Inteligencia Artificial

Informe Final: 2D-Strip Packing Problem (2DSPP)

Felipe Nicolás Vega Valencia

December 18, 2017

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Abstract

En este documento se realiza un estudio del 2D-Strip Packing Problem, el cual busca colocar objetos rectangulares sobre una superficie con ancho definido y largo infinito, buscando encontrar la mejor organización de estos objetos para minimizar el alto de la superficie utilizada. Esto se realiza con el objetivo de entender el problema y su desarrollo, para poder implementar un método para resolver el problema. Primero se brinda una descripción del problema, y un estudio de las principales técnicas utilizadas para resolver el problema, además de presentar dos modelos matemáticos para el problema. Luego, se presenta un método híbrido para resolver el problema, el cual utiliza un algoritmo genético en conjunto a una heurística para decidir la ubicación del objeto. Finalmente se presentan los experimentos realizados con este método y los resultados obtenidos en comparación a otros métodos de la literatura.

1 Introducción

El **2D-Strip Packing problem (2DSPP)** es un problema de optimización combinatoria perteneciente a la familia de los Packing Problems, donde se busca minimizar la altura de una superficie sobre la cual se debe acomodar un conjunto de objetos con dimensiones definidas. El siguiente documento presenta un estudio sobre los principales avances sobre los métodos para resolver el problema, presentando un modelo matemático para este. Además, se muestran los resultados obtenidos al implementar un método híbrido para resolver el problema, incluyendo las representaciones utilizadas y un pseudo-código explicativo del algoritmo desarrollado.

En la sección 2 se brinda una descripción más a fondo del problema, explicando donde se encuentra este problema dentro de los Packing Problems, los elementos que considera el problema a estudiar y otras posibles variaciones al problema. En la sección 3 se analizan métodos de resolución utilizados para resolver este problema, principalmente enfocado en heurísticas

de colocación (Placement Heuristics) que brindan reglas para posicionar los objetos sobre la superficie, y los métodos híbridos, los cuales combinan las heurísticas de colocación con méta-heurísticas de búsqueda y exploración. En la sección 4 se presentan dos modelos matemáticos, uno basado en niveles (Lodi (2002) [12]) y otro basado en coordenadas cartesianas (Riff (2009) [16]). En la sección 5 se detalla la representación utilizada para el 2DSPP, presentando en la sección 6 una descripción del algoritmo híbrido que utiliza la representación presentada en la sección 5. En la sección 7 se detallan los experimentos sobre el algoritmo implementado para mostrar los resultados obtenidos en la sección 8. Finalmente en la sección 9 se presentan las conclusiones del estudio realizado, en base al estado del arte realizado en la sección 3 y a los resultados obtenidos en la sección 8.

2 Definición del Problema

Los Packing Problems son problemas clásicos en el mundo de la optimización combinatoria, los cuales tienen dos sub-divisiones, dependiendo del objetivo que se desea alcanzar. El primer objetivo es encontrar el ordenamiento óptimo de un grupo de objetos para minimizar los contenedores a utilizar para empacar dichos objetos, este problema de conoce como Bin Packing Problem. El segundo objetivo es organizar un grupo de objetos de la manera más densa posible en un solo contenedor de dimensiones infinitas, lo que se conoce con el nombre de Strip Packing Problem.

Además, se pueden considerar distintos tipos de problemas dependiendo de lo que se desee minimizar. 1D Packing Problem desea acomodar objetos en el menor número de contenedores, 2D Packing Problem busca organizar objetos sobre una superficie minimizando el espacio perdido y 3D Packing Problem busca minimizar el volumen utilizado al distribuir objetos en tres dimiensiones. Existen otras variaciones del problema donde los objetos a organizar no necesariamente son rectangulares [1]. Otra variación son los denominados Guillotine 2DSPP, donde se deben realizar cortes paralelos a los ejes sobre la superficie utilizada [14].

Este documento se centra en el estudio del denominado **2D Strip Packing Problem (2DSPP)**, el cual busca encontrar una forma de organizar distintos objetos en una superficie con una de sus dos dimensiones de largo definido y la otra de largo infinito, buscando minimizar el espacio desperdiciado, lo que se logra minimizando el largo de la dimensión infinita. Esta versión de Packing Problem tiene aplicaciones industriales, por ejemplo en la industria del vidrio, donde se busca cortar moldes de este material minimizando el material desperdiciado.

Este problema en particular considera los siguientes elementos (Baker (1980) [17]):

- Una cantidad determinada de objetos rectangulares (objetos, items), cada uno con un ancho y un alto definido.
- Una única superficie con ancho fijo y largo indefinido (o viceversa)
- Los objetos se pueden rotar en 90 grados.
- Los objetos no se pueden sobreponer.

3 Estado del Arte

El 2DSPP es un problema que ha sido abordado de muchas maneras. Se comenzó con la utilización de métodos exactos (Branch and Bound, Árboles de Búsqueda, etc), los cuales funcionan de buena manera pero para instancias pequeñas del problema. También se ha intentado abordar mediante la utilización de meta-heurísticas (Algoritmos Genéticos, Simulated Annealing, etc), donde se pueden resolver instancias un poco mayores, a cambio de no siempre obtener la

solución óptima. Sin embargo, estos métodos necesitan de mucho tiempo de procesamiento para instancias grandes.

Este estudio se centra en los métodos utilizados para la resolución de instancias grandes, los cuales generalmente utilizan heurísticas de colocación, las cuales definen reglas para posicionar los objetos en la superficie. También se realiza un estudio sobre métodos híbridos, los cuales mezclan las heurísticas de colocación con meta-heurísticas de búsqueda y exploración.

3.1 Heurísticas de Colocación

Dada la dificultad del 2DSPP (NP-HARD [6]), generalmente se utilizan heurísticas para poder resolver instancias grandes. Una de ellas son las heurísticas de colocación (heuristic placement algorithms), donde se determina la posición de los objetos en la superficie. Existen de muchos tipos, las cuales difieren dependiendo del criterio con el cual colocan y organizan los objetos.

3.1.1 Bottom-Left (BL), Bottom-Left-Fill (BLF), Bottom-Left-Decreasing (BLD*)

Bottom-Left (BL) (Jakobs (1996)[9]) es una heurística basada colocar los objetos lo más cerca que se pueda de la esquina inferior izquierda. Al momento de colocar un objeto sobre la superficie se parte desde la esquina superior izquierda, desde ahí se debe descender lo más posible hasta alcanzar un tope. Una vez alcanzado el tope se debe correr el objeto hacia la izquierda lo que más se pueda. Este proceso de debe repetir hasta que el objeto a posicionar no se pueda mover más hacia la izquierda ni hacia abajo. La figura 1 presenta como BL posiciona un objeto en la superficie.

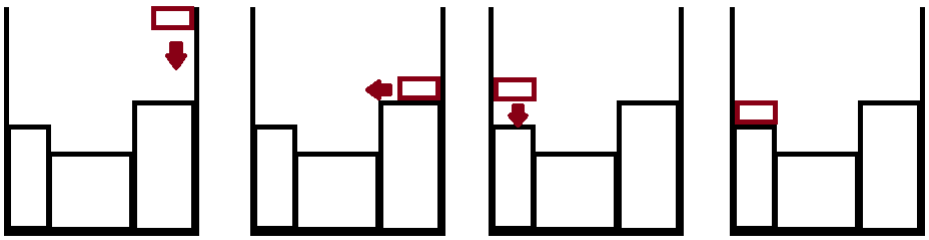


Figure 1: Ejemplo gráfico para las reglas de posicionamiento de la heurística Bottom-Left.

Esta heurística se utiliza con un orden dado para los objetos a empacar, el cual puede ser por altura, ancho, área, perímetro, etc. Donde por lo general el ordenamiento que mejores resultados trae es el ordenamiento descendiente por altura. Un ejemplo de esto es el estudio de Hopper y Turton (2001) [7], donde realizan pruebas utilizando BL con ordenamiento por altura, por ancho y sin ordenamiento, donde en 5 de 7 instancias es el ordenamiento por altura el que mejores resultados trae.

Esta técnica tiene una complejidad $O(n^2)$ por lo cual se utiliza bastante en conjunto con otros métodos de resolución, por ejemplo, actuando como método de decodificación en algoritmos genéticos (Hopper y Turton (2001)[7]). Lo malo de esta heurística es que tiende a crear espacios libres cuando se utilizan objetos muy grandes, pues estos tienden a cortar el paso a item más pequeños.

Bottom-Left-Fill (BLF) (Chazelle (1983) [4]), es una heurística donde se colocan los objetos en el lugar más cercano al fondo posible y luego se justifica hacia la izquierda. Esta técnica genera mejores resultados que BL dado que produce soluciones más densas (se desperdicia menos espacio). La figura 2 presenta un ejemplo de posicionamiento utilizando BLF.

La principal desventaja es que necesita de más procesamiento computacional que BL (complejidad $O(n^3)$). Hopper y Turton [7], al comparar esta técnica con BL obtienen que BLF

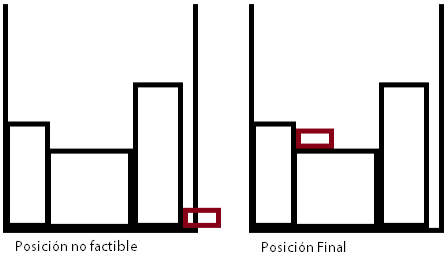


Figure 2: Ejemplo gráfico para las reglas de posicionamiento de la heurística Bottom-Left-Fill.

obtiene mejores soluciones por un 10% a 30%, pero que para instancias más grandes el tiempo de procesamiento es mucho mayor.

Al igual que BL, utiliza un orden predeterminado para organizar los objetos, donde nuevamente el orden que mejores resultados consigue es el por altura descendiente. Existen implementaciones donde se calcula el BLF para orden descendiente por altura, ancho, perímetro y área y se retorna el que obtenga mejor resultado.

Otra heurística de colocación es **Bottom-Left-Decreasing* (BLD*)**, la cual actua de la misma manera que BLF, pero cambia el orden de los objetos bajo una cierta probabilidad "p". Fue propuesto por Lesh (2005) [10], donde comparan su heurística contra BLF y contra el trabajo de Iori (2003) [8], el cual utiliza un método híbrido entre Tabu Search y Algoritmos Genéticos. Los resultados de las pruebas realizadas muestran que BLD* obtiene mejores resultados que los obtenidos por Iori en las 14 instancias obtenidas desde la literatura.

Además, en conjunto con esta heurística, desarrollan un sistema interactivo utilizando Java, con lo cual permiten que un usuario guíe la búsqueda realizada por BLD*, donde este puede reordenar piezas mientras el algoritmo se ejecuta, también pueden concentrar el trabajo en ciertas secciones de la superficie, donde un humano se puede dar cuenta que se puede lograr un mejor ordenamiento. Los resultados obtenidos al probar este sistema muestran que se puede reducir la distancia a la solución óptima en un 1% en comparación a la solución encontrada por BLD* sin interacción humana, además reducen en el tiempo de procesamiento drásticamente, aunque se debe considerar que el algoritmo no puede funcionar tan rápido pues el usuario no alcanzaría a interactuar con las soluciones encontradas.

3.1.2 Best-Fit (BF)

Best-fit (BF) es otra heurística, la cual se basa en elegir dinámicamente el mejor objeto a colocar en cada paso. Esta heurística es propuesta por Burke (2003) [2] y su idea es buscar cual es el espacio a menor altura disponible, luego recorrer toda la lista de objetos para ver cual es el que mejor llena el espacio disponible y colocarlo.

Una vez colocados todos los objetos se buscan los que están a mayor altura y se verifica que sucede si se quita este de la superficie, se rotara en 90 grados y se colocara nuevamente. De esta manera se puede mejorar aún más la solución encontrada.

Al tener que recorrer todo el ancho de la superficie para encontrar el espacio disponible a menor altura, además de toda la lista de objetos para encontrar el que mejor se adecue a dicho espacio, la complejidad temporal de este algoritmo es bastante. Sin embargo, el método utiliza un arreglo de tamaño W (ancho de la superficie), donde cada casilla guarda que altura posee dicha porción del ancho, de esta manera encontrar el espacio disponible consiste solo en encontrar el menor valor dentro del arreglo, lo que reduce el tiempo de procesamiento.

Al comparar FH con BL y BLF, utilizando las instancias propuestas por Hopper y Turton (2001) [7], se obtiene que FH funciona mejor que BLF, aún cuando se permite un orden establecido para BLF (descendiente por ancho y por largo). También comparan FH con los métodos

híbridos propuestos por Hopper y Turton, los cuales son Algoritmos Genéticos o Simulated Annealing en conjunto con BLF para la decodificación. Los resultados que obtienen es que BF iguala o supera a los métodos híbridos, aunque BF se ve sobrepasado para instancias pequeñas (menor a 50 objetos).

3.1.3 Fast Heuristic Algorithm (FH)

Por último, la siguiente heurística es la utilizada por el denominado **Fast Heuristic Algorithm (FH)**, propuesto por Leung (2009) [11], la cual se basa en la forma de construcción de las murallas de ladrillos. Esta técnica comienza colocando un objeto, desde donde determina una línea de referencia. Luego, comienza apilando objetos sobre el inicial, actualizando la línea de referencia hasta alcanzar un cierto valor LB definido por el problema. Finalmente, el algoritmo determina la posición "p" más cercana al borde inferior bajo la línea de referencia y comienza a colocar los objetos dependiendo de su "*fitness value*", el cual se determina dependiendo de la cantidad de vertices que calcen justo con las paredes del espacio determinado por p.

Con esto construye FH, el cual comienza ordenando los objetos por perímetro descendente, calcula la altura necesaria utilizando la heurística para luego realizar swaps entre objetos, partiendo con el primero con todos los sucesores, luego el segundo y su sucesores, así hasta llegar al objeto $n-1$. Si en algún momento un swap produce una altura menor, se guarda este valor y se actualiza la secuencia de objetos desde la cual se realizaran los intercambios de posición.

Este algoritmo tiene una gran performance con instancias grandes, Leung y Zhang comparan su algoritmo con un método híbrido que utiliza BF con Simulated Annealing [3], donde obtienen mejores resultados en un menor tiempo computacional.

3.2 Métodos Híbridos

Estos métodos se caracterizan por utilizar heurísticas de colocación en conjunto con metaheurísticas de búsqueda y exploración, tales como Algoritmos Genéticos, Simulated Annealing, Tabu Search, entre otras.

3.2.1 Algoritmo genético (GA)/Simulated Annealing (SA) + BLF

Hopper y Turton (2001) [7] realizan un estudio donde comparan BL con BLF, además de proponer el uso de estas heurísticas en conjunto con **Algoritmos Genéticos (GA), Simulated Annealing (SA)** y Naïve Search, realizando comparaciones entre estas técnicas bajo el mismo set de instancias. En dichos casos utilizan BL y BLF como método de decodificación, transformando el problema a uno donde deben encontrar la secuencia de objetos de minimizando la altura, la ventaja que trae esto es que no se deben pensar en movimientos específicos para el problema, dado que se pueden usar los movimientos presentes en la literatura para problemas de ordenamiento.

Para el caso de la utilización de GA, la mutación utilizada es la rotación de algún objeto con cierta probabilidad, el método de cruzamiento mezcla las secuencias de los dos padres (se verifica que la secuencia sea correcta).

Cuando utilizan SA generan el vecindario realizando intercambios aleatorios entre objetos, además de rotar un objeto con cierta probabilidad. Utilizan una función de temperatura geométrica, que disminuye un 10% a la vez.

Al comparar estos métodos se obtiene que la versión GA + BLF funciona mejor que SA + BLF al principio de la búsqueda, pero al final de esta SA + BLF sobrepasa a GA + BLF (1% a 8% de mejora). Esto se explica porque SA se centra en una solución a la vez y GA utiliza un conjunto de soluciones. Además, ambos métodos híbridos obtienen mejores resultados que BL y BLF.

3.2.2 Best Fit + Simulated Annealing (BF + SA)

Best Fit + Simulated Annealing (BF + SA) es un método híbrido desarrollado por Burke (2009) [3] para mejorar la heurística BF propuesta por él mismo [2]. La técnica propuesta es un método de dos pasos; El primero utiliza Best Fit para colocar una primera porción de objetos. El segundo paso utiliza SA para encontrar el mejor orden usando SA.

La razón de la separación de estos dos pasos es evitar el desorden que produce BF en los últimos pasos de ejecución pasando el trabajo a SA, lo que a su vez permite utilizar este tipo de técnicas para instancias más grandes.

Burke prueba este método frente a BF y los métodos híbridos propuestos por Hopper y Turton (2001)[7] utilizando 47 instancias del problema presentes en la literatura, donde su método consigue mejoras en el 87% de estas. Los resultados obtenidos para instancias en la literatura con esta técnica se convirtieron en los mejores hasta ese momento (2009).

3.2.3 Heurística tipo Greedy + Tabu Search

Lijun Wei (2011)[18] proponen un algoritmo híbrido basado en una heurística greedy junto con Tabu Search para resolver un Packing Problem con una altura determinada , además de utilizarlo para resolver el 2DSPP utilizando búsqueda binaria. Su heurística se basa en la utilización de Skylines, las cuales son "líneas" ubicadas en la parte superior de los rectángulos a colocar, donde los siguientes items a empacar solo se pueden colocar sobre estas Skylines. Junto con esta heurística utilizan Tabu Search para recorrer el espacio de búsqueda, su algoritmo genera un conjunto de posibles secuencias de objetos realizando swaps aleatorios entre objetos, para luego encontrar la mejor de esas secuencias utilizando su heurística.

El método anterior es utilizado para resolver instancias donde la superficie a utilizar tiene dimensiones definidas. Para resolver el 2DSPP realizan búsqueda binaria, utilizando una altura mínima, dada por la instancia del problema de resolver, y una altura máxima esperada, calculada mediante métodos estadísticos.

Al probar este método frente a las mejores técnicas desarrolladas hasta ese momento (BF, BF+SA, FH, etc), bajo las mismas instancias, obtienen que su método de resolución funciona de mejor manera, solo siendo superado en una de ocho instancias utilizadas.

3.3 Comentarios

Dentro de los trabajos actuales en este problema se encuentra el trabajo de Babaoglu (2017) [19], el cual utiliza el algoritmo FOA (Fruit Fly Optimization Algorithm, Pan (2012) [15]), que emula el comportamiento de la "mosca de la fruta". Baboaglu combina este método en conjunto a BLF para colocar los objetos. Sin embargo, los resultados obtenidos al comparar este método contra los métodos híbridos de Hopper y Tupper (2001) muestran que este enfoque para el problema funciona de peor manera, quedando por debajo de SA+BLF y GA+BLF.

Este último trabajo, sumado a los trabajos analizados anteriormente, muestra que los mejores avances en este problema se logran al utilizar métodos híbridos. El uso de heurísticas de colocación en conjunto a las técnicas de búsqueda local más clásicas permite encontrar soluciones buenas para instancias de tamaño considerable (200+ objetos), en donde las mejoras se producen al mejorar y/o desarrollar heurísticas de colocación.

4 Modelo Matemático

Para presentar los modelos matemáticos primero es necesario definir los elementos del problema de manera más formal.

- Conjunto de n objetos, cada uno con un ancho w_j y un alto h_j, $j \in (1, 2, ..., n)$.

- Una superficie o tira, con un ancho definido W y un alto $H > 0$ infinito en un principio.
- Cada uno de los n objetos cumple con $w_j \leq W \wedge h_j \leq H, j \in (1, 2, ..., n)$.

4.1 Modelo por niveles

El siguiente modelo es presentado por Lodi (2002) [12], el cual se basa en una representación por niveles del problema. El primer nivel es la parte más baja de la superficie ($H = 0$), y los siguientes niveles están determinados por el objeto más alto que se encuentre sobre el nivel anterior.

Primero, se deben hacer tres suposiciones sobre estos niveles.

1. En cada nivel, el objeto que está más a la izquierda es el más grande.
2. El nivel inferior es el más grande.
3. Los objetos están ordenados de manera descendente por h_j.

4.1.1 Variables

La definición de las variables considera las tres suposiciones anteriores.

$$y_i = \begin{cases} 1 & \text{si el objeto i inicializa el nivel i.} \\ 0 & \text{si no.} \end{cases} \tag{1}$$

$$x_{ij} = \begin{cases} 1 & \text{si el item j está en el nivel i.} \\ 0 & \text{si no.} \end{cases} \tag{2}$$

4.1.2 Función Objetivo

Dada la definición de las variables, la función objetivo debe minimizar la suma de la altura de todos los niveles, pues eso lleva a minimizar la altura total necesaria para la superficie.

$$min \sum_{i=1}^{n} h_i y_i \tag{3}$$

4.1.3 Restricciones

$$\sum_{i=1}^{j-1} x_{ij} + y_i = 1, j \in (1, 2, ..., n) \tag{4}$$

Esta restricción obliga a cada objeto inicialice el nivel que le corresponde, o que pertenezca a uno de los niveles inferiores.

$$\sum_{j=i+1}^{n} w_j x_{ij} \leq (W - w_i)\, y_i, i \in (1, 2, ..., n-1) \tag{5}$$

La segunda restricción obliga a que se mantenga la restricción de ancho por nivel, donde la suma de los anchos de todos los objetos del nivel debe ser menor al ancho de la superficie menos el ancho utilizado por el objeto que inicializa el nivel. Hay que notar que en la sumatoria aparece el subíndice $j = i + 1$, esto se debe a las suposiciones 1 y 3, que producen que si $y_i = 1$ los objetos pertenecientes al nivel i solo puedan ser los objetos $i + 1, i + 2$, etc.

Hay que destacar que esta restricción hace que no se sobrepongan los objetos en el nivel, pues la única manera de que se de esta situación es que los anchos de los objetos del nivel sobrepase el ancho del nivel.

$$y_i, x_{ij} \in \{0,1\} \forall i, j \tag{6}$$

La última restricción representa la naturaleza binaria de las variables y_i y x_{ij}.

4.2 Modelo por coordenadas cartesianas

El siguiente modelo fue presentado por Riff (2009) [16], el cual considera las coordenadas cartesianas de la esquina inferior izquierda de cada rectángulo.

4.2.1 Variables

Las variables a utilizar por el modelo son las siguientes:

$$H = \text{Alto de la superficie} \tag{7}$$

Esta variable determina el alto final que tendrá la superficie al colocar todos los objetos.

$$\begin{aligned} x_j &= \text{coordenada en el eje x de la esquina inferior izquierda del objeto j} \\ y_j &= \text{coordenada en el eje x de la esquina inferior izquierda del objeto j} \end{aligned} \tag{8}$$

Estas dos variables definen la posición sobre la superficie para cada uno de los n objetos.

4.2.2 Función Objetivo

Dada la formulación de las variables la función objetivo solo debe buscar minimizar la cantidad de material utilizado.

$$min \quad H \tag{9}$$

4.2.3 Restricciones

$$x_j + w_j \leq W, \forall j \in (1, 2, ..., n) \tag{10}$$

$$y_j + h_j \leq H, \forall j \in (1, 2, ..., n) \tag{11}$$

Las restricciones (10) y (11) restringen a los objetos a mantenerse dentro de los límites de la superficie. Cabe destacar que la restricción (11) es la que termina definiendo el alto de la superficie.

$$x_j + w_j \leq x_i \vee x_i + w_i \leq x_j \vee y_j + h_j \leq y_i \vee y_i + h_i \leq y_j \forall i, j \in (1, 2, ..., n), i \neq j \tag{12}$$

La restricción (12) busca que los objetos no se sobrepongan entre ellos, cabe destacar que son cuatro restricciones unidas por "o lógicos". Las dos primeras buscan que no se sobrepongan horizontalmente cada par de objetos, las últimas dos buscan que cada par de objetos no se sobrepongan verticalmente.

5 Representación

Para abordar el problema se utilizará BLF como heurística para la colocación de objetos. De esta manera, el objetivo es encontrar el mejor orden para los objetos, de manera que al ser empacados por BLF se minimice la altura utilizada. A continuación se presenta la representación utilizada para cada solución, la cual está compuesta por una representación para cada objeto, una para el orden en que cada objeto es colocado en la superficie y la última para la ubicación que utiliza cada objeto sobre la superficie.

5.0.1 Representación de los objetos

Para representar los objetos se utiliza una clase Objeto, donde los atributos de esta son los siguientes.

- **Id** del objeto.
- **Alto** del objeto.
- **Ancho** del objeto.
- Coordenada **x** de la esquina inferior izquierda del objeto.
- Coordenada **y** de la esquina inferior izquierda del objeto.
- **Rotación**.

Donde el id es utilizado para reconocer el objeto aún cuando se cambie el orden de empaque. Los siguientes cuatro atributos son utilizados por BLF para empacar los objetos. El atributo rotación indica si el objeto se encuentra rotado con respecto a la declaración inicial de estos.

5.0.2 Representación de orden

Para representar el orden se utiliza un vector de tamaño "n", donde n es igual a la cantidad de objetos a empacar. Cada posición "i" del arreglo contiene el Objeto que ingresa en paso i de BLF. Dado que se trabaja con un algoritmo genético, cada individuo mantiene su propio vector de orden.

La figura 3 muestra un ejemplo del vector utilizado para representar soluciones, donde el objeto 1 (O1) es el primero en ser ingresado a la superficie y el objeto 7 (O7) es el último en ser empacado.

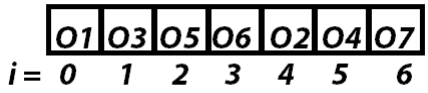


Figure 3: Vector utilizado para representar el orden en que son empacados los objetos.

5.0.3 Representación en la superficie

Finalmente, para ver la ubicación de cada objeto en la superficie se utilizan los atributos **x** e **y** de cada objeto, en conjunto con las dimensiones de este para que no se sobrepongan unos con otros.

La figura 4 muestra como se guardan las coordenadas de tres objetos, en conjunto con un ejemplo gráfico del empaque realizado por BLF.

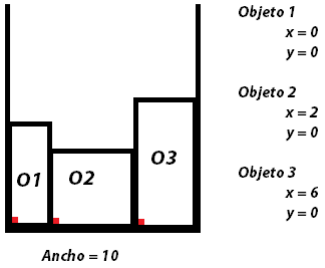


Figure 4: Ejemplo de colocación de tres objetos sobre la superficie, incluyendo la representacion de las coordenadas de estos.

6 Descripción del algoritmo

El algoritmo implementado para resolver el 2DSPP es un algoritmo genético, el cual utiliza la heurística BLF para colocar los objetos sobre la superficie y calcular la altura necesaria dependiendo del orden de los objeto ingresados.

Algoritmo 1 Algoritmo genético + BLF

Entrada: TamañoPoblación, CantidadIteraciones.
Salida: Individuo con mejor calidad.

```
1:  Altura Actual
2:  generarPoblacion()
3:  evaluarPoblacionActual()
4:  for i = 0; i < CantidadIteraciones; i++ do
5:    seleccionarPadres()
6:    for j = 0 ; j < TamanoPoblacion ; j++ do
7:      cruzar()
8:    end for
9:    for j = 0 ; j < TamanoPoblacion ; j++ do
10:     mutar()
11:   end for
12:   evaluarProximaPoblacion()
13:   if Altura Actual > Altura Mejor Individuo then
14:     Altura Actual = Altura Mejor Individuo
15:   end if
16: end for
17: Return Mejor Individuo
```

El algoritmo 1 muestra el pseudo-código de la solución. Para generar la población inicial se generan ordenes aleatorios de los objetos de la instancia a resolver, por lo que toda la población inicial contendrá soluciones factibles.

En las líneas 3 y 12 se realizar el calculo de la altura para cada individuo de la población. Para esto se utiliza la heurística de colocación BLF implementada por Charles Dubout (2012)[5], la cual se basa en la implementación presentada por Chazelle (1983) [4]. En la implementación se mantiene un conjunto de espacios vacíos en la superficie, los cuales internamente están ordenados sobre un árbol, por lo cual al recorrer el conjunto se recorren primero los espacios vacíos que se encuentran más cercanos al comienzo de la superficie. De esta manera, por cada objeto a insertar se recorre el conjuntos de espacios vacíos hasta encontrar uno donde pueda entrar, se registran las coordenadas de los objetos y se actualiza el conjunto de espacios. Al mismo tiempo,

se va guardando la altura máxima al ingresar cada objeto.

Como operador de selección (línea 5) se utiliza **2-Torneo**. Con este operador elegimos el individuo con menor altura entre dos, favoreciendo la intensificación del algoritmo, pero sin aumentar demasiado la presión de selección.

Para el cruzamiento se utiliza el operador **Order-Based Crossover**, el cual genera dos puntos de cortes aleatorios en los padres (los mismos para ambos) y se traspasan los objetos a los hijos en la misma posición en la que aparecían en los padres. Luego, se recorre en orden el padre del que no traspasó sus objetos y se copian al hijo todos los objetos que no estén ingresados actualmente. La figura 5 muestra un ejemplo de cruzamiento para generar un hijo, para el segundo se invierten los roles de los padres.

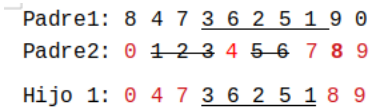


Figure 5: Ejemplo Order-Based Crossover

Este operador se seleccionó porque permite generar soluciones factibles al realizar el cruzamiento. Además, no requiere utilizar operaciones adicionales en los vectores para su funcionamiento, por lo que su procesamiento es más rápido que otros operadores de cruzamiento.

Cabe destacar que este operador se aplica bajo una cierta probabilidad de cruzamiento, generando número aleatorios bajo una distribución uniforme. En caso de que se sobrepase la probabilidad seleccionada, se agregan los padres tal como están a la próxima generación.

Para las mutaciones se recorre el vector de orden del individuo, rotando cada objeto bajo una cierta probabilidad de mutación. De esta manera se pueden encontrar mejores soluciones al ingresar un objeto rotado a la superficie, aumentando la diversificación del algoritmo.

Para finalizar, se define un criterio de término para cuando el algoritmo converja a una solución, el cual interrumpe la ejecución del algoritmo cuando pasan 3000 iteraciones sin que se mejore la calidad de las soluciones.

7 Experimentos

Se realizaron dos tipos de experimentos con el algoritmo desarrollado. Una primera parte fue la sintonización de los parámetros del algoritmo, donde se realizaron pruebas para determinar el tamaño de la población, la probabilidad de mutación y la cantidad de iteraciones. La segunda parte de los experimentos realizados fueron para determinar el desempeño del algoritmo implementado, para esto se realizó una comparación con el algoritmo genético + BLF propuesto por Hopper y Turton (2001) [7] y con la técnica FOA + BLF propuesta por Babaoglu (2017) [19].

7.1 Sintonización de parámetros

Los primeros experimentos para la sintonización de los parámetros fueron para determinar el tamaño de población a utilizar. Para esto se realizaron pruebas con una población de tamaño 25, 50 y 100, utilizando las instancias *BENG01*, *NGCUT01* y *CGCUT01*[13]. Las instancias anteriores se seleccionaron debido a que poseen óptimos conocidos y porque son instancias con distintos números de objetos a colocar, lo que permite obtener mejores resultados que al probar con instancias con el mismo número de objetos.

El método utilizado para estos experimentos, dada la naturaleza estocástica de la técnica utilizada, fue correr 100 veces el algoritmo por cada tamaño de población y calcular el promedio de la solución entregada. Esto se realizó con las tres instancias seleccionadas.

Una vez determinado el tamaño de población a utilizar se realizaron pruebas para determinar la probabilidad de mutación del algoritmo. Para esto se realizó el mismo método que el experimento anterior, probando con probabilidades de 0.2, 0.3 y 0.4. En este caso se seleccionó la instancia *GCUT04*[13], la cual posee 50 objetos con dimensiones bastante grandes, lo cual hace que sean más apreciables los efectos que se producen al rotar un rectángulo.

Finalmente se realizaron pruebas para determinar la cantidad de iteraciones a utilizar, se experimentó con 10000 y 5000 iteraciones. Para esto se utilizó la instancia *BENG10*[13], debido a que es una instancia grande para este problema (200 objetos). En este experimento se registró la mejor solución obtenida en cada iteración con el objetivo de ver el comportamiento del algoritmo a medida que se avanza con el procedimiento.

7.2 Desempeño del algoritmo

Para este tipo de experimentos se utilizaron las *"instancias C"* propuestas por Hopper y Turton (2001)[7], las cuales son 7 instancias con 3 problemas cada una. Esto debido a que dichas instancia son de las más utilizadas en la literatura desde su aparición.

Estos experimentos consistieron en correr el algoritmo una cierta cantidad de veces por instancia; 100 veces para las instancias *C1* a *C3*, 25 veces para la instancia *C4*, 10 veces para las instancias *C5*, 25 veces para la instancia *C6* y 5 veces para la instancia*C7*. Para las instancia *C6* se utilizaron solo 1000 iteraciones debido al tiempo de ejecución que tiene el algoritmo, para *C7* se decidió bajar el número de repeticiones pero manteniendo las 10000 iteraciones. Este procedimiento se hizo para cada uno de los tres problemas por instancia, calculando una altura promedio por problema y luego una altura promedio entre problemas de la misma instancia.

También, se realizaron pruebas utilizando la instancia *BENG10*[13] debido a la cantidad de objetos que posee (200), lo que puede entregar luces del comportamiento del algoritmo para instancias grandes.

Por último, se realizó una prueba para ver como se comporta el tiempo de ejecución con respecto al número de objetos a empacar. Se utilizaron problemas de las siete instancias *C*, registrando el tiempo de ejecución para cada una. La razón para elegir estas instancias es la diferencia entre objetos entre cada una (desde 20 hasta 197 objetos), lo que entrega una buena muestra del comportamiento temporal del algoritmo.

8 Resultados

A continuación se presentan los resultados para los distintos experimentos detallados en la sección 7. Para la realización de estos se utilizó un notebook con procesador Intel Core i7-4510U de cuatro núcleos con 2.00 GHz y 8 gigabytes de memoria RAM.

Debido a la naturaleza estocástica del algoritmo, se presentan el promedio de los resultados obtenidos para las distintas instancias. Como la altura obtenida debe ser un número entero, para cada experimento se entregan los resultados redondeados.

8.1 Sintonización de parámetros

En la tabla 1 se puede observar los resultados obtenidos en las pruebas para determinar el tamaño de población a utilizar. Se puede observar que para las instancias utilizadas los mejores resultados con respecto al óptimo son para los casos en que se usa una población de 100 individuos. Durante la ejecución se pudo notar que a menor población la solución converge más rápido. Sin embargo se obtienen peores resultados, por lo que la diversificación lograda al aumentar el tamaño de población ayuda a explorar de mejor manera el espacio de búsqueda y encontrar mejores soluciones.

Para el caso de la probabilidad de mutación, el experimento realizado muestra que es mejor utilizar una probabilidad de mutación igual a 0.3. Debido a que la orientación del rectángulo

Instancia	Óptimo	Población de 25	Población de 50	Población de 100
BENG01	30	31	31	**30**
NGCUT01	20	**20**	**20**	**20**
CGCUT01	23	**23**	**23**	**23**

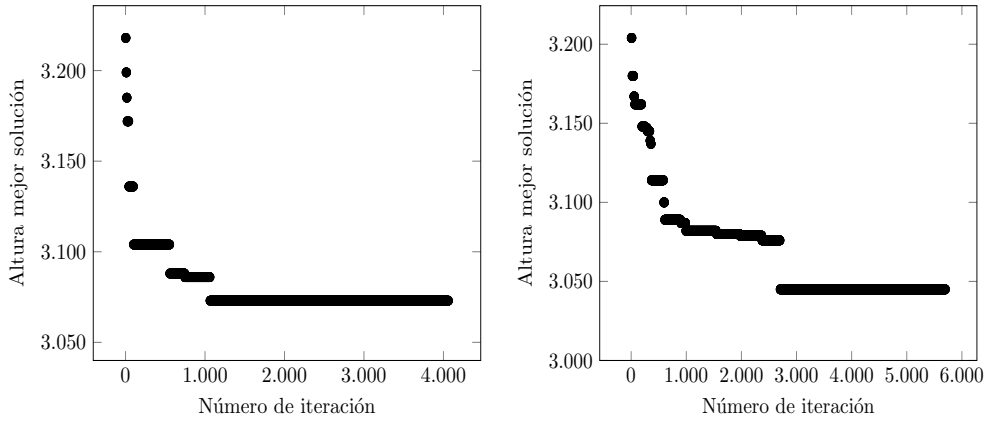
Table 1: Resultados obtenidos para la sintonización del tamaño de población. Los números en negrita son los mejores resultados obtenidos por instancia.

es determinante para este tipo de instancia, donde las dimensiones de los objetos están en el mismo orden que el ancho de la superficie, se toma la decisión de mantener esta probabilidad para todos los tipos de instancias. Los resultados se pueden observar en la tabla 2.

Instancia	Repeticiones	Probabilidad 0.2	Probabilidad 0.3	Probabilidad 0.4
GCUT04	10	3075	**3068**	3078

Table 2: Resultados obtenidos para la sintonización de la probabilidad de mutación. El resultado en negrita corresponde a la mejor solución encontrada para las tres diferentes probabilidades.

Finalmente, para la cantidad de iteraciones se corre el algoritmo con 5000 y 10000 iteraciones, utilizando un criterio de término adicional el cual se activa cuando pasan 3000 iteraciones sin mejorar la calidad de la solución. En las figuras 6.a y 6.b se puede observar gráficos que detallan como va mejorando la calidad de la solución conforme pasan las iteraciones del algoritmo. Si bien, para el caso de 5000 iteraciones el algoritmo converge a las 1000 iteraciones aproximadamente, se puede observar en el gráfico de 10000 iteraciones que la solución puede seguir mejorando pasado dicho número. Por lo anterior, se decide utilizar 10000 iteraciones, manteniendo el criterio de término en 3000 iteraciones ante posibles casos en que aún se pueda seguir mejorando la solución actual.



(a) 5000 iteraciones.

(b) 10000 iteraciones.

Figure 6: Gráfico de la altura del mejor individuo en función del número de iteraciones para ejecuciones con 5000 y 10000 iteraciones.

8.2 Desempeño del algoritmo

Los resultados obtenidos para la comparación entre técnicas se presentan en la tabla 3. El algoritmo desarrollado obtiene mejores resultados que las técnicas de Hopper y Babaoglu en las primeras tres instancias, mostrando la efectividad del método para instancias con menos de 30

objetos. Para las instancias *C4* y *C5* se produce un empate entre los dos algoritmos genéticos y en las instancias *C6* y *C7* la técnica que obtiene mejores resultados en el algoritmo genético de Hopper.

Instancia	Cantidad de Objetos	Óptimo	GA + BLF*	GA + BLF	FOA + BFL
C1	16/17	20	**1**	4	2
C2	25	15	**1**	7	7
C3	28/29	30	**2**	5	6
C4	49	60	**3**	**3**	7
C5	73	90	**4**	**4**	7
C6	97	120	6	**4**	7
C7	196/197	240	10	**5**	6

Table 3: Resultados obtenidos para la distancia con el óptimo para distintas técnicas. GA + BLF* corresponde a la técnica implementada, GA + BLF corresponde a la implementación de Hopper [7] y FOA + BLF corresponde a la técnica de Babaoglu [19].

Considerando los resultados obtenidos para la instancia *C7*, en donde la técnica desarrollada se posiciona en último lugar, se realizaron pruebas con la instancia BENG10, debido a que posee un número similar de objetos y se conoce su óptimo. Se realizaron 5000 iteraciones, repitiendo 10 veces este procedimiento, llegando a un promedio de altura de 160, lo cual se separa en cuatro unidades del óptimo reportado para la instancia (altura de 156).

La diferencia de comportamiento del algoritmo para las instancias *C7* y *BENG10* es considerable. Esto se debe principalmente al ancho de la superficie de cada instancia, donde se tiene que la instancia *C7* tiene cuatro veces el ancho de la instancia *BENG10*. Esto produce que a una misma altura se tengan múltiples opciones para colocar un objeto, lo que al final produce un estancamiento de la calidad de las soluciones encontradas en cada iteración independiente del orden de los objetos, y provoca que el algoritmo termine sin poder seguir explorando el espacio de búsqueda.

Finalmente, en el gráfico de la figura 7 se puede ver como se comporta temporalmente el algoritmo, donde para una instancia de 197 objetos se demora 1328 segundos (22 minutos aproximadamente), lo cual dependiendo del contexto puede ser bastante.

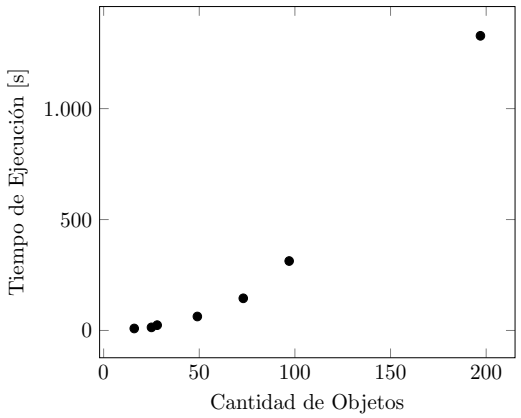


Figure 7: Gráfico que presenta como se comporta temporalmente el algoritmo en función de la cantidad de objetos.

El tiempo de procesamiento alto está directamente relacionado al hecho de que se trabaja con poblaciones de soluciones, por lo que BLF debe ser aplicado a cada uno de estos en cada

iteración, lo que debido a la complejidad computacional del algoritmo, toma mucho tiempo de procesamiento en comparación a otras técnicas que solo lo aplican con un vecindario reducido de una posible solución.

Esto, sumando a los resultas obtenidos en la comparación anterior indican que el método desarrollado funciona mejor para instancias pequeñas y medianas del problema, entregando la solución en un tiempo aceptable (2 minutos para una instancia de 73 objetos).

9 Conclusiones

Si bien todos los métodos estudiados en el estado del arte encuentran soluciones al problema, no todos funcionan de la misma manera para el mismo tipo de instancia. Comparando entre heurísticas de posicionamiento, se obtiene que BLF funciona de mejor manera que BL (entre 10% a 30%). BLD* al agregar una componente aleatoria para alterar el orden de la secuencia ingresada a BLF puede mejorar aún más los resultados. Con respecto a BF, al compararlo con BL y BLF, se obtiene que BF supera a los dos anteriores en 19 de 21 instancias presentes en el set de Hopper. La última heuristica de posicionamiento estudiada es la utilizada por el Fast Heuristic Algorithm (FH) propuesto por Leung (2009) [11], la cual se basa en el método de construcción de murallas de ladrillos. Este método obtiene grandes resultados al ser utilizado con instancias grandes del problema, superando al método híbrido (BF + SA) propuesto por Burke (2009) [3].

Las grandes mejoras a los métodos existentes a lo largo del tiempo se producen cuando se desarrollan métodos híbridos combinando métodos heurísticas de colocación con meta-heurísticas. Hopper y Turton (2001)[7] presentan métodos utilizando BL/BLF y las meta-heurísticas GA y SA. Las cuales al ser comparadas antes las técnicas incompletas predominantes en ese tiempo obtienen mejores resultados. La desventaja que tenían los métodos híbridos de Hopper y Turton es que requieren de mucho tiempo de procesamiento. Otros ejemplos de esta situación son los métodos desarrollados por Burke (2009)[3] y Lijun Wei (2011)[18], los cuales usan sus heurísticas Best Fit y el método de Skyline respectivamente, utilizando métodos de búsqueda local como Simulated Annealing y Tabu Search.

Dado lo anterior, es que se desarrolla un método híbrido mezclando un algoritmo genético con la heurística BLF. Al comparar esta técnica con el mismo método, desarrollado por Hopper y Turton (2001)[7] y con el método FOA + BLF, desarrollado por Babaoglu (2017)[19], se obtiene que para instancias pequeñas y medianas del problema el algoritmo desarrollado funciona de gran manera, obteniendo mejores o iguales resultados que los otros dos métodos. Para las instancias grandes con las que se realizaron las pruebas (*C6* y *C7* [7]) el algoritmo no entrega buenos resultados, por lo que se realiza una nueva prueba con la instancia *BENG10*[13], la cual posee un número similar de objetos. Los resultados de la última prueba indican que el desempeño del algoritmo para dichas instancias está relacionado con el ancho de la superficie. A mayor ancho más le cuesta al algoritmo mejorar la calidad de la solución debido a que para diferentes ordenamientos se obtiene una altura similar, provocando que el algoritmo se detenga debido a su criterio de detención.

La gran desventaja de este método es el tiempo de ejecución que tiene para instancias grandes, debido a que trabaja con poblaciones de individuos, por lo que hay que utilizar BLF para cada individuo de la población en cada iteración, provocando que el tiempo de ejecución de la técnica sea bastante mayor al de otras técnicas de búsqueda local.

Lo anterior indica el trabajo futuro con respecto los algoritmos genéticos para resolver el 2DSPP. Se debe trabajar en crear heurísticas de colocación con baja complejidad computacional que funcionen con la representación de orden para este problema. De esa manera, se podría mejorar el desempeño del algoritmo al aumentar la cantidad de iteraciones para instancias grandes con superficies grandes, logrando que el algoritmo entregue buenos resultados para todo tipo de instancias.

10 Bibliografía

References

[1] J. Błażewicz, P. Hawryluk, and R. Walkowiak. Using a tabu search approach for solving the two-dimensional irregular cutting problem. *Annals of Operations Research*, 41(4):313–325, Dec 1993.

[2] E. K. Burke, R. S. R. Hellier, G. Kendall, and G. Whitwell. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Operations Research*, 52(4):655–671, July 2004.

[3] Edmund K. Burke, Graham Kendall, and Glenn Whitwell. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. *INFORMS Journal on Computing*, 21(3):505–516, 2009.

[4] B. Chazelle. The bottomn-left bin-packing heuristic: An efficient implementation. *IEEE Trans. Comput.*, 32(8):697–707, August 1983.

[5] Charles Dubout. C++ blf 2d rectangle packing algorithm. url-http://www.dubout.ch/en/code/blf.cpp.

[6] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, January 1985.

[7] E Hopper and B.C.H Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. *European Journal of Operational Research*, 128(1):34 – 57, 2001.

[8] Manuel Iori, Silvano Martello, and Michele Monaci. Metaheuristic algorithms for the strip packing problem. 01 2003.

[9] Stefan Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165 – 181, 1996.

[10] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. New heuristic and interactive approaches to 2d rectangular strip packing. *J. Exp. Algorithmics*, 10, December 2005.

[11] Stephen CH Leung and Defu Zhang. A new heuristic approach for the stock-cutting problems. *Proceedings of World Academy of Science, Engineering & Technology*, 41:688–693, 2009.

[12] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241 – 252, 2002.

[13] Silvano Martello, Michele Monaci, and Daniele Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[14] N. Ntene and J.H. van Vuuren. A survey and comparison of guillotine heuristics for the 2d oriented offline strip packing problem. *Discrete Optimization*, 6(2):174 – 188, 2009.

[15] Wen-Tsao Pan. A new fruit fly optimization algorithm: Taking the financial distress model as an example. *Knowledge-Based Systems*, 26(Supplement C):69 – 74, 2012.

[16] María Cristina Riff, Xavier Bonnaire, and Bertrand Neveu. A revision of recent approaches for two-dimensional strip-packing problems. *Engineering Applications of Artificial Intelligence*, 22(4):823 – 827, 2009.

[17] Brenda S. Baker, Ed Coffman, and Ronald L. Rivest. Orthogonal packings in two dimensions. 9:846–855, 11 1980.

[18] Lijun Wei, Andrew Lim, and Wenbin Zhu. A skyline-based heuristic for the 2d rectangular strip packing problem. In *Proceedings of the 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems Conference on Modern Approaches in Applied Intelligence - Volume Part II*, IEA/AIE'11, pages 286–295, Berlin, Heidelberg, 2011. Springer-Verlag.

[19] İsmail Babaoğlu. Solving 2d strip packing problem using fruit fly optimization algorithm. *Procedia Computer Science*, 111(Supplement C):52 – 57, 2017. The 8th International Conference on Advances in Information Technology.