Inteligencia Artificial

Irregular Strip Packing Problem

Matías Valenzuela Ibarra

Resumen

El Irregular Strip Packing Problem consiste en ubicar un conjunto de polígonos irregulares dentro de un contenedor con ancho fijo, de tal forma que su largo sea lo menor posible. Este problema es de alto interés en industrias como la textil y metalúrgica, pero puede aplicarse en otras áreas de conocimiento/producción. En el presente documento se realizará su estado del arte, abarcando tanto su origen como los diversos algoritmos y heurísticas que se han desarrollado para encontrarle soluciones en tiempos de cómputo razonables, junto con los modelos matemáticos que tienen asociados. Además, se presentará un algoritmo basado en *Hill Climbing* para resolverlo y se discutirán los resultados obtenidos. Se finalizará con algunas conclusiones y propuestas de trabajo futuro.

1. Introducción

Irregular Strip Packing Problem (ISPP) pertenece a la clase más general de problemas de optimización combinatoria: los problemas de corte y empaque (Cutting & Packing Problems) [8]. Específicamente, los problemas de corte se caracterizan por objetos grandes (por ejemplo hojas o rollos) que deben cortarse en elementos pequeños (figuras en 2D). Los objetos residuales que se producen en el patrón y que no pertenecen a la lista de pedidos se denominan *pérdida de recorte*. El objetivo de la mayoría de los problemas C&P es minimizar la pérdida o desperdicio de las molduras [8]. Otra forma de describir el mismo objetivo es minimizar el largo de un contenedor de ancho fijo, el cual contiene un conjunto de figuras. En este caso en particular, la motivación es acomodar un conjunto de polígonos irregulares de tal forma que el contenedor (de ancho fijo) tenga el menor largo posible y que genere la menor cantidad posible de desperdicios.

La estructura del presente documento es la siguiente: en la Sección 2 se explicará el ISPP de forma general, junto con la descripción de algunas de sus variantes. En la Sección 3 se hará un recorrido por el estado del arte, estudiando tanto los orígenes de la definición del problema como los primeros métodos diseñados para darle solución. Se revisará una gama de técnicas, cada una de las cuales ha significado avances en el objetivo de computar soluciones en un tiempo bajo. En la Sección 4 se revisarán algunos modelos matemáticos que se han planteado para definir formalmente el problema. Desde la Sección 5 a la Sección 8 se presentará el algoritmo desarrollado para resolver el problema, los experimentos realizados y los resultados obtenidos. En la Sección 9 se presentarán las conclusiones obtenidas del trabajo realizado.

2. Definición del Problema

Dados n polígonos y un contenedor rectangular con ancho constante W y largo variable H, este problema requiere encontrar una ubicación factible de los polígonos dados en el contenedor

utilizando el menor largo posible. Un patrón se dice *factible* si ningún polígono se superpone con ningún otro, o sobresale del contenedor. Tiene tres variaciones con respecto a las rotaciones de los polígonos: (1) se permiten rotaciones de cualquier ángulo, (2) se permite un número finito de ángulos y (3) no se permite la rotación [9]. A diferencia de los problemas de *packing* en donde los polígonos son rectángulos, el manejo de intersecciones entre polígonos irregulares es considerablemente más complejo. Algunos algoritmos heurísticos usan la aproximación de las formas dadas, mientras que muchos de los algoritmos recientes usan una técnica geométrica llamada polígonos *No-Fit* [9].

Gran parte de las variantes de este problema son del tipo NP-Duro, dado que contienen un caso particular de Bin Packing Problem, el cual es un famoso problema NP-Completo [9].

Clasificación

El objetivo de un problema de *packing* es la asignación eficiente de polígonos en un contenedor, sin que éstos se superpongan. Por lo tanto, su complejidad está fuertemente relacionada con la forma geométrica de los artículos a empacar. En cuanto a la geometría, se pueden distinguir dos tipos de formas: *formas regulares* (por ejemplo, rectángulos y círculos) y *formas irregulares*, incluidas asimetrías y concavidades [8].

En *packing* regular, los siguientes tipos de diseño se pueden distinguir en función de la geometría de los artículos que se van a embalar (Hinxman 1980) [8]. En el caso de artículos regulares, los patrones de empaque pueden ser *ortogonales*, describiendo cortes solo paralelos a los lados del contenedor y *no ortogonales*. El corte ortogonal distingue adicionalmente entre diseños *guillotinables* y *no guillotinables* [8].

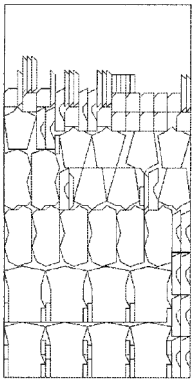


Figura 1: *Packing* irregular. Fuente: [8]

Los problemas guillotineables incluyen una restricción, que prescribe que el diseño debe ser procesado por una serie de cortes rectos en toda la longitud del objeto restante. Este tipo de problema de corte ocurre, por ejemplo, en la industria del vidrio y el papel. Como los problemas no guillotinables no están restringidos por esta regla, un elemento se puede colocar en cualquier posición disponible, lo que da como resultado un diseño no superpuesto.

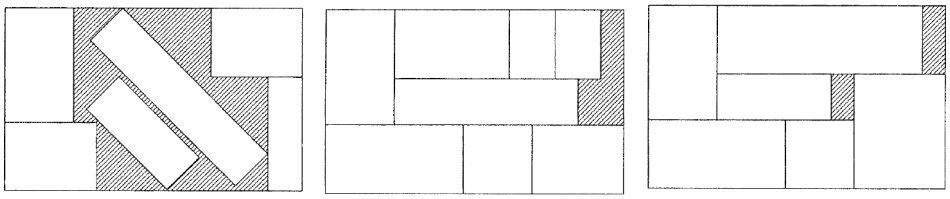


Figura 2: Patrón no ortogonal, guillotinable y no guillotinable. Fuente: [8]

También se pueden clasificar según la rotación de las piezas. Se dice que se resuelve el *Strip Packing* orientado si no se permite rotar las piezas para situarlas dentro del *strip*. Si es que se permite rotar un *item* para ordenarlo dentro del *strip* se dice que se resuelve el problema de *Strip Packing* con rotación.

Por último, si se tienen todas los *items a priori*, se tiene un problema de *Strip Packing Offline*, contrario al *Strip Packing Online* donde los *items* se presentan a medida que se van posicionando en el *strip*. La mayoría de los métodos presentes en la literatura se enfocan en resolver la variante *Offline*.

3. Estado del Arte

Origen del problema

El problema fue propuesto en el año 1966 por Richard Art [2], el cual era conocido en la industria de la ropa como *Pattern Marking*. De forma general, lo describe de la siguiente forma:

> "el problema general es organizar varias piezas que constituyen varias prendas completas de tal manera que consuman una cantidad mínima de material de un ancho dado."
>
> Richard Art. Fuente: [2]

Su propósito era encontrar un método para resolver el problema que fuese comercialmente competitivo (no buscaba el óptimo). Dentro de la definición del problema, Art especifica las condiciones que deben cumplir las piezas:

- Cada pieza debe estar completamente dentro del ancho del material y no debe solaparse con ninguna otra pieza.
- Todas las piezas deben estar entre dos líneas paralelas dibujadas perpendiculares a la *grain line* del marcador.
- Debido a una diferencia en el acabado de los dos lados del material, las piezas deben colocarse boca arriba.

El autor menciona que es un caso particular del problema de ubicar un conjunto de figuras en un plano de tal forma de minimizar el área circunscrita utilizada. Además, agrega que una solución computacional podría ser de gran ayuda para superar la deficiencia de humanos que realizaran dicho trabajo profesionalmente. También menciona que los métodos clásicos para resolver el Knapsack Problem no son suficientes para tratar las formas complejas características de este problema.

Representación de las piezas

En [2] se menciona que cada una de las piezas puede ser representada como un conjunto ordenado de segmentos de recta que se aproximan a la precisión deseada. Dicho conjunto de puntos tiene 4 características importantes que son utilizadas en el método propuesto por Art:

1. La *grain line* del material se posiciona de tal forma que quede paralela al eje x. Por lo tanto, el ancho del material es paralelo al eje y).
2. La pieza está hecha para que quepa completamente en el primer cuadrante, por lo que no hay valores negativos para x ni para y.
3. Hay al menos un punto en el conjunto cuya coordenada x es igual a 0 (lo mismo aplica para el eje y).
4. El conjunto de puntos comienza y termina con el punto con el mayor valor de la coordenada y para el cual el valor x es cero. Dicho punto permite enumerar los puntos en el perímetro en sentido anti-horario. El punto de referencia para cada pieza es este primer y último punto introducido de forma redundante.

El método propuesto en [2] utiliza la representación convexa de las piezas debido a que el número de puntos requeridos para representar una pieza es algo menos de la mitad en el caso promedio, y porque el tiempo de cálculo requerido para una solución con las piezas convexas es inferior a una décima parte de lo que se requeriría para una solución con las piezas "completas" [2].

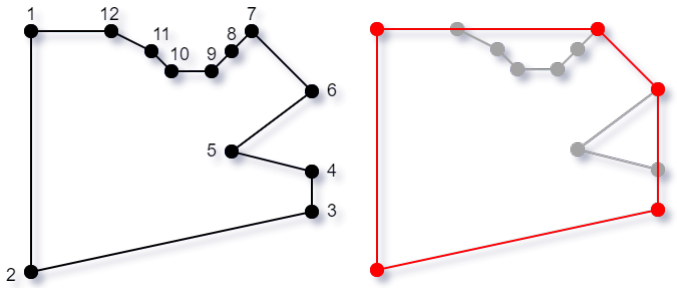


Figura 3: Ejemplo de pieza y su respectiva representación convexa. Fuente: Elaboración propia

Ubicación de las piezas

El autor define para cada una de las piezas un lugar geométrico llamado *envoltura* (*envelope*). Dicho lugar es el perímetro válido dentro del cual el punto de referencia puede caer. Se representa como un conjunto ordenado de pares (x, y) que son los puntos finales de los segmentos de línea que lo comprenden. En la figura 4 se puede visualizar dicho perímetro, en donde la línea negra corresponde al contorno del material.

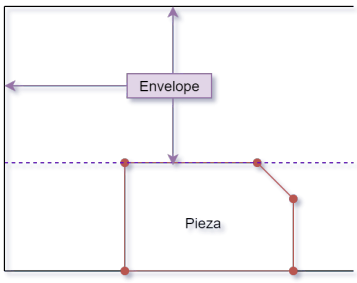


Figura 4: Ejemplo de envoltura. Fuente: Elaboración propia basada en [2]

Técnica de solución

En [2], Art destaca que las piezas pueden ubicarse en infinitas posiciones, por lo que realiza ciertos supuestos para acotar dicha cantidad. Los supuestos realizados se listan a continuación:

- Cada pieza se colocará de modo que esté en contacto con el borde del material o con una pieza colocada previamente.
- Se desestiman aquellas soluciones en las que se define un orden predeterminado de colocación de las piezas (por ejemplo, ubicar las piezas de la más grande a la más chica).
- Se escoge arbitrariamente el inicio del marcador en el borde izquierdo. Una pieza debe tocar dicho borde (en la práctica varias piezas lo hacen). Una ubicación rechazada es aquella en la que la pieza toca el borde superior (o inferior) de una pieza, pero no el borde izquierdo de alguna de ellas.
- Las primeras piezas colocadas deben tocar tanto el borde izquierdo como el borde superior o inferior. La parte superior e inferior son puntos de partida equivalentes en el mismo sentido que los lados derecho e izquierdo. Se elige el borde inferior de forma arbitraria.

Art define el *desperdicio de material* como la suma del área entre las piezas. Si se desea minimizar el desperdicio general, el autor plantea que es razonable intentar minimizar el desperdicio entre

piezas individuales a medida que se colocan. Para ello, realiza analogías con un rompecabezas para clasificar una solución como "buena" [2]:

- Una pieza con dos bordes rectos es identificada como una esquina. En todos los casos hay algunas piezas con bordes rectos paralelos a la *grain line*.
- Algunas de estas piezas tendrán esquinas en ángulo recto. En la mayoría de los casos, corresponden a las piezas más grandes.
- Hay combinaciones de piezas o "meta-piezas" que se consideran eficientes cuando están juntas porque tienen bordes muy coincidentes. No obstante, no hay garantía de que las meta-piezas sean más eficientes en términos de la solución general. Si se utilizan, se debe considerar: 1) el desperdicio probable de la meta-pieza (es decir, el desperdicio probable de cuando se coloca, más el desperdicio inevitable implícito en la formación de la meta-pieza); 2) el desperdicio probable de las piezas que componen la meta pieza (es decir, la suma del desperdicio en esta ubicación particular y el probable desperdicio de las otras piezas cuando se colocan), y; 3) el desperdicio involucrado de ubicar la meta-pieza en la siguiente mejor alternativa sin romperla.

Una ubicación que parece eficiente al principio puede ser bastante ineficiente si se considera el área que ha sido excluida de un uso posterior. Esta área puede ser más grande que la pieza misma debido a la geometría de la solución [2]. También menciona que se deben tener consideraciones adicionales. Citando textualmente a Art: *"podemos asegurar que el lado izquierdo del marcador es bastante eficiente, pero el número reducido de opciones a medida que uno se mueve hacia la derecha nos da una solución progresivamente menos eficiente"* [2]. Es por esto que a medida que se construye la solución se debe observar el área que requieren las piezas restantes como un indicador general de la longitud de la solución. Así, el autor establece que una pieza más grande debe colocarse antes que una más pequeña, dado que la probabilidad de encontrar otro lugar que sea tan bueno o mejor para una pieza grande es menor que la de una pieza pequeña.

Método programado

Ante la falta de investigaciones previas, el autor reafirma que su método no busca la solución óptima, sino que busca lograr resultados comercialmente competitivos, comparados con la mano de obra humana. Por ello, propone un *algoritmo no iterativo* para ubicar las piezas, el cual puede ser extendido a conjuntos de reglas de ubicación más sofisticados. Las reglas de selección de piezas propuestas por el autor son las siguientes:

1. Seleccionar aquellas piezas que tengan un valor x mínimo en la envoltura dentro de un valor de tolerancia dado en pulgadas (parámetro TOL 1) del mínimo disponible de las piezas restantes. La regla determina las piezas que se pueden colocar a la izquierda del área disponible.
2. Seleccionar de las piezas aceptables según la regla 1, aquellas que tengan un área mayor que una fracción fija (parámetro TOL 2) del área de la pieza con el área máxima entre las piezas aceptables según la regla 1. Esta regla determina la pieza más grande que se puede colocar a la izquierda.
3. Seleccionar de aquellas piezas aceptables bajo la regla 2 aquellas con un mínimo de "desperdicio probable" en el lado izquierdo. El desperdicio probable se obtiene dividiendo el área que se encuentra entre la pieza y un rectángulo que contiene a la pieza dibujado con lados paralelos a los bordes del material, dividido por el ancho de la pieza. Esta regla es más importante en las primeras etapas de ubicación, cuando la mayoría de las piezas tienen un valor de x mínimo, equivalente al borde izquierdo del material; por lo tanto, el autor afirma que se requiere una regla más precisa.

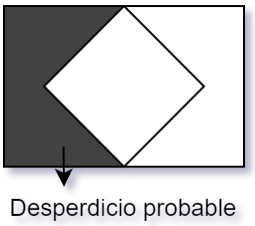


Figura 5: Desperdicio probable. Fuente: [2]

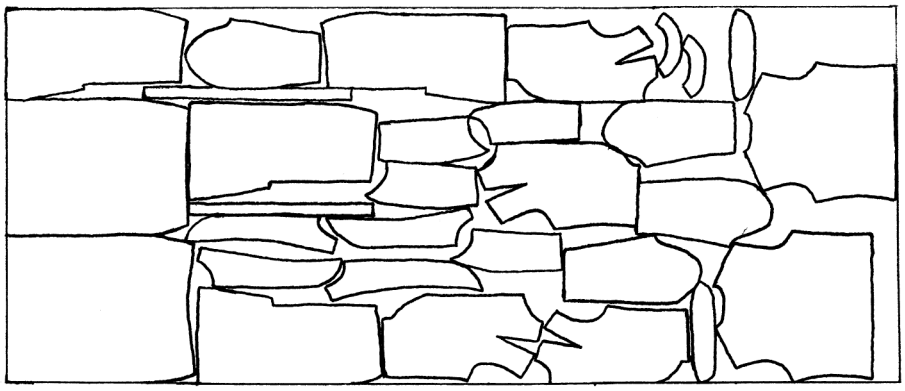
4. Aunque rara vez se necesita (según [2]), la cuarta regla es seleccionar entre aquellos que han pasado todas las reglas anteriores la pieza que tiene el valor y más bajo en el punto donde la envoltura tiene un mínimo de x. Esta regla ubica una pieza en la parte inferior izquierda que tiende a tener una ubicación más compacta.

Para probar el método propuesto, el autor utilizó los patrones para un traje de mujer en dos tamaños distintos. Los patrones de trajes están compuestos por 22 piezas cada uno, y según el autor son los más complejos de la industria (de la época). Se utilizó una tela de 58[in] de ancho en la mayoría de las ejecuciones de prueba. En dichos casos, el autor menciona que se descuidan las piezas más pequeñas porque se les puede dar un mejor uso para llenar el área que sobra al cambiar de la representación convexa a la representación completa, y su inclusión en el problema empeora el tiempo de ejecución en un 60 % [2]. También se ejecutaron casos con telas de 56[in] y 60[in] de ancho para confirmar la suposición de que la longitud disminuiría si se aumentara el ancho y viceversa.

Otros resultados que obtuvo fueron los siguientes:

- En uno de sus experimentos, el autor obtuvo un largo de 137.5 pulgadas con un desperdicio del 26 % utilizando una representación completa, mientras que con una representación convexa obtuvo un desperdicio de 18.7 %
- El valor de tolerancia para el x mínimo de la envoltura pareciera afectar la solución solo para valores extremos. Si dicho valor es menor que 0.1, las reglas de selección se concentran en encontrar pequeños lugares para ubicar las piezas. El marcador obtenido suele ser un mosaico en el que las piezas más pequeñas dominan las primeras etapas después de que se haya llenado el borde izquierdo, lo que resulta en una ubicación ineficaz de piezas más grandes y anchas en las etapas posteriores. Por otro lado, si la tolerancia se relaja hasta el punto donde el área es el factor dominante, las piezas grandes dominan las primeras etapas de una manera igualmente ineficiente, desperdiciando mucha área en el lado izquierdo.
- El autor afirma, además, que las variaciones en la tolerancia para la relación de tamaños indican que (generalmente) es más eficiente seleccionar la pieza más grande disponible. La razón entregada por Art de por qué la variación en las tolerancias solo tiene una ligera relación con la calidad de los resultados es que cada par de tolerancias (TOL1, TOL2) en particular proporciona una de las 450 soluciones posibles que pueden generarse con las reglas de ubicación de la técnica programada [2].
- Añadir pequeñas rotaciones puede mejorar el resultado. En [2] se menciona que un ejemplo de la ventaja de ignorar la restricción de la *grain line* (es decir, permitir que las piezas giren en un ángulo pequeño) es que al trabajar con un conjunto en miniatura de piezas convexas el autor obtuvo una solución 4 % mejor que el marcador de prueba dibujado con piezas en representación completa.

Finalmente, el autor comenta que permitiendo la rotación de las piezas, junto con el desarrollo de métodos iterativos pueden mejorar el rendimiento, pero agregarlo al esquema que propuso no es fácil; es más, en sus propias palabras: *el problema ya no se puede tratar de la misma manera.*



Sample marker output from programmed method. Scale length = 137.5 inches

Figura 6: Resultado obtenido. Fuente: [2]

Cambio de enfoque: Algoritmos Genéticos

En 1993, Stefan Jakobs propone un algoritmo genético para ubicar polígonos en un tablero rectangular, el cual es mejorado mediante la combinación con métodos deterministas. Al igual que [2], la motivación es el problema que tienen la industria textil y metalúrgica, estampar figuras poligonales en un tablero rectangular. El autor define el problema de la siguiente forma:

> Dado un número finito de rectángulos $r_i, i = 1, \ldots, n$ y un tablero rectangular, un patrón de empaque ortogonal requiere por definición ubicar los rectángulos de forma disyuntiva en el tablero de tal manera que los bordes de r_i sean paralelos a los ejes x e y, respectivamente. El cálculo del patrón de empaque ortogonal con altura mínima se llama Ortogonal Packing Problem (OPP).
>
> Stefan Jakobs. Fuente: [10]

Nótese que esta definición es más formal (matemáticamente hablando) que la propuesta por Art en [2].

Para resolver el problema, el autor propone un algoritmo evolutivo con tres clases principales independientemente desarrolladas. La primera se llama programación evolutiva (EP), la segunda se llama estrategias evolutivas (ES) y la tercera, algoritmo genético (GA).

El autor propone dos enfoques para la extensión a los polígonos. El primero de ellos, llamado extensión directa, aplica el algoritmo genético directamente sobre los polígonos. Sin embargo, este método da como resultado un tiempo de computación bastante largo. Una alternativa es aplicar el algoritmo genético a los rectángulos en los que están incrustados los polígonos; posteriormente, el uso de un paso de encogimiento determinista acerca los polígonos entre sí [10].

Caracterización del problema

El tamaño del espacio de búsqueda de OPP es infinito, ya que cada movimiento de un rectángulo en una dirección factible crea un nuevo patrón [10]. Con el fin de reducir el número de posibles patrones se introduce la llamada *Condición Inferior Izquierda* (*BL-condition*). El patrón de empaque cumple la condición BL si un rectángulo no se puede desplazar más hacia abajo y hacia la izquierda [10].

Jakobs menciona que el OPP es una generalización natural de Bin-packing en una dimensión, y que de hecho, si se requiere que todos los rectángulos tengan la misma altura entonces los dos problemas coinciden. Por otro lado, el caso en el que todos los rectángulos tengan el mismo ancho corresponde al conocido Makespan Minimization Problem. Se sabe que ambos problemas son

NP-Completos. El autor menciona que Sleator, a pesar de no utilizar la condición BL, demuestra que OPP puede reducirse a PARTITION [10].

Estructura de datos

En [10] se representan los rectángulos como una permutación $\pi = (i_1, \ldots, i_n)$, donde i_j corresponde al índice del rectángulo r_{i_j}.

Dicha permutación representa la secuencia en la cual los rectángulos son puestos sobre el tablero. El autor indica que las ventajas de esta representación es la facilidad con la que se pueden crear nuevas permutaciones al cambiar la secuencia. Una desventaja, por otro lado, es que cada permutación debe asignarse a un patrón único, por lo que la decodificación del genotipo requiere más esfuerzo que la conversión de la representación natural (2 vértices opuestos del rectángulo), por lo que el objetivo en [10] es crear un algoritmo de decodificación rápido.

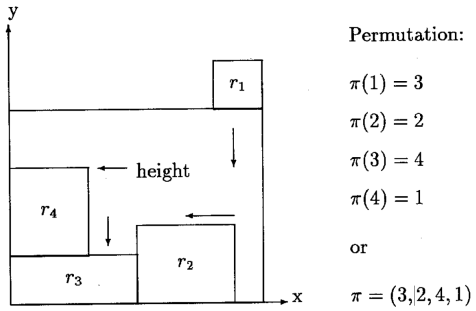


Figura 7: Ejemplo de permutación y condición BL. Fuente:[10]

Jakob presenta en su estudio algunas propiedades del algoritmo BL. La primera es una cota superior a los posibles patrones de empaque. Dados n rectángulos, hay a lo más $2^n \cdot n!$ patrones posibles. Esto es una consecuencia de la naturaleza combinatorial del problema. En la práctica, el algoritmo BL puede crear menos patrones que dicho número [10]. El costo del algoritmo BL es $O(n^2)$. Esto se basa en el hecho de que cada rectángulo r_i se puede desplazar un máximo de i veces, porque cada cambio está limitado por uno de los $(i-1)$ rectángulos colocados o por las esquinas del tablero. Por lo tanto, el costo de colocar el rectángulo r_i es $O(i)$ y el costo total asciende a $O(n^2)$ [10].

Algoritmo genético

El algoritmo genético requiere una evaluación del patrón. Para ello, el autor define una función *fitness* $f : \pi \to \mathbb{R}_+$ que cumple la propiedad $f(\pi_i) > f(\pi_j)$ si π_i es un mejor patrón de empaque que π_j. El autor la define como el área disponible que queda tras ubicar los rectángulos en el orden π, es decir: $f(\pi) = \text{Área}(\text{RestoContiguo}(\pi))$

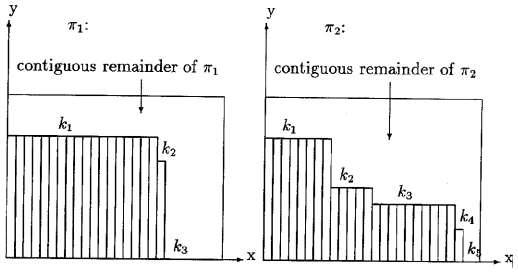


Figura 8: Ejemplo de resto contiguo. Fuente: [10]

En este caso el algoritmo genético trabaja con m permutaciones $(\pi_1, \ldots, \pi_m)$, a las cuales se le asigna su valor *fitness*:

$$f_i = f(\pi_i), \quad i = 1 \ldots, m$$

De esta forma, cada individuo A_i queda definido por la permutación y su valor *fitness*: $A_i = (\pi_i, f_i)$. Los operadores del algoritmo genético son los siguientes:

- **Selección proporcional**: se divide el intervalo $I = [0, 1)$ en m subintervalos, de tal forma que cada individuo tenga uno. Los límites de cada intervalo son determinados de forma aleatoria.

- **Cruzamiento**: se seleccionan dos individuos π_i y π_j. Desde la posición p (elegida aleatoriamente), el cruzamiento copia q elementos desde π_i a la nueva permutación. El resto de los elementos son extraídos de π_j en el mismo orden en el que se encuentran.

- **Mutación**: este operador rota los rectángulos en 90^o de forma aleatoria con una probabilidad de p_m.

Tras completar estas operaciones genéticas (cuyo costo total es $O(t \cdot m \cdot n^2)$, con t el número de iteraciones permitidas), la descendencia se convierte en el fenotipo (es decir, en el patrón). Luego se calcula el valor de *fitness*, se identifica el peor individuo y se reemplaza por la descendencia. Todos estos procedimientos se repiten hasta que se alcanza un máximo de iteraciones o no se nota una mejora adicional durante un período de tiempo determinado [10].

Extensión a polígonos

El autor menciona que existen algoritmos deterministas para realizar la extensión a polígonos, pero son costosos (mayores a $O(n^2)$) y el algoritmo genético debe ejecutarlo en cada iteración, por lo que no lo aconseja [10]. Es por ello que propone una alternativa más rápida: un algoritmo de incrustación-contracción (*Embedding-Shrinking Algorithm*).

En la figura 9 presentada como resultado en [10] se observa que el algoritmo reduce la altura del patrón de *packing* a través del mejoramiento del valor de *fitness*, lo que implica que los *gaps* entre figuras son más pequeños [10].

En la práctica, la combinación de algoritmos deterministas y genéticos proporciona un posible escape fuera de los mínimos locales [10]. Una ventaja adicional, según Jakobs, es su fácil implementación. El autor sostiene que cualquier algoritmo de *packing* determinista basado en permutaciones podría mejorarse mediante el algoritmo genético que propuso. La mejora del algoritmo BL es el primer paso en esta dirección.

No obstante, también reconoce que el algoritmo tiene desventajas. Una de ellas es que existen grupos de rectángulos para los cuales el algoritmo BL no puede generar el patrón óptimo, por lo que es necesario usar un algoritmo determinista mayor y más costoso para concretar dicha transformación.

Surgimiento de una tipología

En 1990, Harald Dyckhoff [6] desarrolla una tipología general que tenía como objetivo integrar los diversos tipos de problemas de corte y empaque (C&P), dado que existían diversos nombres en la literatura para referirse a problemas con una estructura lógica similar (en esencia), los cuales se presentan en la tabla 10.

La tipología se basó en la estructura lógica básica de los problemas C&P, la cual se define de la siguiente manera:

- Hay dos grupos de datos básicos cuyos elementos definen cuerpos geométricos de formas fijas (figuras) en un espacio de números reales de una o más dimensiones: el *stock* de los llamados "objetos (grandes)" y la lista de los llamados "artículos (pequeños)" por otro lado.

- Los patrones son combinaciones geométricas de artículos pequeños asignados a objetos grandes. Las piezas residuales, es decir, las figuras que aparecen en los patrones y que no corresponden a elementos pequeños, generalmente se tratan como "pérdida de corte".

De esta forma Dyckhoff describe 4 características importantes de los problemas C&P:

1. **Dimensionalidad:** define el número mínimo de dimensiones necesarias para describir la geometría del patrón. Se obtienen problemas con más de tres dimensiones cuando se expanden a dimensiones no espaciales, por ejemplo, tiempo o peso.

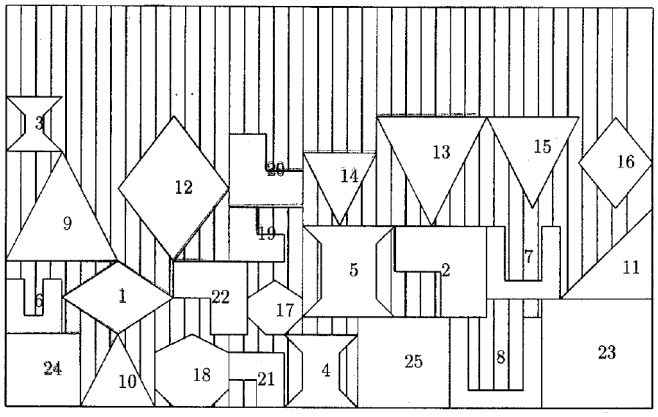


Fig. 25. Packing pattern generated by GA (1000 steps); height = 17.

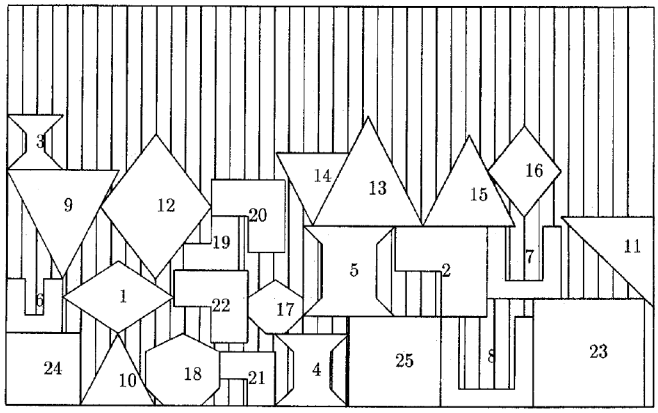


Fig. 26. Packing pattern generated by the shrinking algorithm; height = 16.

Figura 9: Resultado obtenido tras aplicar GA con *Shrinking*. Fuente: [10]

2. El tipo de asignación describe si se deben asignar todos los objetos y elementos o solo una selección.

3. **Surtido de objetos:** esta característica distingue entre problemas que tienen objetos de forma idéntica o diferente.

4. El surtido de artículos se refiere a la forma y al número de artículos. Los problemas pueden consistir en pocos elementos, elementos congruentes, muchos elementos de muchas formas diferentes y muchos elementos de relativamente pocas formas diferentes

El autor postula además que los problemas C&P pertenecen al campo de combinatoria geométrica, y postula que también pueden considerarse en un sentido abstracto y generalizado en dimensiones no espaciales. Así plantea los siguientes ejemplos aplicados en distintas áreas de conocimiento:

- Knapsack (Dantzig, 1957) y carga de vehículos (Eilon y Christiofides, 1971) para la dimensión del peso
- Equilibrio de línea de montaje (Wee y Magazine, 1982) y programación multiprocesador (Coffman et al., 1978) para la dimensión temporal
- Presupuesto de capital (Lorie y Savage, 1955) y creación de cambios (Martello y Toth, 1980; Stehling, 1983) para las dimensiones financieras
- Asignación de memoria de computadora para las dimensiones de almacenamiento de datos (cf. Garey y Johnson, 1981).

Table 1
Surveys on special aspects of C&P

Author(s)	Year	Notion(s)	Discipline
Brown	1971	Packing, depletion	Computer Science
Salkin/de Kluyver	1975	Knapsack	Logistics
Golden	1976	Cutting stock	Industrial Engineering
Hinxman	1980	Trim loss, assortment	Operational Research
Garey/Johnson	1981	Bin packing	Combinatorial Optimization
Israni/Sanders	1982	Cutting stock, layout	Manufacturing
Rayward-Smith/Shing	1983	Bin packing	Mathematics
Coffman et al.	1984	Bin packing	Computer Science
Dowsland	1985	Packing	Operational Research
Dyckhoff et al.	1985	Trim loss	Management
Israni/Sanders	1985	Parts nesting	Production
Berkey/Wang	1987	Bin packing	Operational Research
Dudzinski/Walukiewicz	1987	Knapsack	Operational Research
Martello/Toth	1987	Knapsack	Mathematics
Rode/Rosenberg	1987	Trim loss	Engineering/Production
Dyckhoff et al.	1988	Cutting stock	Production

Figura 10: Literatura de problemas C&P Fuente: [6]

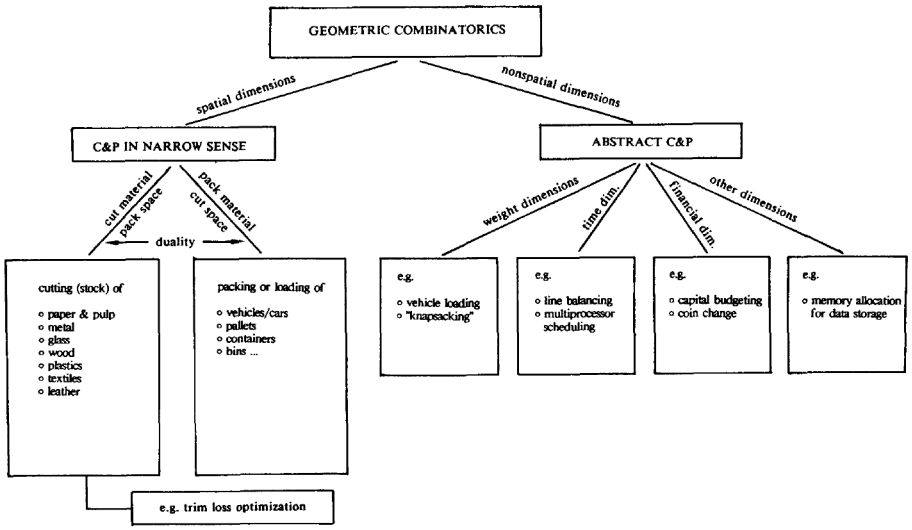


Figure 4. Phenomenology of cutting and packing problems

Figura 11: Clasificación de problemas C&P. Fuente: [6]

Recorriendo los enfoques

Hopper y Burton [8] hacen un recorrido por los diversos enfoques desarrollados para resolver los problemas de *packing* de dos dimensiones con diferentes técnicas (en su mayoría algoritmos genéticos). No obstante, presentan información respecto a otras técnicas, como *Simulated Annealing* y *Tabú Search* para resolver el problema.

- **Simulated Annealing**: La mayoría de los enfoques de *Simulated Annealing* para ISPP no utilizan ninguna técnica de codificación. El problema del *packing* se representa como una asignación de artículos 2D, que deben compactarse. Por lo general, se permite la superposición durante el proceso de búsqueda y se penaliza en la función de evaluación. La técnica de representación para este enfoque difiere de las que involucran algoritmos genéticos. Con una excepción (Ratanapan y Dagli 1997b, 1998) los métodos genéticos de la literatura operan en la codificación. La superposición generalmente se evita mediante la aplicación de reglas de ubicación y solo se permite en algunos enfoques [8].

- **Tabú Search**: Blazewicz (1993) fueron los primeros en aplicar *Tabú Search* a ISPP. Comenzando con una solución de diseño factible que se produce mediante un procedimiento de colocación simple, se utiliza un proceso de búsqueda tabú para mejorar aún más el diseño

existente. Después de seleccionar un solo elemento, se prueban varias posiciones nuevas y se mantiene la mejor. El movimiento describe un cambio en la asignación de un artículo de una posición a otra, prohibiendo configuraciones superpuestas. Los elementos que han cambiado su posición durante las iteraciones recientes son miembros de la lista tabú. El mejor movimiento admisible está determinado por la función objetivo que apunta a colocar los elementos más a la derecha en las áreas vacías del diseño. En comparación con el algoritmo de búsqueda heurística de Albano y Sapuppo (1980), la búsqueda tabú logró mejores resultados [8].

- **Redes Neuronales Artificiales (ANN)**: Han y Na (1996) usaron una red neuronal para producir una solución inicial para un problema irregular 2D. Después de obtener una solución inicial "buena", el diseño no superpuesto se mejora aún más mediante *Simulated Annealing*. El algoritmo de aprendizaje de la red neuronal se basa en una red *Kohonen*. Al comienzo del proceso de anidamiento, todas las formas se asignan alrededor del centro del objeto mediante la asignación de pequeños valores aleatorios a sus vectores de posición que describen la distancia al centro. La red neuronal modifica los vectores de posición, que indican la dirección del movimiento de los elementos. La posición finita se determina de manera tal que la superposición de los elementos sea mínima utilizando la colocación más baja a la izquierda. La función de costo es una combinación entre un término de penalización para la superposición y los momentos de los elementos de conducción de área a la izquierda y al lado inferior del objeto [8].

Respecto al tiempo de cómputo, en [8] se menciona que la decodificación tiene una gran influencia en el esfuerzo computacional del algoritmo híbrido, principalmente porque las meta-heurísticas son costosas debido a la gran cantidad de evaluaciones de funciones. Esto da como resultado tiempos de ejecución largos, especialmente en problemas irregulares, donde los cálculos geométricos necesarios para el proceso de anidamiento requieren mucho tiempo. El tipo y la implementación de algoritmos geométricos contribuyen al tiempo de cálculo, especialmente cuando se utiliza una alta precisión para la aproximación y descripción de la forma.

La rotación rara vez se considera para cualquiera de los problemas de *packing* irregular que no sean en factores de 90^o. Esto se debe principalmente a las restricciones impuestas por la aplicación industrial, por ejemplo, la industria textil y metalúrgica [8].

El estudio [7] es la base de varios estudios posteriores. En él se sigue el enfoque que aplica algoritmos híbridos, los cuales combinan meta-heurísticas y modelos de compactación de programación lineal (Bennell y Dowsland, 2001) [7].

Gomes y Olivera utilizan un algoritmo de *Simulated Annealing* para guiar la búsqueda a través del espacio de solución, siendo la estructura de vecindad basada en modelos de compactación de programación lineal.

En la tabla 12 presentada se observa que la técnica desarrollada por Gomes y Olivera mejoró los resultados de desempeño publicados en la literatura para la mejor solución en un promedio de 6,8 % con GLSHA (*Greedy Search Local Hybrid Algorithm*) y un 8,84 % con SAHA (*Simulated Annealing Hibryd Algorithm*). En palabras de los mismos autores:

> Los mejores resultados publicados previamente en la literatura se mejoraron para todas las instancias de problemas utilizadas en las pruebas computacionales. Los tiempos de cálculo del algoritmo híbrido de *Simulated Annealing* son relativamente altos, especialmente con las grandes instancias tomadas de la industria textil. Sin embargo, una estrategia de búsqueda local *greedy*, con la misma estructura de vecindario, puede usarse para producir diseños muy buenos en un tiempo más razonable.
>
> Miguel Gomes, José Olivera. Fuente: [7]

En [3] se presenta una técnica llamada *Beam Search* Para ello, los autores mencionan que implementaron un método alternativo a TOPOS (Olivera et. al., 2000) para generar el NFP de

Table 3
Hybrid algorithms comparisons

Problem instance	Best solution improvement (%)			Average solution improvement (%)			Average time ($\frac{SAHA}{GLSHA}$)
	GLSHA vs. BRP	SAHA vs. BRP	SAHA vs. GLSHA	GLSHA vs. BRP	SAHA vs. BRP	SAHA vs. GLSHA	
FU	4.28	7.84	3.72	2.43	3.82	1.43	6.24
JAKOBS1	7.69	4.44	−3.53	−2.31	0.52	2.76	8.88
JAKOBS2	7.80	11.44	3.95	6.49	8.31	1.95	8.29
SHAPES0	1.59	2.54	0.97	−1.73	−0.23	1.47	6.30
SHAPES1	3.39	5.08	1.75	0.49	1.41	0.93	5.31
SHAPES2	3.30	4.39	1.13	1.39	2.81	1.43	6.14
DIGHE1	12.43	27.60	17.33	5.25	11.68	6.78	5.57
DIGHE2	25.40	25.40	0.00	2.69	10.83	8.37	6.33
ALBANO	0.48	1.63	1.16	−3.50	−1.56	1.88	10.82
DAGLI	9.57	11.29	1.90	6.85	9.44	2.78	6.41
MAO	11.62	13.23	1.82	10.05	11.29	1.37	10.61
MARQUES	3.72	6.12	2.49	1.39	4.75	3.41	9.50
*SHIRTS**	1.46	1.44	−0.01	0.06	0.15	0.09	6.40
*SWIM**	8.03	8.70	0.72	6.26	6.80	0.57	5.05
*TROUSERS**	1.16	1.48	0.32	−0.24	0.43	0.67	6.68
Average	6.80	8.84	2.25	2.37	4.70	2.39	7.13

* Solved with the multi-stage approach.

Figura 12: Comparación de resultados. Fuente: [7]

modo que se puedan utilizar las posiciones de *interlocking* y los espacios entre piezas. También afirman que mejoraron la velocidad computacional de generar posiciones de colocación a través del NFP. La variante implementada de *Beam Search* utiliza la evaluación local y global, donde se basa la evaluación local únicamente en la evaluación de agregar la siguiente pieza a la solución parcial y la evaluación global genera un diseño completo y evalúa la longitud total del diseño.

En la tabla 13 se presentan los resultados obtenidos por Benell y Song [3]. A pesar que no obtiene los mejores resultados, en [3] aseguran que *Beam Search* es un mecanismo de búsqueda eficaz para ISPP que produce resultados competitivos, dado que tiene la ventaja de ser determinista y producir varias soluciones de alta calidad en una sola ejecución (los resultados obtenidos son reproducibles en cualquier computador ya que no requieren una *seed* de números aleatorios predeterminada).

En [5] se propone 3PM (*3-Phase Matheuristic*). Dicha técnica se basa en el modelo *Dotted Board* de Toledo e. al. (2013) y el modelo de compactación de Gomes y Olivera (2006). La heurística consta de 3 fases:

- Fase constructiva: se busca una solución factible usando el modelo *Dotted Board*.
- Fase de mejora: se mejora la solución con el mismo modelo
- Fase de compactación: se mejora la mejor solución encontrada usando el modelo de compactación lineal.

Durante las pruebas se permitía la rotación de los polígonos en ángulos definidos. En la tabla 14 se presentan los resultados obtenidos. Se observa que HS2 y SCM (*Semi Continuous Model*) son mejores para instancias pequeñas, ya que el método exacto puede encontrar y probar rápidamente la optimalidad de una solución, mientras que 3PM debe completar todas sus fases [5]. Aún así, 3PM funciona mejor para instancias más grandes (por ejemplo ALBANO y MAO). En palabras de sus autores, la heurística *no depende en gran medida de las dimensiones de la instancia, lo que indica que es un buen enfoque para abordar grandes instancias* [5].

4. Modelos Matemáticos

Un modelo para el problema es presentado en [9]. Sea $P = \{P_1, P_2, \dots P_n\}$ un conjunto de n polígonos y un tablero rectangular o *container* C de ancho W^* (constante) y alto H (variable

Table 3 Best beam search results over all runs compared with best in literature

Data set	Beam search				Best in literature		
	Criteria	BW/FW	Max util	Time (s)	Author	Max util	Time (s)
Albano	Sum	10000/10	**87.88%**	5,460	E	**87.88%**	21,600
Dagli	BL-Vec	10000/10	**87.97%**	17,331	G	87.14%	5,110 (×20)
Dighe1	BL-Sum	10/5	**100.00%**	1.4	G	**100%**	83 (×20)
Dighe2	Pri/Vec	10/5	**100.00%**	0.3	G	**100%**	22 (×20)
Fu	Pri	10000/15	90.28%	1,192	E	**92.03%**	21,600
Jakobs1	Pri	400/15	85.96%	2,193	E	**89.07%**	600 (×20)
Jakobs2	Sum	10/20	80.40%	75	E	**80.41%**	600 (×20)
Mao	Sum	10000/10	84.07%	16,757	E	**85.15%**	600 (×20)
Marques	Pri	10000/15	88.92%	10,692	E	**89.82%**	21,600
Poly(5B)	Pri	400/10	**79.51%**	52,514	B	75.82%	677 (×40)
Shapes0	BL-Sum	400/4	64.35%	8	E	**67.09%**	600 (×20)
Shapes1	BL	400/8	72.55%	398	E	**73.84%**	600 (×20)
Shapes2	Pri	10000/10	81.29%	5,603	G	**83.59%**	2,257 (×20)
Shirts	BL-Sum	400/10	**89.69%**	6,217	E	87.38%	12,600
Swim	BL-Vec	400/10	**75.04%**	15,721	G	74.37%	6,937 (×20)
Trousers	BL-Sum	400/10	90.38%	5,988	E	**90.46%**	21,600

Figura 13: Resultados obtenidos para *Beam Search*. Fuente: [3]

no negativa). La ubicación del polígono P_i se describe mediante las coordenada de su *punto de referencia* (el punto de referencia es cualquier punto del polígono, como por ejemplo un vértice o el centro de gravedad). El vector $v_i = (x_i, y_i)$ es llamado *vector de traslación*, y sus componentes corresponden al punto de referencia del polígono. Por conveniencia, en [9] se considera cada polígono y el *container* como el conjunto de puntos (incluidos los puntos interiores y de contorno), cuyas coordenadas se determinan a partir del punto de referencia colocado en el origen $O = (0, 0)$. Luego, se describe el polígono P_i colocado en v_i por la suma de Minkowski

$$P_i \otimes v_i = \{p + v_i | p \in P_i\}$$

Para un polígono S, se define su interior como $int(S)$, ∂S como su contorno, $\overline{S}$ como su complemento y $cl(S)$ como la clausura (es decir, el conjunto cerrado más pequeño que contiene S). Así, el Irregular Strip Packing Problem (ISPP) se describe formalmente como:

$$\begin{array}{lll} \text{mín} & H & (1) \\ \text{sujeto a} & int(P_i \otimes v_i) \cap (P_j \otimes v_j) = \varnothing \quad 1 \leq i < j \leq n, & (2) \\ & (P_i \otimes v_i) \subseteq C(W^*, H), \quad 1 \leq i \leq n & (3) \\ & H \geq 0, & (4) \\ & v_i \in \mathbb{R}^2 \quad 1 \leq i \leq n & (5) \end{array}$$

La función objetivo (1) busca minimizar la altura, y en (4) se restringe que ésta debe ser positiva. La restricción (2) prohíbe que las piezas se solapen entre sí, mientras que la restricción (3) obliga a que las piezas queden totalmente contenidas en el tablero.

Con este modelo, los autores de [9] representan las soluciones de ISP como una $n-$tupla $v = (v_1, v_2, \ldots, v_n)$, que es la parte esencial de las variables de decisión porque la altura mínima H del tablero está determinada por:

$$H_{\text{mín}}(v) = \text{máx}\{y | (x, y) \in P_i \otimes v_i, P_i \in P\} - \text{mín}\{y | (x, y) \in P_i \otimes v_i, P_i \in P\}$$

Instance	HS2[1]		SCM[2]		DBM			3PM	
	Solution	Time	Solution	Time	Solution	Time (find)	Time	Solution	Time
Blaze1	–	–	7.5	4.9	7.5	12.0	23.3	**7.4**	23.3
Blaze2	–	–	**13.8**	TL	14.0	15.1	15.2	14.0	68.9
Blaze3	–	–	20.7	TL	21.0	80.3	674.0	**20.5**	340.2
Blaze4	–	–	29.0	TL	**27.0**	1068.0	1239.2	27.9	517.6
Blaze5	–	–	37.7	TL	**34.0**	540.5	TL	**34.0**	395.2
Shapes_T2	–	–	**14.0**	6.6	16.0	0.5	1.7	**14.0**	8.1
Shapes_T4	–	–	28.0	TL	**26.0**	58.4	89.1	**26.0**	201.5
Shapes_T5	–	–	34.0	TL	**30.0**	340.6	365.0	31.0	106.0
Shapes_T7	–	–	52.0	TL	**42.0**	2901.0	TL	**42.0**	176.3
Shapes_T9	–	–	57.0	TL	49.0	3482.6	TL	**48.0**	292.3
RCO1	–	–	**8.0**	5.8	**8.0**	0.6	0.7	**8.0**	44.4
RCO2	–	–	**15.0**	TL	**15.0**	1.2	1.3	**15.0**	254.5
RCO3	–	–	23.0	TL	**22.0**	10.7	13.2	**22.0**	264.7
RCO4	–	–	32.3	TL	**29.0**	16.7	394.0	**29.0**	83.1
RCO5	–	–	38.0	TL	**36.0**	164.6	936.2	36.7	210.2
Albano	–	–	–	–	11088.0	592.4	592.4	**10608.0**	1614.1
Fu	–	–	–	–	35.0	53.1	53.1	**32.0**	252.3
Jakobs1	–	–	–	–	18.0	3285.3	TL	**12.0**	612.9
Jakobs2	–	–	–	–	30.0	596.2	TL	**26.0**	1939.0
Mao	–	–	–	–	2452.0	99.7	TL	**1927.2**	2621.8
Marques	–	–	–	–	88.0	1827.9	TL	**85.0**	527.9
Shapes0	–	–	78.0	TL	64.0	3590.5	TL	**60.0**	239.1
Shapes1	–	–	–	–	80.0	98.9	TL	**58.0**	1132.7
Shapes2	–	–	–	–	**27.0**	900.4	TL	27.6	310.7
Trousers	–	–	–	–	495.0	218.4	TL	**286.0**	1403.7
Poly1a0	16.6	TL	16.7	TL	17.0	3586.2	TL	**15.8**	1048.8
Shapes_AV4	**24.0**	0.0	–	–	**24.0**	1.7	1.7	**24.0**	2.2
Shapes_AV8	**26.0**	272.0	–	–	28.0	18.5	21.5	**26.0**	186.9
Fu5	**17.9**	0.1	**17.9**	76.3	20.5	2.8	3.4	**17.9**	1.0
Fu6	**23.0**	0.5	**23.0**	442.9	24.0	6.0	10.4	**23.0**	31.7
Fu7	**24.0**	1.0	**24.0**	TL	28.0	0.1	0.2	**24.0**	5.0
Fu8	**24.0**	1.3	**24.0**	TL	28.0	0.2	1.0	**24.0**	20.9
Fu9	**25.0**	70.0	**25.0**	TL	28.0	0.4	0.4	**25.0**	52.3
Fu10	**28.7**	3064.0	30.0	TL	30.0	0.8	0.9	**28.7**	265.9
Fu12	**31.2**	TL	34.4	TL	40.0	1.0	1.0	32.0	186.7
threep1w7	**6.0**	0.8	**6.0**	0.3	6.5	0.3	0.3	**6.0**	2.5
threep2w7	**9.3**	3.9	9.7	1.0	11.0	0.7	0.8	**9.3**	12.4
threep3w7	**13.5**	3394.0	14.0	852.7	14.5	1.3	1.3	**13.5**	183.1
threep2w9	**8.0**	8.5	**8.0**	3.1	8.5	1.4	1.6	**8.0**	36.2
threep3w9	**11.0**	TL	11.3	TL	13.0	0.2	0.2	**11.0**	191.2

Figura 14: Resultados obtenidos para 3PM. Fuente: [5]

y $(P_i \otimes v_i) \subseteq C(W^*, H_{\text{mín}}(v))$ se cumple para todo $P_i \in P$ sí y sólo si

$$W^* \geq \text{máx}\{x|(x,y) \in P_i \otimes v_i, P_i \in P\} - \text{mín}\{x|(x,y) \in P_i \otimes v_i, P_i \in P\}$$

Bajo este mismo modelo, la técnica geométrica de los polígonos *No-Fit* (NFP) queda definida de la siguiente forma:

$$\text{NFP}(P_i, P_j) = int(P_i) \otimes (-int(P_j)) = \{u - w|u \in int(P_i), w \in int(P_j)\}$$

Los polígonos *No-Fit* tiene las siguientes propiedades, las cuales son importantes para el problema [9]:

- $P_j \otimes v_j$ se superpone con $P_i \otimes v_i$ sí y sólo sí $v_j - v_i \in \text{NFP}(P_i, P_j)$.
- $P_j \otimes v_j$ toca a $P_i \otimes v_i$ sí y sólo si $v_j - v_i \in \partial\text{NFP}(P_i, P_j)$.
- $P_i \otimes v_i$ y $P_j \otimes v_j$ están separados sí y sólo si $v_j - v_i \notin cl(\text{NFP}(P_i, P_j))$

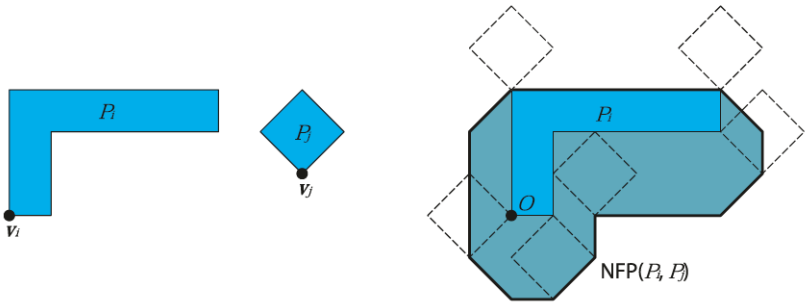


Figura 15: Un ejemplo de NFP(P_i, P_j), donde O es el origen. Fuente: [9]

En [1], los autores presentan el modelo matemático utilizado por Gomes y Olivera (2006) y dos propuestas basadas en las ideas de Fischetti y Luzzi (2009). En todos los casos se busca minimizar el largo del tablero acomodando las N piezas (cada una de largo l_i y ancho w_i) sin que se superpongan. Además, los autores sostiene que todas las formulaciones que presenta cuentan con dos tipos de restricciones: las primeras previenen que una pieza exceda las dimensiones del tablero y las otras prohíben que las piezas se solapen.

Modelo de Gomes y Olivera

$$
\begin{array}{lll}
\text{mín} & L & (1) \\
\text{sujeto a} & x_i \leq L - l_i, \quad 1 \leq i \leq N, & (2) \\
& y_i \leq W - w_i, \quad 1 \leq i \leq N & (3) \\
& \alpha_{ijk}(x_j - x_i) + \beta_{ijk}(y_j - y_i) \leq q_{ijk} + M(1 - v_{ijk}) \quad 1 \leq i < j \leq N, \; k = 1, \ldots, m_{ij} & (4) \\
& \sum_{k=1}^{m_{ij}} v_{ij} \geq 1 \quad 1 \leq i < j \leq N & (5) \\
& v_{ijk} \in \{0, 1\} \quad 1 \leq i < j \leq N & (6) \\
& x_i, y_i \geq 0 \quad 1 \leq i \leq N & (7)
\end{array}
$$

La función objetivo (1) minimiza el largo del tablero. Las restricciones (2), (3) y (7) son *restricciones de borde*, es decir, evitan que las piezas salgan del tablero. La restricción (4) evita que las piezas se solapen (aquí, M es una constante positiva muy grande que se utiliza para penalizar, mientras que $\alpha_{ijk}(x_j - x_i) + \beta_{ijk}(y_j - y_i) = q_{ijk}$ es la ecuación de la recta que incluye la $k-$ésima arista del NPF$_{ij}$. Los autores de [1] definen m_{ij} como la cantidad de aristas que tiene NPF$_{ij}$). La restricción (6) indica que las variables v_{ijk} son enteras y que al menos una de ellas debe valer 1 para cada NFP. Una variable v_{ijk} toma el valor 1 si los puntos de referencia de la pieza j y la pieza i están en diferentes lados de la $k-$ésima arista de NFP$_{ij}$, o si el punto de referencia de la pieza j incluso está en esa línea, en cualquier otro caso vale 0.

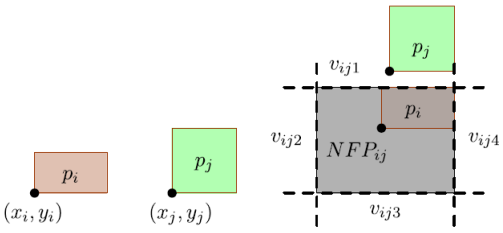


Figura 16: Definición de variables v_{ijk}. Fuente: [1]

No obstante, en [1] se menciona que este modelo tiene una potencial desventaja a la hora de evitar el solapamiento de piezas, ya que no limita la posición relativa de las piezas de manera muy estricta. Según los autores, "si usamos esta formulación en un procedimiento *Branch and Bound*, muchas ramas diferentes pueden contener la misma solución y eso puede ralentizar la búsqueda"[1]. Es por ello que presentan dos modelos de programación lineal entera mixta, y que se encuentran a continuación:

Modelo HS1

$$
\begin{array}{lll}
\text{mín} & L & \\
\text{sujeto a} & x_i \leq L - l_i, & 1 \leq i \leq N, \\
& y_i \leq W - w_i, & 1 \leq i \leq N \\
& \alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \leq \sum_{h=1}^{m_{ij}} \delta_{ij}^{kfh} b_{ijh} & 1 \leq i < j \leq N \\
& & k = 1, \ldots, m_{ij},\ f = 1, \ldots, t_{ij}^k \\
& \sum_{k=1}^{m_{ij}} b_{ijk} = 1 & 1 \leq i < j \leq N \\
& b_{ijk} \in \{0, 1\} & 1 \leq i < j \leq N \\
& x_i, y_i \geq 0 & 1 \leq i \leq N
\end{array}
$$

Este modelo se basa en la formulación propuesta por Fischetti y Luzzi (2009), y difiere del modelo anterior en el uso que le da al NFP para la definición de variables [1]. Ellos consideran la región exterior de cada NFP_{ij} como particionada en m_{ij} regiones convexas disjuntas llamadas *slices* S_{ij}^k, y definen una variable binaria b_{ijk} tal que ésta vale 1 si el punto de referencia de p_j se ubica dentro de S_{ij}^k, y 0 en cualquier otro caso.

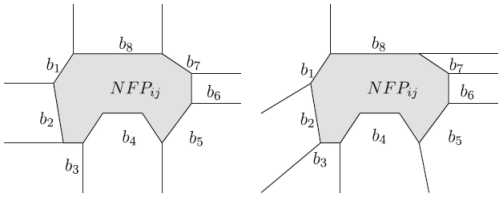


Figura 17: Diferentes formas de definir *slices*. Fuente: [1]

Cada *slice* es un poliedro de 2 dimensiones definido por un conjunto de t_{ij}^k inecuaciones, y para asegurar que éstas siempre sean válidas, los autores de [1] introducen la variable δ_{ij}^{kfh} para reemplazar a la constante M, y la definen como:

$$\delta_{ij}^{kfh} = \max_{(v_i - v_j) \in S_{ij}^h \cap C} \alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i)$$

La interpretación de las restricciones de este modelo son las mismas que el anterior.

Modelo HS2

$$
\begin{array}{lll}
\text{mín} & L & \\
\text{sujeto a} & \sum_{k \in R_{ij}} \underline{x}_{ijk} b_{ijk} \leq x_i \leq L - l_i - \sum_{k \in LS_{ij}} \underline{\lambda}_{ijk} b_{ijk} & 1 \leq i < j \leq N \\
& \sum_{k \in U_{ij}} \underline{y}_{ijk} b_{ijk} \leq y_i \leq W - w_i - \sum_{k \in DS_{ij}} \underline{\mu}_{ijk} b_{ijk} & 1 \leq i < j \leq N \\
& \alpha_{ij}^{kf}(x_j - x_i) + \beta_{ij}^{kf}(y_j - y_i) \leq \sum_{h=1}^{m_{ij}} \delta_{ij}^{kfh} b_{ijh} & 1 \leq i < j \leq N \\
& & k = 1, \ldots, m_{ij}, f = 1, \ldots, t_{ij}^k \\
& \sum_{k=1}^{m_{ij}} b_{ijk} = 1, & 1 \leq i < j \leq N \\
& b_{ijk} \in \{0, 1\}, & 1 \leq i < j \leq N \\
& x_i, y_i \geq 0 & 1 \leq i \leq N
\end{array}
$$

En este modelo se cambian las restricciones de HS1 que son idénticas al modelo de Gomes y Olivera usando la interacción entre pares de piezas. Para ello, se definen las siguientes variables:

- $\overline{Y}_{ij}, \underline{Y}^{ij}$: corresponde al máximo (mínimo) valor de la coordenada y del NFP_{ij}. La definición es análoga para la coordenada x.
- $\overline{x}_{ijk}, \underline{x}^{ijk}$: máximo (mínimo) valor que puede tomar x_j en el sistema de coordenadas del NFP_{ij} cuando $b_{ijk} = 1$. La definición para la coordenada y es análoga.

Para dichas variables se definen algunos conjuntos específicos. Sea $U_{ij}(D_{ij})$ el conjunto de variables asociadas con el NFP_{ij} que están sobre (bajo) el punto de referencia de p_i. De forma análoga, se definen R_{ij} y L_{ij} para el conjunto de variables que están a la derecha o izquierda (respectivamente) del punto de referencia p_i. Aquel conjunto que contiene todas las variables asociadas a un NFP_{ij} se

denomina como VNFP_{ij}, y los anteriores quedan formalmente definidos como:

$$U_{ij} = \{b_{ijk} \in \text{VNFP}_{ij} | \underline{y}_{ijk} \geq 0\}$$
$$D_{ij} = \{b_{ijk} \in \text{VNFP}_{ij} | \overline{y}_{ijk} \leq 0\}$$
$$R_{ij} = \{b_{ijk} \in \text{VNFP}_{ij} | \underline{x}_{ijk} \geq 0\}$$
$$L_{ij} = \{b_{ijk} \in \text{VNFP}_{ij} | \overline{x}_{ijk} \leq 0\}$$

En la figura 18, se ve que $U_{ij} = \{b_{ij1}, b_{ij2}, b_{ij8}\}$, $D_{ij}\{b_{ij4}, b_{ij5}, b_{ij6}\}$, $R_{ij} = \{b_{ij6}, b_{ij7}, b_{ij8}\}$ y $L_{ij} = \{b_{ij2}, b_{ij3}, b_{ij4}\}$.

Con lo anterior en mente se redefinen las restricciones (2), (3) y (7) del modelo de Gomes Olivera, las cuales son llamadas en [1] como *lifted bound constraints*:

$$DS_{ij} = \{k | \mu_{ijk} > 0\}, \mu_{ijk} = w_i - (\overline{y}_{ijk} - \underline{Y}^{ij})$$
$$DS_{ij} = \{k | \lambda_{ijk} > 0\}, \lambda_{ijk} = l_i - (\overline{x}_{ijk} - \underline{X}^{ij})$$

Según los autores, con este modelo se puede utilizar un algoritmo de *Branch and bound* en el que cada nodo del árbol de búsqueda la relajación lineal permita obtener una cota inferior, y si el nodo no está comprometido el *branching* construirá 2 nodos: uno con $b_{ijk} = 0$ y otro con $b_{ij} = 1$. Además, mencionan que con diferentes estrategias de *branching* pueden obtener mejoras significativas en el rendimiento del algoritmo [1].

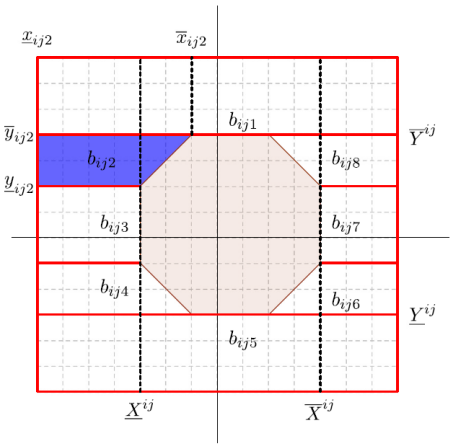


Figura 18: Definiciones en el sistema de coordenadas de NFP_{ij}. Fuente: [1]

En [13] se utiliza la idea de Jakobs [10] de incrustar los polígonos en rectángulos. Los autores postulan que cada rectángulo tiene 8 direcciones posibles, denotadas como $p = (1, \ldots, 8)$. n_{p_i} y m_{p_i} denotan el bisel vertical y horizontal (respectivamente) del rectángulo i en la dirección p. A lo anterior se suma la definición de un conjunto de variables binarias F_{ij}^p, las cuales denotan si el item i precede al item j en la dirección p.

→ Si $F_{ij}^p = 1$ y p es impar, i precede a j y están posicionados como rectángulos

→ Si $F_{ij}^p = 1$ y p es par, podría haber una interferencia entre los rectángulos que encierran a los respectivos polígonos.

Un ejemplo de valores y configuraciones de X_{ij}^2 e Y_{ij}^2 para $F_{ij}^2 = 1$ se presentan en la figura 19.

Así, el modelo en [13] se define: sea I un conjunto de n *items* ($i \in I, i = 1, \ldots, n$), cada uno de los cuales tiene un largo l_i y un ancho w_i. La posición de los *items* es un par de variables de decisión (x_i, y_i), que corresponden a las coordenadas del vértice superior derecho del rectángulo que contiene al respectivo *item*. Luego, el modelo es:

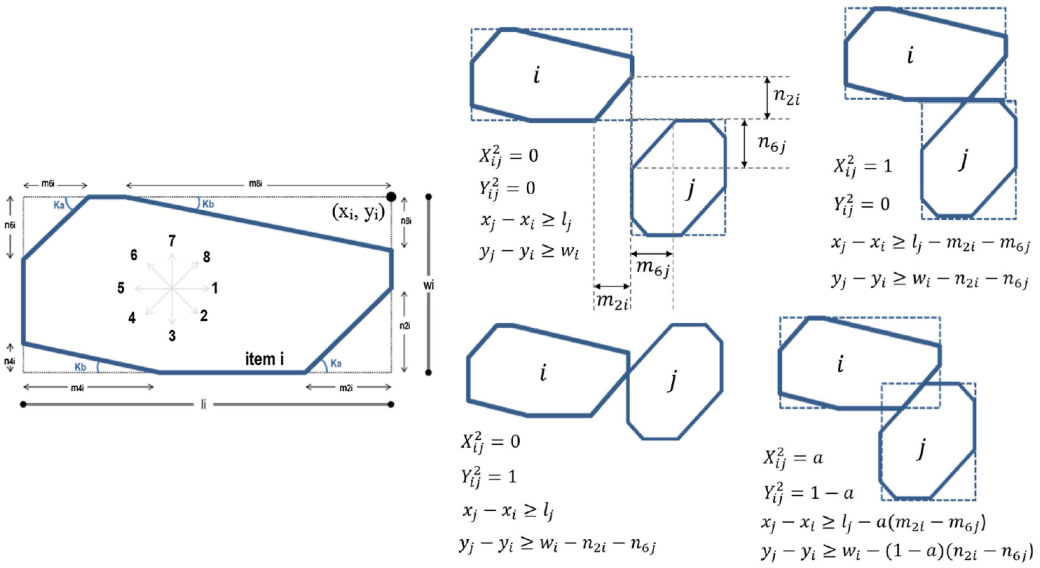


Figura 19: Ejemplo de items y configuraciones. Fuente: [13]

$$\textbf{min} L \tag{1}$$

$$\textbf{sa}:$$

$$x_j \leq L \quad j = 1, \ldots, n \tag{2}$$

$$y_j \leq W \quad j = 1, \ldots, n \tag{3}$$

$$x_j \geq l_j \quad j = 1, \ldots, n \tag{4}$$

$$y_j \geq w_j \quad j = 1, \ldots, n \tag{5}$$

$$F_{ij+}^1 F_{ij+}^2 F_{ij+}^3 F_{ij+}^4 F_{ij+}^5 F_{ij+}^6 F_{ij+}^7 F_{ij}^8 = 1 \quad i, j = 1, \ldots, n \quad i < j \tag{6}$$

$$x_j - x_i - l_j + \left(1 - F_{ij}^8 - F_{ij}^1 - F_{ij}^2\right) M_x + X_{ij}^8 \left(m_{4j} + m_{8i}\right) + X_{ij}^2 \left(m_{2i} + m_{6j}\right) \geq 0i, \quad j = 1, \ldots, n \quad i < j \tag{7}$$

$$x_i - x_j - l_i + \left(1 - F_{ij}^4 - F_{ij}^5 - F_{ij}^6\right) M_x + X_{ij}^4 \left(m_{4i} + m_{8j}\right) + X_{ij}^6 \left(m_{6i} + m_{2j}\right) \geq 0i, \quad j = 1, \ldots, n \quad i < j \tag{8}$$

$$y_j - y_i - w_j + \left(1 - F_{ij}^6 - F_{ij}^7 - F_{ij}^8\right) M_y + Y_{ij}^8 \left(n_{8i} + n_{4j}\right) + Y_{ij}^2 \left(n_{6i} + n_{2j}\right) \geq 0i, \quad j = 1, \ldots, n \quad i < j \tag{9}$$

$$y_i - y_j - w_i + \left(1 - F_{ij}^2 - F_{ij}^3 - F_{ij}^4\right) M_y + Y_{ij}^4 \left(n_{4i} + n_{8j}\right) + Y_{ij}^6 \left(n_{2i} + n_{6j}\right) \geq 0i, \quad j = 1, \ldots, n \quad i < j \tag{10}$$

$$X_{ij}^2 - Y_{ij}^2 \leq F_{ij}^2 \quad i, j = 1, \ldots, n \quad i < j \tag{11}$$

$$X_{ij}^4 - Y_{ij}^4 \leq F_{ij}^4 \quad i, j = 1, \ldots, n \quad i < j \tag{12}$$

$$X_{ij}^6 - Y_{ij}^6 \leq F_{ij}^6 \quad i, j = 1, \ldots, n \quad i < j \tag{13}$$

$$X_{ij}^8 - Y_{ij}^8 \leq F_{ij}^8 \quad i, j = 1, \ldots, n \quad i < j \tag{14}$$

$$0 \leq X_{ij}^p \leq 1 \quad i, j = 1, \ldots, n \quad i < j \quad p = 2, 4, 6, 8 \tag{15}$$

$$0 \leq Y_{ij}^p \leq 1 \quad i, j = 1, \ldots, n \quad i < j \quad p = 2, 4, 6, 8 \tag{16}$$

$$x_j, y_j, L, X_{ij}^p, Y_{ij}^p \in R, \quad F_{ij}^p \in \{0, 1\} \tag{17}$$

La función objetivo (1) es minimizar L, que es la longitud del tablero. Por la restricción (2) se asegura que L es más grande que las coordenadas x de todos los *items*. Las restricciones (2), (3), (4) y (5) aseguran, por otro lado, que se respeten los límites fijos del objeto. Las restricciones (6)-(10) son las que impiden la superposición. La ecuación (6) asegura que el elemento j viene después del elemento i en solo uno de los 8 lados posibles de los polígonos. En otras palabras, solo uno de los ocho posibles F_{ij}^p es igual a 1. El conjunto de ecuaciones (7)-(10) son las restricciones disyuntivas referidas a la condición de no superposición en los ejes x e y. Asegura que la distancia mínima para no solaparse se respete en las 8 direcciones. M_x y M_y, los llamados valores *big-M*, son valores que son lo suficientemente grandes como para hacer que las restricciones (7–10) sean ficticias cuando

no se multiplican por cero. El mínimo M_x es la suma de todos los l_i y el mínimo M_y es W. Las interferencias (X_{ij}^p y/o Y_{ij}^p son mayores que cero) solo deben ocurrir si el correspondiente F_{ij}^p es igual a 1, lo cual está asegurado por las restricciones (11)-(14). Además, deben asumir un valor entre 0 y 1, como se restringe en (15)-(16). Finalmente, en la declaración (17), se establece el dominio de las variables, donde solo F_{ij}^p es binaria.

En [4] se presentan dos modelos de programación lineal entera mixta. En el primer modelo (DTM), las restricciones no superpuestas se establecen en base a la trigonometría directa, mientras que en el segundo modelo (NFP-CM) las piezas se descomponen primero en partes convexas y luego las restricciones que evitan la superposición de piezas se escriben en base a polígonos *No-fit* de las partes convexas. Ambos enfoques, según los autores, son robustos en términos del tipo de geometrías que pueden abordar, considerando cualquier tipo de polígono no convexo con o sin agujeros.

***Direct Trigonometry Model* (DTM)**

$$\min L \tag{5}$$

$$\text{s.t. } l_i^{\min} \le x_i \le L - l_i^{\max}, \quad i = 1, \ldots, N, \tag{6}$$

$$h_i^{\min} \le y_i \le H - h_i^{\max}, \quad i = 1, \ldots, N, \tag{7}$$

$$\begin{aligned} & C_{ij}^{pqk} + (a_x^k - b_x^k)(y_i - y_j) \\ & \quad + (a_y^k - b_y^k)(x_i - x_j) \le (1 - v_{ij}^{pqk}) M_{ij}^{pqk}, \quad i, j = 1, \ldots, N, \ i \ne j, \\ & k \in K_{ip}, \\ & p \in P_i, \quad q \in P_j, \end{aligned} \tag{8}$$

$$\begin{aligned} & \sum_{k \in K_{ip}} v_{ij}^{pqk} + \sum_{k \in K_{jq}} v_{ji}^{qpk} = 1, \quad 1 \le i < j \le N, \\ & \quad p \in P_i, \quad q \in P_j, \end{aligned} \tag{9}$$

$$(x_i, y_i) \in \mathcal{R}^2, \quad i = 1, \ldots, N, \tag{10}$$

$$\begin{aligned} & v_{ij}^{pqk} \in \{0, 1\}, \quad i, j = 1, \ldots, N, \ i \ne j, \\ & \quad k \in K_{ip}, \ p \in P_i, \ q \in P_j. \end{aligned} \tag{11}$$

La función objetivo (5) busca minimizar el largo del contenedor. Las restricciones (6) y (7) aseguran que las piezas quedan contenidas en el tablero. Las restricciones (8) y (9) son las que impiden el solapamiento de piezas: en (8) solo permite que las piezas estén separadas o se toquen, mientras que en (9) se determina que solo 1 inecuación relacionada con un par de piezas se cumpla. En (10) y (11) se determina el dominio de las variables.

Este modelo no necesita estructuras geométricas especiales, como los polígonos *No-Fit*, para ser construido [4]. Según los autores, esta característica es interesante ya que al usar estructuras más simples puede ser más fácil (versus los modelos más complejos) agregar nuevas restricciones al modelo o cambiar las existentes.

Aquí, H y L son la altura y el largo del tablero, respectivamente. Se deben ubicar N polígonos en total. $l_i^{\text{mín}}$ y $l_i^{\text{máx}}$ corresponden a las distancias en el eje x desde x_i hasta el vértice que se encuentra más a la izquierda (derecha), respectivamente. De forma análoga se definen $h_i^{\text{mín}}$ y $h_i^{\text{máx}}$, pero con respecto al eje y. (x_i, y_i) corresponde al punto de referencia del polígono i.

También se define K_{ip} como el conjunto de aristas de la parte p de la pieza i, mientras que los puntos a^k y $b^k \in \mathbb{R}^2$ corresponden respectivamente al vértice inicial y final de la arista $k \in K_{ip}$. Con

ambas definiciones se plantea la constante C_{ij}^{pqk}, que es igual a:

$$C_{ij}^{pqk} = \max_{r}\{(a_x^k - b_x^k)(a_y^k - g_y^{jrqj}) - (a_y^k - b_y^k)(a_x^k - g_x^{jrqj})\}$$

donde g^{jrqj} es la distancia entre el punto de referencia de la pieza j y el vértice r de la parte q de la pieza j. Finalmente, se define la variable binaria v_{ij}^{pqk}, la cual toma el valor 1 sí y sólo si la desigualdad para la arista k de la parte p de la pieza i es satisfecha con respecto a la parte q de la pieza j. Asociado a esta variable se define la constante M_{ij}^{pqk}, cuyo valor es suficientemente grande como para hacer válida la inecuación correspondiente siempre y cuando $v_{ij}^{pqk} = 0$.

***NFP Covering Model* (NFP-CM)**

$$\min L \tag{13}$$

$$\text{s.t. } l_i^{\min} \le x_i \le L - l_i^{\max}, \quad i = 1, \ldots, N, \tag{14}$$

$$h_i^{\min} \le y_i \le H - h_i^{\max}, \quad i = 1, \ldots, N, \tag{15}$$

$$\begin{aligned} &\overline{C}_{pk}^{ij} - (a_{ij,x}^p - b_{ij,x}^p)(y_i - y_j) \\ &\quad + (a_{ij,y}^p - b_{ij,y}^p)(x_i - x_j) \le (1 - v_{ij}^{pk})M_{ij}^{pk}, \quad i = 1, \ldots, N-1, \\ &\quad j = i+1, \ldots, N, \\ &\quad p \in Q_{ij}, \quad k \in K_{ij}^p, \end{aligned} \tag{16}$$

$$\sum_{k \in K_{ij}^p} v_{ij}^{pk} = 1, \quad i = 1, \ldots, N-1, \tag{17}$$

$$\begin{aligned} &j = i+1, ..., N, \\ &p \in Q_{ij}, \end{aligned} \tag{18}$$

$$(x_i, y_i) \in \mathcal{R}^2, \quad i = 1, \ldots, N, \tag{19}$$

$$\begin{aligned} &v_{ij}^{pk} \in \{0, 1\}, \quad i, j = 1, \ldots, N, \\ &\quad k \in K_{ij}^p, \ p \in Q_{ij}. \end{aligned} \tag{20}$$

En este modelo se utilizan polígonos *No-fit*, por lo que se deben realizar algunas definiciones. K_{ij}^p es el conjunto de aristas dirigidas del NFP_{ij}^p; Q_{ij} es el número de partes del NFP_{ij} y a_{ij}^p, b_{ij}^p corresponden a 2 vértices consecutivos del NFP_{ij}^p.

Con respecto al modelo DTM, la única diferencia es que se redefinen la constante C_{ij}^{pqk} y la variable v_{ij}^{pqk}. La constante queda definida como

$$\overline{C}_{ij}^{pk} = (a_{ij,x}^p - b_{ij,x}^p)a_{ij,y}^p - (a_{ij,y}^p - b_{ij,y}^p)a_{ij,x}^p$$

mientras que la variable binaria v_{ij}^{pk} es 1 sí y sólo sí el punto de referencia de la pieza j está al lado derecho o sobre la linea k del NFP_{ij}^p.

Así, en este modelo los elementos del mismo tipo no se consideran elementos diferentes, lo que implica que se evita calcular soluciones que sean simétricas. Esto se logra imponiendo que $x_i \le x_j$ para todo $i < j \in N$ si las piezas i y j son del mismo tipo. Estas restricciones aseguran que las piezas respetarán un orden de precedencia y reducirán significativamente la simetría del espacio de la solución [4].

Los últimos modelos que se presentarán en el siguiente escrito serán los que se proponen en [12]. Dicho estudio se basa en el modelo *dotted-board*, un modelo de programación entera mixta para el ISPP en el que el tablero está representado por una cuadrícula. Toledo calculó un límite superior (L) para la longitud del tablero y aplicó una discretización usando una resolución apropiada g_x (horizontal) por g_y (vertical). Dichos valores influyen directamente en la calidad de la solución, ya que una discretización refinada aumenta el número de restricciones y variables binarias utilizadas.

Dotted-board Model

$$\min \quad z \tag{1}$$

$$\text{s.t.} \quad (c_d\, g_x + x_t^M)\, \delta_t^d \le z, \qquad \forall t \in \mathcal{T}, \forall d \in \mathcal{IFP}_t, \tag{2}$$

$$\sum_{d \in \mathcal{IFP}_t} \delta_t^d = q_t, \qquad \forall t \in \mathcal{T}, \tag{3}$$

$$\delta_t^d + \delta_u^e \le 1, \qquad \forall t, u \in \mathcal{T}, \forall d \in \mathcal{IFP}_t, \forall e \in \mathcal{NFP}_{t,u}^d, \tag{4}$$

$$\delta_t^d \in \{0, 1\}, \qquad \forall t \in \mathcal{T}, \forall d \in \mathcal{IFP}_t, \tag{5}$$

$$z \ge 0, \tag{6}$$

En el modelo, cada pieza de tipo t está representada por un polígono y tiene una demanda q_t. El polígono se describe mediante un conjunto de vértices cuyas coordenadas se definen a partir de un punto de referencia. Además, cada pieza tiene un recinto rectangular que define, respectivamente, los límites inferiores x_t^m e y_t^m y los límites superiores x_t^M e y_t^M para los ejes x e y. El concepto de polígono *Inner-Fit* se utiliza para garantizar que cada pieza esté completamente dentro del tablero. Toledo utilizó el concepto de polígono *No-fit* para garantizar que dos piezas no se superpongan [12]. Por último, se define la variable binaria δ_t^d, la cual vale 1 sí y sólo sí una pieza de tipo t está asignada en el punto d del tablero.

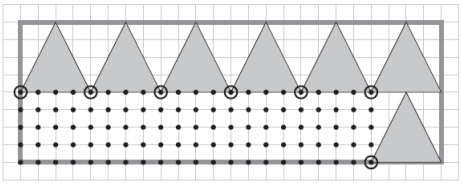
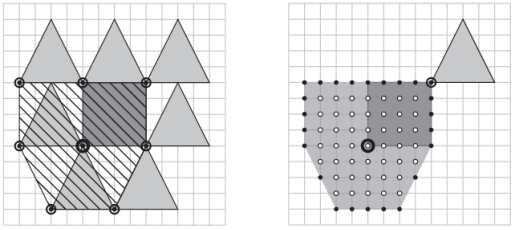


Fig. 2. Example of an inner-fit polygon on the grid.



(a) $NFP_{t,u}$ (b) $\mathcal{NFP}_{t,u}^d$

Fig. 3. Construction of a nofit polygon on the grid.

Figura 20: Ejemplo de polígono *Inner-fit* y polígono *No-Fit*. Fuente: [12]

La función objetivo (1) junto con las restricciones (2) minimizan la longitud del tablero utilizado para cortar todas las piezas. Las restricciones (3) aseguran que se satisfaga la demanda de todas las piezas, mientras que las restricciones (4) evitan la superposición entre las piezas. Finalmente, las restricciones (5) y (6) definen los dominios variables.

Clique Covering Model

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

$$\text{s.t.} \quad \sum_{(d,t)\in K} (c_d\, g_x + x_t^M)\, \delta_t^d \le z, \qquad \forall K \in \mathcal{V}, \tag{10}$$

$$\sum_{d\in \mathcal{IFP}_t} \delta_t^d = q_t, \qquad \forall t \in \mathcal{T}, \tag{11}$$

$$\sum_{(d,t)\in K} \delta_t^d \le 1, \qquad \forall K \in \mathcal{E}, \tag{12}$$

$$\delta_t^d \in \{0,1\}, \qquad \forall t \in \mathcal{T}, \forall d \in \mathcal{IFP}_t. \tag{13}$$

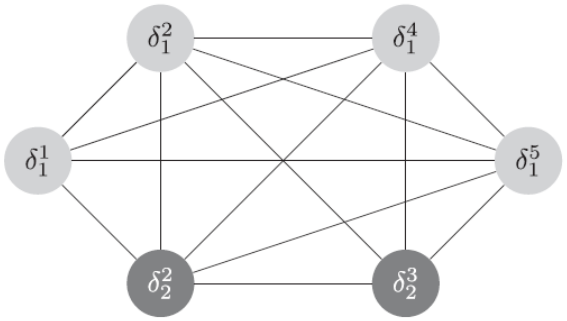


Figura 21: Ejemplo de grafo de conflictos. Los vértices asociados con las variables $\{\delta_1^1, \delta_2^3\}$ describen una solución factible del problema. Fuente: [12]

Aquí, $K \subseteq \mathcal{V}$ es un clique de G, donde $\mathcal{V}$ y $\mathcal{E}$ son un vértice y una arista del clique.

La función objetivo (9) y las restricciones (11) y (13) son las mismas del modelo anterior. Las restricciones de vértice de *clique covering* (10) definen la longitud del tablero utilizado para cortar todas las piezas, mientras que las restricciones de arista de *clique covering* (12) imponen la no superposición entre las piezas. Se debe tener en cuenta que el modelo anterior es un caso particular de este modelo si las cubiertas triviales se usan para $\mathcal{V}$ y $\mathcal{E}$.

Este modelo se basa en la idea de que la sobreposición de las piezas en un tablero de puntos se puede describir como un grafo de conflictos. La sustitución de las restricciones de borde (4) por *cliques* (7) conlleva muchas ventajas. Primero, para el problema estudiado, el número de restricciones utilizadas para representar la superposición entre piezas se reduce drásticamente. En segundo lugar, las restricciones de *clique* también reducen el número de valores distintos de cero del modelo, lo que favorece los métodos para matrices dispersas. Por último, los *cliques* pueden mejorar significativamente el límite inferior obtenido de la relajación lineal del modelo [12].

5. Representación

Dada la naturaleza geométrica del problema, se definió una clase general para representar una figura cualquiera:

```
class Figura {
  public:
    int ID;
    vector<vector<int>> vertices;
};
```

Código 1: Definición de clase `Figura`

La estructura de la clase 1 es la siguiente: cada figura posee un identificador numérico y un vector con cada uno de sus vértices, los cuales están dispuestos en sentido anti-horario y se ubican sólo en el primer cuadrante del plano euclideano. Por ejemplo, para el polígono 22 su vector de vértices

sería `[(0,0),(2,0),(2,2),(0,2)]`. Se utiliza la misma estructura para cargar los polígonos desde las instancias.

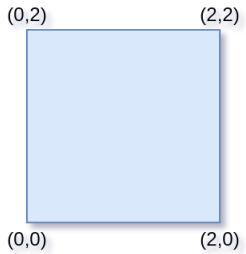


Figura 22: Polígono. Fuente: Elaboración propia

Para resolver el problema, se adoptó una estrategia similar a la planteada en [10]. En vez de trabajar con los polígonos directamente, se trabajó con el rectángulo que contiene a todos los vértices del respectivo polígono. Dicho rectángulo queda representado por dos vértices opuestos (sin pérdida de generalidad se escogieron los vértices inferior izquierdo y superior derecho), donde cada una de las coordenadas corresponde al mínimo (máximo) valor que toman los vértices del polígono. En la figura 23 se ilustra esta situación.

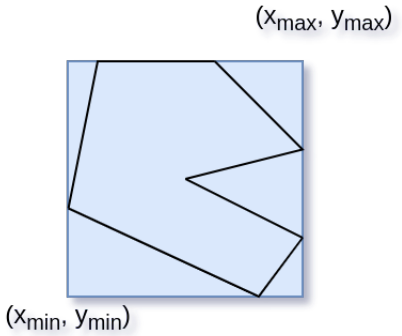


Figura 23: Rectángulo contenedor. Fuente: Elaboración propia

La estructura de datos escogida para resolver el problema es un *vector* de instancias de la clase `Figura` (concretamente, es un vector de rectángulos contenedores de polígonos), lo cual representará una de las posibles permutaciones de rectángulos que serán ubicados en el contenedor siguiendo la heurística BL (*Bottom-Left*, es decir, se ubica el polígono lo más a la izquierda y lo más abajo posible). La permutación es respecto al orden en el que se insertarán los rectángulos, tal como se ilustra en la figura 7.

Se escoge esta representación ya que el algoritmo ubicará los rectángulos emulando al juego *tetris* (siguiendo las condiciones de la heurística BL). De esta forma, se manejan las restricciones:

- Los polígonos quedan contenidos dentro de los márgenes del *container*.
- Los polígonos no se solapan entre sí.

6. Descripción del Algoritmo

El algoritmo es una implementación de *Hill Climbing* con alguna mejora (*First Improvement Hill Climbing*).

6.1. Solución inicial

Dado que HC es una técnica reparadora, se debe idear un algoritmo para obtener una solución inicial. En este caso en particular, se genera una **permutación aleatoria** (dado que hay diferentes tipos de polígonos, los rectángulos contenedores serán diferentes y el orden en el que se ubican en el contenedor aplicando la heurística BL influye en el resultado final).

6.2. Algoritmo de posicionamiento de rectángulos

Algoritmo 1 putRectangles(permutación de rectángulos)

```
1: x := ancho actual
2: h := altura actual
3: L := altura máxima
4: heights[w] := vector para almacenar altura actual
5: for cada rectángulo de la permutación do
6:     l_i := altura del rectángulo actual
7:     w_i := anchura del rectángulo actual
8:     if x + w_i excede el ancho del container then resetear el valor de x a 0
9:     if no hay solapamiento entre el rectángulo actual y los ya posicionados then
10:        actualizar el valor de heights en w_i casillas
11:    else obtener altura máxima actual y actualizar valor de heights en w_i casillas
12:    Aumentar el valor de x en w_i unidades
13: return L
```

Dado que los rectángulos poseen coordenadas enteras, se utiliza un arreglo de enteros para almacenar la altura actual en cada coordenada x (el tamaño del arreglo es w, el ancho del *container*). Para ilustrar esta idea obsérvese el siguiente ejemplo con $w = 8$:

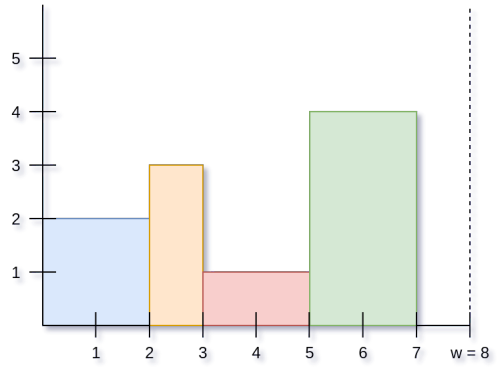


Figura 24: Ejemplo de ubicación de rectángulos. Fuente: Elaboración propia

Para el caso presentado, `heights=[2,2,3,1,1,4,4,0]`. De esta forma se puede determinar fácilmente dónde posicionar el siguiente rectángulo. Continuando con el ejemplo, si el siguiente rectángulo no cabe dentro del *container* se cumplirá que la suma entre x (que vale 7) y el ancho del rectángulo será mayor a 8, por lo que se reiniciará el valor de la x en 0 y se intentará ubicar en la siguiente posición más abajo posible (siguiendo la heurística BL). En la figura 25 se presentan los dos casos de ubicación infactible, y la siguiente ubicación factible:

Para la ubicación factible del ejemplo, $x = 3$ y `heights=[5,5,5,1,1,4,4,0]`. Si este fuese el total de rectángulos a ubicar, se obtendría que el largo mínimo del *container* es L = 5.

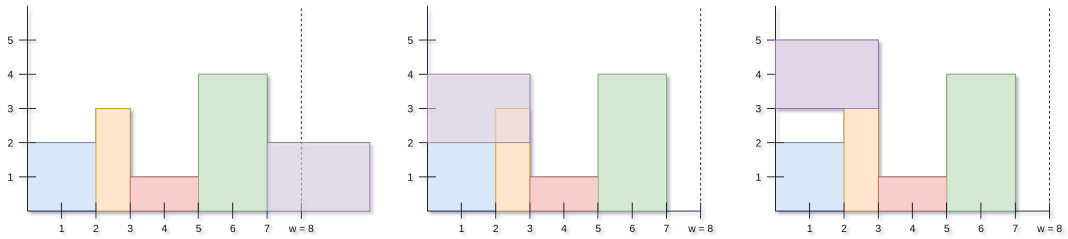


Figura 25: Ejemplo de ubicación de rectángulos. Fuente: Elaboración propia

6.3. Algoritmo HC

Algoritmo 2 Hill Climbing con Alguna Mejora, con *restart*

```
1: while #Iteraciones < k do
2:     r := permutación aleatoria de rectángulos (solución actual)
3:     L* := putRectangles(r)
4:     for cada vecino de la solución actual do
5:         Calcular L*_i del vecino actual
6:         if L*_i < L* then
7:             L* := L*_i
8:             Almacenar en memoria el vecino actual
9:             Generar otra solución inicial e incrementar en 1 el valor de k
10: return mejor solución encontrada
```

Los pasos que sigue el algoritmo son los siguientes:

- **Paso 1**: se genera una permutación aleatoria, la cual se utiliza como solución inicial.
- **Paso 2**: se insertan los rectángulos contenedores en el *container* y se obtiene el largo requerido, el cual es mantenido en memoria junto con la permutación.
- **Paso 3**: se aplica el movimiento *swap* sobre la solución inicial, generando paulatinamente su vecindario. Se posicionan los rectángulos en el *container* y se verifica si se obtiene un largo menor. Si hay una mejora en la solución, se almacena como la mejor solución y se genera otra solución inicial (*restart*); en caso contrario, se obtiene el siguiente vecino. Este paso iterativo se realiza k veces (donde k indica la cantidad de *restarts* y es un parámetro entregado por el usuario).

7. Experimentos

Para realizar los experimentos se entregaron las instancias de la tabla a continuación, las cuales fueron preprocesadas para que los vértices de todos los polígonos se ubicaran en el primer cuadrante del plano euclideano. Además de los vértices de los polígonos, se entregan las dimensiones del *container*, donde el ancho es fijo y el largo debe ser minimizado.

Nombre de Instancia	**Cantidad de polígonos**	**Ancho de *container***	**Largo de *container***
albano	24	4900	29000
blaz	11	15	90
dagli	30	60	200
jakobs1	25	40	200
jakobs2	25	70	250
marquez	24	104	400
poly2a	30	4000	5000
poly3a	45	4000	7000
poly4a	60	4000	8000
poly5a	75	4000	10000
shirts	99	40	1000
swim	48	5752	50000
trousers	64	79	1000

Cuadro 1: Datos de las instancias utilizadas en los experimentos

Dado que el algoritmo implementado es *Hill Climbing*, el único parámetro que se modificó es la cantidad de *restarts* (entre 10 y 1000 *restarts*, ambos incluidos). El objetivo es favorecer la exploración del algoritmo, y observar cómo se comporta a medida que se aumentan las iteraciones. También se buscaba observar el comportamiento de HC respecto al estancamiento en óptimos locales.

Se buscaba experimentar con la rotación de los polígonos en ángulos específicos (90°, 180°, 270°) pero por tiempo no se alcanzó a implementar y probar correctamente.

8. Resultados

Se ejecutó el algoritmo *Hill Climbing* con Alguna Mejora con k restarts, donde $k \in [10, 1000]$. Los resultados obtenidos se presentan en la siguiente tabla:

	10 restarts	50 restarts	100 restarts	250 restarts	500 restarts	750 restarts	1000 restarts
albano	13410	13640	13715	13346	**13243**	13747	13243
blaz	**39**	40	40	40	40	40	40
dagli	104	107	105	106	106	107	**101**
jakobs1	20	20	**19**	20	19	19	20
jakobs2	44	44	**42**	42	44	44	44
marquez	118	114	114	116	115	**112**	122
poly2a	4997	**4996**	4996	4997	5094	5000	4996
poly3a	7292	7296	7394	**7099**	7295	7298	7296
poly4a	9597	9597	9594	9595	9595	9692	**9593**
poly5a	**11892**	11997	11991	11894	11994	11895	11894
shirts	91	**90**	92	93	92	93	93
swim	11134	11081	10947	11229	11006	11152	**10818**
trousers	**365**	365	365	365	365	366	365

Cuadro 2: Largos obtenidos. En negrita se identifican los mejores valores obtenidos para cada instancia

Se puede observar que el algoritmo tiende a quedarse atrapado en óptimos locales (un ejemplo evidente es el caso de la instancia `trousers`), ya que todas las soluciones obtenidas son similares.

Respecto a los tiempos de cómputo versus la cantidad de *restarts*, se obtuvieron los siguientes resultados:

	10 restarts	50 restarts	100 restarts	250 restarts	500 restarts	750 restarts	1000 restarts
albano	18,3515	18,2907	18,2419	18,2259	18,1568	18,1238	20,3166
blaz	2,27639	2,30226	2,39546	2,39864	2,30139	2,38586	2,40228
dagli	3,3289	3,36758	3,49425	3,32363	3,31898	3,29032	3,35511
jakobs1	1,70344	1,62953	1,60917	1,64551	1,61634	1,7279	1,6914
jakobs2	1,59285	1,59502	1,61794	1,65873	1,58695	1,60142	1,74834
marquez	1,70693	1,71407	1,71194	1,71974	1,71035	1,72628	1,79269
poly2a	17,4715	17,5201	17,473	17,4734	17,3631	17,163	17,3534
poly3a	63,6189	63,4119	64,1949	64,5257	64,4862	63,5241	64,1475
poly4a	152,446	151,576	151,313	152,823	153,252	150,472	152,486
poly5a	298,61	300,798	297,842	297,664	299,261	296,621	301,074
shirts	157,103	156,585	157,312	158,291	164,057	158,98	157,428
swim	155,772	157,631	153,983	156,502	156,991	156,874	157,611
trousers	46,6182	46,6633	46,1749	47,3435	46,6263	46,4606	45,8416

Cuadro 3: Tiempos de cómputo (en segundos) de experimentos realizados

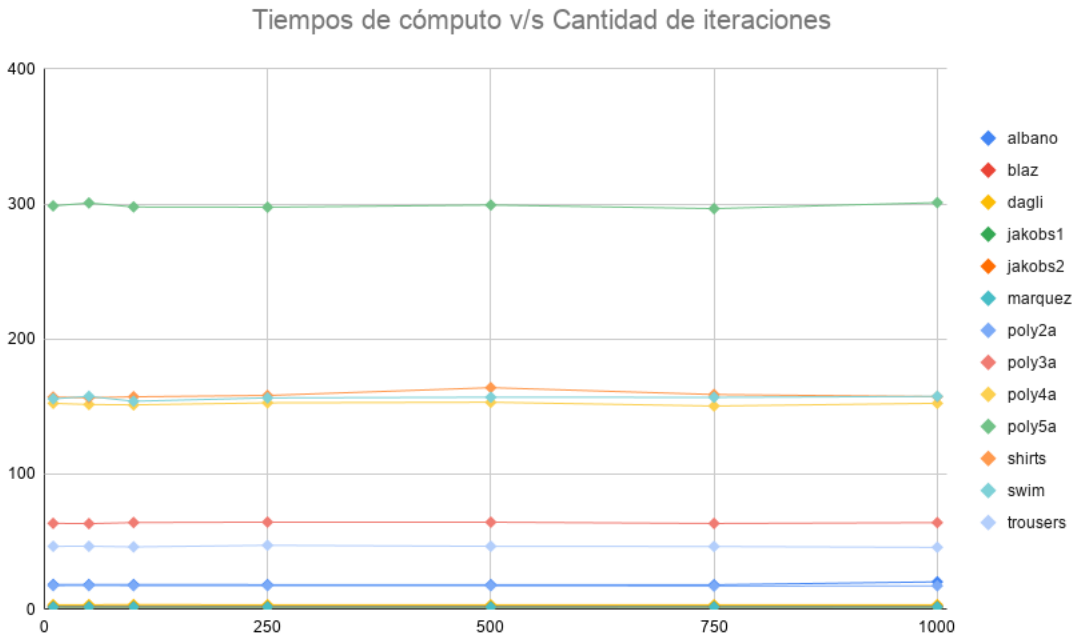


Figura 26: Gráfico de tiempo de cómputo versus cantidad de iteraciones

En 26 se grafican los datos del cuadro 4. Queda en evidencia que a pesar del aumento en la cantidad de iteraciones, los tiempos de cómputo son *constantes*.

En el cuadro 4 se comparan los largos obtenidos versus los largos entregados en las instancias. También se entregan los porcentajes de mejora y el tiempo de cómputo.

	Mejor resultado obtenido	Largo entregado	% de mejora	Tiempo de cómputo [s]
albano	13243	29000	54,33	18,1568
blaz	39	90	56,67	2,27639
dagli	101	200	49,50	3,35511
jakobs1	19	200	90,50	1,60917
jakobs2	42	250	83,20	1,61794
marquez	112	400	72,00	1,72628
poly2a	4996	5000	0,08	17,5201
poly3a	7099	7000	-1,41	64,5257
poly4a	9593	8000	-19,91	152,486
poly5a	11892	10000	-18,92	298,61
shirts	90	1000	91,00	156,585
swim	10818	50000	78,36	157,611
trousers	365	1000	63,50	46,6182

Cuadro 4: Largos obtenidos versus Largos entregados

Nótese que sólo para las instancias `poly3a`, `poly4a` y `poly5a` no se obtienen resultados satisfactorios. Para el resto de las instancias se obtienen soluciones de calidad en tiempo de cómputo bajos.

En [11] se presentan los mejores resultados obtenidos para cada instancia junto con sus respectivos tiempo de cómputo. En el cuadro 5 se realiza la comparación con los resultados obtenidos en el presente trabajo:

	Mejor resultado obtenido	Tiempo de cómputo [s]	Mejor resultado en [11]	Tiempo de cómputo [s]	% de mejora de solución	% de mejora de tiempo
albano	13243	18,1568	10032,24	124,39	-32,00	85,40
blaz	39	2,27639	25,41	25,42	-53,48	91,04
dagli	101	3,35511	56,9	139	-77,50	97,59
jakobs1	19	1,60917	12,99	19,3	-46,27	91,66
jakobs2	42	1,61794	24,25	53,67	-73,20	96,99
marquez	112	1,72628	84,65	58,11	-32,31	97,03
poly2a	4996	17,5201	26,16	87,53	-18997,86	79,98
poly3a	7099	64,5257	39,01	773,32	-18097,90	91,66
poly4a	9593	152,486	50,99	1621,54	-18713,49	90,60
poly5a	11892	298,61	63,66	1773,85	-18580,49	83,17
shirts	90	156,585	62,19	1808,73	-44,72	91,34
swim	10818	157,611	5661,95	431,97	-91,06	63,51
trousers	365	46,6182	249,35	603,36	-46,38	92,27

Cuadro 5: Comparación con mejores resultados en literatura

Nótese que no se obtienen mejores resultados que los publicados en [11]. No obstante, los tiempo de cómputo son mucho mejores. Por lo tanto, el algoritmo presentado en este documento podría emplearse cuando el recurso tiempo es limitado y se puede hacer un *trade-off* con la calidad de la solución (por ejemplo, para las instancias *albano* y *marquez* la pérdida de calidad es razonable).

9. Conclusiones

Tras realizar este documento, es claro que ISPP es uno de los problemas C&P más difíciles de resolver, por lo que los algoritmos y heurísticas desarrolladas hasta el momento son un gran avance hacia la búsqueda de soluciones óptimas en tiempos de cómputo razonables.

En este documento se presentó un algoritmo basado en *Hill Climbing* con Alguna Mejora, y tras los experimentos realizados y los resultados obtenidos se desprenden las siguientes conclusiones:

- Se evidencia experimentalmente el estancamiento de HC en óptimos locales.
- El enfoque de los rectángulos contenedores sumado a la heurística BL entrega resultados de calidad en tiempos de cómputo bajo (en la mayor parte de las instancias y sacrificando la calidad de la solución). La eficiencia en tiempo de cómputo se debe en gran parte al enfoque alguna mejora, ya que no se construye completamente el vecindario de la solución candidata, sino que se construye paulatinamente.
- El tiempo de cómputo es constante respecto a la cantidad de iteraciones (*restarts*).
- Se evidencian experimentalmente los beneficios de las heurísticas: obtener soluciones de calidad en tiempo de cómputo bajos.
- También se evidencia experimentalmente una de las principales ventajas de las técnicas incompletas, que es que permiten obtener soluciones sin recorrer todo el espacio de búsqueda.
- Se experimentó la complejidad de la búsqueda de soluciones óptimas de un problema NP-Completo.
- AL desarrollar el algoritmo se verificó la importancia de dotarlo de métodos de exploración y explotación para obtener buenas soluciones.

Para futuros trabajos, se puede considerar el efecto de añadir ángulos de rotación permitidos, así como el diseño de un algoritmo que, en base al patrón que entrega el algoritmo desarrollado aquí, permita acercar los polígonos entre sí lo máximo posible sin que se superpongan entre sí.

Referencias

[1] R. Alvarez-Valdes, A. Martinez, and J.M. Tamarit. A branch & bound algorithm for cutting and packing irregularly shaped pieces. *International Journal of Production Economics*, 145(2):463 – 477, 2013.

[2] Richard Art. An approach to the two dimensional irregular cutting stock problem. *Massachusetts Institute of Technology. Alfred P. Sloan School of Management Thesis*, 1966.

[3] Julia A. Bennell and Xiang Song. A beam search implementation for the irregular shape packing problem. *Journal of Heuristics*, 16(2):167–188, 2010.

[4] Luiz H. Cherri, Leandro R. Mundim, Marina Andretta, Franklina M.B. Toledo, José F. Oliveira, and Maria Antónia Carravilla. Robust mixed-integer linear programming models for the irregular strip packing problem. *European Journal of Operational Research*, 253(3):570 – 583, 2016.

[5] Luiz Henrique Cherri, Maria Antónia Carravilla, and Franklina Maria Bragion Toledo. A Model-based Heuristic for the Irregular Strip Packing Problem. *Pesquisa Operacional*, 36:447 – 468, 12 2016.

[6] Harald Dyckhoff. A typology of cutting and packing problem. *European Journal of Operational Research*, 44(2):145 – 159, 1990.

[7] A. Miguel Gomes and José F. Oliveira. Solving Irregular Strip Packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171(3):811 – 829, 2006.

[8] E. Hopper and B. C. H. Turton. A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems. *Artificial Intelligence Review*, 16(4):257–300, 2001.

[9] Toshihide Ibaraki, Shinji Imahori, and Mutsunori Yagiura. Hybrid Metaheuristics for Packing Problems. *Hybrid Metaheuristics: An Emerging Approach to Optimization*, pages 185–219, 2008.

[10] Stefan Jakobs. Theory and methodology on genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165 – 181, 1996.

[11] Jeinny Peralta, Marina Andretta, and José Fernando Oliveira. Solving Irregular Strip Packing Problems with Free Rotations using Separation Lines. *Pesquisa Operacional*, 38:195 – 214, 08 2018.

[12] Marcos Okamura Rodrigues and Franklina M.B. Toledo. A clique covering MIP model for the irregular strip packing problem. *Computers & Operations Research*, 87:221 – 234, 2017.

[13] Miguel Cezar Santoro and Felipe Kesrouani Lemos. Irregular packing: MILP model based on a polygonal enclosure. *Annals of Operations Research*, 235(1):693–707, 2015.