Revisión de Heurísticas para el Problema Bin Packing de una y dos dimensiones

Priscila Angulo López
A00813460@itesm.mx
Ciencias Computacionales
Tecnológico de Monterrey

Marcel Valdez Orozco
A00790834@itesm.mx
Ciencias Computacionales
Tecnológico de Monterrey

1 Introducción

El problema de *Bin Packing* sirve para caracterizar muchos problemas de la vida real, este problema consiste en tener un conjunto ilimitado de contenedores con capacidad W, y un conjunto de n piezas para empacar, con pesos $w_1, w_2, w_3, \ldots, w_n$; los pesos de las piezas pueden repetirse, sin embargo, se debe cumplir la restricción $0 \leq w_i \leq W$. La tarea consiste en acomodar las piezas dentro de los contenedores de tal modo que se minimice el número total de contenedores utilizados [1].

Aunque este problema sea fácil de entender, su solución es NP difícil, lo cual implica que llegar a la solución óptima en todas las instancias del problema implica un esfuerzo computacional exponencial. Es por este motivo que existe amplia investigación para llegar a soluciones aceptables en tiempo polinomial, y aquí es donde entran las heurísticas e hiperheurísticas, las primeras llegan a soluciones aproximadas y aceptables para una familia particular de instancias del problema, las segundas permiten la selección de heurísticas según las características de la instancia.

El presente reporte muestra la aplicación de dos heurísticas de selección, una para el problema *Bin Packing* de una sola dimensión, y otra con dos dimensiones regular. El contenido está organizado alrededor de esas heurísticas, en la primera sección se aborda la heurística *First Fit Decreasing* (FFD), la cual tiene su aplicación en el problema de *Bin Packing* de una dimensión. Esta heurística fue implementada y probada con 3 instancias de problemas. Los objetivos que se siguieron al implementar esta heurística fueron: hacer un análisis detallado del algoritmo que implementa la heurística, probar su efectividad en las instancias propuestas y comprobar que se cumple con la cota superior para el número de contenedores (objetos) obtenidos en el peor caso.

En la segunda sección de este reporte se trata el problema de *Bin Packing* de dos dimensiones regular, para este problema se consideraron 20 instancias de problemas, las cuales fueron resueltas aplicando dos versiones de la heurística de selección de piezas de Djang y Finch (DJD) [2]: $\text{DJD}_{1/3}$ y $\text{DJD}_{1/4}$, teniendo que la primera aplica un llenado inicial del 33% y la segunda del 25%. Dados los resultados de estas heurísticas se procedió a proponer una hiperheurística que selecciona cuál de las dos versiones es mejor para cada tipo de instancia.

En el contenido de ambas secciones se incluyen detalles de las heurísticas utilizadas, descripción de los algoritmos desarrollados y su complejidad, datos de las instancias con las que se experimentó y los resultados obtenidos.

2 FFD para *Bin Packing* de una dimensión

La heurística FFD es un algoritmo muy popular porque, aunque no siempre obtiene una solución óptima, tiene un buen comportamiento aún en el peor caso, estudios han probado que nunca usará más de $^{11}/_{9}\,OPT(L) + 1$ contenedores, dónde $OPT(L)$ es la cantidad mínima, u óptima, de contenedores necesarios para la instancia del problema. Originalmente esta cota superior estaba fijada en $^{11}/_{9}\,OPT(L) + 4$, pero se fue reduciendo con investigaciones posteriores [1] [3].

Los pasos que el algoritmo sigue son los siguientes:

1. Ordena descendentemente las piezas a empacar por su peso, con el algoritmo *counting sort*.
2. Busca la siguiente pieza más grande disponible.
   2.1. Recorre los contenedores disponibles en orden de creación.
      2.1.1. Si en el contenedor actual hay espacio suficiente, acomoda la pieza y regresa al paso 2. De lo contrario va al inciso siguiente.
      2.1.2. Si el contenedor actual es el último disponible agrega uno nuevo y agrega la pieza, regresa al paso 2.
      2.1.3. Si ninguna de las condiciones anteriores se cumplió simplemente regresa al paso 2.

Estos pasos fueron implementados en el programa desarrollado para probar la heurística. A continuación se describe la metodología utilizada para implementar esta heurística y obtener los resultados.

2.1 Metodología

Primeramente se procedió a la revisión teórica de la heurística y de su algoritmo, esto permitió definir el pseudocódigo del algoritmo a seguir. Como herramientas de desarrollo se utilizaron el lenguaje JAVA, el IDE NETBEANS y la plataforma JAVA SE 1.7; como metodología de desarrollo se utilizó "Test Driven Development" [4], la cual consiste en desarrollar primero las pruebas que definen y aseguran el comportamiento deseado, por lo que primero se crearon las pruebas utilizando JUNIT, posteriormente se desarrollaron las clases necesarias para el Algoritmo. Para la revisión de los resultados se optó por imprimir los datos en la consola del

IDE. El trabajo de identificar la Instancia 4 se llevó a cabo después de haber resuelto las primeras 3 instancias.

En las secciones siguientes se aborda la complejidad del algoritmo implementado y los experimentos realizados.

2.2 Análisis de la Complejidad

En la Tabla 1 se menciona la complejidad de cada uno de los pasos del algoritmo, no se incluye la impresión o presentación de los resultados en este análisis de complejidad:

Tabla 1. Análisis de Algoritmo FFD

Paso	Descripción	Complejidad
1	Ordenamiento de *n* piezas (*counting sort*)	$\theta(n)$
2	Ciclo que recorre cada pieza	$\theta(n)$
2.1	Ciclo que recorre cada contenedor existente.	$\theta(j) \approx O(n)$ En el peor de los se requerirá un contenedor por pieza.
2.1.1	Si la pieza cabe, agregarla.	$\theta(1)$ Operaciones de tiempo constante: comparar y agregar.
2.1.2	De lo contrario, revisar si ya se acabaron las posibilidades de contenedores y no fue posible agregar la pieza a ninguno, de ser así, agregar nuevo contenedor.	$\theta(1)$ Operaciones de tiempo constante: comparar y agregar.

La complejidad total que se obtiene en el peor caso es entonces: $\theta(2n+2)$, y en términos generales $\theta(n)$. Esta complejidad linear es mucho mejor de la que se obtendría si se intentara probar todas las posibles combinaciones de piezas para su acomodo.

2.3 Experimentos

Primeramente se trabajó con 3 instancias de problemas, adicionalmente, a la instancia 1 se le hizo una modificación: se eliminó la pieza de peso 46, de este modo se generó una segunda versión de la instancia, esto se hizo con el propósito de probar un comportamiento inconsistente que se ha documentado [1] que puede ocurrir con el algoritmo, este comportamiento es discutido más adelante. El origen de estas primeras instancias es de otros trabajos de investigación, no fueron generadas por nosotros.

Después de revisar el comportamiento obtenido con el cambio en la instancia 1, se buscó la forma de replicar ese comportamiento con una instancia de problema diferente, se logró identificar otra instancia que se comporta de la misma manera, la cual fue nombrada Instancia 4, y se obtiene el comportamiento inconsistente al remover la pieza de peso 38.

Las instancias con las que se trabajó están mostradas a continuación.

Instancia 1

Capacidad Contenedor: 524

Cantidad de Piezas: 33

37	10	106	85	252	252
252	10	37	10	46	127
127	252	252	10	12	12
442	9	252	106	10	106
252	127	106	84	127	9
12	10	127			

Instancia 2

Capacidad Contenedor: 1,000

Cantidad de Piezas: 60

351	357	269	252	261	350
303	473	347	262	287	252
474	258	366	410	271	275
370	298	273	366	419	444
372	299	445	439	259	272
315	251	430	320	450	273
472	395	275	288	292	269
298	495	274	252	355	307
350	366	283	466	414	361
363	255	272	254	263	268

Instancia 3

Capacidad Contenedor: 10,000

Cantidad de Piezas: 57

4812	4122	3326	2067	1492	468
4812	4122	3326	2067	1492	246
4812	3959	3168	1912	1308	246
4783	3787	3168	1897	1308	117
4778	3534	3168	1762	1274	117
4769	3534	3168	1762	1274	63
4769	3534	2649	1762	724	63
4738	3412	2317	1594	511	55
4199	3412	2317	1574	511	26
4199	3412	2156			

Instancia 4

Capacidad Contenedor: 1300

Cantidad de Piezas: 43

665	561	12	12	12	150
500	500	280	10	10	212
500	500	280	10	10	9
500	500	280	10	10	37
500	280	280	222	9	38
243	211	200	200	37	158
189	162	150	150	150	154
197					

Los experimentos consistieron en ingresar de forma individual cada una estas instancias al algoritmo implementado y registrar los resultados obtenidos, los datos que se obtuvieron fueron: cantidad de contenedores utilizados, piezas en cada uno de los contenedores y espacio no utilizado del contenedor.

El algoritmo implementado no tiene prerrequisitos de entrenamiento, siempre aplica el mismo método para resolver cada instancia. En la siguiente sección se comparan los resultados obtenidos para cada instancia con los resultados óptimos conocidos para cada una de ellas, también se menciona el comportamiento inconsistente obtenido en las instancias 3 y 4.

2.4 Resultados

En general los resultados obtenidos por FFD para las 3 instancias iniciales son muy buenos, el 66% de los resultados fueron óptimos, en 2 de las 3 instancias revisadas se obtuvo un resultado óptimo. Esto concuerda con los resultados de otros estudios realizados sobre esta heurística. La cuarta instancia no se contempla en esta tabla ya que fue generada de tal modo que el resultado fuera óptimo al resolverse con esta heurística en particular. La siguiente tabla muestra los resultados obtenidos para cada instancia y permite compararlos con el resultado óptimo conocido de las mismas:

Tabla 2. Resultados FFD

Instancia	Cantidad de contenedores FFD	Cantidad de contenedores óptima
1	7	7
2	23	20
3	15	15

Para las instancias 1 y 3, es claramente observable que la cota superior de la heurística se cumple, para la instancia 2 se puede aplicar la fórmula para corroborar que se cumpla:

$$L \leq {}^{11}/_{9}\, OPT(L) + 1$$
$$23 \leq {}^{11}/_{9}\,(20) + 1$$
$$23 \leq {}^{11}/_{9}\,(20) + 1$$
$$\mathbf{23 \leq 25.4}$$

El comportamiento inconsistente observado en la Instancia 1 y replicado en la Instancia 4, es que con las piezas originales se logra un empacado óptimo, con cero desperdicio de espacio, pero al *remover* uno de los elementos en particular, sucede que la cantidad de contenedores aumenta, dejando una gran cantidad proporcional de espacio no utilizado.

Este comportamiento es mencionado por [1], quien comenta que esto es una aparente discontinuidad en el desempeño de la heurística, la instancia mencionada en ese artículo es la Instancia 1 considerada en este reporte. Las únicas características que en ese artículo se identifican como factores para que este comportamiento ocurra son:

- FFD debe ser capaz de dar una solución sin desperdicios de espacio para la instancia.
- El elemento a eliminar no debe ser igual en peso que la suma de 2 o 3 elementos menores.

Sin embargo, esas características no son lo único requerido para replicar este comportamiento, de hecho identificar una segunda instancia requirió de considerables intentos. Dadas las instancias 1 y 4, se identificaron ciertas características que tienen ambas tienen en común y que pudieran influir en que el comportamiento ocurre. Estas características son adicionales a las mencionadas en el párrafo anterior. Para describir las características es necesario el siguiente marco de referencia:

- Las piezas están ordenadas ascendentemente y se contabiliza cuantas repeticiones hay para cada número; la lista a la que se hace referencia más adelante es de los números sin repeticiones.
- Las posiciones de las piezas en la lista ordenada se considera que inician en 1, no en 0.

Mencionado lo anterior, las características identificadas son:

- El número que se remueve es el que está en la posición 5 y debe tener una sola incidencia en la instancia. En la Instancia 1 este número es el 46, en la Instancia 4 es el 38.
- El número en la posición 4 debe ser menor al número en la posición 5 (Instancia 1: 37<46, Instancia 4: 37<38).
- La suma del número en la posición 3 y del número en la posición 1, debe ser mayor al número en la posición 2 multiplicado por 2 (Instancia 1: 12+9>10*2, Instancia 4: 12+9>10*2).
- Los contenedores dónde se utilice el número en la posición 2, se debe utilizar 2 veces (Instancia 1: contenedores 2, 3 y 4, Instancia 4: contenedores 2, 3 y 4).
- El número de veces que presente el número de la posición 1 debe ser igual al número de renglones en que se presente el número 2.
- Los números en las posiciones del 1 al 5 no deben ser múltiplos.
- La pieza que se remueve debe pertenecer al primer contenedor.
- El 40% de los números mayores de los contenedores en la solución óptima debe ser el mismo (Instancia 1: 252 encabeza 4 de 7 contenedores, Instancia 4: 500 encabeza 4 de 7 contenedores).

Estas características no son definitivas ni suficientes para identificar una instancia que pudiera tener ese comportamiento. Se intentó generar una instancia más siguiendo estas reglas, pero no fue posible encontrarla. Sin embargo, se incluyen en el reporte por ser evidencia del estudio realizado a las instancias identificadas.

Las tablas 3-8 muestran el detalle de los resultados arrojados por el algoritmo para cada instancia, para las instancias 1 y 4 con la pieza removida, se denota en cursiva el contenedor que fue añadido:

Tabla 3. Resultados Instancia 1

Cantidad de Contenedores utilizados: 7

Contenedor	Piezas	Espacio no utilizado
1	442, 46, 12, 12, 12	0
2	252, 252, 10, 10	0
3	252, 252, 10, 10	0
4	252, 252, 10, 10	0
5	252, 127, 127, 9, 9	0
6	127, 127, 127, 106, 37	0
7	106, 106, 106, 85, 84, 37	0

Tabla 4. Resultados Instancia 1 (Removiendo pieza 46)

Cantidad de Contenedores utilizados: 8

Contenedor	Piezas	Espacio no utilizado
1	442, 37, 37	8
2	252, 252, 12	8
3	252, 252, 12	8
4	252, 252, 12	8
5	252, 127, 127, 10	8
6	127, 127, 127, 106, 10, 10, 10	7
7	106, 106, 106, 85, 84, 10, 10, 9	8
8	*9*	*515*

Tabla 5. Resultados Instancia 2

Cantidad de Contenedores utilizados: 23

Contenedor	Piezas	Espacio no utilizado
1	495, 474	31
2	473, 472	55
3	466, 450	84
4	445, 444	111
5	439, 430	131
6	419, 414	167
7	410, 395	195
8	372, 370, 258	0
9	366, 366, 268	0
10	366, 363, 271	0
11	361, 357, 275	7
12	355, 351, 292	2
13	350, 350, 299	1
14	347, 320, 315	18
15	307, 303, 298	92
16	298, 288, 287	127
17	283, 275, 274	168
18	273, 273, 272	182
19	272, 269, 269	190
20	263, 262, 261	214
21	259, 255, 254	232
22	252, 252, 252	244
23	251	749

Tabla 6. Resultados Instancia 3

Cantidad de Contenedores utilizados: 15

Contenedor	Piezas	Espacio no utilizado
1	4812, 4812, 246, 117	13
2	4812, 4783, 246, 117, 26	16
3	4778, 4769, 63, 63, 55	272
4	4769, 4738, 468	25
5	4199, 4199, 1594	8
6	4122, 4122, 1574	182
7	3959, 3787, 2156	98
8	3534, 3534, 2649	283
9	3534, 3412, 2317, 724	13
10	3412, 3412, 3168	8
11	3326, 3326, 3168	180
12	3168, 3168, 2317, 1308	39
13	2067, 2067, 1912, 1897, 1762	295
14	1762, 1762, 1492, 1492, 1308, 1274, 511	399
15	1274, 511	8215

Tabla 7. Resultados Instancia 4

Cantidad de Contenedores utilizados: 7

Contenedor	Piezas	Espacio no utilizado
1	665, 561, 38, 12, 12, 12	0
2	500, 500, 280, 10, 10	0
3	500, 500, 280, 10, 10	0
4	500, 500, 280, 10, 10	0
5	500, 280, 280, 222, 9, 9	0
6	243, 212, 211, 200, 200, 197, 37	0
7	189, 162, 158, 154, 150, 150, 150, 150, 37	0

Tabla 8. Resultados Instancia 4 (Removiendo pieza 38)

Cantidad de Contenedores utilizados: 8

Contenedor	Piezas	Espacio no utilizado
1	665, 561, 37, 37	0
2	500, 500, 280, 12	8
3	500, 500, 280, 12	8
4	500, 500, 280, 12	8
5	500, 280, 280, 222, 10	8
6	243, 212, 211, 200, 200, 197, 10, 10, 10	7
7	189, 162, 158, 154, 150, 150, 150, 150, 10, 10, 9	8
8	*9*	*1291*

2.5 Discusión y Conclusiones sobre FFD

Los pasos del algoritmo de FFD son simples y fue posible implementarlos en un solo método. La observación y comparación de los resultados fueron también actividades sencillas. La parte que requirió un mayor esfuerzo fue la identificación de la Instancia 4, ya que no existe un conjunto de reglas establecidas para identificar las instancias que pueden incurrir en ese comportamiento. Al respecto se considera que lo mencionado por el autor en [1] requiere de mayor investigación.

Fue posible comprender porque esta Heurística es utilizada en herramientas comerciales, la garantía que ofrece su cota superior satisface las necesidades reales de la industria.

3 DJD para *Bin Packing* de dos dimensiones regular

El problema de *Bin Packing* de dos dimensiones regular consiste en acomodar un conjunto de piezas rectangulares en contenedores cuadrados, los cuales tienen un área mayor que cualquiera de las piezas. En este problema el objetivo también es minimizar la cantidad de contenedores a usar. Para que la solución a una instancia de este problema sea factible no deben existir traslapes entre piezas ni debe haber piezas que sobresalgan a los límites del contenedor [5].

Para poder solucionar este tipo de problema se requiere de dos fases: la primera fase es para la selección de la pieza y del contenedor donde se acomodará, la segunda fase es el acomodo de la pieza. Para cada una de las fases se requiere al menos una heurística. En este caso se utilizó la heurística de Djang y Finch (DJD) [2] para la fase de selección, y la heurística Fondo-Izquierda (Bottom-Left) para la fase de acomodo.

La heurística de selección DJD consiste en, dado un nuevo contenedor, agregarle piezas, empezando por la más grande disponible, hasta alcanzar un porcentaje de llenado deseado, a este porcentaje se llama *Capacidad Inicial*; posteriormente se intenta encontrar 1 pieza, o la combinación de 2 o 3 piezas que puedan llenar el espacio disponible sin sobrepasar el desperdicio *w* establecido; en este caso no se rotaron las piezas para acomodarlas. Originalmente, la Capacidad Inicial que fue propuesta por los autores era de 33%, sin embargo, existen variantes de esta heurística que proponen otros porcentajes [1]. En el presente trabajo se consideraron dos capacidades iniciales: 33% y 25%, la primera es llamada $DJD_{1/3}$ y la segunda $DJD_{1/4}$. El desperdicio *w* es primero establecido en 0, pero si no se encuentra un conjunto de piezas que cumplan con el requerimiento, se aumenta su valor en incrementos de 1/20 del área total del objeto.

El algoritmo de DJD se compone básicamente de los siguientes pasos:

1. Ordenar las piezas descendentemente.
2. Abrir un nuevo contenedor.
3. Mientras el área utilizada del contenedor sea menor o la *Capacidad Inicial* deseada, intentar agregar la pieza de mayor tamaño disponible.
4. Buscar 1, 2 o 3 piezas que llenen el espacio disponible, intentar acomodarlas, todo esto minimizando el espacio a desperdiciar.
    a. Si se encuentra una combinación aceptable, ir al paso siguiente.
    b. Si ninguna combinación cumple con los requisitos del desperdicio, aumentar el valor del desperdicio. Regresar al inicio del paso 4.
5. Si sigue habiendo piezas disponibles, regresar al paso 2, de lo contrario ir al paso siguiente.
6. Termina el algoritmo.

La heurística Fondo-Izquierda es un algoritmo para el acomodo de piezas que realiza lo siguiente: se empieza posicionando la pieza a agregar sobre la esquina superior derecha del contenedor, posteriormente se desliza la pieza hacia abajo tanto como sea posible, después se desliza la pieza hacia la izquierda tanto como sea posible, se repite este proceso hasta que la pieza no se mueva más [5]. Se debe verificar que las posiciones asumidas por la pieza no incurran en traslapes ni salgan de los límites. Si desde un principio la pieza no logra acomodarse dentro de los límites no se agrega al contenedor. La figura 1 muestra el funcionamiento de esta heurística.

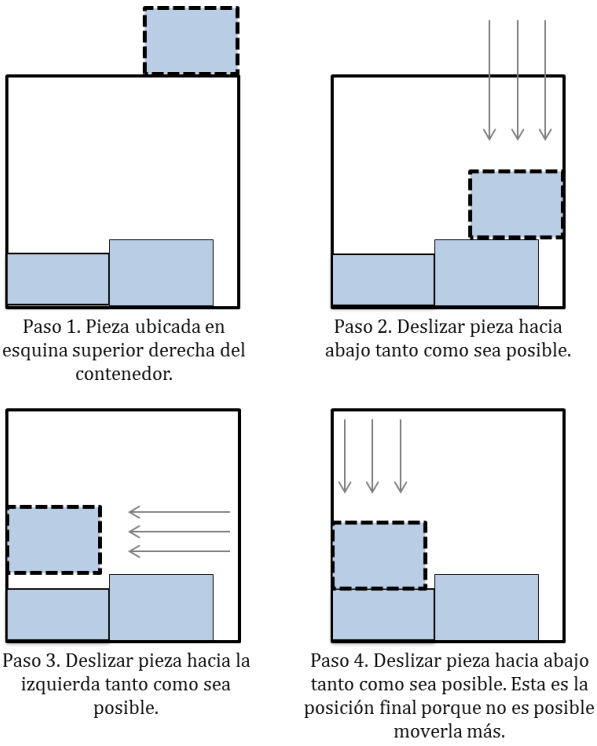


Figura 1. Funcionamiento heurística Fondo-Izquierda

Ambas heurísticas fueron implementadas en el programa desarrollado para resolver problemas de dos dimensiones regulares. A continuación se describe la metodología utilizada para la implementación y para obtener los resultados.

3.1 Metodología

Primeramente se procedió a la revisión teórica de las fases de solución del problema, de las heurísticas y de sus algoritmos, esto permitió definir el pseudocódigo del algoritmo general a seguir. Como herramientas de desarrollo se utilizaron el lenguaje JAVA, el IDE (Ambiente Integrado de Desarrollo, por sus siglas en inglés) NETBEANS y la plataforma JAVA SE 1.7; como metodología de desarrollo se utilizó "Test Driven Development" [4], por lo que primero se crearon las pruebas utilizando JUNIT, posteriormente se desarrollaron las clases necesarias para el Algoritmo.

Para la revisión de los resultados se imprimieron los datos en la consola del IDE, adicionalmente se almacenaron los resultados en archivos de texto, los cuales son mostrados

visualmente en una pequeña aplicación web que se desarrolló. Después de analizar los resultados obtenidos con las dos variantes de la heurística DJD, se procedió a definir las reglas para la hiperheurística y a implementarla. Finalmente se compararon los resultados obtenidos con las heurísticas individuales y la hiperheurística.

En las secciones siguientes se aborda la complejidad del algoritmo implementado y los experimentos realizados.

3.2 Análisis de la Complejidad

En tabla 9 se menciona la complejidad de cada uno de los pasos del algoritmo simple desarrollado, este algoritmo contiene la ejecución de la heurística DJD y la heurística Fondo-Izquierda, y sirve para resolver una instancia del problema a la vez. En el análisis de complejidad no se incluye el tiempo de procesamiento computacional requerido para obtener los datos de las instancias de los archivos de texto ni para desplegar los resultados.

Tabla 9. Análisis Algoritmo DJD y Fondo-Izquierda

Paso	Descripción	Complejidad
1	Ordenamiento de *n* piezas	$\theta(n \log n)$
2	Ciclo que se ejecuta mientras existan piezas sin acomodar.	$n(\theta(1) + O(n^2 + n) + O(n^4))$
2.1	Generar un nuevo contenedor.	$\theta(1)$
2.2	Llenar el contenedor hasta su capacidad inicial.	$O(n^2 + n)$ *Ver Desglose 1.*
2.3	Llenar el resto del contenedor.	$O(n^4)$ *Ver Desglose 3.*

Desglose 1: 2.2 Llenar el contenedor hasta su capacidad inicial.

Paso	Descripción	Complejidad
1	Ciclo que recorre piezas disponibles.	$O(n)$
1.1	Revisar si la pieza cabe y no se ha rebasado la capacidad inicial, si es así ir al paso siguiente, de lo contrario ir al paso 2	$\theta(1)$
1.2	Usar heurística Fondo-Izquierda para intentar acomodar. Implica revisar que no se traslape con las otras piezas contenidas.	$O(n)$ *Ver Desglose 2.*
2	Ciclo que recorre piezas contenidas en el contenedor.	$O(n)$
2.1	Remueve pieza de la lista de piezas disponibles.	$\theta(1)$

Desglose 2: Heurística Fondo-Izquierda

Paso	Descripción	Complejidad
1	Ciclo para hacer movimientos fondo e izquierda.	$\theta(1)$ La cantidad de movimientos varía pero es constante.
2	Revisar si está dentro de los límites del contenedor. Si lo cumple ir al paso siguiente, de lo contrario ir al paso 4.	$\theta(1)$ Es constante, sólo requiere 4 comparaciones.
3	Revisar que no haya traslapes. Si lo cumple regresar nueva posición, de lo contrario ir al paso siguiente.	$O(n)$ Requiere compararse con todas las piezas en el contenedor, en el peor de los casos es n.
4	Como la pieza no pudo ser acomodada dentro del contenedor, regresarla a su posición original.	$\theta(1)$

Desglose 3: 2.3 Llenar el resto del contenedor.

Paso	Descripción	Complejidad
1	Establecer valor del máximo desperdicio.	$\theta(1)$
2	Ciclo que intenta acomodar piezas mientras que el máximo desperdicio sea menor o igual al área disponible.	$\theta(1)$ El desperdicio aumenta de forma constante en $1/20$
2.1	*Método recursivo para intentar acomodar 1, 2 o 3 piezas que cumplan con el máximo desperdicio actual. La recursión permite agregar de una en una las piezas a la combinación. En el peor de los casos se hacen 3 recursiones, que ocurre cuando se busca la combinación de 3 piezas. Por lo que la complejidad es de $O(n^4)$ en el peor caso*.	
2.1.1	Ciclo para buscar y añadir pieza candidata a la combinación requerida.	$O(n)$ En el peor de los casos recorre todas las piezas.
2.1.1.1	Usar heurística Fondo-Izquierda para intentar acomodar. Implica revisar que no se traslape con las otras piezas contenidas. Si no es posible acomodar va a paso 2.1.1.4, de lo contrario va al paso siguiente.	$O(n)$ *Ver Desglose 2.*
2.1.1.2	Si hacen falta más piezas para completar la combinación volver a hacer recursión, ir al paso 2.1.1, de lo contrario va al paso siguiente.	NA
2.1.1.3	Regresa un verdadero al nivel de recursión superior, se mantienen los cambios.	$\theta(1)$
2.1.1.4	Regresa un falso al nivel de recursión superior para deshacer cambios.	$\theta(1)$
2.2	Sale de la recursión, si el resultado final recibido fue verdadero termina el proceso, de lo contrario va al paso siguiente.	$\theta(1)$
2.3	No fue posible encontrar piezas que cumplieran con el	$\theta(1)$

	desperdicio establecido. Se aumenta el máximo desperdicio permitido. Regresa al paso 2.	

La complejidad total que se obtiene es entonces: $O(n \log n + n + n^3 + n^2 + n^5)$ y en términos generales $O(n^5)$. Aunque esta complejidad es polinomial, está dada por el peor caso, y se estima que el caso promedio es mucho mejor, sin embargo, no se incluye en este reporte.

La complejidad del algoritmo analizado no ve afectada por añadir la lógica de la hiperheurística, ya que esta tiene un costo adicional $O(3n)$. En la sección 3.5, se explica el funcionamiento de la hiperheurística generada.

No descartamos la posibilidad de que con un análisis amortizado del algoritmo recursivo de posicionamiento de piezas se podría encontrar que el algoritmo tiene una complejidad con un orden exponencial menor a n^4, dado que el peor caso no se presenta tan frecuentemente.

3.3 Experimentos $DJD_{1/3}$ y $DJD_{1/4}$

Se trabajó con 20 instancias de problemas, todas con contenedores de 100x100, y con piezas rectangulares. Los datos de las instancias se encontraban contenidos en archivos de texto, uno por cada instancia. La información proporcionada para cada instancia era: el número de piezas, el ancho y alto del contenedor y el detalle de las piezas: número de vértices y coordenadas de la pieza. El detalle de las instancias no se incluye en el texto de este reporte por su extensión.

Cada instancia fue resuelta con las dos variantes de la heurística, los resultados fueron observados tanto en texto (en la consola del IDE) como visualmente (en la aplicación web generada). En la sección siguiente se mencionan los resultados obtenidos de estos experimentos.

3.4 Resultados $DJD_{1/3}$ y $DJD_{1/4}$: Generación de una hiperheurística

La tabla 10 muestra en un consolidado la cantidad de instancias en que cada heurística fue mejor, en los casos donde con ambas capacidades se obtuvo el mismo resultado, aparece la leyenda "Igual". La tabla 11 muestra en términos generales con cuál capacidad inicial se obtuvieron los mejores resultados para cada instancia.

Tabla 10. Consolidado de Resultados

Heurística (Capacidad Inicial)	Cantidad de instancias con mejores resultados
$DJD_{1/3}$ (0.33)	7
$DJD_{1/4}$ (0.25)	6
Igual	7

En la tabla 12 es posible comparar el desempeño de las variantes de DJD utilizadas, entre los datos mostrados están:

- Cant. Piezas. Es la cantidad de piezas en la instancia.
- OPC. Es la "Ocupación promedio por pieza en contenedor", indica una proporción promedio de las piezas en el contenedor.
- NTC. Es el "Número teórico de contenedores", este número es obtenido con la siguiente fórmula:

$$NTC = \left(\frac{AP}{CA}\right)$$

Donde AP es la suma del área total de las piezas de la instancia, y CA es el área del contenedor.

- CPA. Es la "Cantidad de Piezas más anchas que la Capacidad Inicial" considerada.

Tabla 11. Mejores resultados por instancia

Instancia	Mejores resultados obtenidos con Heurística:
PF01	Igual
PF02	DJD 1/4
PF03	Igual
PF04	DJD 1/3
PF05	DJD 1/3
PF06	Igual
PF07	Igual
PF08	DJD 1/3
PF09	Igual
PF10	DJD 1/3
PF11	DJD 1/3
PF12	DJD 1/4
PF13	DJD 1/4
PF14	DJD 1/4
PF15	DJD 1/3
PF16	DJD 1/3
PF17	DJD 1/4
PF18	Igual
PF19	Igual
PF20	DJD 1/4

Tabla 12. Resultados de cada heurística por instancia

Inst.	Cant. Piezas	OPC	NTC	$DJD_{1/3}$		$DJD_{1/4}$	
				Cant. Conts.	CPA	Cant. Conts.	CPA
PF01	8	0.5	4	4	2	4	2
PF02	30	0.2	6	8	2	7	2
PF03	28	0.25	7	8	0	8	0
PF04	36	0.1666	6	7	0	8	0
PF05	40	0.2	8	9	0	10	0
PF06	30	0.1666	5	6	0	6	0
PF07	27	0.1111	3	5	0	5	0
PF08	24	0.125	3	4	0	5	0
PF09	36	0.1666	6	8	0	8	0
PF10	35	0.2	7	8	0	9	0
PF11	35	0.2	7	8	0	9	0
PF12	35	0.2	7	9	4	8	4
PF13	25	0.2	5	7	1	6	1
PF14	25	0.2	5	7	1	6	1
PF15	30	0.1666	5	6	0	7	0
PF16	30	0.1666	5	6	0	7	0

PF17	25	0.2	5	7	2	6	2
PF18	25	0.2	5	6	1	6	1
PF19	30	0.1666	5	7	0	7	0
PF20	25	0.2	5	7	0	6	0

Los resultados de la tabla 3 permiten destacar en qué escenarios se comporta mejor una heurística que otra, esta información permitió definir una hiperheurística de preselección para resolver las instancias, es decir la hiperheurística revisa las características de la instancia y determina que heurística es la más apropiada para resolverla.

El proceso de selección que sigue la hiperheurística es el siguiente:

1. Obtiene la cantidad de piezas que son tan anchas que no cabe nada a un lado de ellas y su área es mayor que el 25% del área del contenedor, a esta cantidad la llamaremos C25.
2. Obtiene la cantidad de piezas que son tan anchas que no cabe nada a un lado de ellas y su área es mayor que el 33% del área del contenedor, a esta cantidad la llamaremos C33.
3. Si C25 es mayor que C33, se selecciona la heurística $DJD_{1/4}$ y termina, de lo contrario continua al paso siguiente. La situación contemplada por esta regla indica que la instancia contiene más piezas que su área a ocupar está en el rango del 25% al 33%, esto implica que, después del llenado inicial, es factible encontrar 1, 2 o 3 piezas que llenen el contenedor con mínimo desperdicio.
4. Si la cantidad total de piezas en la instancia es mayor o igual a 35 o su área promedio en proporción con el área del contenedor es menor a 20%, se selecciona la heurística $DJD_{1/3}$, si esta condición no se cumple ir al paso siguiente. Esta regla contempla el caso dónde las instancias contienen muchas piezas o piezas pequeñas, y que por lo tanto es mejor tener un llenado inicial mayor, porque posteriormente se dificultará encontrar 1, 2 o 3 piezas que llenen con mínimo desperdicio.
5. Como última opción, si no se seleccionó una heurística en los pasos anteriores, se selecciona la heurística $DJD_{1/4}$, se llega hasta este paso cuando las características de la instancia indicaron que no hay más de 35 piezas o que el área de las piezas es mayor al 20% del área del contenedor.

La siguiente sección muestra los resultados obtenidos con la hiperheurística generada.

3.5 Resultados de la hiperheurística

La tabla 13 muestra la heurística que seleccionó la hiperheurística para resolver cada instancia y permite compararla con la que era la mejor opción, la cual fue identificada en la sección anterior en la tabla 11. Como es posible observar, la hiperheurística seleccionó en todos los casos la heurística correcta, lo cual prueba su efectividad en los casos estudiados.

Tabla 13. Resultados de Hiperheurística

Instancia	Mejores resultados obtenidos con Heurística:	Heurística seleccionada por HH:
PF01	Igual	DJD 1/4
PF02	DJD 1/4	DJD 1/4
PF03	Igual	DJD 1/4
PF04	DJD 1/3	DJD 1/3
PF05	DJD 1/3	DJD 1/3
PF06	Igual	DJD 1/3
PF07	Igual	DJD 1/3
PF08	DJD 1/3	DJD 1/3
PF09	Igual	DJD 1/3
PF10	DJD 1/3	DJD 1/3
PF11	DJD 1/3	DJD 1/3
PF12	DJD 1/4	DJD 1/4
PF13	DJD 1/4	DJD 1/4
PF14	DJD 1/4	DJD 1/4
PF15	DJD 1/3	DJD 1/3
PF16	DJD 1/3	DJD 1/3
PF17	DJD 1/4	DJD 1/4
PF18	Igual	DJD 1/4
PF19	Igual	DJD 1/3
PF20	DJD 1/4	DJD 1/4

Las figuras 2 a la 21 permiten apreciar de forma visual los resultados obtenidos para cada instancia. Esta representación gráfica fue obtenida a través de la aplicación web que se desarrolló para fines de este proyecto.

Figura 2. Resultados Instancia 1

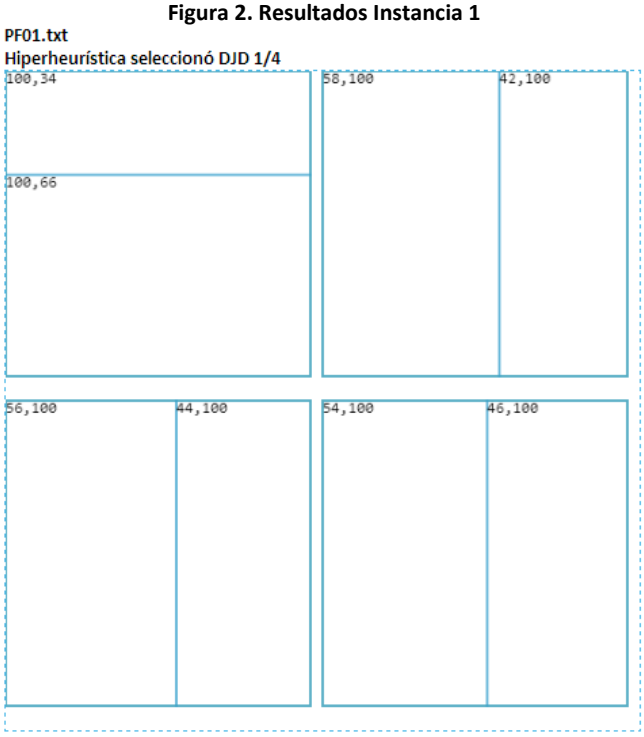


Figura 3. Resultados Instancia 2

PF02.txt
Hiperheurística seleccionó DJD 1/4

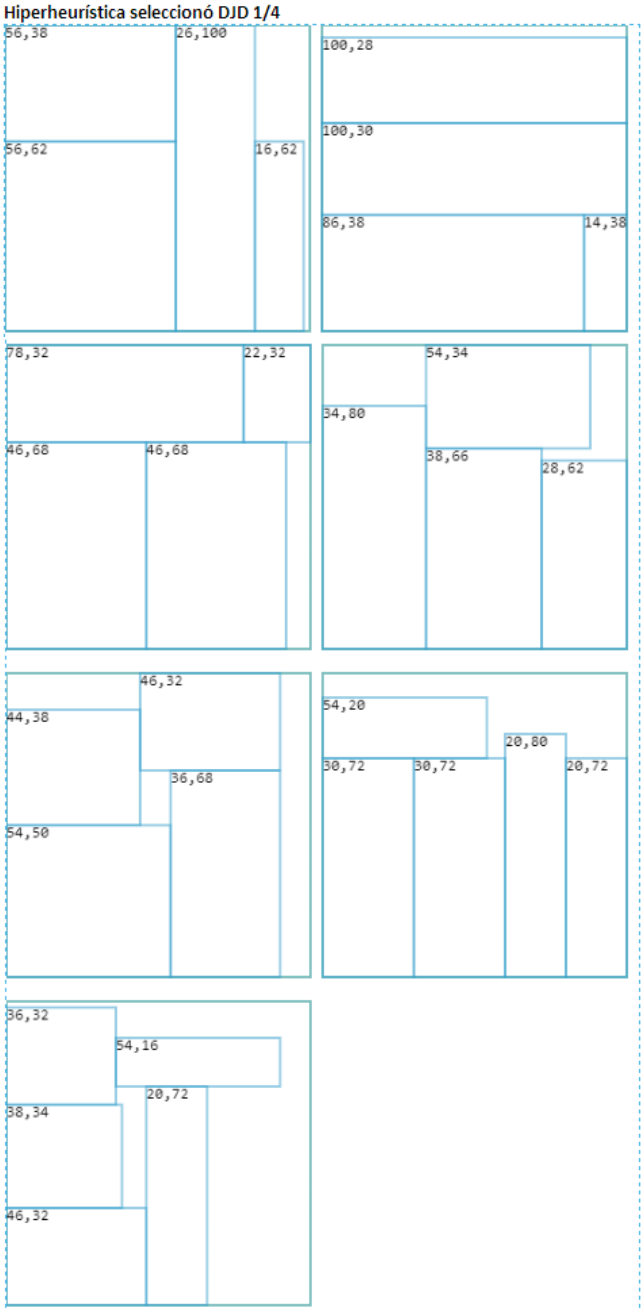


Figura 4. Resultados Instancia 3

PF03.txt
Hiperheurística seleccionó DJD 1/4

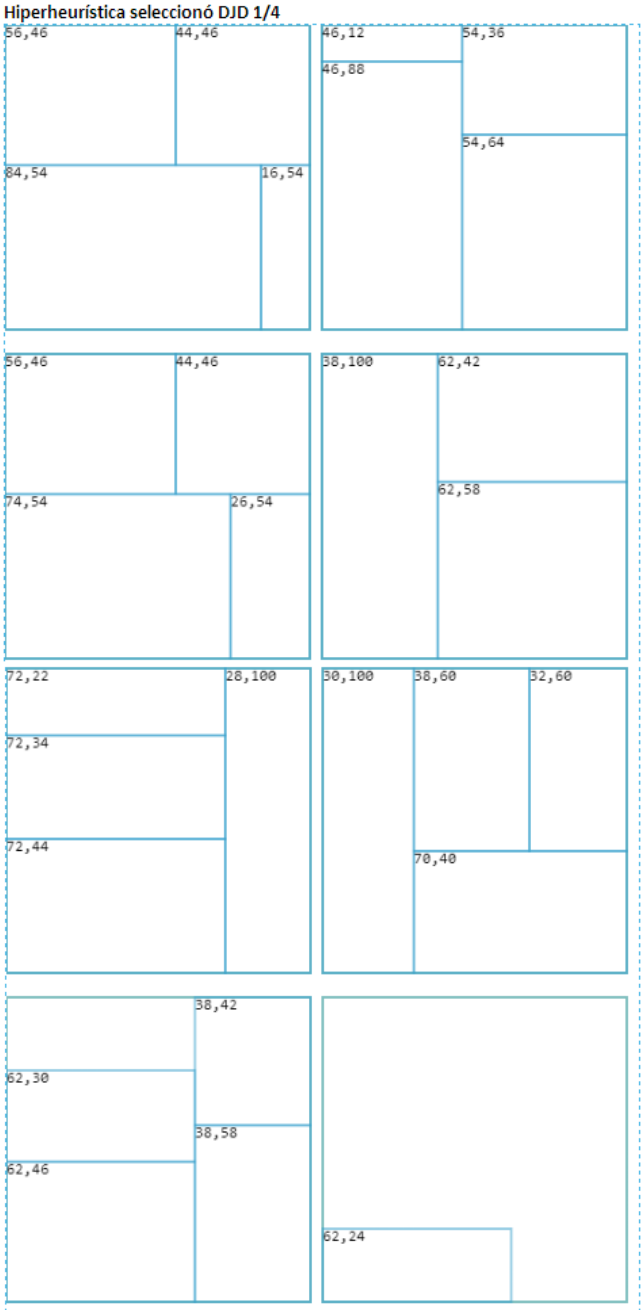


Figura 5. Resultados Instancia 4

PF04.txt
Hiperheurística seleccionó DJD 1/3

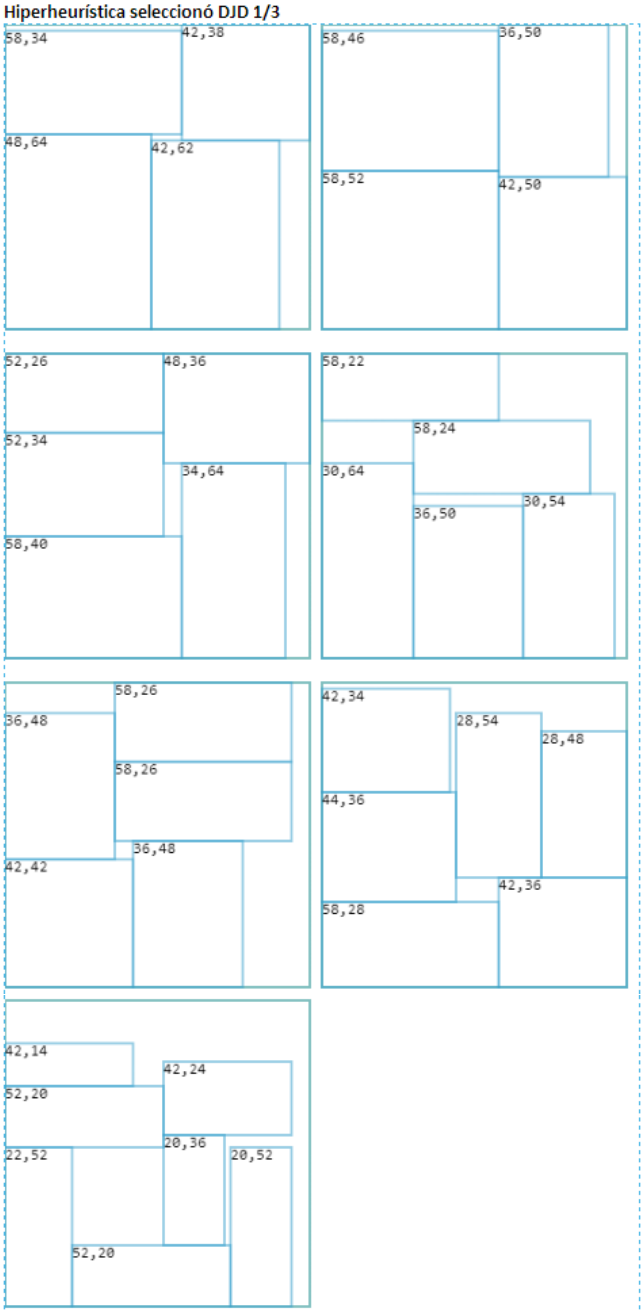


Figura 6. Resultados Instancia 5

PF05.txt
Hiperheurística seleccionó DJD 1/3

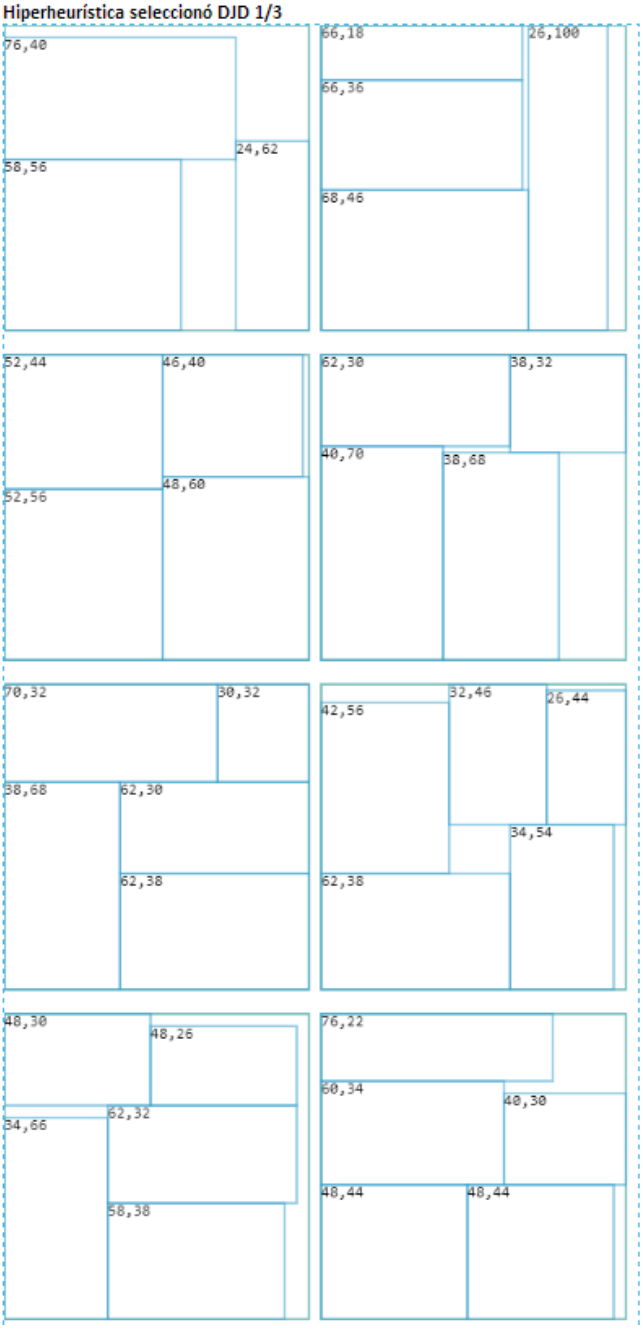


Figura 7. Resultados Instancia 6

PF06.txt
Hiperheurística seleccionó DJD 1/3

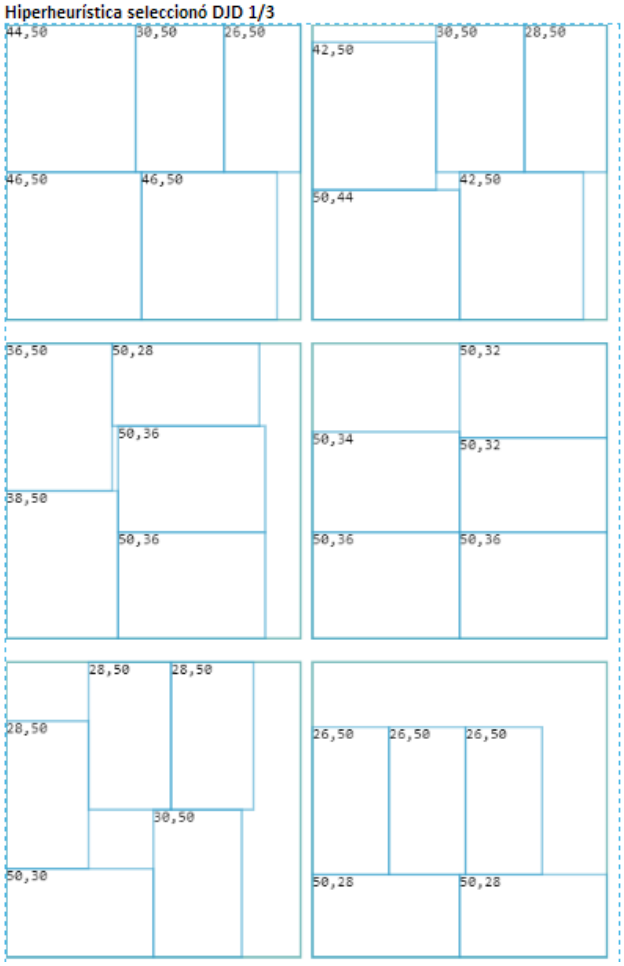


Figura 8. Resultados Instancia 7

PF07.txt
Hiperheurística seleccionó DJD 1/3

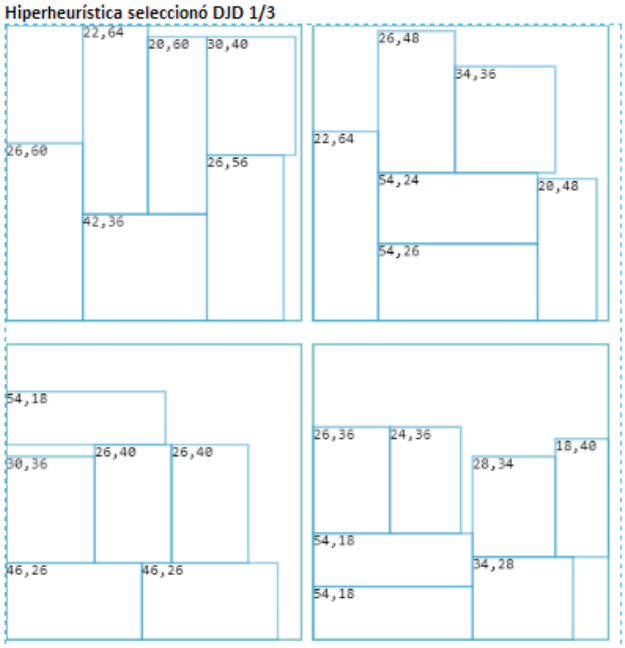


Figura 9. Resultados Instancia 8

PF08.txt
Hiperheurística seleccionó DJD 1/3

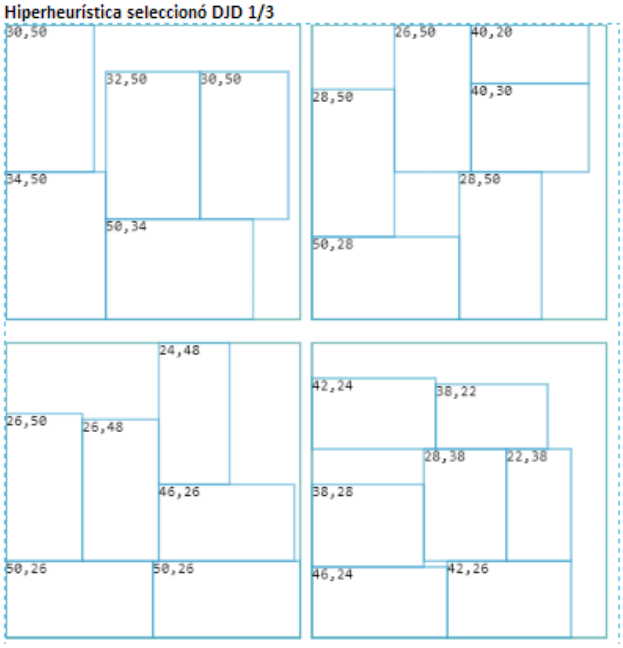
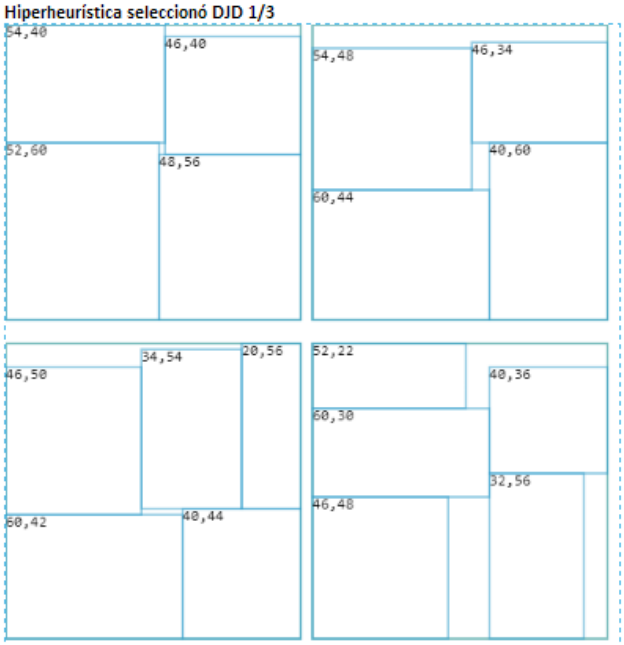


Figura 10. Resultados Instancia 9

PF09.txt
Hiperheurística seleccionó DJD 1/3



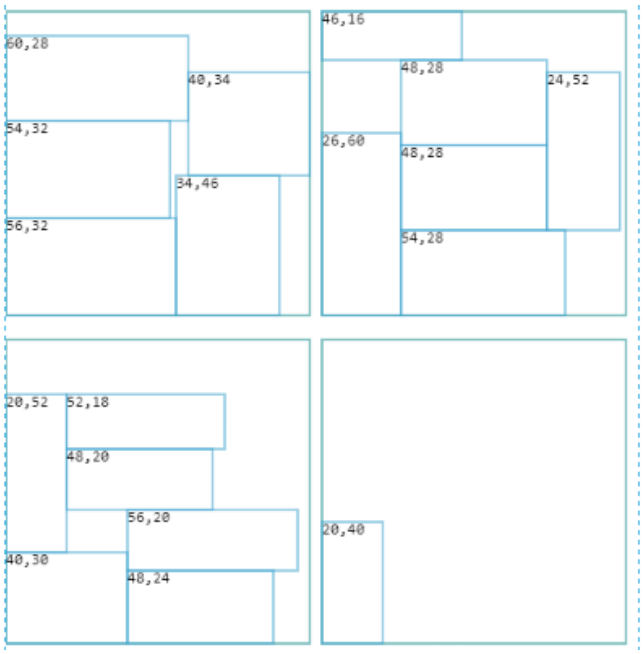
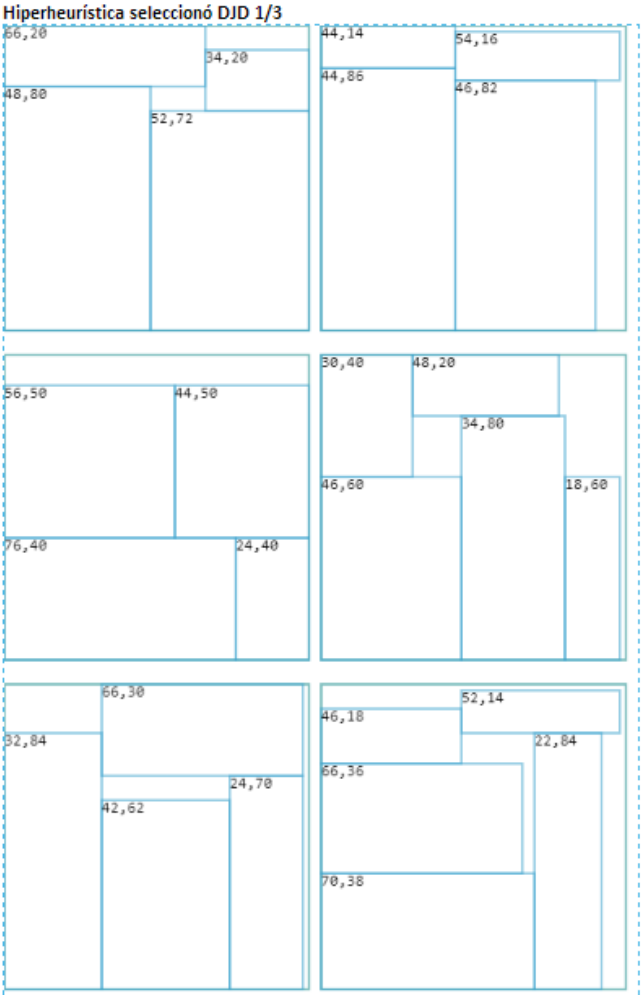


Figura 11. Resultados Instancia 10

PF10.txt
Hiperheurística seleccionó DJD 1/3



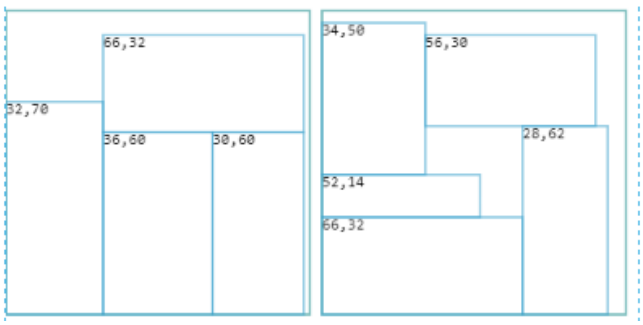


Figura 12. Resultados Instancia 11

PF11.txt
Hiperheurística seleccionó DJD 1/3

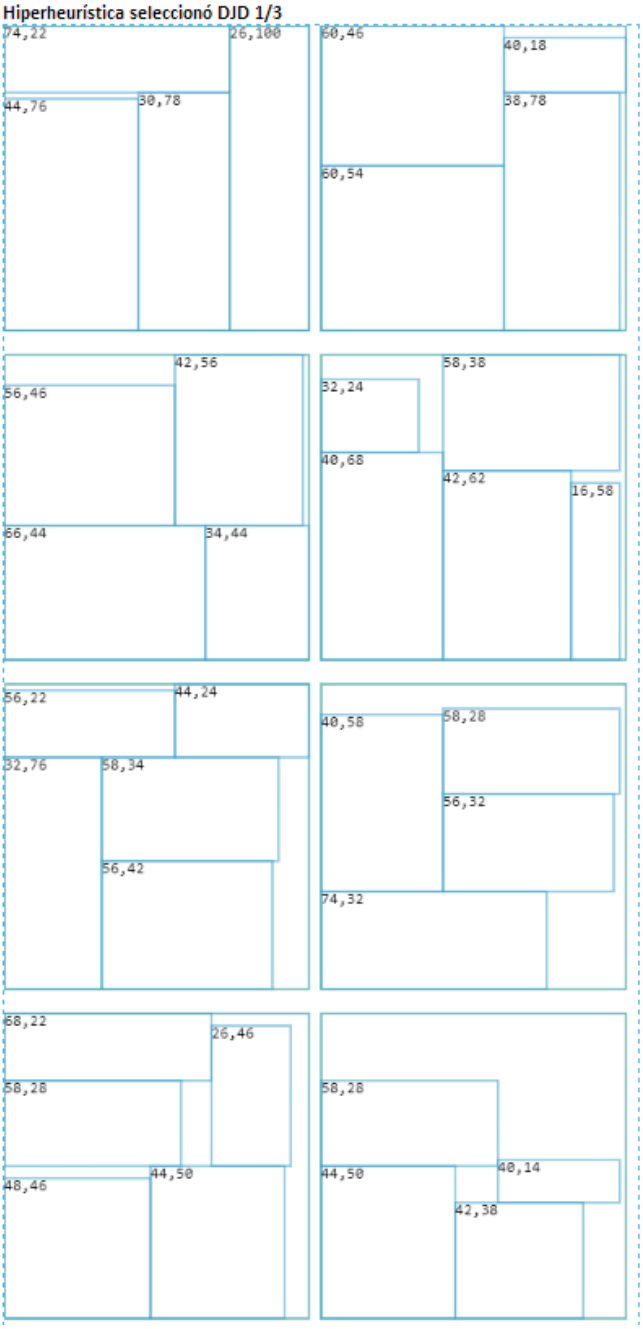


Figura 13. Resultados Instancia 12

PF12.txt
Hiperheurística seleccionó DJD 1/4

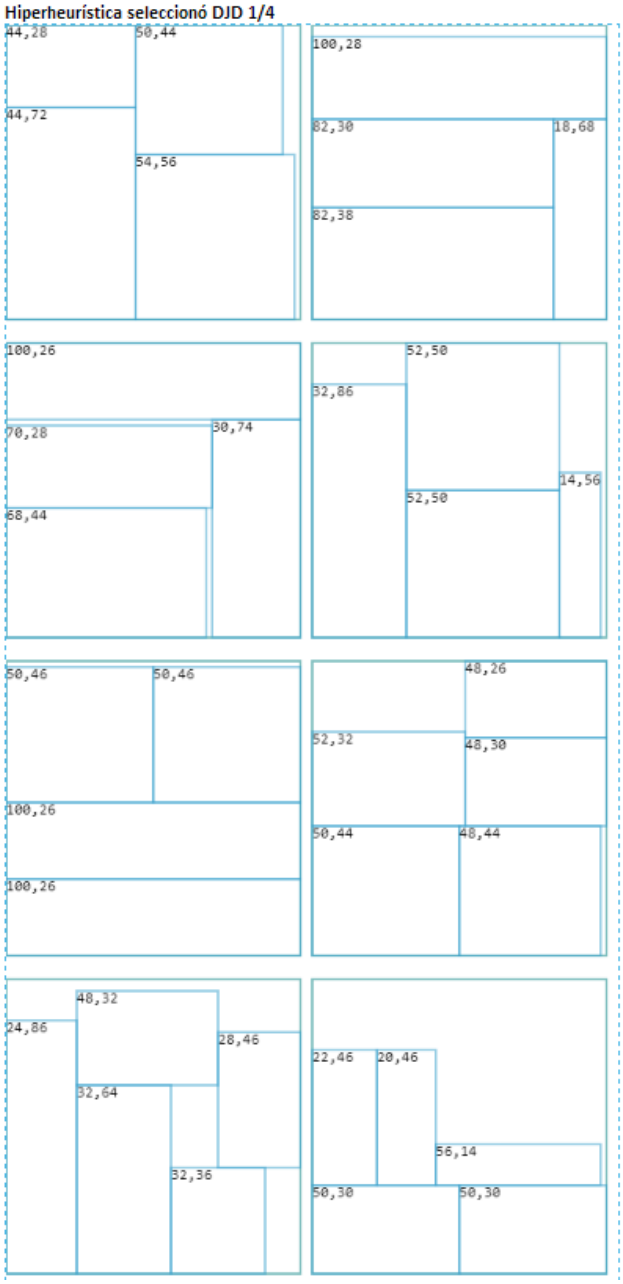
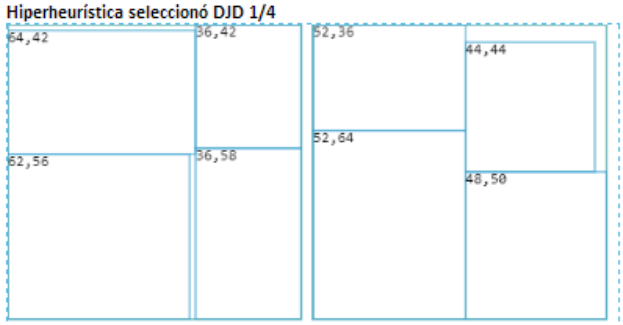


Figura 14. Resultados Instancia 13

PF13.txt
Hiperheurística seleccionó DJD 1/4



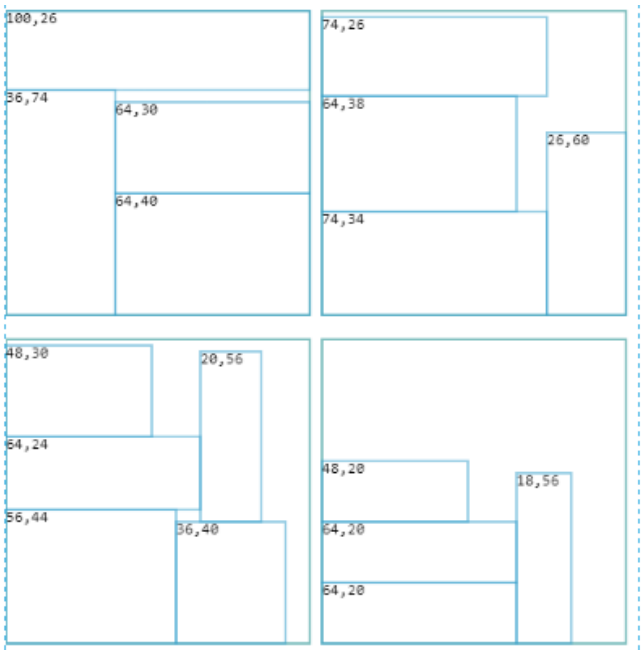


Figura 15. Resultados Instancia 14

PF14.txt
Hiperheurística seleccionó DJD 1/4

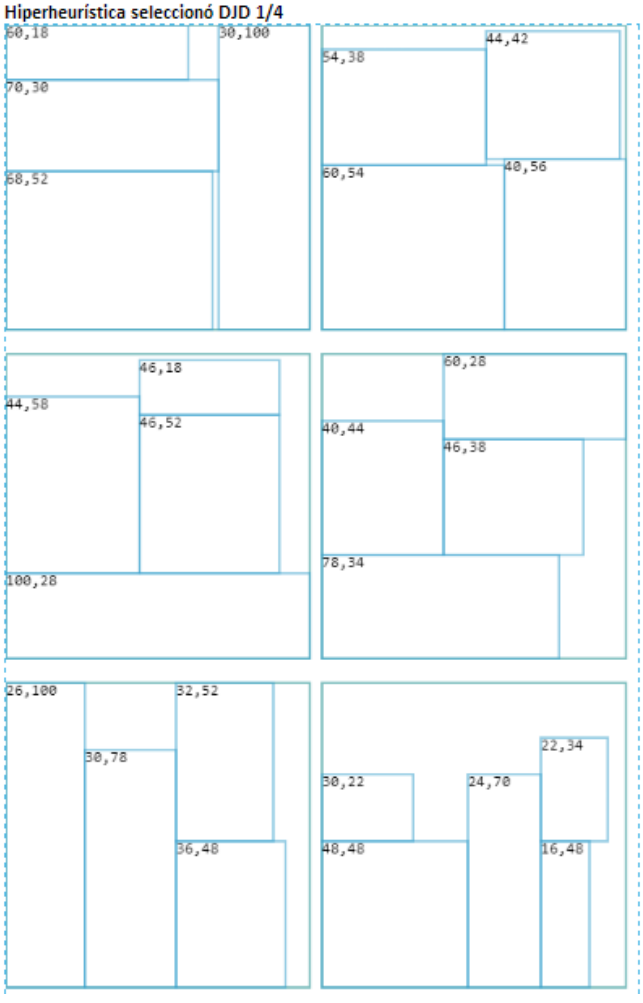


Figura 16. Resultados Instancia 15

PF15.txt
Hiperheurística seleccionó DJD 1/3

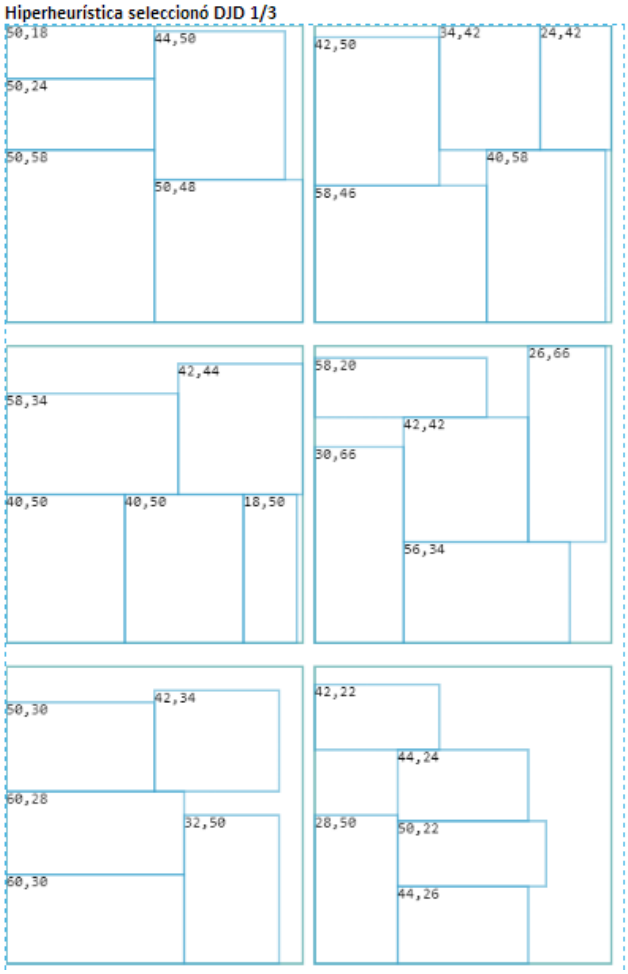
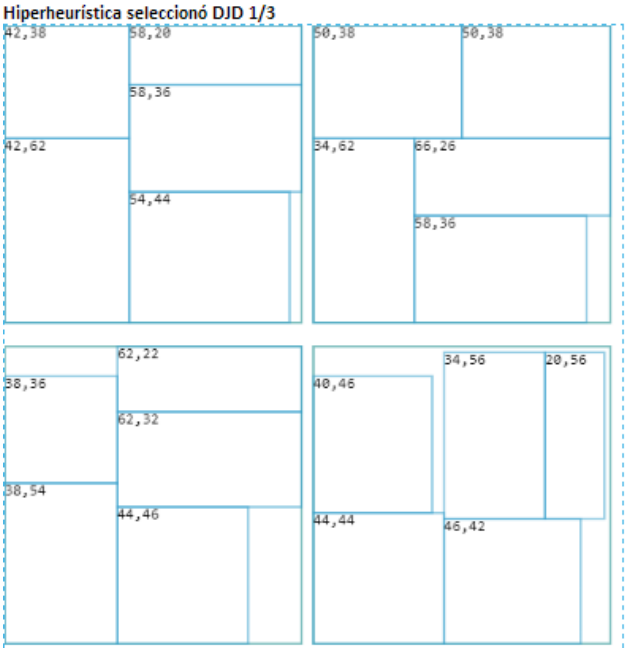


Figura 17. Resultados Instancia 16

PF16.txt
Hiperheurística seleccionó DJD 1/3



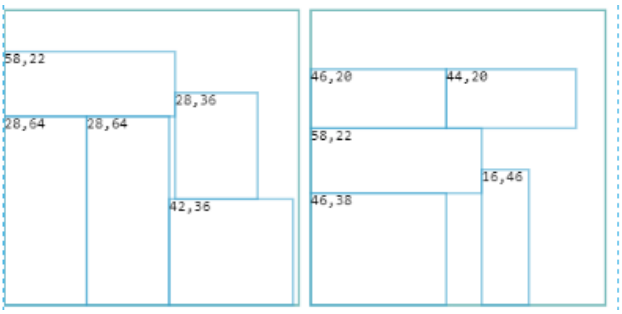


Figura 18. Resultados Instancia 17

PF17.txt
Hiperheurística seleccionó DJD 1/4

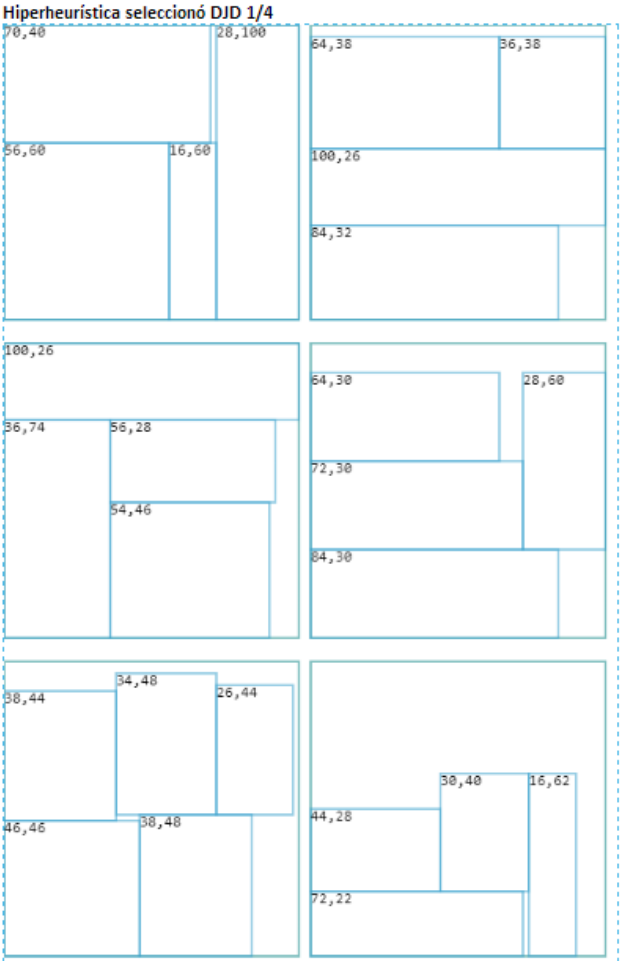
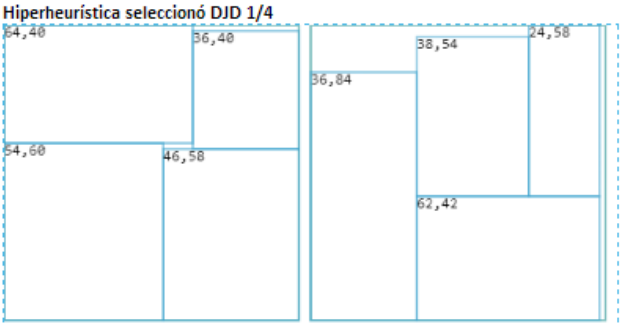


Figura 19. Resultados Instancia 18

PF18.txt
Hiperheurística seleccionó DJD 1/4



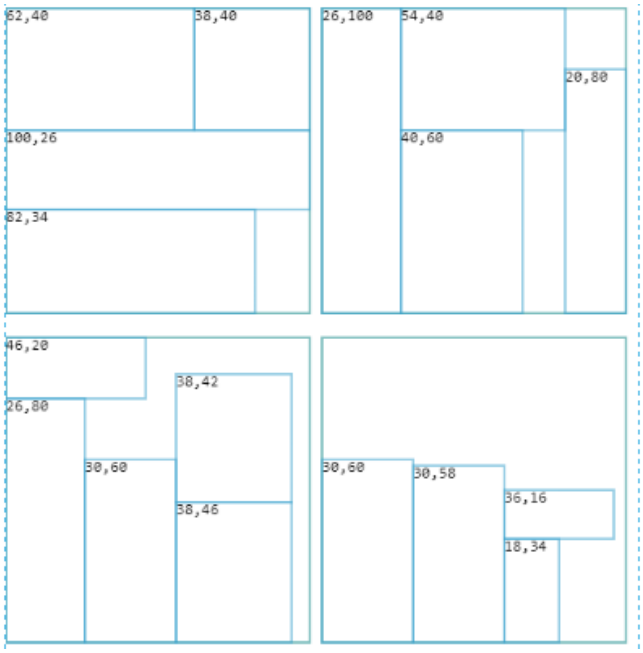


Figura 20. Resultados Instancia 19

PF19.txt
Hiperheurística seleccionó DJD 1/3

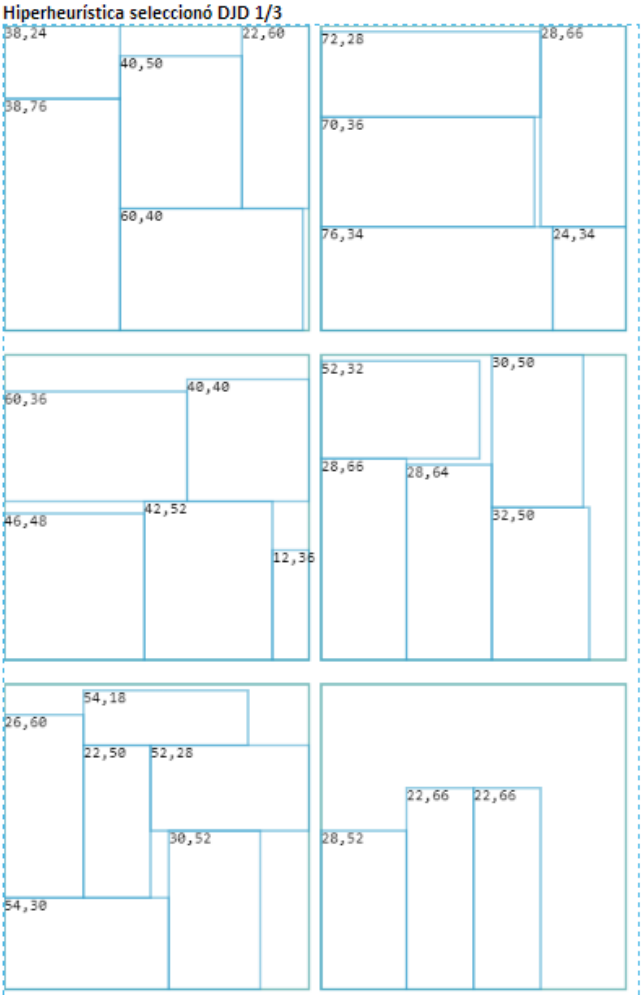
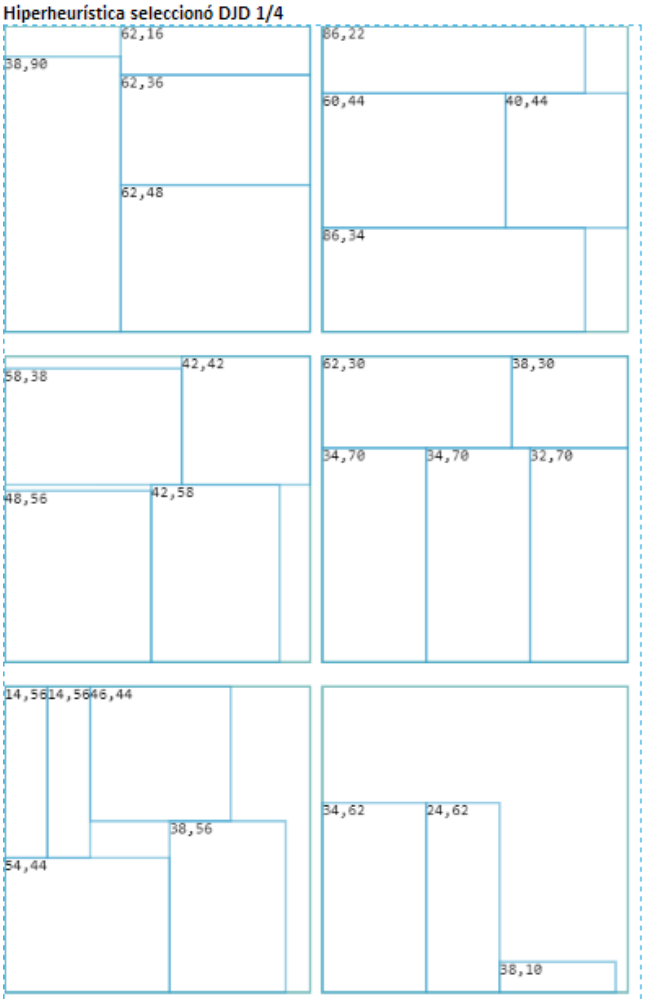


Figura 21. Resultados Instancia 20

PF20.txt
Hiperheurística seleccionó DJD 1/4



3.6 Discusión y Conclusiones sobre DJD: Heurísticas e Hiperheurística

Aunque considerar dos capacidades iniciales fue un factor relevante en la obtención de resultados correctos, existen otros factores que, de ser considerados, pudieran generar mejores resultados, entre ellos están:

- Rotación de piezas. Fue posible observar que en algunos casos se hubieran logrado mejores resultados si las piezas se hubieran rotado. Incluir esta operación no aumenta el orden asintótico del algoritmo.

- Hiperheurística para acomodo. Fue posible observar algunos casos en los que si, por ejemplo, se hubiera intentando acomodar la pieza empezando por la esquina superior izquierda, se hubiera logrado un mejor acomodo. Si al algoritmo desarrollado se le agregaran otras opciones de heurísticas de acomodo, con complejidad lineal, para probar si con alguna es posible agregar la pieza al contenedor, no se afectaría el orden asintótico del algoritmo.
- Consideración de otras capacidades de llenado inicial. En casos dónde las piezas son muy pequeñas es mejor tener un llenado inicial mayor, para así aumentar las posibilidades de encontrar la combinación de 1, 2 o 3 piezas que terminen de llenar el contenedor con mínimo desperdicio. Esto tampoco afectaría el orden asintótico del algoritmo ya que la decisión recaería en el método que selecciona de forma estática la capacidad inicial a utilizar.

Los resultados obtenidos con la hiperheurística sólo fueron basados en las instancias consideradas, por lo que se requeriría realizar pruebas con más instancias para revisar su eficacia y aplicar los refinamientos correspondientes. Otro punto a denotar como conclusión es que la hiperheurística desarrollada sólo funcionaría para instancias de problemas en los que las piezas son rectangulares o cuadradas, el algoritmo desarrollado no tiene las capacidades necesarias para resolver de forma eficaz instancias con figuras irregulares.

4 Referencias

[1] P. Ross, "Hyper-heuristics," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, New York, Springer, 2005, pp. 529-556.

[2] E. Lopez-Camacho, G. Ochoa and H. Terashima-Marin, "An effective heuristic for the two-dimensional irregular bin packing," *Annals of Operations Research,* pp. 1-24, 2013.

[3] M. Yue, "A simple proof of the inequality FFD (L) ≤ 11/9 OPT (L) + 1, ∀L for the FFD bin-packing algorithm," *Acta Mathematicae Applicatae Sinica,* vol. 7, no. 4, pp. 321-331, 1991.

[4] K. Beck, Test-Driven Development: By Example, Boston: Addison-Wesley Professional, 2003.

[5] E. Hopper and B. Turton, "An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem," *European Journal of Operational Research,* vol. 128, no. 1, p. 34–57, 2001.