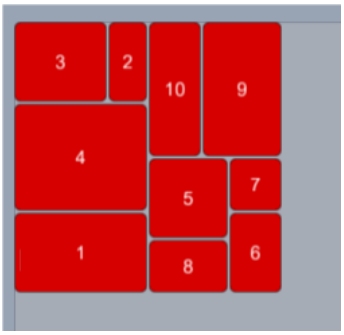


Figura 15: Una solución óptima al problema utilizado para realizar las pruebas.

[18] Existen varias soluciones óptimas, ésta es una de ellas.

El valor de los parámetros en las diferentes pruebas realizadas son los siguientes:

- Prueba #1:
  - Tamaño de la población: 30
  - Probabilidad de mutación: 0.2
- Prueba #2:
  - Tamaño de la población: 50
  - Probabilidad de mutación: 0.2
- Prueba #3:
  - Tamaño de la población: 70
  - Probabilidad de mutación: 0.2
- Prueba #4:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.2
- Prueba #5:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.05
- Prueba #6:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.35
- Prueba #7:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.5
- Prueba #8:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.35
  - Conjunto de 20 piezas diferentes
  - Pesos $w_f = 5$ y $w_g = 2$

Para todas las pruebas cuyos pesos de los objetivos no se han especificado, los pesos son de $w_f = 5$ y $w_g = 3$.

Y para cada prueba, se ha medido:

- Diferencia entre la solución óptima y la mejor solución alcanzada para un tiempo razonable, en este caso establecido a dos minutos de tiempo de cómputo (ver en la página 56),
- tendencia[19] de la aptitud del mejor individuo con el paso de las generaciones, mostrada en las gráficas
- tiempo de cálculo de diez mil generaciones (ver en la página 56),

[19] Los valores obtenidos consisten realmente en la aptitud del mejor individuo en cada generación. Al haber muchas generaciones en las que la mejor aptitud no cambia, el resultado sin interpolar es difícil de interpretar, ya que la línea descrita en el gráfico toma una forma abrupta con escalones. Por esa razón, se ha decidido interpolar y seguir la tendencia de la mejor aptitud para facilitar la lectura de las gráficas, aunque la línea existente entre dos puntos que representan datos no muestren el valor real del experimento.

- Prueba #1:
    - Tamaño de la población: 30
    - Probabilidad de mutación: 0.2

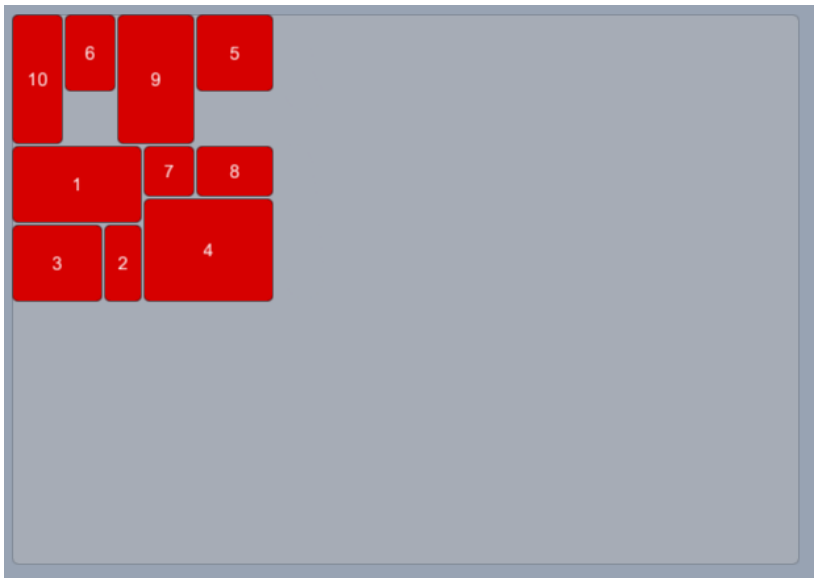


Figura 16: Fenotipo de la mejor solución obtenida en la prueba número uno

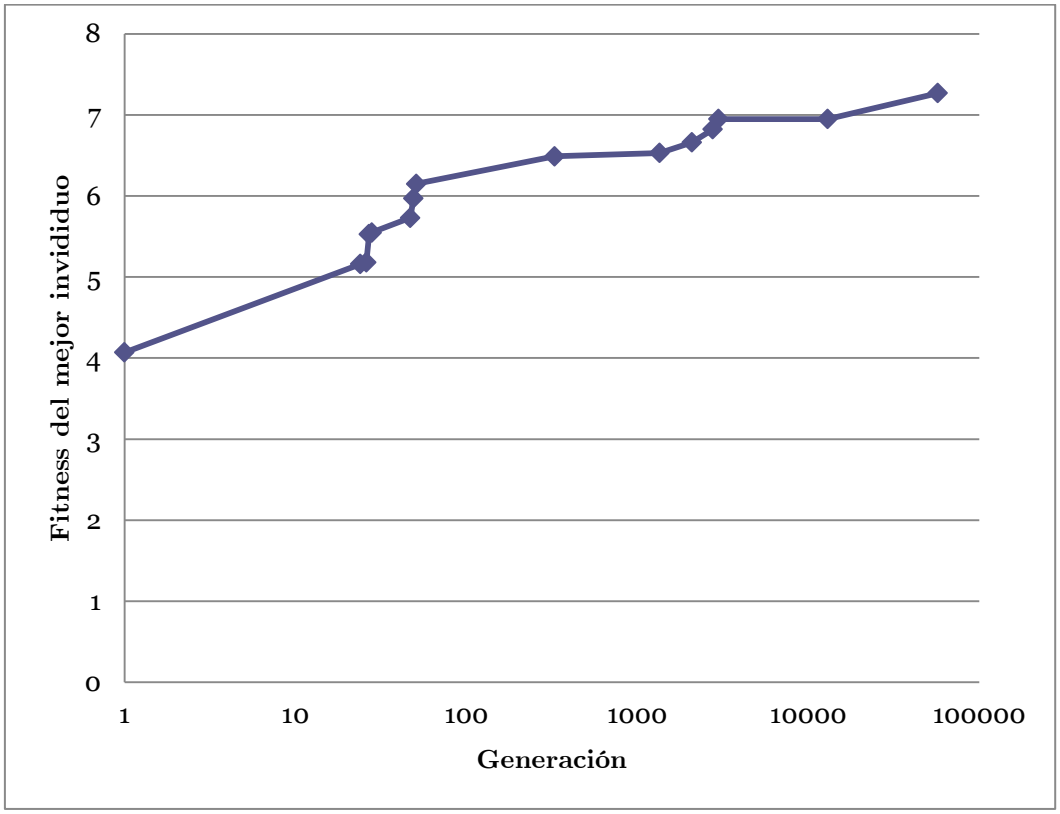


Gráfico 1: Evolución de la aptitud del mejor individuo por cada generación en la prueba número uno

- Prueba #2:
  - Tamaño de la población: 50
  - Probabilidad de mutación: 0.2

Figura 17: Fenotipo de la mejor solución obtenida en la prueba número dos

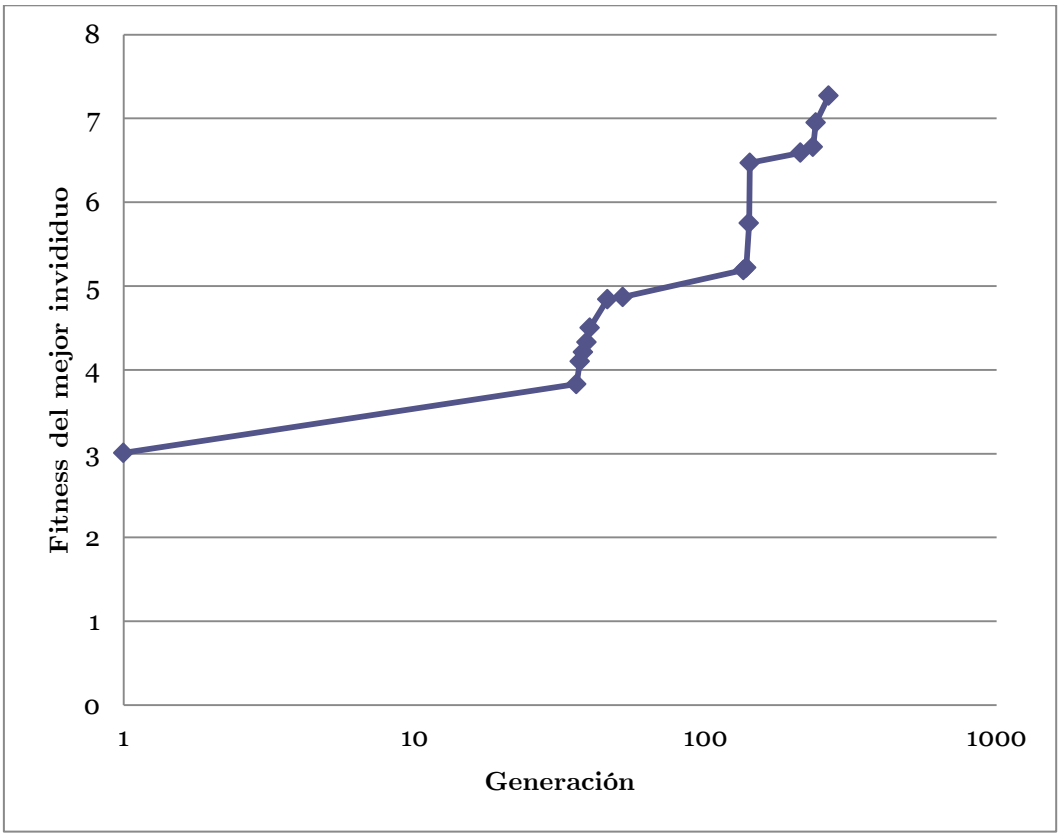


Gráfico 2: Evolución de la aptitud del mejor individuo por cada generación en la prueba número dos

- Prueba #3:
    - Tamaño de la población: 70
    - Probabilidad de mutación: 0.2

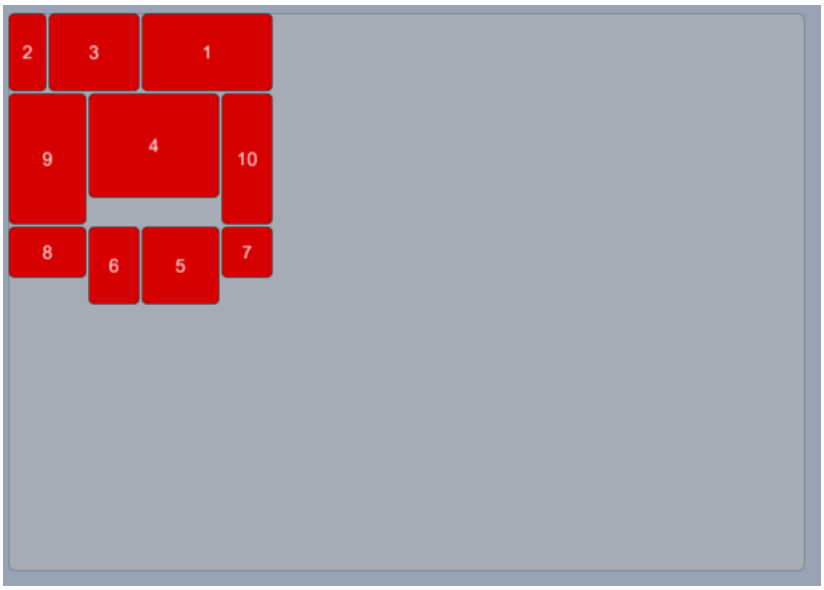


Figura 18: Fenotipo de la mejor solución obtenida en la prueba número tres

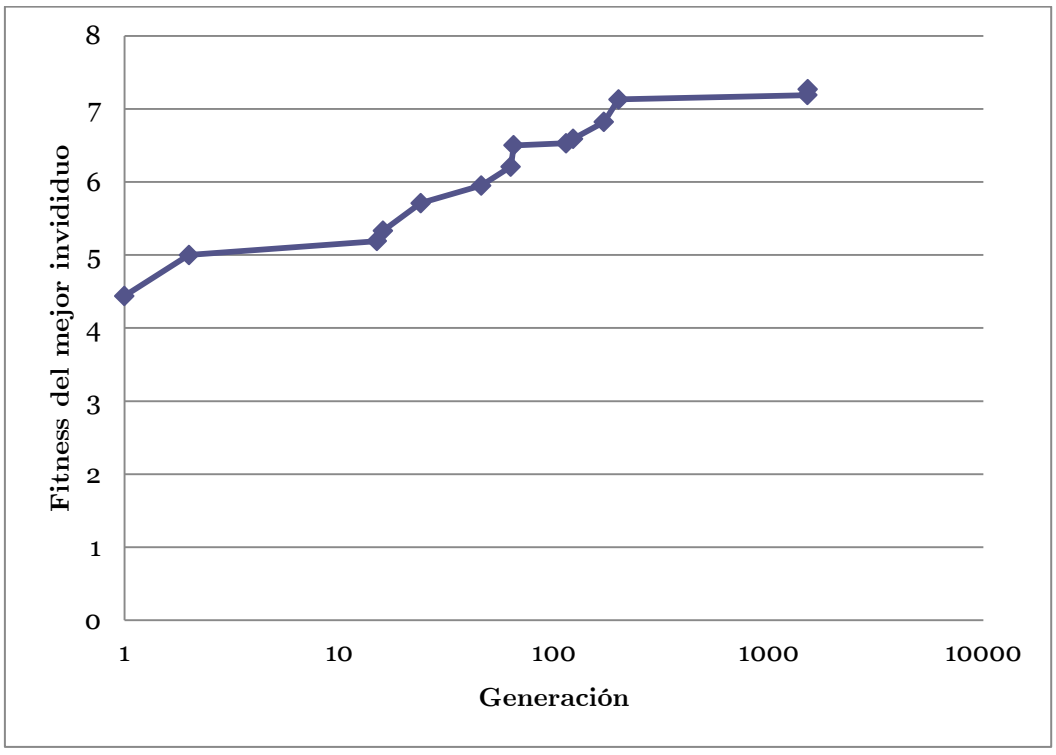


Gráfico 3: Evolución de la aptitud del mejor individuo por cada generación en la prueba número tres

- Prueba #4:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.2

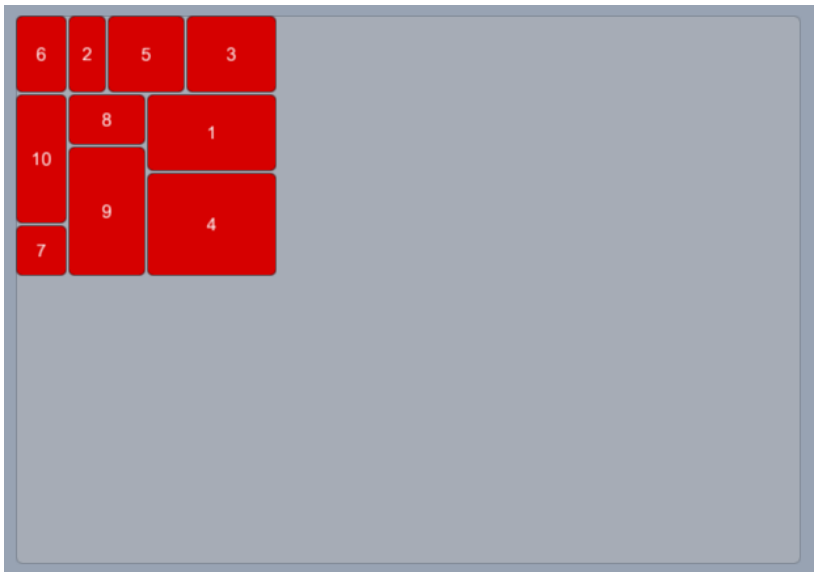


Figura 19: Fenotipo de la mejor solución obtenida en la prueba número cuatro

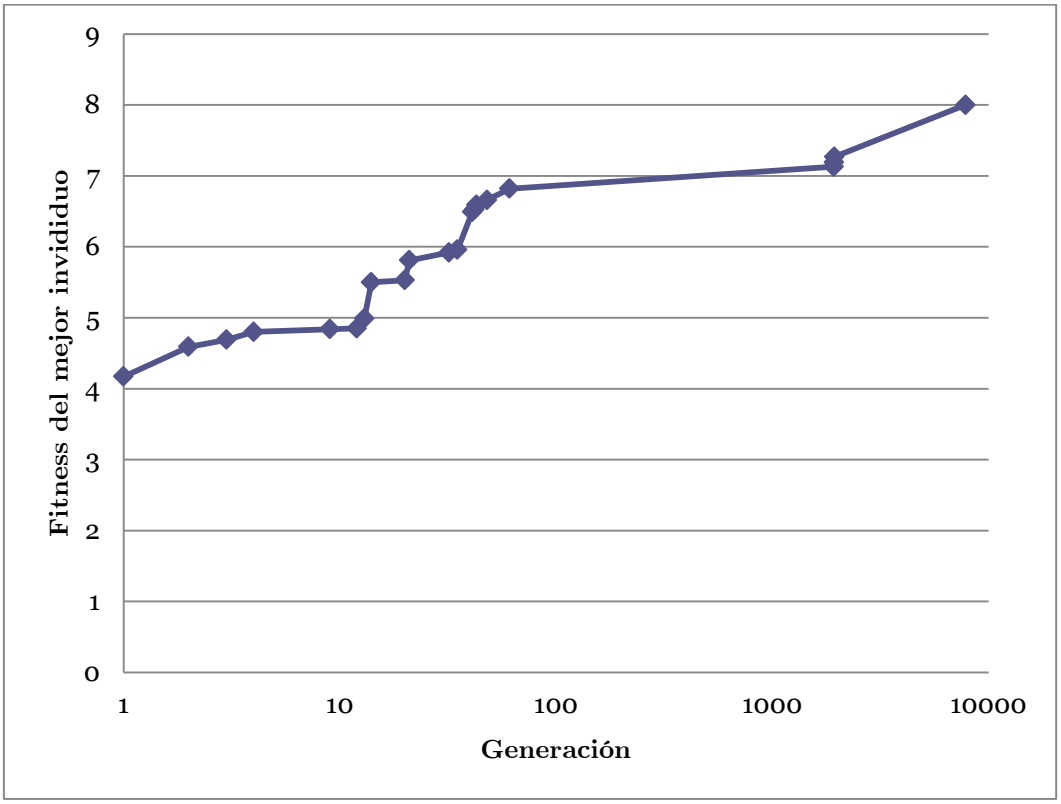


Gráfico 4: Evolución de la aptitud del mejor individuo por cada generación en la prueba número cuatro

- Prueba #5:
    - Tamaño de la población: 90
    - Probabilidad de mutación: 0.05

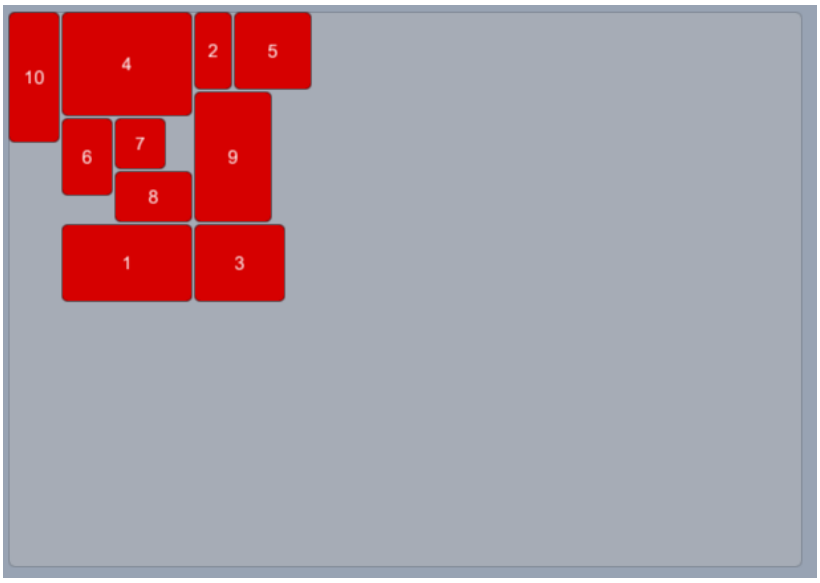


Figura 20: Fenotipo de la mejor solución obtenida en la prueba número cinco

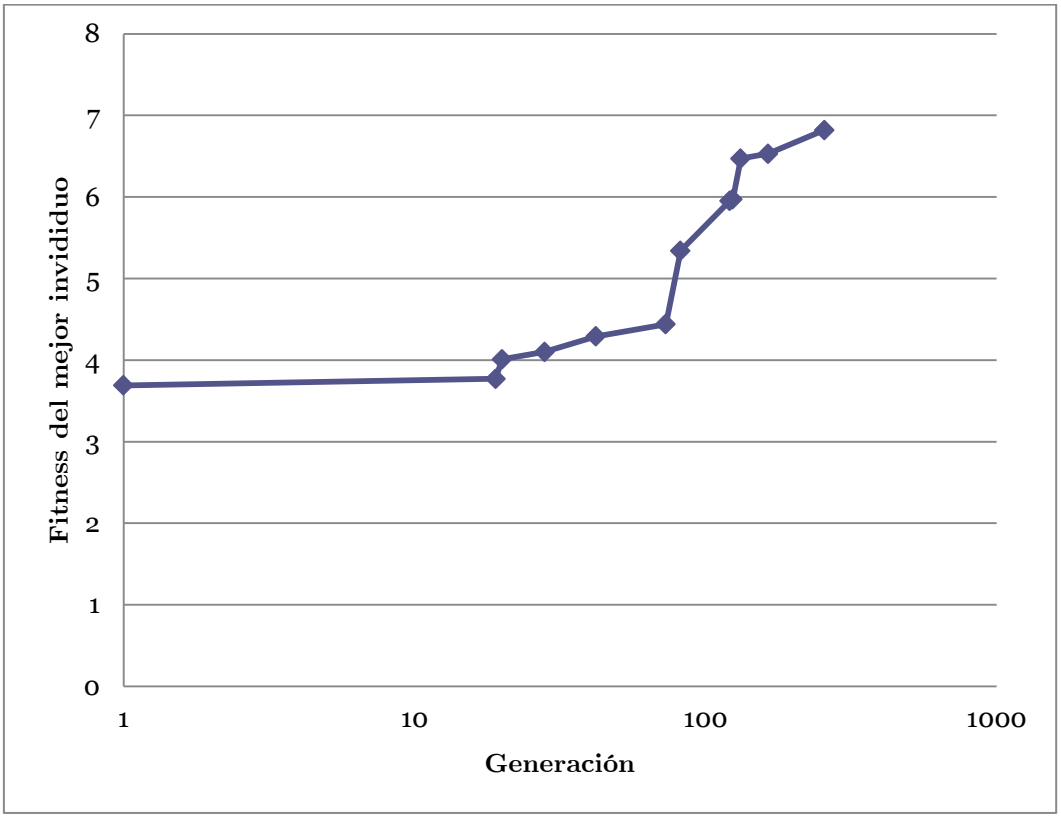


Gráfico 5: Evolución de la aptitud del mejor individuo por cada generación en la prueba número cinco

- Prueba #6:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.35

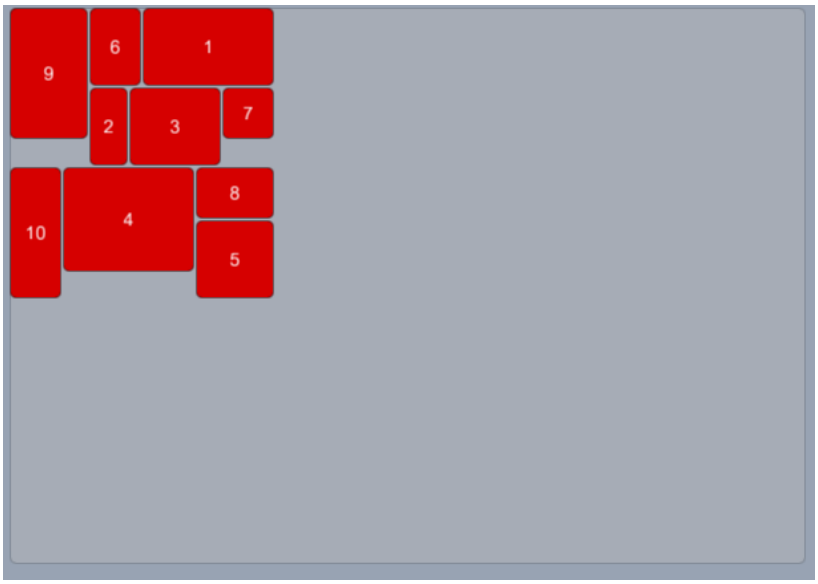


Figura 21: Fenotipo de la mejor solución obtenida en la prueba número seis

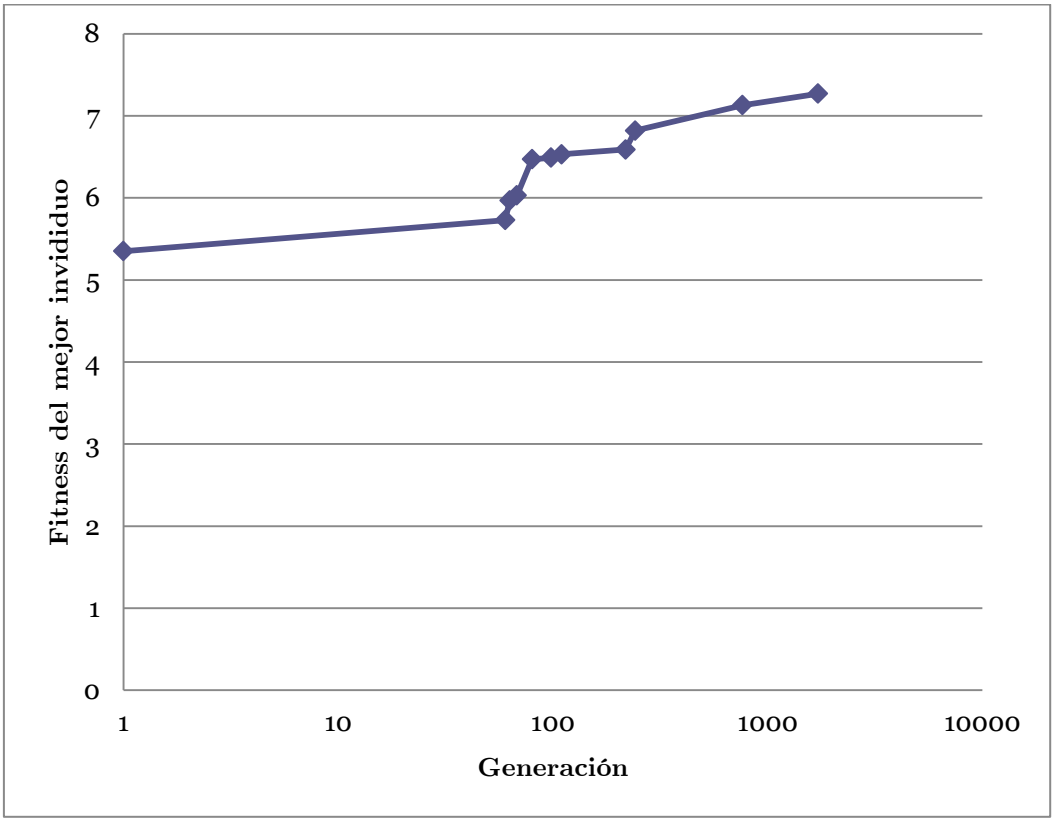


Gráfico 6: Evolución de la aptitud del mejor individuo por cada generación en la prueba número seis

- Prueba #7:
    - Tamaño de la población: 90
    - Probabilidad de mutación: 0.5

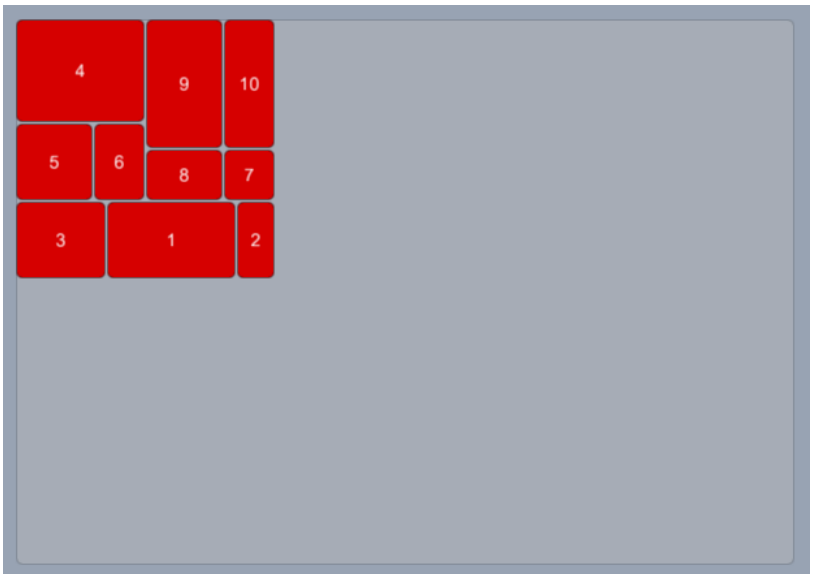


Figura 22: Fenotipo de la mejor solución obtenida en la prueba número siete

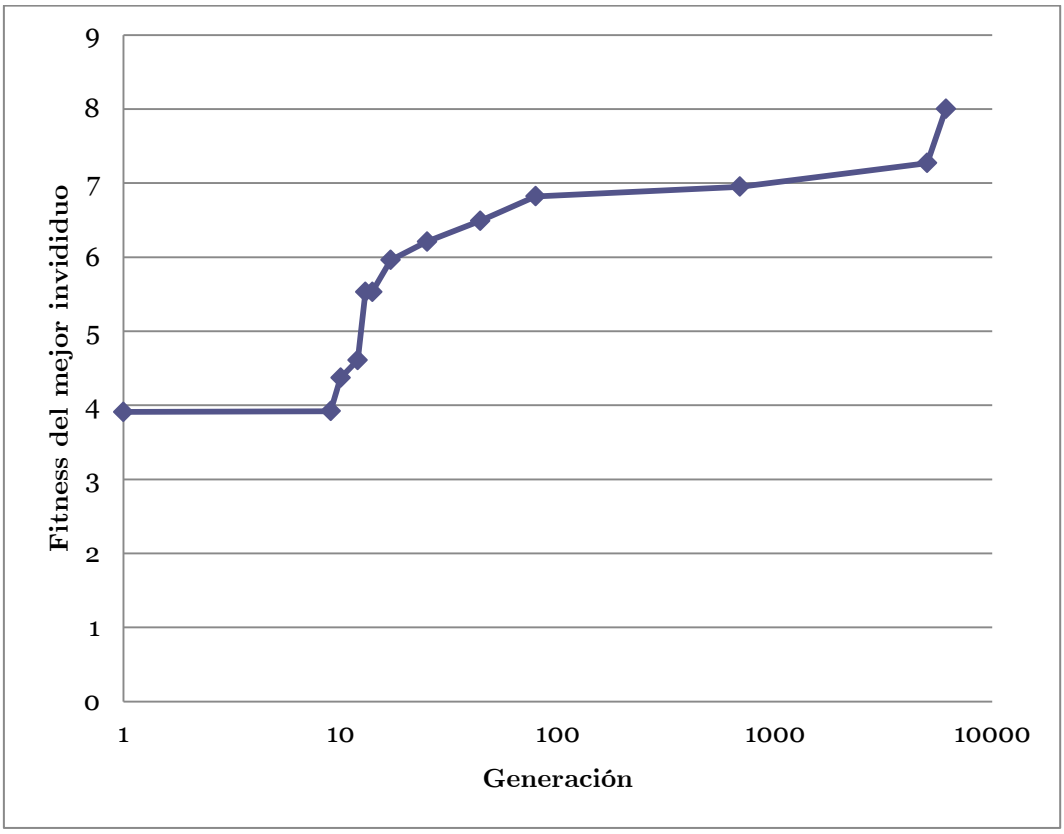


Gráfico 7: Evolución de la aptitud del mejor individuo por cada generación en la prueba número siete

- Prueba #8:
  - Tamaño de la población: 90
  - Probabilidad de mutación: 0.35
  - Conjunto de 20 piezas diferentes
  - Pesos $w_f = 5$ y $w_g = 2$

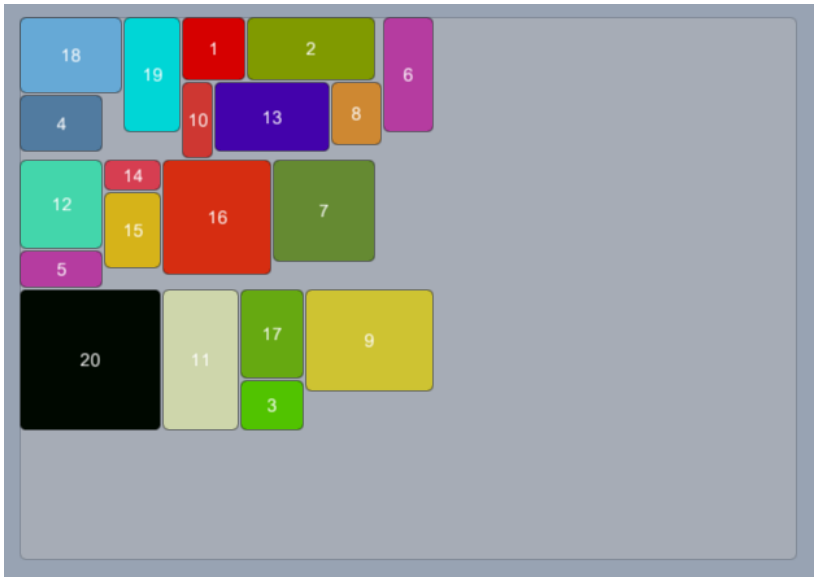


Figura 23: Fenotipo de la mejor solución obtenida en la prueba número ocho

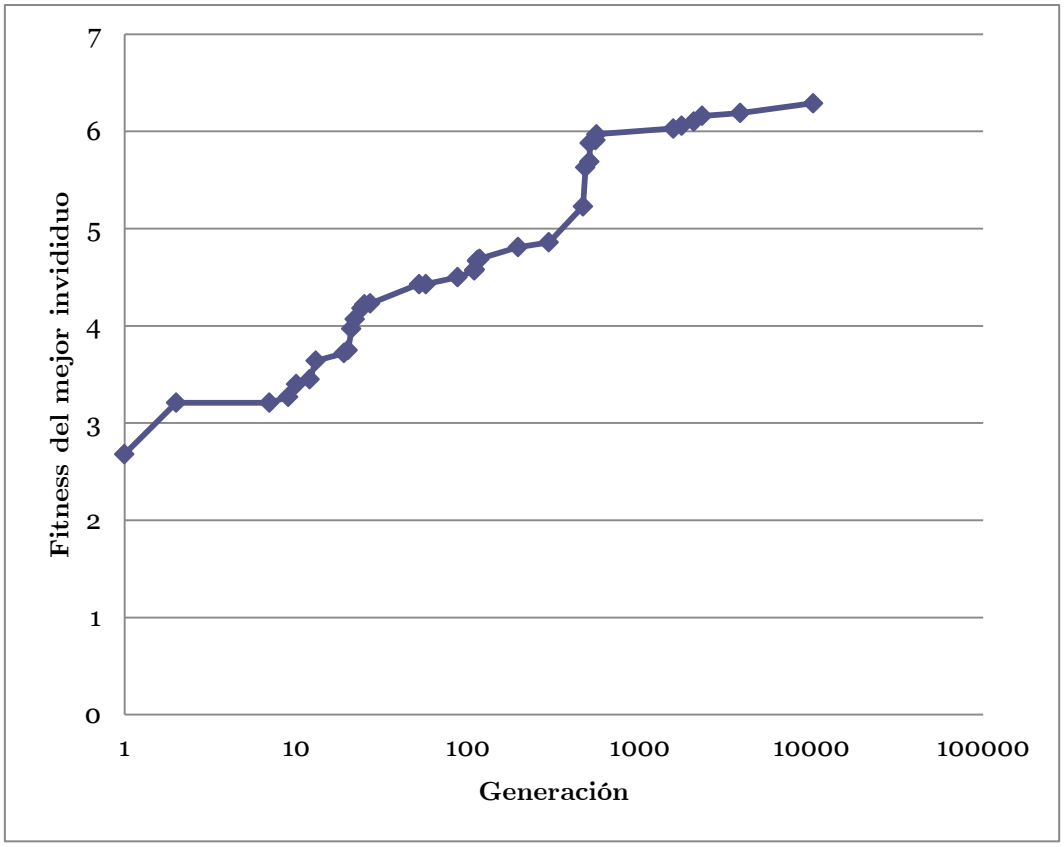


Gráfico 8: Evolución de la aptitud del mejor individuo por cada generación en la prueba número ocho

Comparación de configuraciones:

Configuración	Tamaño de la población	Número de piezas	Probabilidad de mutación	Peso del objetivo 1	Peso del objetivo 2
#1	30	10	0.2	5	3
#2	50	10	0.2	5	3
#3	70	10	0.2	5	3
#4	90	10	0.2	5	3
#5	90	10	0.05	5	3
#6	90	10	0.35	5	3
#7	90	10	0.5	5	3
#8	90	20	0.35	5	2

Comparación general de los resultados:

Configuración	Tiempo medio de cómputo de 10.000 generaciones (segundos)	Generación en la que se ha alcanzado la mejor solución	Tiempo aproximado de cómputo (segundos)	Aprovechamiento de material[20]
#1	**8.5**	57322	48.7	90.87 %
#2	14.9	265	**0.4**	90.87 %
#3	21.5	1520	3.2	90.87 %
#4	26.7	7835	20.9	**100 %**
#5	23.2	**256**	0.59	85.25 %
#6	30.7	1717	5.2	90.87 %
#7	32.6	6117	19.9	**100 %**
#8	47.5	10229	48.5	89.85 %

Conclusiones de la evaluación:

- La prueba con menor probabilidad de mutación es en la que se obtiene el peor resultado, y una de las dos pruebas en las que se obtiene el óptimo es con una tasa de mutación muy alta (0.5), con lo cual se deduce que la mutación ayuda a expandir el espacio exploratorio y obtener máximos locales mejores o incluso el máximo global.
- Aumentar la tasa de mutación aumenta ligeramente el tiempo de cálculo.
- Aumentar el número de piezas aumenta considerablemente el tiempo de cálculo.
- Con los datos obtenidos se induce que el tamaño de la población no impacta en el resultado final, pero sí en el tiempo de cálculo.

[20] Se calcula como máxima aptitud alcanzada dividido entre máxima aptitud alcanzable. La máxima aptitud alcanzable es la suma de w_f y w_g.

- La tendencia más o menos lineal que sigue el progreso de la mejor aptitud en las gráficas es realmente logarítmico ya que el eje de abscisas está en escala logarítmica. Esto significa que cada vez es necesario que transcurran más generaciones para que se haya una mejora de la aptitud.
- Las mejoras de aptitud obtenidas a partir de un número alto de generaciones son mayormente debidas a una mutación que da lugar a un individuo mejor, y no debido a un cruce, ya que en puntos tan avanzados, la población está relativamente convergida.

7. Conclusiones

A pesar de que ya existen algoritmos exactos para resolver problemas derivados de corte de material, éstos no son viables para problemas complejos, sobre todo si el tamaño de los problemas a resolver es relativamente grande. Sin embargo, la aplicación de algoritmos genéticos en este caso es tan viable como para cualquier otro problema *NP-Completo* de optimización, y no sólo es viable, sino que en algunos casos es incluso deseable. Los distintos algoritmos exactos existentes para el corte de material, son algoritmos específicos, diseñados para un problema en concreto, normalmente problemas pequeños, mientras que un algoritmo genético es fácilmente adaptable y sirve para cualquier variante del problema de optimización siempre y cuando se pueda transformar convenientemente.

Aunque los resultados obtenidos no son siempre óptimos, son considerablemente buenos en cuanto al cumplimiento de los objetivos en un tiempo computacional admisible, que es el objetivo de la aplicación de técnicas *metaheurísticas* como la que se ha utilizado para resolver el problema en este trabajo. No obstante, al tratarse de un algoritmo de carácter *anytime*, generalmente se puede esperar una mejor solución si se le permite más tiempo al algoritmo.

Además, dados los resultados obtenidos en las pruebas, se puede observar que es un proceso no determinista, que significa que el hecho de que se encuentre la solución óptima o que la población evolucione hacia unos individuos relativamente bien adaptados es un proceso aleatorio, que únicamente es guiado por la función objetivo, por eso en cada ejecución del algoritmo se obtiene una solución diferente. Una posible extensión del algoritmo que mejore las solución obtenida tomando el aspecto mencionado a su favor es convertir el algoritmo genético en un algoritmo genético paralelo, que sea capaz de empezar con un número determinado de poblaciones separadas (islas) y juntarlas cuando converjan, para asegurar una mayor divergencia poblacional cuando las poblaciones se hayan atascado en un óptimo local y ampliar el espacio explorado mediante el cruce de las poblaciones que han convergido.

8. Bibliografía

[1] Kantorovich, L.V. (1960), “Mathematical methods of organizing and planning production”, reprinted in *Management Science* 6, pp. 366 – 422.

[2] Gilmore, P.C., and Gomory, R.E (1961), “A linear programming approach to the cutting stock problem”, *Operations Research* 9, pp. 848 - 859.

[3] Gilmore, P.C., and Gomory, R.E (1961), “A linear programming approach to the cutting stock problem, Part II”, *Operations Research* 11, pp. 863 - 888.

[4] K. –H. Küfer, V. Maag, and J. Schwientek (2015): Maximale Materialausbeute bei der Edelsteinverwertung. Chapter 8 in H. Neunzert and D. Prätzel-Wolters (eds.): Mathematik im Fraunhofer-Institut. pp. 239-301 (in German, End of 2015 in English)

[5] Sweeney, P.E., and Paternoster, E.R. (1991), “Cutting and packing problems: An updated literature review”, Working Paper No.654, University of Michigan, School of Business.

[6] Dyckhoff, H. (1990), “A typology of cutting and packing problems”, *European Journal of Operational Research* 44, pp. 145 – 159.

[7] A. Dinges, P. Klein, K.-H. Küfer, V. Maag, J. Schwientek, and A. Winterfeld, “How to solve difficult cutting and packing problems? By semi-infinite optimization!”, Fraunhofer Institute for Industrial Mathematics ITWM.

[8] Lodi, A.; Martello, S.; Vigo, D. (1999), “Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems”, Informs Journal on Computing 11, pp. 345 - 357.

[9] Garey, M. R. and Johnson, D. S. (1979), “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W. H. Freeman & Co., New York, NY, USA.

[10] Chauny, F. et al. (1991), “A Two-phase Heuristic for the Two-dimensional Cutting-stock Problem”, J. Opl Res., Vol.42, No.1, pp. 39-47.

[11] Ghandforoush, P. and Daniels, J.J. (1992), “A Heuristic Algorithm for the Guillotine Constrained Cutting Stock Problem”, ORSA Journal on Computing, Vol.4, No.3, pp. 351-356.

[12] Morabito, R. and Arenales, M. N. (1996), “Staged and constrained two-dimensional guillotine cutting problems: An AND/OR-graph approach”, European Journal of Operational Research 94, pp. 548-560.

[13] MacLeod, B. et al. (1993), “An algorithm for the 2D guillotine cutting stock problem”, European Journal of Operational Research 68, pp. 400-412.

[14] Christofides, N. and Hadjiconstantinou, E. (1995), "An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts", European Journal of Operational Research 83, pp. 21-38

[15] Vasko, F.J. (1989), "A computational improvement to Wang's two-dimensional cutting stock algorithm", Computers ind. Engng, Vol.16, No.1, pp. 109-115.

[16] Andrade, R., Birgin E. G., Morabito R. (2013), "Two-stage two-dimensional guillotine cutting stock problems with usable leftover"

[17] Riehme, J. et al. (1996), "The solution of two-stage guillotine cutting stock problems having extremely varying order demands", European Journal of Operational Research 91, pp. 543- 552

[18] Hashemi, N. F. et al. "A heuristic approach for rectangular Guillotine Cutting-stock Problem", Iran University of Science and Technology.

[19] Morabito, R. and Arenales, M. N. (1994), "An AND/OR-graph Approach to the Container Loading Problem", European Journal of Operational Research. Vol. 1, No. 1, pp. 59-73

[20] Beasley, J. E., (1985), "An exact two-dimensional non-guillotine cutting tree search procedure", Operations Research 33 (1) pp. 49-64

[21] Cintra, C. F. et al. (2008), "Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation", European Journal of Operational Research. 191, pp. 61-85

[22] Alvarez-Valdes, R. et al. "A Tabu Search algorithm for two-dimensional non-guillotine cutting problems", University of Valencia and University of Castilla-La Mancha.

[23] Alvarez-Valdes, R. et al. "A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems", University of Valencia and University of Castilla-La Mancha.

[24] Wang, P.Y. (1983), "Two algorithms for constrained two-dimensional cutting stock problems", Operations Research 32, 573-586.

[25] Haessler, R. W. and Sweeney P. E. (1991), "Cutting stock problems and solution procedures". European Journal of Operational Research 54, 141-150.

[26] Goldberg, D. E., (1989), "Genetic Algorithms in Search, Optimization and Machine Learning", New York: Addison – Wesley

[27] Holland, J. H., (1992), "Genetic Algorithms", Scientific American Journal.

[28] Kalyanmoy Deb, "An Introduction To Genetic Algorithms", Sadhana, Vol. 24 Parts 4 And 5.

[29] Mitchell, M. (1999), "An Introduction To Genetic Algorithms", Cambridge, Massachusets. London, England. The MIT Press.

[30] Michalewicz, Z. 1994. Genetic Algorithms + Data Structures = Evolution Programs, Second, Extended Edtion, Springer-Verlag, p. 340

[31] Ono, T. “Optimizing Two-Dimensional Guillotine Cut by Genetic Algorithms”, Fukuoka Institute of Technology.

9. Anexo

A parte de este documento, se hace entrega de un anexo que contiene los siguientes elementos:

- Un documento adjunto en forma de anexo que contiene el código fuente de los dos *scripts* de C# implementados,
- El proyecto de Unity3D,
- La aplicación desarrollada en formato ejecutable, lista para ser utilizada.