Inteligencia Artificial

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

Fabian Marcerlo Guillermo Riquelme Macias

17 de enero de 2020

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):	_____

Resumen

Este documento consiste en un estudio del Two-Dimensional Strip Packing Problem (2DSPP) el cual consiste en ubicar un conjunto de objetos rectangulares de tamaño ajustado dentro de una tira(strip). Se hará mención a las primeras nociones del problema y sus equivalentes, desarrollando y mostrando sus variantes, para así poder ahondar en las soluciones, heurísticas y meta-heurísticas propuestas en la literatura de estudio. Finalmente se llevará acabo una propuesta para la resolución del problema basada en Backtracking guiado por Conflictos (BT+CBJ), incluyendo exhaustivas pruebas, analisis sobre su factibilidad y uso práctico.

1. Introducción

Los problemas de Corte y Empaque/Encaje (Cutting & Packing) son un bien conocido ejemplo de problemas de optimización combinatoria. Tienen variadas aplicaciones industriales como en manufactureras de corte de rollos de papel o bien cortando tiras(strip) de tela en la industria textil.

2D Strip Packing Problem (2DSPP)[5] consiste en que se tiene n elementos rectangulares y una tira(strip) de material de ancho fijo W y alto infinito. Sin perdida de generalidad podemos afirmar que las dimensiones de los elementos y la tira son números enteros, ya que cualquier conjunto de dimensiones en forma de fracción pueden ser escaladas a números enteros (x :

$[0{,}3, 0{,}03] \rightarrow x' : [30, 3]$). El objetivo es minimizar la altura H utilizada por los n elementos luego de su encaje, no permitiendo que los objetos se traslapen y permitiéndoles rotar únicamente en $90°$.

La importancia del problema reside en las grandes industrias, donde un pequeño porcentaje de material inutilizado podría presentar pérdidas de millones de dólares o bien dinero que no está siendo bien utilizado.

En estricto rigor (2DSPP) y sus variantes pertenecen a la categoría de problemas NP-HARD[10], dificultando y tal vez imposibilitando el desarrollo de métodos exactos [19]. Debido a lo anterior se han tenido que desarrollar técnicas heurísticas y meta-heurísticas[20] a fin de poder encontrar óptimos locales y a su vez con cierto grado de aceptación en tiempos considerables de computación, que serán mencionadas en secciones posteriores de este documento.

Muchas funciones objetivo pueden ser propuestas dependiendo de las necesidades industriales específicas, pero para este documento se guiará únicamente por el objetivo clásico de menor desperdicio Trim Loss Problem (TLP)[1].

De esta sección en adelante la estructura del documento consistirá en una pequeña definición y ejemplificación gráfica del problema ahondando en sus características principales y variantes populares. Luego en el **Estado del Arte** se verá el contexto del problema, el contexto del estudio del problema, y el desarrollo que han tenido las soluciones a lo largo del tiempo. Mas adelante se enseñará el modelo matemático respectivo a un modelo de programación lineal. Mas adelante vendrán secciones que explicarán como se llevó a cabo la implementación del algoritmo, tanto en forma gráfica como en pseudocódigo. Y finalmente se expondrán los experimentos realizados y los resultados obtenidos con riqueza en gráficos y tablas, para finalmente llevar a cabo una conclusión breve sobre lo investigado del problema y sobre la factibilidad de la implementación.

Cabe mencionar que el propósito de este documento es ser un estudio y recopilación general de información sobre este problema y como aporte a la variedad de soluciones que pueden ser encontradas. Finalmente todo el codigo puede ser encontrado en el repositorio remoto [1].

2. Definición del Problema

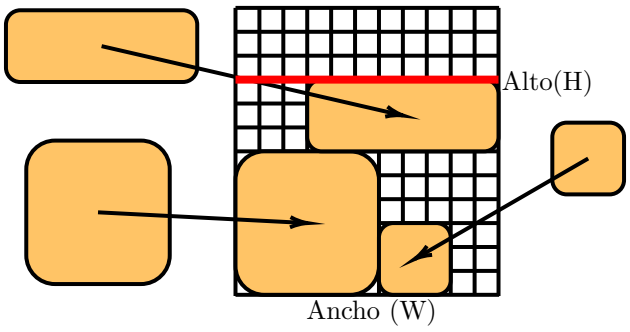


Figura 1: 2DSPP

Two-Dimensional Strip Packing Problem (2DSPP) consiste en encajar(packing) elementos generalmente rectangulares dentro de una tira(strip) de ancho fijo y alto infinito (Figura 1). El objetivo es encontrar la mínima altura a utilizar por los elementos luego de su encaje. Donde

[1] https://github.com/keleron/2DSPP

las variables de elección normalmente son un punto de partida y un punto final, que normalmente representan las esquinas inferior-izquierda y superior-derecha. Otra variable aunque mas indirecta es finalmente la altura H, la cual viene determinada por las anteriores.

El problema está muy relacionado con su análogo, Two-Dimensional Bin Packing (2DBP)[18] Problem, que diferencia del (2DSPP) este tiene una cantidad ilimitada de contenedores(o tiras) y se busca minimizar la cantidad de ellos [8].

Variaciones, análogos y similares comunes

(2DSPP) es un problema muy específico y a lo largo de la historia han surgido muchas variaciones, donde cada una depende de las características elementales del problema, las cuales son Dimensionalidad, Discreción de valores, Forma y Orientación de las figuras y Condiciones Especiales.

La variación mas conocida y frecuentemente estudiada es el Bin Packing Problem (BPP)[18] tal y como fue mencionada antes. (BPP) puede extenderse para $1 \rightarrow N$ dimensiones. Mientras que (SPP) necesita $2 \rightarrow N$ dimensiones, aunque de todas formas un One-Dimensional Bin Packing (1DBP) Problem con objetivos referentes a la altura podría verse como un (1DSPP).

Otros autores [17] se plantean problemas un tanto mas complicados tales como el Two-Dimensional Irregular Strip Packing Problem (2DISPP), el cual a diferencia de (2DSPP) se tienen polígonos irregulares con posibilidad de total rotación en vez de rectángulos de longitudes ajustadas. Otro caso relevante es cuando cada elemento i tiene un beneficio asociado $p_i > 0$, y se busca encontrar el subconjunto de objetos que maximice el beneficio asociado al contenedor, tal problema es conocido como Two-Dimensional Cutting Stock (2DCS) [18] a pesar de ser introducido como Two-Dimensional Cutting Knapsack (2DCK) [12] (1965).

Dimensionalidad

Parte importante de la estructura lógica de los (2DSPP), determina la estructura física de los elementos a almacenar.

- 1-Dimensional: (Objetos unidimensionales) Generalmente (1DBPP) y sus variantes.
- 2-Dimensionales: (Orientado en objetos planos, papel o telas)
- 3-Dimensionales (Figuras volumétricas como cubos): La idea es la misma, pero esta vez los elementos son objetos tridimensionales encajando en un contenedor.
- Multi-Dimensionales (Enfoques mas complejos mas allá de las dimensiones físicas de los elementos) Destacan el problema de la mudanza y de despacho (Traveler Salesman + 3DSPP)(TSP+3DSPP) o problemas que necesitan un orden determinado de extracción[23] posterior a su encaje.

Posicionamiento Discreto o Continuo

Consiste en la idea de que cada elemento y a su vez la tira siguen o no un espectro continuo o discreto tanto en su posicionamiento como en sus dimensiones. [6]

- Discreto : Cada elemento está conformado por dimensiones ajustadas, es decir, que pueden ser escritas como múltiplos de una unidad básica. Este enfoque siempre es aplicable, pero su utilidad radica en poliedros regulares.
- Continuo : Cada elemento, tira o contenedor puede ser conformado por cualquier medida no ajustada dentro del intervalo de los números reales.

El primer enfoque permite una mejor formulación como problema de programación lineal entera, delimitando el espacio de búsqueda a un número finito de regiones factibles, mientras que el segundo enfoque provee un espacio de búsqueda infinito. Además es posible convertir sin perdida de generalidad un modelo real en uno entero multiplicando por el escalar adecuado.

Forma y orientación de las figuras

La forma y orientación delimitan completamente la trata del problema, ya que complican la capacidad de encaje y búsqueda de patrones.

- Regulares: El uso de polígonos o poliedros regulares como objetos a encajar.
- Irregulares [17]: El uso de polígonos y poliedros irregulares, informalmente cualquier forma es permitida dentro de esta categoría.
- Rotaciones [16]: Complementarías y compatibles con los puntos anteriores.
  - Sin rotaciones : Generalmente usada cuando se tiene una tira(strip) con un patrón de diseño en particular y se desea que todos los elementos finales posean la misma orientación en diseño.
  - Rotaciones 90°
  - Rotación libre : Generalmente necesaria para problemas donde las figuras son irregulares.

Requisitos adicionales

Esta sección recopila condiciones o restricciones que normalmente varían el problema.

- Guillotinable [20]: Los cortes deben ser hechos de lado a lado (Edge to Edge).
- Formas repetidas: Se pueden generar patrones de encaje.
- Tiempo de Encaje: No siempre se puede tener la información de cada uno de los elementos, y por ende hay que tomar la mejor decisión para el momento. Estos problemas se llaman Strip Packing On-line Problem, claro ejemplo es el 2D Strip Packing Tetris Problem (2DSPTP) [11], también conocidos como problemas sobre-la-linea.
- Respecto a cortes: Cuando los cortes tienen un costo asociado $C_i > 0$ es preciso darse cuenta que son un factor importante si el objetivo es minimizar gastos. Situaciones en las que la perdida de material se ve opacada por el precio del corte.
- Restricción de orden: Como antes mencionado hay problemas como los de la mudanza o despacho en que se necesita ordenar eficientemente la posición de los elementos dentro de un contenedor para de esa forma desembarcar y entregar a los clientes en la menor cantidad de movimientos. Two-Dimensional Strip Packing Problem with Unloading Constraints (2DSPPUC).[23]

Finalmente el caso de estudio de este documento será el 2D Strip Packing Problem Bi-Dimensional con dimensiones y medidas discretas y elementos rectangulares, rotaciones en 90 y sin ninguna condición especial.

3. Estado del Arte

Métodos no-exactos

Las primeras heurísticas propuestas fueron Bottom-Left (BL) y Bottom-Left-Fill(BLF)[4], ambas heurísticas consisten en encajar(packing) cada elemento con los siguientes movimientos; primero se sitúa un objeto en la esquina superior derecha, se desplaza el objeto lo mas posible hacia abajo y luego lo mas posible hacia la izquierda. La diferencia radica en que (BLF) rellena espacios intermedios que hayan quedado vacíos. Ambas heurísticas pueden o no implementar ordenamiento de los elementos (Por dimensiones, área o algún otro parámetro).

Mas tarde llegó el enfoque por niveles [10](1980) como extensión de (BL) y (BLF), una forma de ver el (2DSPP) como si se tratase de un (2DBPP) con contenedores(o tiras) de alturas que se deciden en tiempo de ejecución según los elementos introducidos. El primero First-fit Decreasing Heigh (FFDH), tiene que cada elemento es encajado(packing) en el primer nivel que se esté disponible. Next-fit Decreasing Height (NFDH) consiste en ir introduciendo elementos hasta que no encaje, cuando ocurra, se cierra el nivel. Finalmente Best Fit decreasing height (BFDH) busca que cada elemento minimice la distancia entre su lado derecho y el borde derecho de la tira(strip), en otras palabras minimiza el desperdicio horizontal. Complementario a las tres propuestas antes mencionadas existe además el Less Flexibility First (LFF), que tal y como su nombre lo indica busca almacenar primero los elementos menos flexibles.

Las anteriores propuestas tienen una característica en común muy ventajosa, y es que su implementación es muy sencilla y eficaz, ya que no necesita computar nada que no sea colisiones entre rectángulos (Llevando a que la complejidad encontrada no tenga grandes constantes escondidas) y además es altamente flexible para variantes del problema. La desventaja claro está en que son métodos menos inteligentes y normalmente tomarán peores decisiones que métodos mas sofisticados.

Cierto estudio [13] recolecta y explica los enfoques anteriormente mencionados incluyendo complejidades y tiempos de ejecución. Además de incluir detalles sobre heurísticas y metaheurísticas tales como Genetic Algorithm (GA), Naive Evolution(NE), Hill Climbing (HC) y Simulated Annealing(SA). Otros autores mas contemporáneos han utilizado heurísticas mas nuevas tales como Fruit Fly Optimization + Bottom-Left (FOA+BL) [3](2017), heurísticas Skyline + Búsqueda Tabú (TS)[22] (2011) y Algoritmos Genéticos (GA) [21](2013).

Finalmente lo más moderno que se encontró en torno a soluciones fue el uso de Hiper-Heurísticas basadas en el uso de grillas (Grid) combinado con algoritmos genéticos. (Evolutionary hyper-heuristic for solving the strip-packing problem)

Contrastando a las primeras propuestas, estás soluciones son mucho mas sofisticadas y requieren de un gran esfuerzo por parte del algoritmo para hacer un buen uso de la información, tal vez resultando en constantes de computo enormes. Pero al contrario de las anteriores estás dan soluciones candidatas considerablemente mas ideales. Así que finalmente habría que llevar un estudio para saber cual usar o no usar en determinadas instancias.

[9]

Métodos exactos

Debido a la complejidad del problema generalmente no se habla de métodos exactos, pero ciertos estudios[5] comparativos hacen buena mención a algoritmos basados en Branch and Bound (B&B), Dichotomous Search (DS), Column Generation (CG) y Branch and Price(B&P), además de especificar el calculo para cotas inferiores y superiores. Por otro lado, existen autores

que resuelven el problema enfocado a niveles Multi-Level Branch and Bound[2].

Los métodos exactos no son muy populares, pocos autores evidencian su desarrollo y únicamente trabajan en soluciones parciales. Los métodos exactos pueden generar buenas soluciones parciales, pero casi inmediatamente quedan descartados por sus altos tiempos de computo; igualmente existe la posibilidad que un método exacto pueda ser eficiente si una instancia en particular tiene las mejores condiciones. Sin mencionar que estos métodos encuentran el óptimo.

Menciones especiales

Otros autores han preferido gastar sus esfuerzos en investigar el meta-problema y han propuesto modelos[14] basados en minería de datos con Machine Learning (ML) para alcanzar el siguiente nivel de sistemas expertos a fin de evaluar la calidad de la solución para determinar la heurística indicada para cada problema.

Finalmente Random Search (RS), una forma aleatoria de buscar soluciones, recurrentemente usada únicamente con propósitos comparativos.

4. Modelo Matemático

Se parte con una tira (strip) rectangular de ancho W y altura infinita, estableciendo la esquina inferior izquierda como el punto $(0, 0)$ del plano. Y un conjunto de n rectángulos $R = R_1, R_2, ..., R_n$, donde cada $R_i (1 \leq i \leq n)$ es una pieza rectangular de altura h_i y ancho w_i. El objetivo de (2DSPP) es minimizar la altura H luego de encajar cada pieza R_i dentro de la tira(strip), tal que se cumplan las siguientes condiciones:

- Todos los rectángulos son paralelos a los ejes de la tira(strip)
- Ningún rectángulo traslapa a otro
- Un rectángulo puede únicamente rotar en 90°

A continuación se mostrará el siguiente modelo [7] de función objetivo mas restricciones que interpreta las condiciones antes mencionadas.

Variables de Decisión

$$\begin{gathered} x_{li} = \text{Componente X de la esquina inferior-izquierda del elemento i} \\ x_{ri} = \text{Componente X de la esquina superior-derecha del elemento i} \\ y_{li} = \text{Componente Y de la esquina inferior-izquierda del elemento i} \\ y_{ri} = \text{Componente Y de la esquina superior-derecha del elemento i} \\ x_{li}, x_{ri}, y_{li}, y_{ri} \geq 0 \in \mathbb{Z} \end{gathered} \tag{1}$$

Función Objetivo y Restricciones

$$Minimize : H = max(y_{ri}, i = 1, 2, .., n) \tag{2}$$

$$\begin{gathered} 0 \leq x_{li} \leq x_{ri} \leq W \\ 0 \leq y_{li} \leq y_{ri} \leq H \end{gathered} \tag{3}$$

$$max(x_{li} - x_{rj}, x_{lj} - x_{ri}, y_{li} - y_{rj}, y_{lj} - y_{ri} \geq 0) \forall i, j; i \neq j \tag{4}$$

$$
\begin{gathered}
(x_{ri} - x_{li} = w_i \text{ and } y_{ri} - y_{li} = h_i) \\
OR \\
(x_{ri} - x_{li} = h_i \text{ and } y_{ri} - y_{li} = w_i)
\end{gathered} \tag{5}
$$

La ecuación (1) representa que lo que se busca determinar o decidir son los puntos de cada rectángulo R_i correspondientes a la esquina inferior izquierda y esquina superior derecha.

La ecuación (2) muestra la función objetivo en cuestión, la cual busca minimizar la la altura H, tomando como valor el máximo valor y_{ri} que corresponde a la mayor punto superior-derecho en su componente y.

La ecuación (3) se encarga que cada punto esquina inferior izquierda (Bottom-Left-Corner) sea inferior a la esquina superior derecha (Top-Right-Corner) y además que ambos estén dentro de las dimensiones del contenedor(o tira).

La ecuación (4) evita que los rectángulos puedan traslapar los unos a los otros.

La ecuación (5) se encarga que los rectángulos únicamente puedan tener una orientación.

Debido a que se trata de un problema de programación lineal entera el espacio de búsqueda queda definido como : $(W \cdot H)^{2n}$

5. Representación

La representación del problema consiste generalmente en la definición de una clase `SOLUTION` la cual sus atributos mas importantes son una variable que almacena el máximo `HEIGTH` y un `ARRAY/VECTOR` de elementos de la clase `RECTANGLE`.

La clase `RECTANGLE` particularmente está conformada por los atributos por `W` y `H` que representan las dimensiones de la caja, además tiene dos instancias de tipo `POINT`, para los puntos superior-izquierdo e inferior-derecho respectivamente, y finalmente un atributo `BOOL` para identificar la rotación de la pieza.

Donde la clase `POINT` únicamente tiene asociados los atributos `X` e `Y`

Los dominios involucrados en la solución de este problema (al ser una solución completa), corresponden a $height \in \mathbb{N}$ por un lado, y por otro para cada rectángulo las coordenadas de los puntos superior-izquierdo e inferior-derecho pueden tomar valores entre $[0, W]$ y $[0, H]$ respectivamente. Donde en un comienzo $H \rightarrow$ ínf.

6. Descripción del Algoritmo

Pseudo-Código

```
MAIN()
    HEIGHT = INF
    LEVEL = 0
    BACK(LEVEL)
    RECTANGLES = LIST[TODOS LOS RECTANGULOS]

DEF BACK(LEVEL):
    RECT = RECTANGLES[LEVEL]
    CONFLICTOS = SET()
    IF LEVEL==N (POST ULTIMO HIJO):
        IF MEJOR:
            GUARDAR_MEJOR
            UPDATE HEIGHT
        RETURN
```

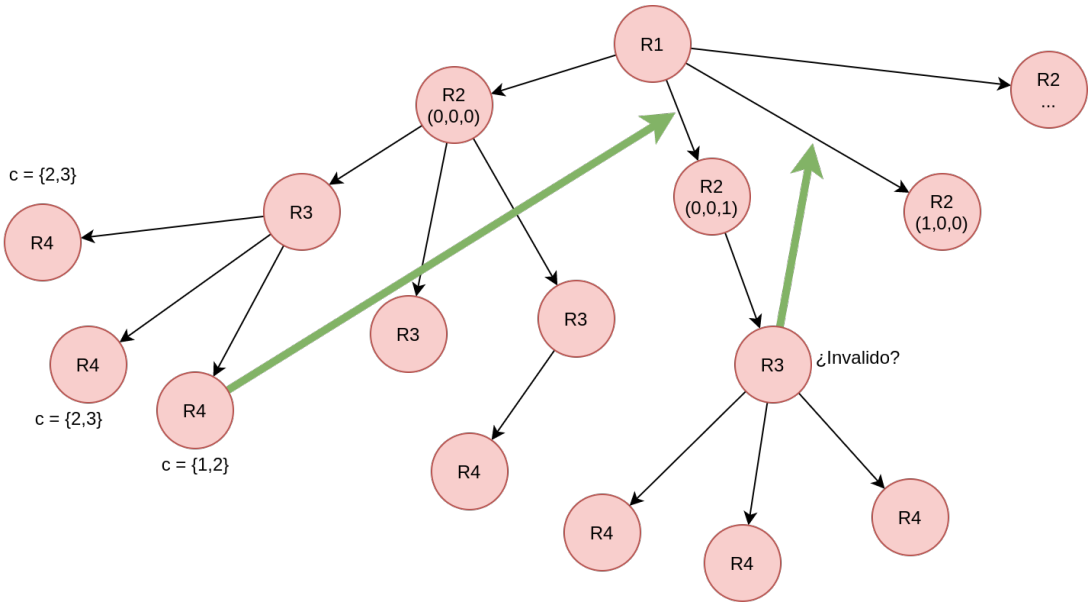


Figura 2: Backtracking + CBJ example

```
    ELSE:
        X,Y = 0,0
        FOR X IN [0,W]:
            FOR Y IN [0,H]:
                IF ¿(X,Y) ESTA LIBRE?:
                    FOR ROT IN [TRUE, FALSE]: //ROTACION
                        VALID, CONFLICT = CHECK(X,Y,ROT,RECT)
                        IF VALID:
                            JUMP = BACK(LEVEL+1)
                            IF JUMP!=LEVEL:
                                RETURN JUMP
                        ELSE:
                            CONFLICTOS.ADD(CONFLICT)
    IF CONFLICTOS!=EMPTY:
        RETURN MAX(CONFLICTOS)
    RETURN (LEVEL-1)

DEF CHECK(X,Y,ROT,RECT):
    CHECKEAR SI EN EL ESTADO ACTUAL DE LA SOLUCION
    INSTANCIAR RECT EN X,Y CON ROTACION ROT GENERA CONFLICTOS CON
    OTROS RECTANGULOS, CON LOS BORDES, O SI BIEN SE ESCAPA DEL ACTUAL HEIGHT

    VALID => TRUE : SI ES POSIBLE INSTANCIARLO
          => FALSE: SI NO ES POSIBLE

    CONFLICTOS => LA PIEZA MAS ANTIGUAMENTE INSTANCIADA
    RETURN (VALID, CONFLICTOS)
```

Mas detalles

El algoritmo utiliza una técnica de exploración completa conocida como BT + CBJ (Backtracking Guiado por Conflictos), técnica la cual será explicada usando como referencia el pseudo código y la imagen 2. En palabras simples el pseudocódigo puede ser explicado como que en un comienzo existe una lista de rectángulos $R[]$, de la cual cada vez se elige un rectángulo $R[i]$, donde i representa el nivel actual del árbol. Luego para tal $R[i]$ se busca la siguiente posición disponible o libre, y se **intenta** insertar en tales coordenadas (con ambas rotaciones), creando así la tupla de instanciación (x, y, rot) que podemos ver en la imagen 2.

En caso de que fallase la instanciación, la instancia retornará el conflicto mas prematuramente instanciado (El de nivel menor) y continuará con la siguiente tupla posible para $R[i]$. Donde como se muestra en la imagen 2 la primera instanciación de $R2$ es $(0, 0, 0)$, luego $(0, 0, 1)$ y luego $(1, 0, 0)$, todas estas instanciaciones son hermanas e hijas de $R1$

Dichos conflictos serán agregados a un conjunto del nodo padre que compartan, y cuando exista un fallo del nivel (que todos los hijos hayan fallado en instanciarse), el padre elegirá el máximo valor de su conjunto de conflictos y saltará hasta esa instanciación; descartando así cualquier nodo intermedio. Anteriormente mencionamos las características básicas que debe tener una implementación de BT+CBJ para este problema, pero además se desarrollaron otras optimizaciones que vale la pena mencionar y que afectan positivamente al desempeño del algoritmo.

- Antes de comenzar a ejecutar BT+CBJ, se prefirió usar una heurística para la primera instanciación que consistía en ordenar la lista de Rectángulos basados en su área. Ya que de esta forma la cota para HEIGTH desde un comienzo es baja, y permite hacer podados inteligentes desde un comienzo.
- Además, cada vez que se logra una instanciación completa, la recesión de BT vuelve al objeto mas prematuramente instanciado que haya alcanzado la nueva altura candidata al óptimo, ya que tal objeto al ser el nuevo limite, siempre debe ser desplazado a una posición inferior o bien rotado.
- Se implementaron además de las condiciones de conflicto sobre colisión de rectángulos, se implementaron otras condiciones de quiebre en momentos donde una pieza simplemente puede descartarte por la estructura de la tira o bien bordes.
- También se implemento un salto inteligente en colisión, ya que el dominio para cada punto era $x \in [0, W]$ y $y \in [0, H]$, e ir incrementando en 1 los índices no era inteligente. Por ende se implementó una método que usando las dimensiones de la caja podía saltar inteligentemente

7. Experimentos

Debido a que la solución implementada es una solución completa se utilizaron instancias medianamente pequeñas (las cuales pueden ser encontrados en el repositorio del proyecto [2]), además como existen instancias en las cuales el tiempo requerido para recorrer todo el espacio de búsqueda es realmente inalcanzable, el algoritmo provee la opción de establecer un tiempo limite de computo.

Por lo tanto los experimentos para cada instancia serán ejecutados bajo el mismo tiempo limite, y cada vez que se encuentre una solución candidata mejor se guardarán las métricas comparativas de **altura**, **espacio inutilizado** y **porcentaje % relativo de espacio inutilizado**

[2] https://github.com/keleron/2DSPP

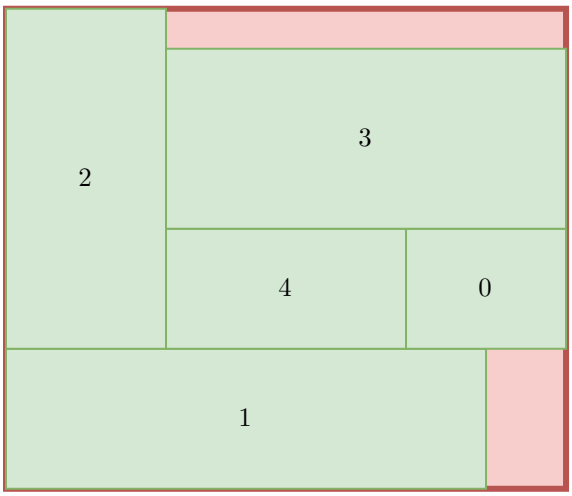


Figura 3: Ejemplo de Experimento

Como se ve en la figura 3 los datos mas relevantes a computar y analizar vendrían siendo la altura de la tira, y el porcentaje que la superficie roja ocupa sobre toda la tira de material. Para la experimentación se utilizarán las instancias del documento [19], el cual contiene una variedad interesante de dimensiones.

8. Resultados

NOMBRE	N	W	INUTILIZADO %	ITERACIONES	ALTURA	Z[19]
BENG01.TXT.log	20	25	4.39 %	42	31	30
BENG02.TXT.log	40	25	5.33 %	19549	60	58
BENG03.TXT.log	60	25	2.79 %	116	86	85
BENG04.TXT.log	80	25	1.00 %	30602	108	108
BENG05.TXT.log	100	25	2.06 %	239	136	134
BENG06.TXT.log	40	40	4.05 %	97	37	37
BENG07.TXT.log	80	40	1.73 %	137	68	67
BENG08.TXT.log	120	40	2.26 %	747	103	101
BENG09.TXT.log	160	40	2.19 %	272	128	126
BENG10.TXT.log	200	40	1.63 %	378	158	156
CGCUT01.TXT.log	16	10	2.17 %	8338	23	23
CGCUT02.TXT.log	23	70	12.60 %	14816	71	67
CGCUT03.TXT.log	62	70	5.12 %	82	670	670
GCUT01.TXT.log	10	250	11.95 %	177	743	1016
GCUT02.TXT.log	20	250	11.86 %	21	1246	1208
GCUT03.TXT.log	30	250	10.94 %	31	1831	1803
GCUT04.TXT.log	50	250	5.93 %	67	3110	3077
HT01.TXT.log	16	20	0.00 %	100539	20	20
HT02.TXT.log	17	20	4.76 %	38	21	20
HT03.TXT.log	16	20	0.00 %	101911	20	20
HT04.TXT.log	25	40	6.25 %	75	16	15
HT05.TXT.log	25	40	6.25 %	46	16	15
HT06.TXT.log	25	40	0.00 %	306171	15	15
HT07.TXT.log	28	60	3.23 %	205	31	31
HT08.TXT.log	29	60	3.23 %	161	31	31
HT09.TXT.log	28	60	3.23 %	608	31	30
NGCUT01.TXT.log	10	10	5.00 %	3015	20	23
NGCUT02.TXT.log	17	10	7.67 %	81	30	30
NGCUT03.TXT.log	21	10	1.07 %	85	28	28
NGCUT04.TXT.log	7	10	10.00 %	1385	18	20
NGCUT05.TXT.log	14	10	7.11 %	290773	38	36
NGCUT06.TXT.log	15	10	0.00 %	29709	29	31
NGCUT07.TXT.log	8	20	12.50 %	18	10	20
NGCUT08.TXT.log	13	20	6.91 %	927	34	33
NGCUT09.TXT.log	18	20	6.35 %	122978	52	50
NGCUT10.TXT.log	13	30	6.01 %	277	61	80
NGCUT11.TXT.log	15	30	11.73 %	36824	56	52
NGCUT12.TXT.log	22	30	3.12 %	42427	79	87

Cuadro 1: Resultados obtenidos para un limite de tiempo de 30[s]

Formato, Condiciones y Entorno

Los resultados preliminares se muestran en la tabla 1, donde las columnas mas importante son INUTILIZADO %, ITERACIONES, ALTURA y Z; las cuales representan, respectivamente, el porcentaje inutilizado con respecto a la altura actual, la cantidad de iteraciones hasta

encontrar la ultima iteración, la altura encontrada en la ultima solución, y el valor del óptimo encontrado para el algoritmo de búsqueda completa del documento [19].

Es importante mencionar que nuestra definición de Ïteración", corresponde estéticamente a la cantidad de veces que el algoritmo ha visitado un nodo. O en términos mas técnicos con respecto al desarrollo del algoritmo, las iteraciones consisten en la cantidad de veces que se hace una asignación factible dentro del espacio.

Es necesario destacar nuevamente que debido a que es una técnica completa muchas de las instancias a tratar simplemente no eran capaces de terminar; ya que la complejidad del problema hacía que la ejecución del algoritmo fuese eterna. Únicamente para la tabla 1 se utilizó un tiempo limite de 30[s], ya que para todos los archivos 30[s] estaba muy cercano al umbral de estancarse en búsqueda de soluciones. De todas formas la implementación es para soluciones exactas y completas, pero para casos prácticos se tratará con un tiempo limite.

Finalmente el Hardware utilizado para correr las pruebas fue un Notebook i5-2450M, GT630M, 8GB RAM 1333[MHZ], mientras que el Software fue una implementación de código de C++(gcc 8.3.0) en entorno Ubuntu 19.04.

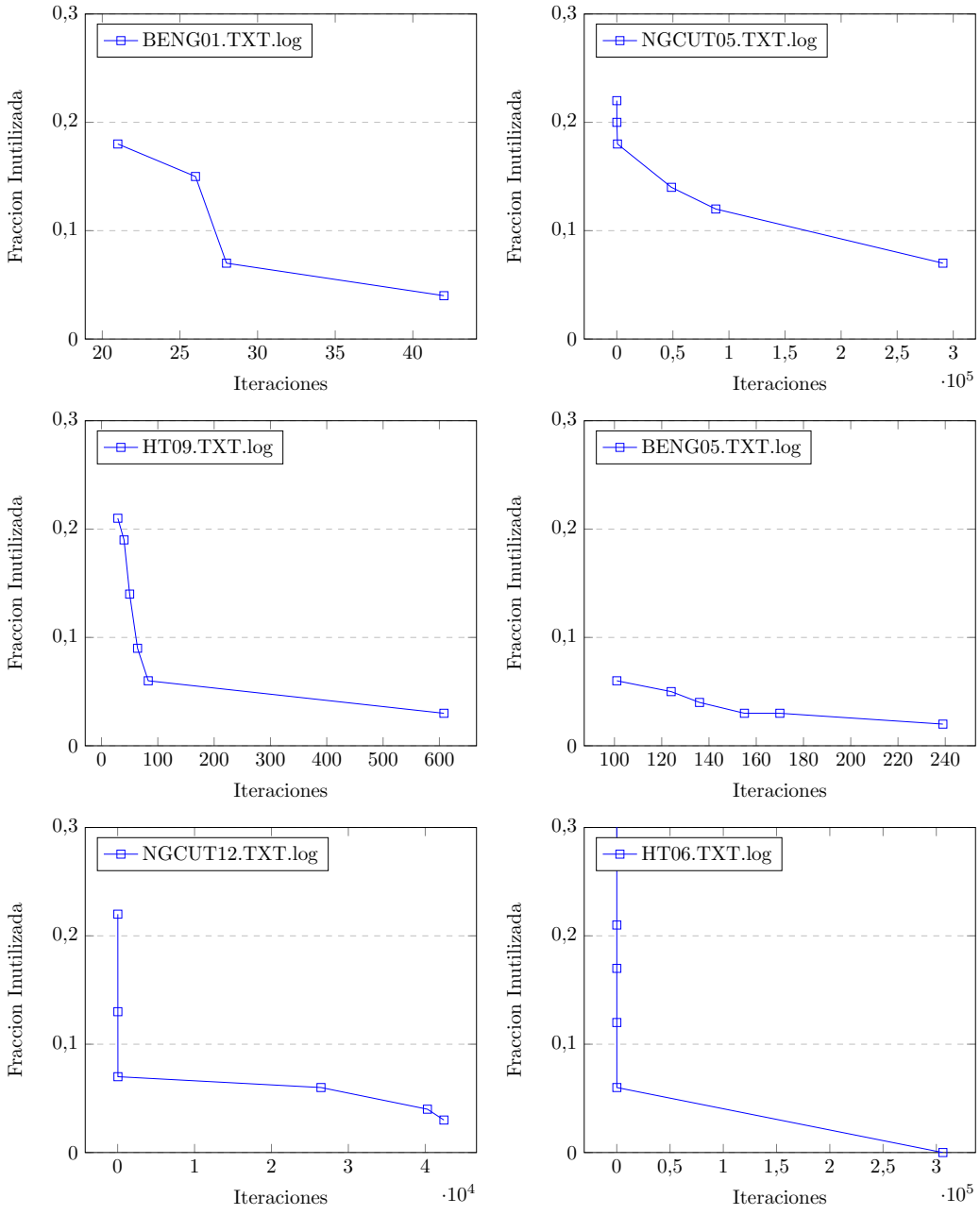
Análisis

En primera instancia, observando la tabla 1 podemos ver que las columnas ALTURA y Z son realmente similares, por lo que al menos superficialmente podemos afirmar que nuestra implementación obtuvo en general un buen rendimiento. Alcanzando incluso en instancias particulares (CGCUT03.TXT.log por ejemplo) mejores valores que los entregados por el documento [19]. Ahora si vemos las columnas INUTILIZADO % nos damos cuenta que las instancias para las cuales se alcanzó un mayor porcentaje generalmente poseen un valor de W generalmente alto, y un valor de N de todo tipo; de esa forma es pertinente concluir que el desempeño del algoritmo se ve mas perjudicado por el valor de W que de N.

Ahora con respecto a la columna ITERACIONES, podemos ver que hay valores pequeños o muy grandes; la explicación de este comportamiento yace en la naturaleza del algoritmo y descarte inteligente. En palabras simples, todos los valores de ITERACIONES bajos corresponden a instancias que fueron solucionadas rápidamente antes de los 30[s] y todo su espacio fue explorado. Tal vez pueda causar dudas como instancias tan grandes puedan terminar tempranamente, pero la explicación es muy sencilla, nuestra implementación calcula una cota mínima y cuando es alcanzada no es necesario continuar. Además a mayor cantidad de elementos, mayor es la probabilidad de que exista uno que rellene los espacios vacíos.

En otros ámbitos el cuadro 2 muestra seis instancias distintas a lo largo del numero de iteraciones; donde cada recuadrado dentro del gráfico representa una instanciación completa encontrada como un punto (iter, % normalizado); es notable darse cuenta que todas las instancias tienen un brusco decaimiento en las primeras iteraciones y luego se estacan por varios ordenes de magnitud en iteraciones para mejorar en una proporción realmente baja. Por ejemplo, mirando el gráfico "HT06.TXT.log"podemos darnos cuenta que en el primer orden de magnitud alcanzó gran parte de la solución y tuvieron que pasar $5 \cdot 10^5$ iteraciones para la siguiente y ultima mejora de la solución. Por otro lado los gráficos "BENG01.TXT.logz "BENG05.TXT.log"no parecen tener ese comportamiento únicamente porque son instancias mas pequeñas.

No se puede esperar que este comportamiento siempre se repita, pero claramente el algoritmo tiene una tendencia por estos resultados; donde claramente la heurística de instanciación inicial se lleva la mayor parte del crédito en acercarse a la solución.



Cuadro 2: Gráficos Iteraciones vs Fracción Inutilizada

9. Conclusiones

Como se ha visto, (2DSPP) es un problema medianamente antiguo, con muchas variantes, muchas propuestas y muchas formas de abordarlo. Se apreció que es un problema realmente importante en industrias textiles y papeleras, donde un pequeño porcentaje de desperdicio puede significan una gran perdida de dinero. Es pertinente decir que las técnicas basadas en algoritmos genéticos y en particular Fruit Fly Optimization Algorithm (FOA) muestran los mejores resultados, tanto en tiempo como en densidad de solución.

Finalmente se puede decir que la propuesta mas prometedora viene de el uso de Maching Learning[14] para construir la evolución de un sistema experto, dicha técnica consiste en utilizar un Framework de Data Mining capaz de evaluar la calidad de una heurística utilizando modelos de regresión comparativos entre distintas técnicas en contraste con predictores (que adivinan). Como se menciona dentro del documento, hay casi 100 años de historia en las industrias papeleras y textiles, y a lo largo de ese tiempo se han generado miles y miles de Datasets sobre cortes y elementos[15]. Y como estamos en la época del Big Data y con elevados tiempos de computo, parecería que el futuro de la resolución de problemas combinatorios apunta al desarrollo, estudio y aplicaciones del Maching Learning.

Con respecto a la solución implementada podemos concluir que en general hace un buen trabajo, es eficiente para encontrar buenas soluciones en poco tiempo, pero normalmente esto es gracias a la heurística aplicada en un comienzo para tener una buena cota superior desde los primeros pasos del algoritmo. Por desgracia, el propósito de este documento era desarrollar una técnica completa, y a pesar de que se hizo, difícilmente pudo ser llevado a la práctica, ya que la mayoría de las instancias eran lo suficientemente grandes como para que nuestra implementación fuese incapaz de terminar en un tiempo considerable.

Bibliografía

Referencias

[1] Musrrat Ali, Chang Wook Ahn, and Millie Pant. Trim loss optimization by an improved differential evolution. *Mathematical Problems in Engineering*, 2013:1–8, 2013.

[2] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit. A branch and bound algorithm for the strip packing problem. *OR Spectrum*, 31(2):431–459, March 2008.

[3] İsmail Babaoğlu. Solving 2d strip packing problem using fruit fly optimization algorithm. *Procedia Computer Science*, 111:52–57, 2017.

[4] Brenda S. Baker, Jr. E. G. Coffman, and Ronald L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, November 1980.

[5] Abdelghani Bekrar, Imed Kacem, and CH CHU. A comparative study of exact algorithms for the two dimensional strip packing problem. 2007.

[6] Andreas Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172(3):814–837, 2006.

[7] Jianli Chen, Wenxing Zhu, and Zheng Peng. A heuristic algorithm for the strip packing problem. *Journal of Heuristics*, 18(4):677–697, 2012.

[8] Mao Chen and Wenqi Huang. A two-level search algorithm for 2d rectangular packing problem. *Computers & Industrial Engineering*, 53(1):123–136, August 2007.

[9] Daniel Domović, Tomislav Rolich, and Marin Golub. Evolutionary hyper-heuristic for solving the strip-packing problem. *The Journal of The Textile Institute*, pages 1–11, 2019.

[10] Jr. E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, November 1980.

[11] Sándor P Fekete, Tom Kamphans, and Nils Schweer. Online square packing. In *Workshop on Algorithms and Data Structures*, pages 302–314. Springer, 2009.

[12] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, February 1965.

[13] EBCH Hopper and Brian CH 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.

[14] Alvaro Neuenfeldt Júnior, Elsa Silva, A Miguel Gomes, Carlos Soares, and José Fernando Oliveira. Data mining based framework to assess solution quality for the rectangular 2d strip-packing problem. *Expert Systems with Applications*, 118:365–380, 2019.

[15] Alvaro Neuenfeldt Júnior, Elsa Silva, A. Miguel Gomes, Carlos Soares, and José Fernando Oliveira. Data mining based framework to assess solution quality for the rectangular 2d strip-packing problem. *Expert Systems with Applications*, 118:365–380, March 2019.

[16] Mitsutoshi Kenmochi, Takashi Imamichi, Koji Nonobe, Mutsunori Yagiura, and Hiroshi Nagamochi. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, 198(1):73–83, 2009.

[17] Stephen C.H. Leung, Yangbin Lin, and Defu Zhang. Extended local search algorithm based on nonlinear programming for two-dimensional irregular strip packing problem. *Computers & Operations Research*, 39(3):678–686, March 2012.

[18] Andrea Lodi, Silvano Martello, and Daniele Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, November 2002.

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

[20] 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, May 2009.

[21] Jaya Thomas and Narendra S. Chaudhari. Hybrid approach for 2d strip packing problem using genetic algorithm. In *Advances in Computational Intelligence*, pages 566–574. Springer Berlin Heidelberg, 2013.

[22] Lijun Wei, Wee-Chong Oon, Wenbin Zhu, and Andrew Lim. A skyline heuristic for the 2d rectangular packing and strip packing problems. *European Journal of Operational Research*, June 2011.

[23] Lijun Wei, Yongsheng Wang, Huibing Cheng, and Jian Huang. An open space based heuristic for the 2d strip packing problem with unloading constraints. *Applied Mathematical Modelling*, 70:67–81, 2019.