Capítulo 3

MÉTODOS HEURÍSTICOS PARA LA SOLUCIÓN

3.1. INTRODUCCIÓN

En el capítulo anterior se han definido los distintos tipos de problemas de corte y se han clasificado según sus estructuras y características, en este capítulo nos centraremos en el problema de corte bidimensional con formas irregulares.

La mayoría de los algoritmos de disposición heurísticos se pueden clasificar en dos grandes grupos: constructivos y genéticos. En este capítulo se comentan las características de cada grupo, después se exponen algunos ejemplos de algoritmos que se encuentran en la literatura.

También se estudiarán durante este capítulo los métodos que se utilizan en los ejemplos para resolver los solapamientos como la suma de Minkowski o los algoritmos de llenado inferior izquierdo.

3.2. ALGORITMOS HEURÍTICOS

En problemas de optimización combinatoria, el objetivo es optimizar una determinada función sujeta a un conjunto de condiciones, no se trata de encontrar la solución ideal, es por esto que el dominio de soluciones factibles, es en general, de gran tamaño. En los problemas de corte bidimensional se pueden generar muchas disposiciones distintas variando en apenas milímetros la posición de cada pieza sobre la hoja. En los últimos años estos problemas han sido abordados con métodos de búsqueda heurísticos. Éstos, visitando apenas una parte del dominio son capaces de encontrar soluciones suficientemente eficientes. En general, tales métodos se pueden clasificar en constructivos y genéticos. Los métodos constructivos, generan de forma incremental la solución para un problema, enumerando implícitamente el dominio de soluciones factibles. Su eficiencia está relacionada con los criterios de selección adoptados. En general, requieren de un gran esfuerzo computacional, tanto en tiempo de procesamiento como, en memoria requerida. Por otro lado, los métodos genéticos, generan cada vez una nueva solución factible a partir de una evolución de la generación de la solución actual. Aunque no garantizan la determinación de una solución óptima, se han mostrado eficientes en la resolución de una gran cantidad de problemas. Por su operatoria, requieren de poca memoria estando el tiempo computacional asociado a la calidad de la solución requerida. A continuación se exponen algunos ejemplos de ambos métodos.

3.3. MÉTODOS CONSTRUCTIVOS

3.3.1. Algoritmo para el Anidado Bidimensional de Piezas Irregulares Basado en la Suma de Minkowski [2]

Este apartado comienza analizando la suma de Minkowski y comprobando como puede ser de utilidad para los problemas de corte bidimensional con piezas irregulares.

En la segunda parte se describe el algoritmo de disposición empleado.

3.3.1.1. La suma de Minkowski

La suma de Minkowski es un útil instrumento en el campo de la geometría computacional. Se utiliza extensamente en planificación del movimiento de robots y análisis de imagen. Utilizando la idea de la suma de Minkowski, una prueba de cruce de polígonos se puede simplificar a una prueba de ubicación de un punto en un polígono. La definición de la suma de Minkowski y el text para el cruce entre dos polígonos se representa abajo gráficamente. Dados dos polígonos, *A* y *B*, su suma de Minkowski $A \oplus B$ es el conjunto de todas sumas vectoriales generadas por todos pares de puntos en *A* y *B* como se define en la ecuación siguiente: $A \oplus B = \{a + b | a \in A, b \in B\}$ donde a y b son los puntos arbitrarios en A y B respectivamente y a + b es el punto que representa la suma vectorial de estos dos puntos. La Figura 3.1 (a) muestra la suma de Minkowski de dos polígonos gráficamente. El área clara de cada polígono representa la parte interior del polígono. En la adición vectorial se consideran todos los puntos en los interiores así como en las fronteras de los dos polígonos. El conjunto de sumas de vectoriales forma el polígono de la suma de Minkowski y su interior se representa por el área oscura.

La diferencia de Minkowski $A \oplus (-B)$ se utiliza para discernir cruces entre polígonos *A* y *B*, donde $(-B) = \{-b | b \in B\}$ es el polígono B girado 180º alrededor del origen (0,0). Como se muestra en la Figura 3.1 (b), el polígono se localiza en el cuadrante inferior izquierdo del sistema de coordenadas. El polígono oscuro es la diferencia de Minkowski $A \oplus (-B)$ de los polígonos *A* y *B*, que incluye el origen. Si existen dos puntos de los polígonos *A* y *B*, en que *a* = *b*, el vector suma *a* + (-*b*) es igual a cero. Se observa que los polígonos *A* y *B* se cruzan si y sólo si $A \oplus (-B)$ contiene el origen.

Figura 3.1. (a) Suma de Minkowski. (b) Detectar intersecciones entre polígonos [2].

Con el uso de la diferencia de Minkowski el cruce de polígonos se discierne fácilmente. En este texto, inicialmente se superponen dos piezas y se computa la diferencia de Minkowski correspondiente. Se computa la distancia de traslación entre las dos partes según la geometría de la suma de Minkowski. Entonces se traslada una pieza con respecto a otra pieza para eliminar el solapamiento entre ellas.

El algoritmo para computar la suma de Minkowski de dos polígonos se realiza a través de la descomposición inicial del polígono en subpolígonos convexos. Con la descomposición de los polígonos se computan los detalles de la parte interna de la suma de Minkowski.

En algunos trabajos recientes, la idea de la "suma de Minkowski" se introduce para ayudar a resolver los problemas de nidificación. La suma de Minkowski es el área generada por la adición de dos áreas y este concepto básico es computacionalmente similar al problema cuya frontera se define como el polígono de no conveniencia (NFP). La geometría de la suma de Minkowski es útil en discernir los cruces de las piezas. Utilizando el polígono de no conveniencia, se puede trasladar una pieza con respecto a otra a unas coordenadas donde estas acaban tocándose la una a la otra y sin superponer.

Existen varios algoritmos para encontrar las orientaciones óptimas de las piezas utilizando la idea de la suma de Minkowski. La anchura y profundidad de la hoja se computan directamente de la geometría del polígono de no conveniencia. Con el uso de la suma de Minkowski, una pieza se orienta en la hoja maximizando la utilización del material.

3.3.1.2. Aplicación de la suma de Minkowski al anidado de piezas complejas

En el diseño de la disposición para piezas complejas el problema de la nidificación se puede partir en dos etapas principales. La nidificación de dos piezas y la formación de la disposición de los pares anidados. En este texto se presenta la nidificación de piezas relativamente complejas para el diseño de la disposición. Los casos típicos de piezas complejas considerados incluyen piezas irregulares con perfiles circulares y características cóncavas. Se realiza primero la descomposición de las piezas en piezas con características cóncavas. Entonces se computa el casco de la

subsuma de Minkowski utilizando el método modificado de copia de fronteras. La unión de todos los cascos de subsuma da la suma general de Minkowski que incluye la geometría interior. En los casos donde la suma general de Minkowski de dos piezas consiste en un borde exterior y un lazo interior, se puede utilizar el lazo interior para lograr una utilización muy alta del material.

3.3.1.2.1. Cálculo del casco de la suma de Minkowski

Primero se toman las formas de las piezas. El punto situado más abajo y a la izquierda de cada forma se toma como punto de referencia. Todas las formas se trasladan para que su punto de referencia coincida con el origen y se superponen. La forma (*A*) es la forma inmóvil y la forma (*B*) es la forma deslizante. La forma (-*B*) se forma girando la forma (*B*) 180º alrededor del origen. El casco de la suma de Minkowski se define como la frontera de la suma de Minkowski.

- *Detección de los 8 puntos extremos*

Se definen cuatro posiciones extremas en el sistema de coordenadas como la más alta, más baja, más a la izquierda, y más a la derecha. Se definen ahora ocho puntos extremos utilizando las cuatro direcciones extremas dos veces. La Figura 3.2 (a) ilustra la idea de los ocho puntos extremos de un octágono. Los ocho puntos extremos se escogen entre el conjunto de los puntos del perfil del octágono. Dados los perfiles de la forma (*A*) y la forma (-*B*), se computan los puntos finales de cada segmento de la frontera y se definen como los puntos de la forma. Se escogen los ocho puntos extremos entre la lista de puntos de la forma. La Figura 3.2 (b) muestra los ocho puntos extremos marcados de la forma (*A*) y la forma (-*B*) respectivamente. Sin embargo, en la construcción del casco de la suma de Minkowski, los puntos de la forma no extremos se utilizarán todavía durante el proceso de copiado de frontera.

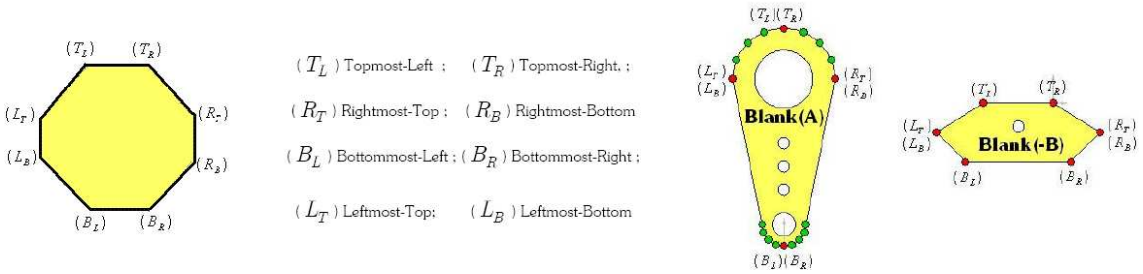


Figura 3.2. (a) Los 8 puntos extremos de un pentágono. (b) Los 8 puntos extremos de la forma (*A*) y la forma (*-B*) [2].

- Formación del dibujo inicial del casco de la suma de Minkowski

En la suma de Minkowski, un punto de la suma vectorial se crea agregando un punto del perfil de una forma a un punto del otro perfil de la otra forma. Después de realizar la adición vectorial de todas las combinaciones de puntos del perfil el resultado es un conjunto de puntos de suma vectorial. No se conoce la secuencia para conectar todas las sumas vectoriales. Por lo tanto, no es posible computar el casco de la suma de Minkowski conectando los puntos de sumas vectoriales directamente.

En la formación de la suma de Minkowski, la geometría de los perfiles de las formas es conocida. Copiando y trasladando las fronteras de los perfiles de las formas se genera un dibujo inicial del casco de la suma de Minkowski. Un punto de la traslación de fronteras es la suma vectorial de un punto de un perfil de una forma con un punto extremo del otro perfil de la otra forma. Considerando cada uno de los puntos extremos se copia un conjunto de fronteras y se traslada al punto correspondiente de traslación. Las Figuras 3.3 (a)-(c) ilustran gráficamente un ejemplo de la copia y el traslado de las fronteras de un perfil de una forma. Se genera el dibujo inicial del casco de la suma de Minkowski y se muestra en la Figura 3.3 (d). El dibujo consiste en varios segmentos de las fronteras de las formas.

El algoritmo detallado es como sigue:

1. Se determina el conjunto de puntos extremos EP_1, EP_2,..., EP_8, y se colocan en la dirección horaria del dibujo.

2. Para cada combinación de tres extremos adyacentes $EP_{i-1}, EP_i, EP_{i+1} | i \in [1..8], EP_0 = EP_8, EP_1 = EP_9$,

(i) Se copian las fronteras de un perfil de una forma entre estos puntos extremos.

(ii) Se computa el punto de la traslación como: punto de traslación = un punto extremo de un perfil de una forma + un punto de la frontera de otro perfil de forma

(iii) Se trasladan las fronteras copiadas del punto extremo al punto de traslación.

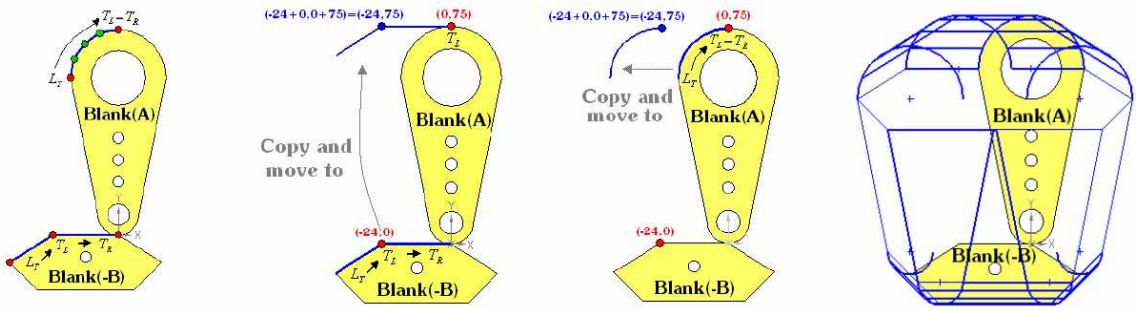


Figura 3.3. (a) Segmentos de la frontera de las formas entre los puntos L_T-T_L-T_R. (b) Copia y traslado de la frontera de la forma (-*B*). (c) Copia y traslado de la frontera de la forma (*A*). (d) Dibujo inicial del casco de la suma de Minkowski [2].

- *Dividir y escoger el contorno del casco de la suma de Minkowski*

Los segmentos de frontera en el dibujo inicial del casco de la suma de Minkowski están conectado juntos. Se determinan todos los puntos de intersección entre segmentos de la frontera. La Figura 3.4 (a) muestra el conjunto de puntos de intersección en color oscuro. La particción de los segmentos se realiza basándose en la lista de puntos de intersección. Cuándo un punto de cruce cae en un segmento de la frontera, se parte este segmento por el punto de cruce dando como resultado un conjunto de minisegmentos de frontera en el dibujo inicial del casco de la suma de Minkowski. Para reducir el tiempo de cómputo, todos los segmentos de frontera redundantes se eliminan automáticamente del dibujo.

El casco de la suma de Minkowski se genera escogiendo los segmentos del contorno en el dibujo inicial. Entre los minisegmentos de frontera, el punto más alto y a

la izquierda en el dibujo se define como el punto de partida de la selección. Escogiendo los puntos en dirección horaria, se determinan los vectores de dirección y los puntos inicial y final de cada segmento. El vector de dirección se representa por medio de un ángulo con respecto al eje *x* positivo. Empezando desde el punto más alto y a la izquierda, se escogen uno a uno un conjunto de segmentos de frontera que forman el contorno. En cada paso de selección, se comparan los vectores de dirección de los segmentos y se escoge un segmento para el contorno. La Figura 3.4 (b) ilustra el comienzo de la selección del casco de la suma de Minkowski. La Figura 3.4 (c) muestra un lazo completo del casco de la suma de Minkowski en líneas continuas y curvas. El área de color claro es la suma de Minkowski de la forma (*A*) y la forma (*-B*).

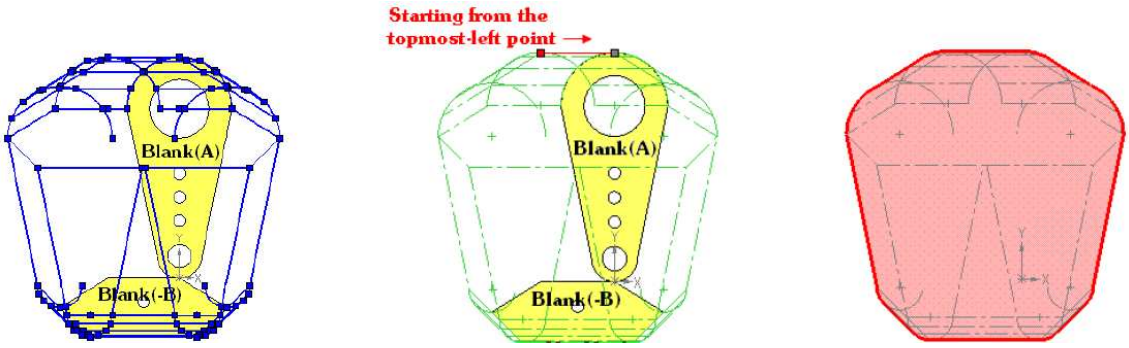


Figura 3.4. (a) Conjunto de puntos de intersección. (b) Selección del contorno de la suma de Minkowski. (c) La suma de Minkowski y su frontera [2].

3.3.1.2.2. Formación de la suma general de Minkowski con la descomposición de piezas cóncavas

En la sección previa, el casco de la suma de Minkowski de dos piezas cualquiera se genera escogiendo su contorno de los minisegmentos de frontera en el dibujo. El resultado es sólo la frontera de la suma de Minkowski y se ignora la geometría interior de la suma de Minkowski. Se pueden generar huecos en la suma de Minkowski cuándo las piezas importadas son cóncavas. La descomposición de piezas en piezas convexas se realiza por lo tanto antes de la construcción de la suma de Minkowski.

- *Algoritmo de descomposición de piezas*

Considerando los ángulos interiores de los puntos del perfil de la forma se dibujan líneas de partición sobre la forma y se utilizan para la descomposición. La Figura 3.5 muestra la forma (*A*) que se utiliza para la demostración. El algoritmo detallado es como sigue:

1. Se determina la concavidad de los puntos del perfil de la forma. El ángulo interior en un vértice cóncavo es siempre mayor que 180º. La Figura 3.5 (a) muestra dos vértices cóncavos marcados.

2. Se escoge uno de los vértices cóncavos para ser el punto de partida de una línea de partición. En las Figuras 3.5 (b) y (c) se escoge el punto 1 como el punto inicial de la línea de partición.

3. Se escoge el punto final de la línea de partición de la lista de vértices consecutivos y se computan los ángulos internos y la concavidad en los puntos inicial y final de la línea de partición.

4. Se repite el paso 3 hasta que los ángulos interiores en los puntos inicial y final sean más pequeños que 180º. La Figura 3.5 (b) muestra una pieza convexa, un rectángulo, generado en la izquierda cuando la línea de partición es "1–6". Cuando se tiene como resultado alguna pieza cóncava se procede con el paso 5. La Figura 3.5 (c) muestra un vértice cóncavo como resultado en el punto inicial 1.

5. Se “extrae” la parte convexa de la forma y se repite el algoritmo otra vez en la parte restante de la forma. Las Figuras 3.5 (d)-(e) muestran el rectángulo oscuro extraído y cómo se repite el algoritmo en resto de la forma. La forma cóncava (*A*) finalmente se descompone en tres formas convexas y se muestra en la Figura 3.5 (f).

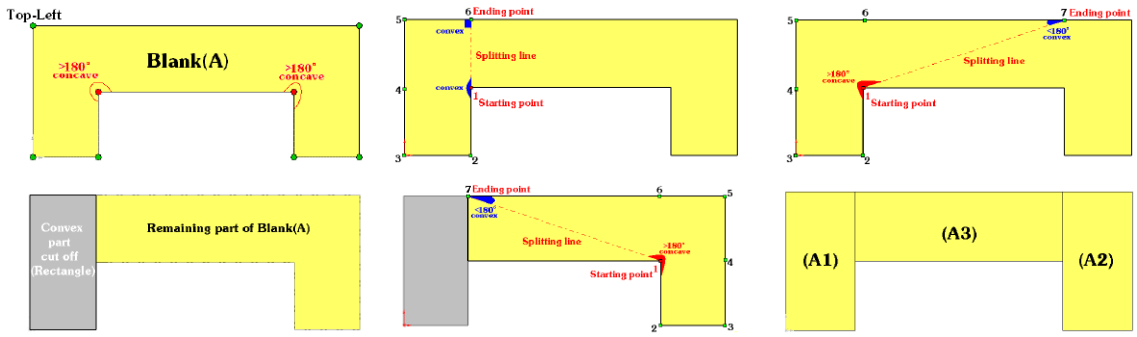


Figura 3.5. (a) Vértices cóncavos y convexos. (b) Línea de partición “1-6” (da formas convexas). (c) Línea de partición “1-7” (da formas cóncavas). (d) Se separa la parte convexa. (e) Descomposición de la parte restante. (f) La forma se descompone en tres formas convexas [2].

- *Ejemplo de la construcción de la suma de Minkowski con descomposición*

La descomposición propuesta de las formas en formas convexas se puede aplicar también a formas con fronteras curvas. La Figura 3.6 (a) muestra dos formas con fronteras curvas, la forma (*A*) y la forma (-*B*). Cada una de ellas se descompone en dos formas convexas. Se observa que sólo se utiliza el perfil de frontera en la descomposición de formas, así que no se incluye la geometría interna de las formas originales en las formas descompuestas. El casco de subsuma de Minkowski se genera realizando la adición de Minkowski en dos formas convexas, una de las formas pertenece a la pieza (*A*) y la otra a la (-*B*). Los cascos de subsuma de Minkowski generados son también convexos. Se computa un conjunto de cascos de subsuma de Minkowski para todas las combinaciones de una forma convexa de la pieza (*A*) con otra forma convexa de la pieza (-*B*). La Figura 3.6 (b) muestra la formación de los cascos de subsuma de Minkowski. La suma general de Minkowski es la unión de todos los cascos de subsuma de Minkowski. La Figura 3.6 (c) muestra la unión de todas las subsumas de Minkowski.

El algoritmo resumido es como sigue:

1. Se descomponen las formas (*A*) y (-*B*) en un conjunto de formas convexas; forma (A_1), forma (A_2),…, forma (A_m) y forma (-B_1), forma (-B_2),…, forma (-B_n) respectivamente.

2. Para cada $i \in [1..m]$ y para cada $j \in [1..n]$, se calcula la subsuma de Minkowski $MSS_{ij} = A_i \oplus (-B_j)$.

3. La suma general de Minkowski $A \oplus (-B)$ es la unión de todas las subsumas $\{MSS_{ij} \mid i \in [1..m], j \in [1..n]\}$.

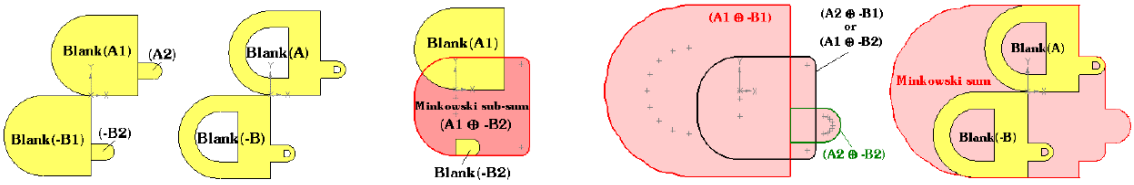


Figura 3.6. (a) Descomposición de las formas. (b) Subsuma de Minkowski. (c) Suma general de Minkowski [2].

3.3.1.3. Anidado de dos piezas

El anidado de dos formas significa agrupar las formas tan cerca como sea posible. Si se puede incluir una parte de una forma en otra forma el resultado es un par de formas anidadas. El algoritmo de anidado consiste en una serie de pasos, que son la descomposición de formas en formas convexas y la construcción de la suma general de Minkowski. La geometría de la suma general de Minkowski se utiliza para calcular los resultados del anidado. En esta sección, se detalla el algoritmo y se muestra un ejemplo de par anidado.

3.3.1.3.1. Algoritmo

El siguiente es un resumen del algoritmo de anidado de dos formas:

1. Se importan dos formas, denominadas como forma (*A*) y forma (*B*). Primero, se trasladan las formas de su punto situado más abajo y a la izquierda al origen (0,0). Se calcula la forma (-*B*) girando la forma (*B*) 180º en el origen.

2. Utilizando los algoritmos de la Sección 3.3.1.2, se descomponen las formas (*A*) y (-*B*) en un conjunto de formas convexas.

3. Se calcula el conjunto de subsumas de Minkowski y la suma general de Minkowski $A \oplus (-B)$.

4. Se genera la geometría de la suma de Minkowski, que incluye el borde exterior y el lazo interior.

5. Se determina la posición central del lazo interior.

6. Se calcula el par anidado de la forma (*A*) y la forma (*B*) trasladando la forma (*B*) del origen a la posición central del lazo interior.

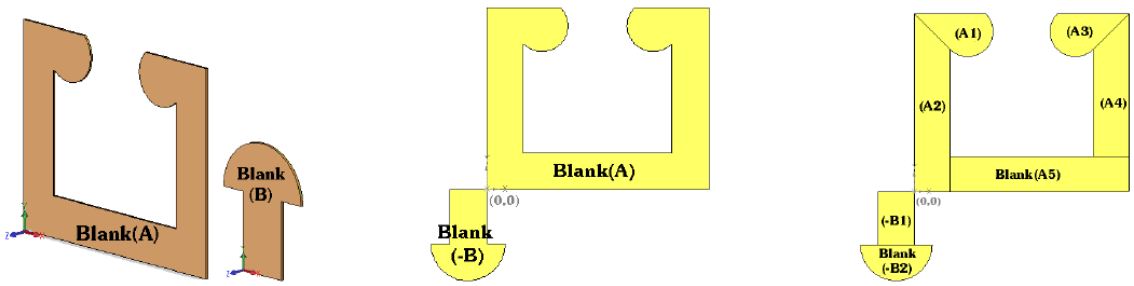


Figura 3.7. (a) Formas (*A*) y (*B*) en vista isométrica. (b) Formas (*A*) y (-*B*). (c) Descomposición de las formas (*A*) y (-*B*) en formas convexas [2].

3.3.1.3.2. Ejemplo de creación de un par anidado

En este ejemplo se utilizan dos formas, una cóncava (*A*) y una convexa (*B*), que se muestran en la Figura 3.7 (a). Se realiza la descomposición de las formas y la formación de la suma general de Minkowski. Se computa la geometría de la suma general de Minkowski y el resultado es un par anidado.

- *Descomposición de las formas (A) y (-B)*

Se importan inicialmente las formas (*A*) y (*B*) al SolidWorks y se gira la forma (*B*) 180º alrededor del origen para engendrar la forma (-*B*). Se extraen los perfiles de forma de la forma (*A*) y la forma (-*B*) y se descomponen las formas en un número de formas convexas. La forma (*A*) y la forma (-*B*) se han dividido en 5 y 2 formas convexas respectivamente. La Figura 3.7 (a)-(c) muestra los resultados de la descomposición de las formas.

- *Construcción de la suma general de Minkowski y del par anidado*

Se calcula un conjunto de subsumas de Minkowski realizando la adición de Minkowski en todas las combinaciones de dos formas convexas, con una forma convexa perteneciente a la forma (*A*) y otra a la forma (-*B*). La Figura 3.8 (a)-(b) muestra todas las subsumas de Minkowski generadas y la unión de todas las subsumas respectivamente. La suma general de Minkowski consiste en un borde exterior y un lazo interior. El lazo interior de la suma de Minkowski es la localización de los puntos trazados por el punto de referencia de la forma (*B*) en los cuales parte de la forma (*B*) se introduce en la forma (*A*). Todos los puntos dentro del lazo interior son las posiciones posibles de la referencia en las que se coloca parte de la forma (*B*) dentro de la forma (*A*) para lograr una utilización mayor del material. Se escoge el centro del lazo interior como posición optima de la forma (*B*) y el resultado es un par anidado de dos formas. En el par anidado, una de las formas se introduce en la otra dejando espacio entre ellas. La Figura 3.8 (c) muestra los resultados del anidado de la forma (*A*) y la forma (*B*) utilizando el centro del lazo interior.

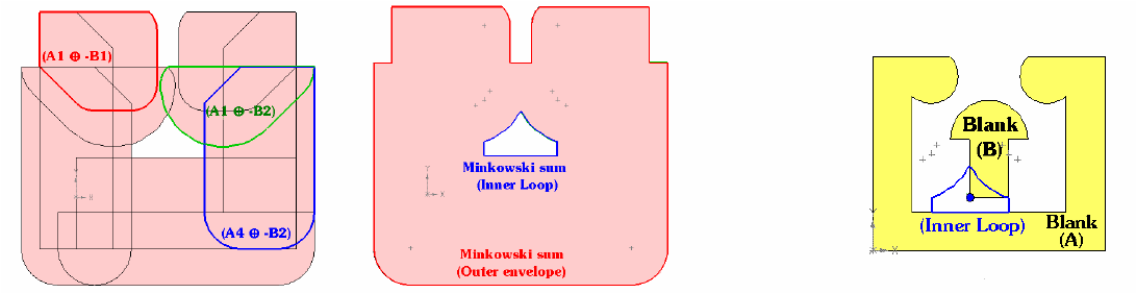


Figura 3.8. (a) Subsumas de Minkowski. (b) Suma general de Minkowski. (c) Anidado resultante de las formas (*A*) y (*B*) tomando el centro del lazo interior como posición óptima para la forma (*B*) [2].

3.3.1.4. Algoritmo de distribución

En el diseño de la disposición para las formas, se producen muchas formas idénticas en la misma hoja. La utilización material depende de la orientación de las formas. La disposición óptima de las piezas es aquella en la que la utilización del material es máxima. La geometría de la suma de Minkowski de dos formas idénticas se utiliza para calcular la disposición óptima de la forma. En esta sección, se detalla el algoritmo propuesto y se presenta un ejemplo de una pieza típica.

3.3.1.4.1. Algoritmo

El siguiente es un resumen del algoritmo de anidado de dos formas:

1. Se importa una forma al SolidWorks, denominada como forma (*A*). Primero, se trasladan las formas de su punto situado más abajo y a la izquierda al origen (0,0). Se computa la forma (-*A*) girando la forma (*A*) 180º alrededor del origen.

2. Se calcula el área A_B de la forma (*A*) utilizando la función de SolidWorks.

3. Se calcula la suma de Minkowski $A \oplus (-A)$ y se genera el borde exterior de la suma de Minkowski.

4: Según el borde exterior de la suma de Minkowski, se determina un conjunto de puntos $MSP_1, MSP_2,..., MSP_m$.

5. Para cada punto $(x_i, y_i) | i \in [1..m]$ de la suma se calculan los parámetros siguientes:

(i) *Sweepline_vector* V_i, es el vector conectado desde origen al punto de la suma de Minkowski MSP_i (x_i, y_i).

(ii) *Strip_pitch* $p_i = \sqrt{x_i^2 + y_i^2}$, es la distancia entre el origen y el punto de la suma de Minkowski MSP_i (x_i, y_i).

(iii) *Strip_width* w_i, es la máxima distancia perpendicular entre puntos del borde exterior y el *sweepline_vector* V_i.

(iv) *Strip_area* $A_{S_i} = p_i \times w_i$.

(v) *Material_utilization* $\rho_i = \dfrac{A_B}{A_{S_i}}$.

6. Se calcula el punto óptimo de la suma de Minkowski $(x_t, y_t) | t \in [1..m]$ en el que el valor de *material_utilization* es el más grande.

7. Se traslada la forma en la dirección del *sweepline_vector* $V_t | t \in [1..m]$.

- *Ejemplo de construcción de una disposición*

La Figura 3.9 (a) muestra una forma, la forma (A), que se utiliza en el ejemplo. Utilizando los algoritmos de la Sección 3.3.1.2, se computa la geometría de la suma de Minkowski y se muestra en la Figura 3.9 (b). Para cada punto de la suma de Minkowski en el borde exterior, se calculan los parámetros de anidado incluyendo *strip_pitch*, *strip_width* y la *material_utilization*. El valor óptimo para el *Material_utilization* (68,4%) se obtiene cuando el ángulo del *sweepline_vector* es de 63,08º como se muestra en los cálculos siguientes:

$A_B = 1650.5 mm^2$

$A_{S_t} = p_t \times w_i = 47.49 \times 51.41 = 2441.46 mm^2$

$\rho_t = \dfrac{A_B}{A_{S_t}} = \dfrac{1650.5}{2411.46} = 68.4\%$

La Figura 3.9 (c) muestra la disposición óptima de tres piezas de la forma (A).

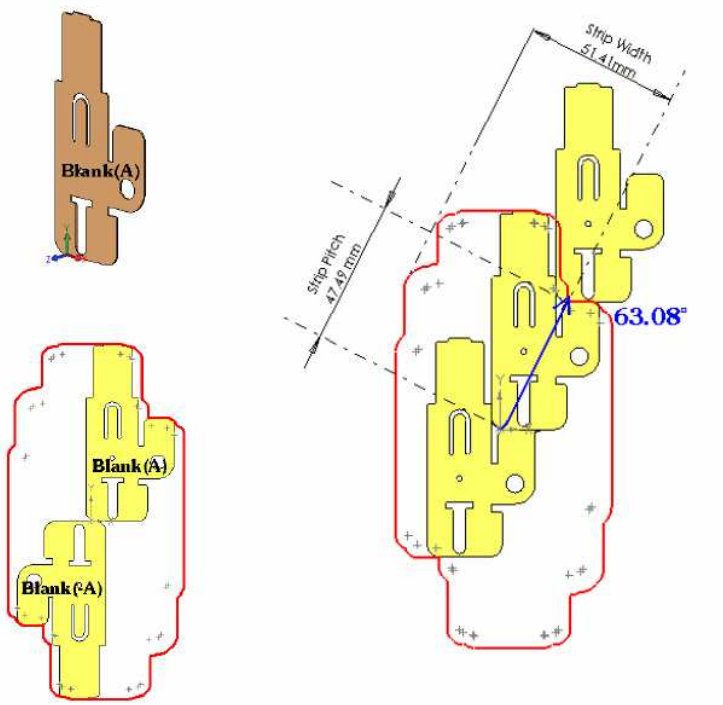


Figura 3.9. (a) Forma (*A*) en vista isométrica. (b) Suma de Minkowski. (c) Distribución ópima de 3 piezas de la forma (*A*) con un *sweepline_vector* de ángulo 63.08º [2].

3.4. ALGORITMOS GENÉTICOS

3.4.1. Sistema Integrado de Visión de Máquina para el Corte de Stock No Convexo con Algoritmos Genéticos [3]

El siguiente algoritmo requiere un paso previo de discretización de las superpies en formas poligonales, o proceso de poligonalización, con el objetivo de poder representar las piezas simplemente mediante las longitudes de sus lados y los ángulos que forman dos lados adyacentes.

En este apartado sólo veremos los algoritmos de disposición utilizados dado que el sistema de captación y el proceso de poligonalización carecen de importancia para este proyecto.

3.4.1.1. Base de datos de las piezas

Tras utilizar el sistema de captación para obtener las formas y haberlas poligonalizado, las siguientes líneas explican cómo dar formato a los datos obtenidos para ser incluidos en una base de datos de las piezas y cómo descomponer las piezas cóncavas en piezas convexas.

La información de los vértices se almacena de forma libre. Uno de los vértices del polígono se escoge primero arbitrariamente y se considera un vértice distintivo. La pieza entonces se gira hasta que uno de los segmentos que proceden del vértice distintivo coincide con el eje *x* de referencia. El segmento se escoge de forma que permita atravesar la frontera en dirección contraria a las agujas del reloj. Un polígono puede entonces ser descrito inequívocamente de forma libre especificando la longitud del segmento distintivo (el que procede del vértice distintivo y que fue escogido para alinearse con el eje *x*) y especificando la longitud los segmentos y los ángulos entre segmentos sucesivos (ver Figura 3.10). Tal descripción evita especificar cualquier eje de coordenadas.

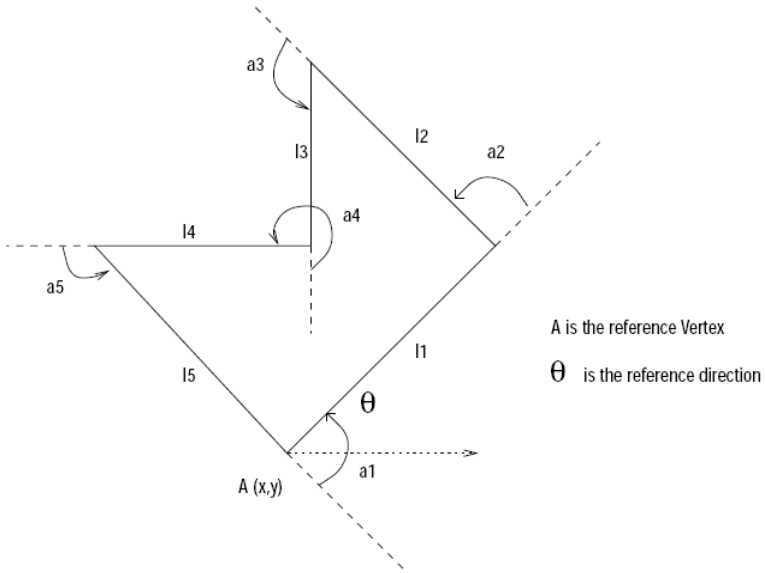


Figura 3.10. Descripción intrínseca de una pieza [3].

Todas las piezas se procesan tal como se plantea arriba, y la información se almacena en una base de datos de las piezas (llamado archivo neutro). A cada tipo de pieza se le da un número inequívoco de pieza (part number) que puede ser utilizado para

nombrar los polígonos. De la misma forma, las hojas procesadas por el sistema de visión son almacenadas en una base de datos de las hojas. Como se describe en la Sección 3.4.1.2, el algoritmo de disposición requerirá alguna información adicional con respecto a las piezas y hojas: el área, el centro de masa, y la descomposición en el subpolígonos convexos. Toda esta información se calcula y se almacena en el archivo neutro antes de aplicar el algoritmo de disposición.

Si una pieza no es convexa, se descompone primero en pedazos convexos para la manipulación matemática. El algoritmo empleado de descomposición trabaja en dos pasos:

1. La descomposición de polígonos simplemente conectados en ángulos cóncavos en subpolígonos de no descomponibles (polígonos convexos o espirales).
2. La descomposición de polígonos espirales en polígonos convexos.

Se asume que cada polígono consiste en dos tipos de esquinas: esquinas-a y esquinas-b. Un vértice C se define como una esquina-a si el ángulo de C en el polígono es más pequeño que 180° (es un vértice convexo); sino es una esquina-b. Un polígono es capaz de descomponerse por el paso 1 si tiene por lo menos dos segmentos-b (un segmento que pertenece a una esquina-b). Un polígono con sólo un segmento-b es un polígono espiral, que se considera polígono primitivo para el primer proceso de la descomposición.

Una vez que el polígono ha sido descompuesto en subpolígonos mediante el paso 1, puede tener o uno o ningún segmento-b. Si tiene un segmento-b, se divide en polígonos convexos hasta que no hay más subpolígonos con vértices cóncavos. La información con respecto a los polígonos convexos no descomponibles con el polígono original se almacena en el archivo neutro. Finalmente, los datos originales de la frontera se conservan ya que serán necesarios para el para el corte automático de las piezas.

3.4.1.2. Método de disposición: el algoritmo genético

Los problemas de corte de stock son realmente una constelación de problemas, y el enfoque genético desarrollado es un enfoque flexible, capaz de ser adaptado a muchas variantes al del problema de corte de stock. Para ilustrar esta flexibilidad, y aclarar la

estructura básica del algoritmo, el algoritmo se describe con todo detalle para su aplicación más sencilla: Colocar piezas en una sola hoja rectangular. La Sección 3.4.1.3 muestra cómo el algoritmo puede ser modificado para manejar muchas variantes diferentes del problema.

3.4.1.2.1. Algoritmos genéticos

Un algoritmo genético opera emulando los mecanismos genéticos de la selección natural. Se da una codificación genética a los elementos del espacio del estado, y se define una función objetivo. Una primera generación, que consiste en múltiples soluciones de prueba, se engendra al azar, se evalúa la función objetivo en cada una, y se ordenan según su eficiencia. Para producir una nueva generación se utilizan los procesos genéticos de clonación, recombinación, y mutación. Este proceso de evaluación y reproducción se itera hasta que se satisface alguna condición de parada. Para poder aplicar la estrategia genética a un problema, se tienen que determinar varias características:

- *Codificación genética*. En cualquier problema de optimización, hay un espacio del estado *S* que consiste en los estados o configuraciones admisibles del problema. Cualquier representación coordenada de este espacio del estado puede ser vista como una codificación genética del problema. Típicamente, las coordenadas naturales asociadas con un problema de optimización implican limitaciones. Si estas limitaciones no son preservadas naturalmente por los procesos genéticos (especialmente por el proceso de recombinación, descrito abajo), entonces los algoritmos genéticos no se pueden aplicar directamente a ese espacio del estado. Sin embargo, no es necesario que la codificación genética de un problema implique a las variables originales del problema. Si se supone un espacio auxiliar *A* y una función de transformación $g: A \rightarrow S$ con la propiedad de que *S* "entra" en *A* (es decir, para cada estado $s \in S$ existe un estado $a \in A$ tal que $g(a) = s$). Entonces *A* sirve también para codificar del espacio porque cada estado de *A* corresponde con un único estado actual $s = g(a)$ en *S*. En particular, con una elección apropiada de *A*, es posible reducir la complejidad del espacio del estado y mover esa complejidad a la función *g*.

- *Función de Eficiencia*. Se asume que el problema de optimización toma la forma de encontrar un extremo global de alguna función f: $S \rightarrow R$. Más allá de la suposición básica de que existe ese extremo, no hay más restricciones en la función. Si un espacio del estado auxiliar A se introduce para simplificar la codificación del problema, entonces la función de eficiencia se convierte en la composición $A \xrightarrow{g} S \xrightarrow{f} R$. Efectivamente, esto transfiere la complejidad del problema de la descripción del espacio del estado a la función de evaluación.

- *Tamaño de generación*. Mientras la complejidad del espacio del estado y la función de eficiencia determinan la complejidad del algoritmo genético, el tamaño del algoritmo está determinado por el tamaño de generación: el número de soluciones producidas en cada etapa. Este es un parámetro que debe ser indicado al principio del problema. Siguiendo la heurística común, el tamaño de la generación se pone igual a la dimensión del espacio del estado empleado.

- *Procedimiento de clonación*. Dado que el propósito del algoritmo genético es el de producir soluciones con eficiencia alta, las mejores soluciones de cada generación se transmiten a la próxima generación. Este proceso, conocido como clonación o reproducción elitista, garantiza que cuando se encuentra una solución de eficiencia alta no se desecha a menos que sea reemplazada por una solución de eficiencia aún más alta. En cada generación, las soluciones se clasifican según su eficiencia, y una proporción fija con valores más altos de eficiencia se transmite a la próxima generación. La proporción escogida (*clone_rate*) es uno de los parámetros que se debe imponer en el problema. Si la tasa de la clonación es demasiado baja se pueden perder soluciones valiosas; si la tasa de la clonación es demasiado alta, se suprime la diversidad y no pueden surgir nuevas soluciones. En este texto se utiliza una tasa de la clonación del 20%.

- *Procedimiento de Recombinación*. Una vez que las *generation_size* × *clone_rate* mejores soluciones de una generación han sido escogidas, las *(1 – clone_rate)* × *generation_size* restantes se producen por recombinación. Este es el rasgo característico del enfoque del algoritmo genético: la combinación de datos (información genética) de dos soluciones diferentes para producir una nueva solución. Este procedimiento de

recombinación se puede llevar a cabo de muchas maneras. El procedimiento de recombinación adoptado es el paso de Bernoulli. Se fija un parámetro entre 0 y 1, el *crossover_threshold*. Entonces, con cada generación, se producen *(1 - clone_rate)* × *generation_size* "niños" para llenar la próxima generación. Estas nuevas soluciones se producen de la forma siguiente. Primero se escogen un par de soluciones al azar (s_i, s_j; los "padres"). Si cada uno de éstos se describe en coordenadas: $s_i = (x_{i1}, x_{i2},..., x_{im})$, $s_j = (x_{j1}, x_{j2},..., x_{jm})$, entonces la operación de paso se lleva a cabo escogiendo números al azar $r_1,..., r_m$. Si r_k está por encima del umbral de paso (*crossover_threshold)*, las variables x_{ik} y x_{jk} se cambian; si r_k está por debajo del umbral, no. De esta manera se forman dos nuevas soluciones (los "niños"). Se evalúa la función de eficiencia en ambas, y la solución más eficiente es trasladada a la próxima generación (la otra se desecha). En este texto, el umbral de paso considerado es del 50%.

- *Procedimiento de mutación*. Los procedimientos de clonación y recombinación modifican la información genética producida en la primera generación, pero en ellos no se tiene en cuenta la introducción de nueva información genética. Para asegurar un alcance más completo del espacio del estado, se introducen nuevas soluciones en cada generación. Este proceso es conocido como mutación o inmigración. Después de que los procesos de clonación y recombinación hayan engendrado una nueva generación, estas soluciones son clasificadas por la función de eficiencia. Un porcentaje fijo, determinado por el parámetro immigration_rate, con eficiencias más bajas son borradas y reemplazadas por soluciones engendradas al azar completamente nuevas.
Estas son clasificados igualmente por eficiencia y combinadas con las otras soluciones para completar la generación.

El algoritmo se resume en el diagrama de flujo en la Figura 3.11. Entre los parámetros que se han de fijar están los siguientes:

generation_size = dim *A*
clone_rate = 20%
crossover_threshold = 50%
immigration_rate = 20%

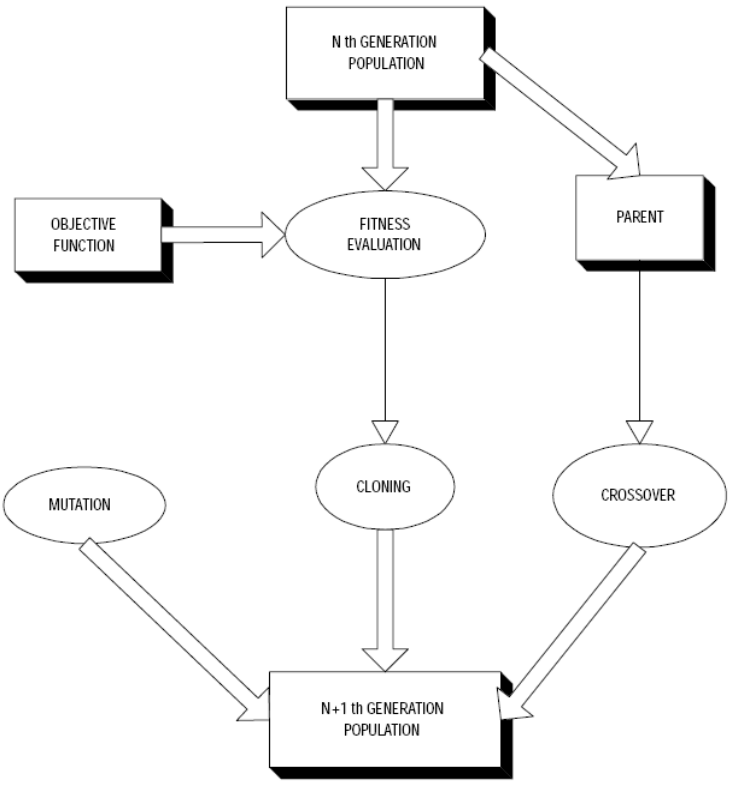


Figura 3.11. Diagrama de flujo del algoritmo genético [3].

3.4.1.2.2. Disposiciones y seudo-disposiciones

Al desarrollar un algoritmo genético para corte de stock, la principal dificultad es crear una codificación genética efectiva para el problema. En particular, se debe crear un espacio con una estructura interna relativamente sencilla para que se puedan aplicar las operaciones genéticas, además, se debe definir una función objetivo. Una vez que se han determinado, el algoritmo se puede aplicar fielmente. En esta sección, se construyen la codificación genética y la función objetivo para el problema; en la Sección 3.4.1.2.3 se completa la descripción del algoritmo de la disposición.

Para describir una disposición se deben especificar las piezas que han de ser colocadas y dónde tienen que ser colocadas. Se asume que la hoja rectangular ha sido orientada con sus lados a lo largo de los ejes positivos x e y. Aún más, como se describió en la Sección 3.4.1.1, se asume que, para cada tipo de pieza, se han escogido un vértice distintivo y un segmento distintivo. Si se han de colocar N piezas, se deben especificar $4N$ coordenadas, estando cada pieza descrita por una cuádrupla de números

(τ_i, x_i, y_i, φ_i). La variable τ_i da el tipo de pieza, (x_i, y_i) dan las coordenadas del vértice distintivo, y φ_i da el ángulo entre el eje *x* positivo y el segmento distintivo.

El problema de esta representación es que hay muchas limitaciones en estas variables, algunas de ellas bastante complicadas. Primero, se debe asegurar que las piezas respetaran las restricciones impuestas por la tarjeta de fabricación. Mientras esta limitación es fácil de formular, las desigualdades resultantes presentarán un inconveniente para la formulación genética del algoritmo. Una limitación mucho más grave está en saber qué configuraciones deben ser excluidas por producir superposición entre del las piezas. Esto es muy difícil de hacer, las restricciones en la ubicación y la orientación de las piezas impuestas por la necesidad de evitar las superposiciones crean un espacio de configuración extraordinariamente compleja. Para esquivar estos problemas, se crea un espacio auxiliar, cuyos elementos son las seudo-disposiciones. Estas seudo-disposiciones son más fáciles de manipular por el algoritmo genético y se pueden convertir en disposiciones reales. Para crear una seudo-disposición, se utilizan unas claves de aproximación aleatorias para determinar el orden en que se escogen las piezas para su colocación. Entonces se colocan las piezas sucesivamente en la hoja sin considerar superposiciones. Si la pieza que ha de ser colocada se superpone con cualquiera de las piezas previamente colocadas, se mueve hasta que no se superponga. En este proceso, puede ser necesario mover la pieza parcialmente o completamente fuera de la hoja. Una vez que todas las piezas han sido colocadas y han sido movidas, aquellas que permanecen completamente en la hoja forman una disposición verdadera.

En el problema de una sola hoja, no se sabe a priori cuántas piezas se colocarán en la hoja. Para crear una seudo-disposición, se comienza con una estimación superior del número de piezas. La estimación más natural es la de computar el área media de las piezas en la tarjeta de fabricación y estimar el número de piezas necesarias para llenar completamente la hoja de la manera siguiente:

$$N = \frac{A_S T}{\sum A_i T_i}$$

En la tarjeta de fabricación de la Figura 3.12 hay T = 32 piezas, con un tamaño medio de $\sum_{i=1}^{8} A_i T_i / T = 4603$. Para una hoja rectangular con área A_s = 105.000 se puede estimar que N = 22,80 que se redondea a N = 23. N resultará una cantidad fundamental que controlará el tamaño (y de ahí la velocidad de la ejecución) del algoritmo.

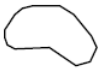
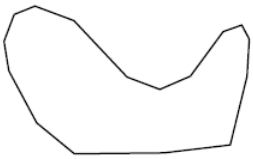
Part Type	Part Drawing	Copies Ti	Area Ai
1		5	2193.6
2		5	4256.8
3		5	6395.1
4		3	13817.2
5		3	3314.5
6		3	3586.4
7		5	1311.3
8		4	3592.4

Figura 3.12. Requerimientos de fabricación e información disponible en la base de datos de las piezas [3].

Una vez que se ha decidido cuántas piezas se va a procurar colocar en la hoja, se utiliza una clave aleatoria para determinar qué N piezas han de ser colocadas y en qué orden. Una seudo-disposición consiste entonces en estas N piezas, colocadas

sucesivamente en la hoja. Cuando se coloca cada pieza, se aplica un algoritmo de intersección entre polígonos para determinar si la pieza se cruza con cualquier pieza previamente colocada. Si hay superposición se elimina con un algoritmo de traslación, que se detalla abajo. Si no hay superposición, se aplica el algoritmo de traslación para trasladar la pieza lo más cerca posible al centro de masa de las piezas previamente colocadas. Esto asegura que la posición final de la pieza toque, pero no se superponga, una pieza previamente colocada. El proceso de clave aleatoria para escoger las piezas, el algoritmo de intersección entre polígonos, y el algoritmo de traslación para eliminar superposiciones se describen en las próximas tres secciones.

- *Claves aleatorias para los tipos de piezas*

La construcción de la seudo-disposición da una representación de la posición que suprime las complicadas limitaciones necesarias para describir las disposiciones reales. La selección de tipos de piezas requiere también algunas limitaciones, e igualmente se introducirá una nueva construcción auxiliar para suprimir esas limitaciones. Esta no es la única limitación de los tipos de piezas: para cada tipo de pieza t_i, no se pueden colocar más piezas que T_i. Mientras estas limitaciones se pueden formular directamente en términos de τ_i hay una desventaja en esta aproximación: las limitaciones no se preservaran en la operación de paso. Por ejemplo, si la tarjeta de fabricación consiste en sólo dos tipos de piezas, con una pieza de cada tipo, los dos valores admisibles para (τ_1, τ_2) son (t_1, t_2) y (t_2, t_1). Pero, si se aplica la operación de paso a éstos, se podrían producir los niños (t_1, t_1) y (t_2, t_2), ninguno de los cuales es admisible.

Para salvar este problema, se usa una aproximación de claves aleatorias. En la tarjeta de fabricación se fija un orden de los tipos de pieza. Este orden será fijo a través del algoritmo genético entero. Para escoger las N piezas en orden, se escogen N números r_1,..., r_N entre 0 y 1. Las piezas se escogen en orden a través del siguiente procedimiento iterativo:

1. Se busca el valor i_0 que hace que $\sum_{i<i_0} T_i < r_j T \leq \sum_{i \leq i_0} T_i$.

2. Se fija $\tau_j = t_{i_0}$ y se disminuye T_{i_0} y T por valor de uno.

Este procedimiento tiene varias ventajas: se garantiza que el proceso satisface todas las limitaciones de tipo de pieza para todos valores (r_1,..., r_N); dado que el conjunto de valores es invariable durante el proceso de recombinación, la recombinación siempre produce selecciones válidas de tipos de piezas; y el procedimiento se adapta fácilmente al problema multihoja de la Sección 3.4.1.3.3.

- *Algoritmo de intersección entre polígonos*

Para comprobar si dos polígonos se intersecan, se utiliza el algoritmo de sujeción entre polígonos. El algoritmo utiliza una estrategia de división resolviendo una serie de problemas sencillos, que cuando se combinan resuelven el problema general de cruce. El problema sencillo, problema de sujeción, trata de buscar intersecciones, o sujeciones, de un polígono a través de un segmento de otro polígono convexo. Al polígono convexo implicado en el problema de sujeción se le llama ventana de sujeción, y el segmento en cuestión de la ventana de sujeción es el límite de sujeción.

El algoritmo de sujeción atraviesa cíclicamente alrededor de los vértices de la ventana de sujeción, en cada paso se examina la relación entre los vértices del otro polígono y el límite de sujeción (definido por dos vértices de la ventana de sujeción). El resultado en cada paso puede dar 0, 1, o 2 puntos, que se añaden a una lista de salida. La lista de salida final constituye los vértices del polígono de sujeción. Entonces se calcula el área de este polígono. Sólo si el área total sujetada por las todas ventanas de sujeción es cero los polígonos no se cruzan.

Para verificar el cruce entre dos polígonos arbitrarios primero se determina si cualquiera de los polígonos es convexo. Si uno lo es, se convierte en la ventana de sujeción y se sujeta contra el otro polígono, que puede ser convexo o no convexo. Si ambos polígonos son no convexos uno se descompone en el subpolígonos convexos; a cada subpolígono se le trata como una ventana de sujeción y se le aplica el algoritmo de sujeción. Los dos polígonos se cruzan si cualquiera de las ventanas de sujeción se cruza con el polígono no convexo (Figura 3.13).

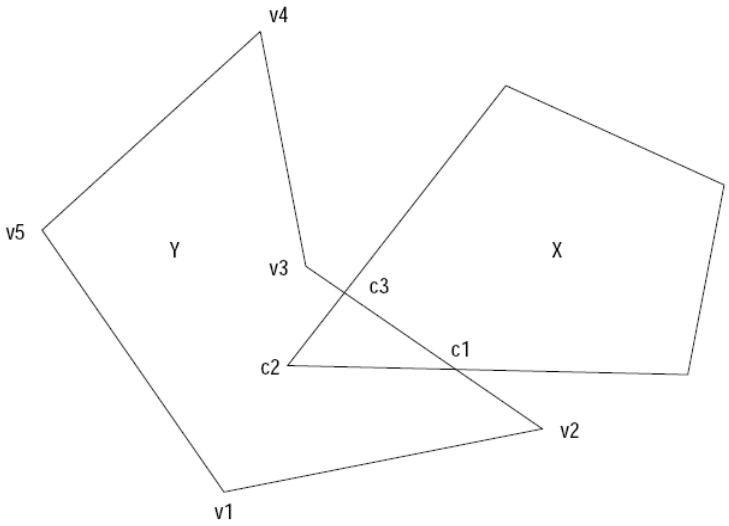


Figura 3.13. El polígono convexo *X* es la ventana de fijación. Los vértices del polígono de sujeción son *c*1, *c*2 y *c*3 [3].

El caso de un polígono que cae completamente dentro de otro polígono se trata con una rutina separada. Antes de comprobar si dos polígonos se intersecan se comprueba si uno está situado dentro del otro. Si este es el caso, se considera un cruce y no es necesario seguir probando. Si un polígono *P* está dentro de otro polígono *Q*, cada vértice de *P* será interno a *Q*. Se aplica el mismo text a todos los vértices *v* de *P*: Se dibuja una línea horizontal por el punto *v*. Si y solo si el número de cruces de esta línea a cualquier lado de *v* con el polígono *Q* es impar el punto cae dentro del polígono.

La misma rutina se utiliza al final del proceso de colocación. El resultado de cada generación es una colección de piezas que pueden o no pueden caer completamente dentro de la hoja. Se verifica si todas las piezas de la disposición caen dentro de la hoja (que se trata de igual forma como un polígono) y sólo aquellas que lo hagan se utilizan para propósitos de cálculo de eficiencia.

- *Algoritmo de traslación*

Como se ha dicho, la característica principal de una seudo-disposición es que no hace tentativa para evitar las superposiciones en la colocación inicial de las piezas, pero en su lugar quita esas superposiciones en un segundo paso de traslación. Para hacer

esto, se introduce una nueva variable para cada pieza: el ángulo de traslación (α_i). α_i es un ángulo medido desde el eje x positivo. Para trasladar una pieza d unidades en la dirección α_i desde su posición inicial, se traslada su vértice distintivo desde su posición (x_i, y_i) a $(x_i + d\cos(\alpha_i), y_i + d\sin(\alpha_i))$. La orientación de la pieza φ_i se deja igual, así que la pieza solo se traslada y no se produce ninguna rotación.

La sustancia del algoritmo de traslación reside en determinar la distancia d_i que se ha de trasladar la pieza. Las características esenciales de las dos aplicaciones (traslación para quitar las superposiciones; traslación para mover hacia el centro de masa) son las mismas. El texto se centra en el proceso de traslación para eliminar las superposiciones. Es útil trabajar en coordenadas (x', y') que son las (x, y) giradas un ángulo α_i. Si los vértices v_{ik} de p_i se expresan en coordenadas (x'_{ik}, y'_{ik}), entonces:

$x'_{imin} = \min \{x'_{ik} \mid (x'_{ik}, y'_{ik})$ es un vértice de $p_i\}$

$x'_{imax} = \max \{x'_{ik} \mid (x'_{ik}, y'_{ik})$ es un vértice de $p_i\}$

$y'_{imin} = \min \{y'_{ik} \mid (x'_{ik}, y'_{ik})$ es un vértice de $p_i\}$

$y'_{imax} = \max \{y'_{ik} \mid (x'_{ik}, y'_{ik})$ es un vértice de $p_i\}$

El espacio definido por $x'_{imin} \leq x'$, $y'_{imin} \leq y' \leq y'_{imax}$ define el rectángulo R_i donde la pieza se podría situar al trasladarla. Para cada pieza p_j que ya ha sido colocada se comprueba sus intersecciones con el rectángulo. Los vértices del conjunto $p_j \cap R_i$ consisten en los vértices v_{jk} de p_j con la coordenada y' entre y'_{imin} e y'_{imax}, y los puntos en los segmentos de p_j con la coordenada y' igual a y'_{imin} o y'_{imax}. Se determina que:

$x'_{ijmin} = \min \{x'_{ij} \mid (x'_{ij}, y'_{ij})$ es un vértice de $p_j \cap R_i\}$

$x'_{ijmax} = \max \{x'_{ij} \mid (x'_{ij}, y'_{ij})$ es un vértice de $p_j \cap R_i\}$

Si $p_j \cap R_i$ tiene más de un componente, se determina x'_{ijmin} y x'_{ijmax} para cada componente por separado. El próximo paso es examinar los espacios entre las piezas. Para encontrar estos espacios, todos los valores $\{x'_{ijmin}, x'_{ijmax}\}$ se clasifican en orden ascendente. Para cada j_0, se comprueba si hay algún x'_{ijmin} o x'_{ijmax} entre x'_{ij0max} y $x'_{ij0max} + x'_{imax} - x'_i$, o dicho de otra manera, se comprueba si existe algún vértice del conjunto $p_j \cap R_i$ en el intervalo que necesita la pieza para ser colocada $[x'_{ij0max}, x'_{ij0max} + x'_{imax} - x'_i]$. Si no es así, entonces hay un espacio después de la pieza j_0 que es suficientemente grande para admitir a p_i (ver Figura 3.14).

Las dos aplicaciones del algoritmo de traslación (quitar las superposiciones, si existen, y mover hacia el centro de masa, si no existen) difieren sólo en este punto. Para quitar las superposiciones, se toma el primer espacio disponible y se mueve al borde más cercano del espacio. Para mover hacia el centro de masa, se toma el espacio más cercano al centro de masa y se mueve al borde más cercano al centro de masa de dicho espacio. A la hora de quitar las superposiciones es posible que el primer espacio disponible no caiga enteramente en la hoja, si esto sucede (se puede probar simultáneamente) la pieza se borra de la disposición y no se coloca.

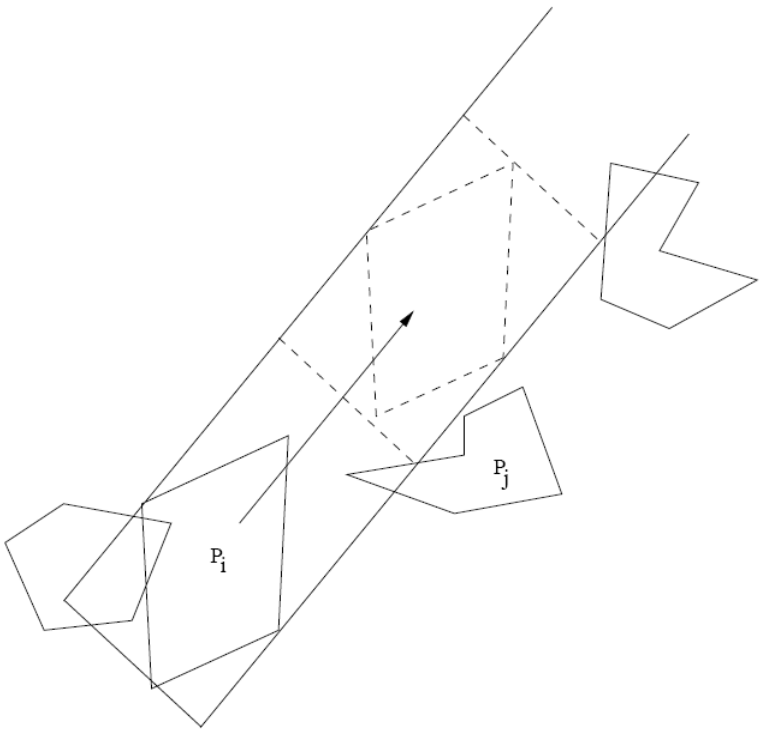


Figura 3.14. Espacios en R_i [3].

- *Espacio de configuración de las seudo-disposiciones*

El espacio de configuración de las seudo-disposiciones P consiste en N quíntuplas en la forma $(r_i, x_i, y_i, \varphi_i, \alpha_i)$ con las limitaciones siguientes:

$$0 \leq x_i \leq S_x$$
$$0 \leq y_i \leq S_y$$

$0 \leq \varphi_i \leq 360$

$0 \leq \alpha_i \leq 360$

$0 \leq r_i \leq 1$

P es un cubo de dimensión 5*N*. La función de eficiencia se evalúa transformando la seudo-disposición en una disposición real y determinando que piezas caen completamente dentro de la hoja. Las áreas de las piezas ya son conocidas, como el área de la hoja, así que es fácil de computar el porcentaje de la hoja utilizada.

Esta representación del problema es el centro del enfoque genético del algoritmo. Su ventaja principal es que, debido a que las limitaciones se incluyen en la conversión de las seudo-disposiciones en disposiciones reales, esas limitaciones siempre se satisfacen. Aún más, se pueden introducir fácilmente variaciones del problema en esta construcción modificando el proceso de conversión o modificando la función objetivo. La desventaja principal de esta representación es que la conversión de la seudo-disposición a la disposición es complicada, a la vez que lenta, y se debe hacer cada vez que se necesita evaluar la eficiencia de la disposición. Esta complejidad es la principal limitación a la habilidad del algoritmo de producir rápidamente disposiciones eficientes.

3.4.1.2.3. Implementación

Una vez que el espacio de configuración de las seudo-disposiciones se ha establecido y el proceso de evaluación se ha definido, la implementación del algoritmo de la disposición es relativamente fiel.

1. Se determina el número de partes *N* que se han de utilizar en cada seudo-disposición.
2. Se generan 5*N* seudo-disposiciones al azar.
3. Se evalúan estas disposiciones y se clasifican según su eficiencia.
4. Las *clone_number* seudo-disposiciones más eficientes se colocan en la próxima generación.
5. Se escogen dos seudo-disposiciones al azar (incluyendo aquellas seleccionadas para la reproducción elitista). Se aplica la operación de paso y se producen dos nuevas seudo-disposiciones. Se evalúan las nuevas seudo-disposiciones según su eficiencia; la mejor de las dos se coloca en la próxima generación. Se itera este proceso hasta que la

próxima generación este completa, con *N* seudo-disposiciones producidas por la reproducción elitista y el otro 4*N* son “hijos” de las seudo-disposiciones y se clasifican todas en orden descendente de eficiencia.
6. Las *immigration_number* seudo-disposiciones menos eficientes se borran y se reemplazan con nuevas seudo-disposiciones engendradas al azar que se evalúan según su eficiencia, se clasifican, y se unen con las seudo-disposiciones restantes para formar la próxima generación.
7. Este proceso se itera hasta que se da la condición de parada (en este caso al alcanzar 250 generaciones).

3.4.1.3. Ajuste del módulo de disposición

Los problemas de corte de stock no son solo un problema sino un grupo de problemas que comparten una característica básica común. El problema viene con muchas variantes debidas a muchas razones. Problemas de corte de stock diferentes tienen limitaciones diferentes, objetivos diferentes, y elementos estructurales diferentes. La meta en esta sección es la de mostrar algunas formas en las que el algoritmo se puede adaptar para ser aplicado a otras variantes del problema. Las variaciones consideradas no son de ninguna manera exhaustivas, pero cubren varios de los problemas más importantes. En ellas se ilustran claramente los procedimientos por los que el algoritmo básico se puede adaptar a nuevos problemas.

3.4.1.3.1. Restricciones de colocación

Hasta ahora se ha asumido que las piezas se pueden colocar en cualquier posición y orientación en la hoja. Este no siempre es el caso. En la industria del cuero, por ejemplo, con frecuencia hay restricciones sobre cómo una pieza puede ser orientada en relación al grano del cuero, o restricciones sobre qué partes de la hoja puede ser utilizadas para una pieza en particular. Restricciones semejantes surgen naturalmente en otras industrias. Estas restricciones surgen independientemente de la aproximación para optimizar el uso de la hoja, y tienden a reducir la eficiencia de la solución.

Sin embargo, hay también situaciones en las que las restricciones de colocación pueden ser introducidas como parte de la estrategia de optimización. Por ejemplo, en el

problema de empaquetar piezas rectangulares en una hoja rectangular normalmente las piezas deben estar orientadas según los ejes de hoja para obtener una solución aceptable. De la misma forma, las piezas pueden exhibir simetrías o relaciones en sus formas que se pueden explotar para mejorar la eficiencia del algoritmo. Por ejemplo, si una región cóncava en una pieza y una región convexa en otra casan bien, se obtienen mejores resultados generalmente creando antes asociaciones entre estas piezas que dependiendo de las operaciones fortuitas del algoritmo genético para producir estas asociaciones (ver Figura 3.15).

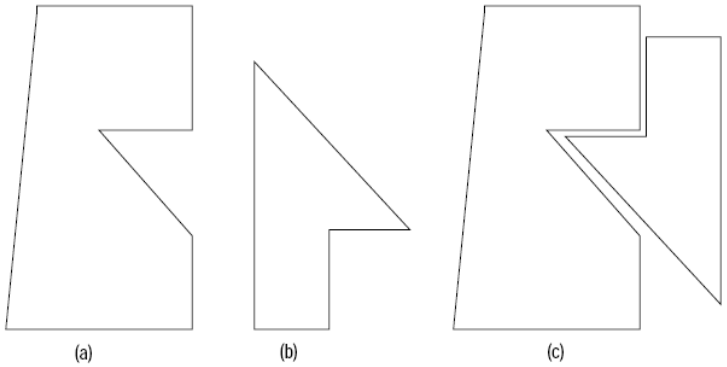


Figura 3.15. Unión de un par de piezas cóncavas y convexas. (a) Pieza 1. (b) Pieza 2. (c) Pieza compuesta [3].

Mientras las restricciones en la colocación puedan surgir o como limitaciones en el problema o como estrategias para mejorar la eficiencia, se aplican de la misma manera. Primero se consideran las restricciones en la orientación y después se agrupan las piezas.

Al imponer restricciones de orientación, hay dos orientaciones a considerar: vertical y horizontal. La orientación horizontal de la pieza es simplemente la variable φ como se definió en la Sección 3.4.1.2.2. La orientación vertical determina si la pieza se coloca “cara arriba” o “cara abajo” (ver Figura 3.16).

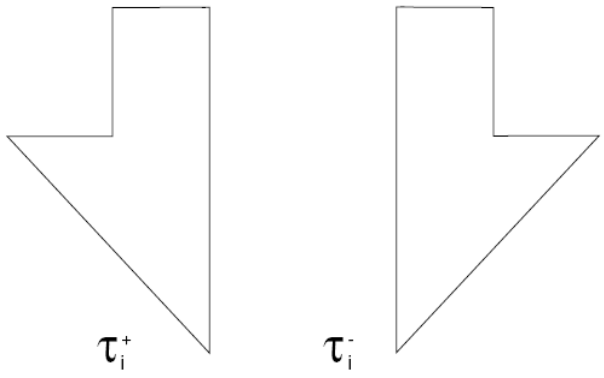


Figura 3.16. Orientaciones verticales de una pieza [3].

En la Sección 3.4.1.2, la orientación vertical de las partes era fija. Para permitir colocaciones con ambas orientaciones verticales, cada tipo de parte t_i debe ser reemplazado por dos tipos de parte t^+_i y t^-_i, uno con orientación hacia "arriba" y otro con orientación hacia "abajo". Hay varias maneras en las que se pueden modificar las claves aleatorias de codificación para permitir esto. Por ejemplo, la paridad del último dígito de r_i puede ser utilizada para determinar la orientación vertical de la pieza *i*-ésima: si el último dígito es par, la pieza se coloca "hacia arriba"; si el último dígito es impar, la pieza se coloca "hacia abajo".

Para poner restricciones en la orientación horizontal de una sola pieza, la descripción de la pieza debe ser modificada en el archivo neutro. Se supone que las piezas del tipo t_i tienen Θ_i orientaciones admisibles $\tau_{i1},...,\tau_{i\Theta_i}$. Estos valores deben ser añadidos al archivo neutro. Entonces, si una pieza del tipo t_i es la pieza *k*-ésima escogida para colocar, la variable φ_i no se interpreta directamente como la orientación de la pieza relativa a los ejes de la hoja sino que se utiliza para escoger uno de los valores τ_{ij}. Si $\frac{j}{\Theta_i} \le \frac{\varphi_k}{360} < \frac{j+1}{\Theta_i}$ entonces la pieza se coloca con orientación τ_{ij}. Este proceso de selección garantiza que todas las colocaciones posibles tienen la misma probabilidad de ser escogidas.

Los pares de piezas se pueden utilizar para aprovecharse de sus uniones especialmente eficientes en las fronteras. Varias preguntas tienen que ser contestadas:

• ¿Cómo se clasifican las “uniones eficientes”?
• ¿Cómo se identifican las uniones eficientes?
• ¿Cuándo se deben usar estas uniones?
• ¿Cómo se incluyen tales uniones en el algoritmo?

Las primeras dos preguntas son los problemas difíciles en si mismas. Se puede hacer una variedad de definiciones de la eficiencia local, y está disponible una variedad de algoritmos para procurar localizar las uniones de la mayor eficiencia. Este texto no entra en detalles, pero se asumirá que (o bien a mano o bien como la producto de un algoritmo) se han identificado ciertos pares de piezas, esto hace probable mejorar la eficiencia de la disposición general.

Para utilizar estos pares, se crea un nuevo de tipo pieza para cada pieza compuesta creada por una unión de piezas. Estos tipos de partes se añaden a la tarjeta de fabricación de la forma siguiente. Para cada par de tipos de piezas t_i y t_j ($i = j$ inclusive), la pieza compuesta es notada por t_{ij}, y al número fijo de componentes admisibles del tipo t_{ij} se le llama T_{ij}:

$$T_{ij} \leq \left\{ \begin{array}{c} \min\{T_i, T_j\} \Leftrightarrow i \neq j \\ \frac{T_i}{2} \Leftrightarrow i = j \end{array} \right\}$$

Si no se permite alguno de los componentes t_i y t_j, T_{ij} es cero. La clave aleatoria de codificación descrita antes se modifica añadiendo las piezas compuestas al final de la lista de piezas fijas. El número total de piezas disponibles se ajusta a $T = \sum_i T_i + \sum_{i \leq j} T_{ij}$. Las variables r_k se interpretan de la misma forma para determinar el valor del k-ésimo tipo de pieza. Si se escoge una pieza simple t_i se cambia T_i por T_i - 1, y para cada j, se cambia T_{ij} por $\min\{T_{ij}, T_i - 1\}$. Si se elige una pieza compuesta t_{ij}, entonces T_i, T_j, y T_{ij} se disminuyen en uno, así como el valor T_{kl} de cualquier conjunto que contenga a t_i o t_j. Para la siguiente pieza se vuelve a calcular T con los valores actuales de T_i, T_j, y T_{ij}.

3.4.1.3.2. Una hoja irregular

La próxima adaptación es crucial para la industria de cuero: colocar las piezas en hojas de forma irregular, en vez que en hojas rectangulares. El término “irregular” incluye hojas convexas y no convexas, incluso hojas con agujeros. Estos agujeros pueden incluir las partes defectuosas de la hoja. Se asume que la hoja tiene una frontera poligonal.

El algoritmo requiere una pequeña modificación para aplicarlo a este problema. La hoja juega un papel activo en el algoritmo sólo en dos ocasiones: cuándo se engendran las colocaciones iniciales aleatorias de las piezas, el vértice distintivo se coloca en la hoja; y después de que el algoritmo de traslación se aplique a una pieza, se prueba si la pieza cae completamente dentro de la hoja. El procedimiento de intersección entre polígonos utilizado para esta prueba (ver la sección del algoritmo de intersección entre polígonos) no requiere que la hoja sea rectangular, ni aún convexa, puede ser aplicado a cualquier hoja poligonal. Para la colocación inicial se puede requerir fácilmente que las colocaciones iniciales de los vértices distintivos caigan o dentro del casco convexo de la hoja o dentro de un rectángulo fijo.

Este procedimiento, sin embargo, no tiene en cuenta las regiones defectuosas de la hoja que deben ser evitadas. Para situar las piezas lejos de tales regiones, se trata estas regiones como piezas colocadas permanentemente. Si S denota la porción de la hoja realmente disponible para la colocación, y R_1, R_2,..., R_i denotan las regiones que no están disponibles. Entonces a $S' = S \cup R_1 \cup ... \cup R_i$ se le trata como la hoja efectiva, y las regiones R_i son piezas ya colocadas en la hoja. En la Figura 3.17 se presenta una hoja no convexa (con una región defectuosa convexa). Esta hoja puede ser rodeada por un rectángulo como el mostrado y entonces ser dividida en dos regiones, disponible y no disponible para la colocación de la pieza.

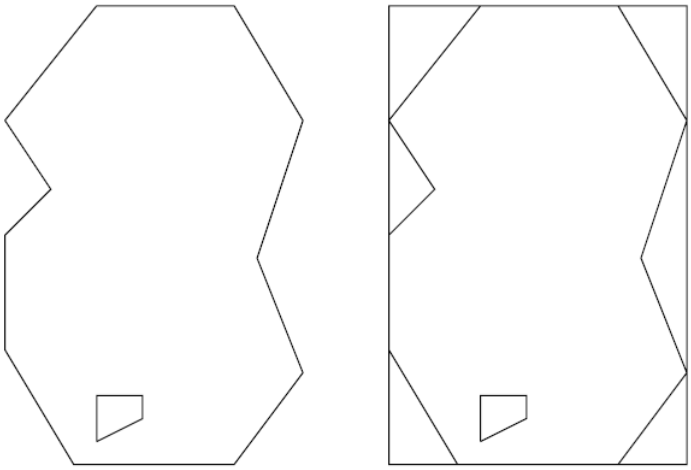


Figura 3.17. Caja de contorno para una hoja irregular [3].

Dado que cada región R_i es poligonal estas pueden ser tratadas como piezas en el algoritmo de disposición. La única modificación es esta, cuando se crea una seudo-disposición todas las “piezas” que son realmente regiones excluidas deben ser listadas antes que las piezas “verdaderas”, y siempre deben aparecer en el mismo orden en cada seudo-disposición (incluyendo aquellas introducidas por la inmigración). Entonces, cuando se aplica el operador de paso, no cambiará ninguno de los datos de las regiones excluidas R_i. Es más, dado que estas “piezas” se colocan primero y está garantizado que no se superponen, la transformación en una disposición verdadera dejará todos estos componentes fijos en su lugar. Aún más, cuando el proceso de transformación alcance las piezas verdaderas, todas las regiones excluidas ya estarán fijadas en su lugar y no estarán disponibles para las piezas verdaderas. De otra forma, cuando el algoritmo intente trasladar las piezas verdaderas para que se sitúen en una porción desocupada de $S´$, la única porción de $S´$ que estará disponible a las piezas será la hoja verdadera S.

La otra modificación que el algoritmo requiere es que si una pieza verdadera tiene una colocación inicial que no se superpone con ninguna pieza existente (incluyendo las regiones excluidas R_i), entonces el algoritmo lo traslada hacia el centro de masas hasta que contacte una pieza previamente colocada. Cuándo el algoritmo se modifica para hojas de forma irregular, el centro de masa debe quedarse sólo con las piezas “verdaderas” y no incluir las regiones excluidas.

3.4.1.3.3. Muchas hojas rectangulares

Para muchas situaciones industriales no es adecuada la solución del problema de una sola hoja. Dada una tarjeta de la fabricación, no es satisfactorio producir tantas piezas como sea posible (que todas las piezas de la tarjeta se tengan que producir es una limitación muy fuerte). El número de hojas requerido (o, más generalmente, el área total de hojas utilizadas) llega a ser desconocido en el problema, y la meta es la de encontrar una disposición que aminore el número de hojas. Este problema, con este objetivo, se nombra como el problema de múltiples hojas.

Se podría aplicar el algoritmo de una hoja para llegar iterativamente a una disposición para el problema entero, pero esta no es la óptima generalmente. Cuándo se aplica iterativamente el algoritmo de una hoja, el objetivo en cada paso es aminorar el desecho en la hoja actual. Este proceso de empaquetar sucesivamente tantas piezas como se pueda en una hoja no es necesariamente el mismo que el de aminorar el número de hojas utilizadas, aunque es claramente semejante. Por lo tanto, el algoritmo de disposición será modificado para incluir el nuevo factor de múltiples hojas y la nueva función objetivo.

El problema de múltiples hojas se considera primero con todas las hojas rectangulares y todas con las mismas dimensiones. Sólo se requiere un pequeño cambio en el espacio de configuración de las seudo-disposiciones. En vez de estimar el número N de piezas que pueden ser colocadas en una sola hoja y engendrar colocaciones para N piezas, las colocaciones son engendradas para todas piezas T. El espacio del estado es entonces un cubo de dimensión $5T$. Los procesos de clonación, recombinación e inmigración son iguales. Las diferencias yacen en la función objetivo utilizada y en el proceso de transformar las seudo-disposiciones en disposiciones verdaderas.

Este proceso se entiende más fácilmente si se conciben las hojas como amontonadas una encima de otra, con todas las piezas inicialmente colocadas en la hoja inferior (ver Figura 3.18). Al igual que en el algoritmo de una hoja, las piezas se colocan sucesivamente en la hoja inferior, se prueba si hay superposición, y se trasladan para procurar eliminar las superposiciones. Sin embargo, si una pieza no puede ser colocada exitosamente en la hoja inferior, no se quita (como en el algoritmo de una

hoja) pero en su lugar se sube a la próxima hoja, utilizando las mismas coordenadas, y se repite el proceso. El proceso se itera hasta que todas las piezas hayan sido colocadas exitosamente. Mediante este proceso, cada pieza de la tarjeta de la fabricación es finalmente colocada. La función objetivo es simplemente el número de hojas utilizadas.

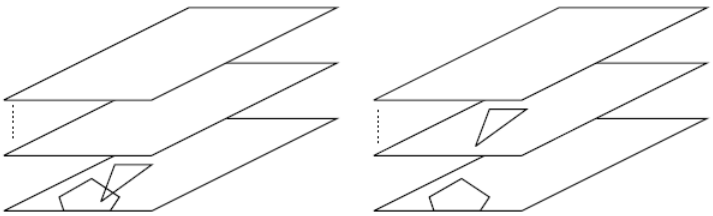


Figura 3.18. Múltiples hojas rectangulares [3].

3.4.1.3.4. Muchas hojas irregulares

Básicamente, esto combina las modificaciones de las dos secciones previas. Sin embargo, la situación es más compleja si las hojas no tienen un área uniforme. En este caso se debe modificar el orden en la que las piezas se colocan en las hojas y el orden en la que las hojas se utilizan para la colocación. Para agregar esta característica adicional de ordenar las hojas se utiliza el mismo método de claves aleatorio indicado antes. Si se dispone de S hojas se añaden nuevas variables σ_1,..., σ_S, que otra vez son números al azar entre 0 y 1. El espacio de configuración es entonces un cubo de dimensión $5T + S$, y el algoritmo opera en este espacio de configuración de manera obvia.

3.4.2. Algoritmo de Anidado Bidimensional Basado en el Polígono de No Conveniencia y el Principio del Mínimo Centro de Gravedad [4]

En el apartado anterior se ha expuesto el método genético; en este apartado se da otro ejemplo de algoritmo genético que, aún manteniendo la misma estructura que el anterior, resulta completamente diferente al modificar algunos de los procedimientos más importantes del algoritmo (función de eficiencia, operación de paso,…).

En la primera parte del apartado se expone un algoritmo para evitar superposiciones basado en el polígono de no conveniencia (NFP) y el principio del mínimo centro de gravedad.

3.4.2.1. Procedimiento de anidado de formas irregulares

El procedimiento para el anidado de formas irregulares incluye principalmente dos pasos: primero, colocar las piezas en la hoja sin superposición; después, regenerar la hoja restando la región ocupada por las piezas ya anidadas en ella. La Figura 3.19 expone el procedimiento de anidado.

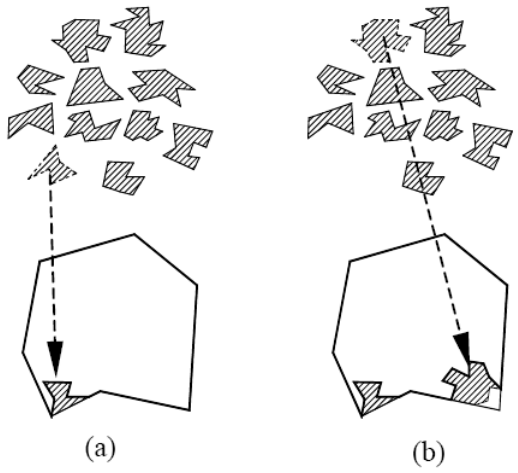


Figura 3.19. Procedimiento de anidado bidimensional. (a) Anidado de una pieza y generación de una nueva frontera. (b) Anidado de otra pieza en la nueva frontera [4].

3.4.2.2. El algoritmo del polígono de no conveniencia y su función en el anidado

Con herramientas de geometría computacional como el polígono de no conveniencia (NFP) y los principios de colocación de las piezas, se pueden colocar con precisión las piezas dentro de la frontera de la hoja contenedora. El algoritmo no fit polygon (NFP) (Adamowicz y Albano, 1976), ha llegado a ser prácticamente un requisito previo para resolver los problemas de empaquetado irregular; acota el espacio accesible sin solapamientos para cada polígono. NFP se define de la siguiente manera:

Dadas una hoja y una pieza con orientación fija, donde la hoja se define como el polígono fijo y la pieza como el polígono que desliza. El NFP es un sendero cerrado que se forma trazando la situación de un punto de referencia en la pieza mientras la pieza desliza alrededor de la frontera interior de la hoja, como se muestra en la Figura 3.20.

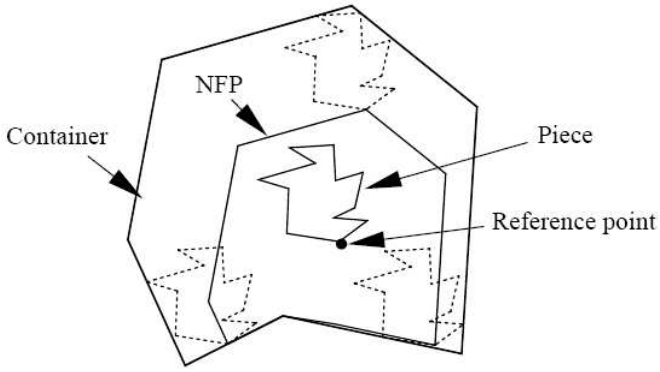


Figura 3.20. NFP es la situación de un punto de referencia mientras la pieza desliza alrededor del contorno de la hoja [4].

La propiedad pertinente del NFP con respecto a la interacción entre la hoja y la pieza es la siguiente: si la pieza se coloca con su punto de referencia en la frontera del NFP entonces la hoja y la pieza se tocarán; si la pieza se coloca con su punto de referencia dentro de la frontera del NFP entonces la pieza estará dentro de la hoja sin superponerse. Así, el interior del NFP representa todas posiciones posibles de colocación para la pieza, y la frontera del NFP representa todas posiciones de contacto. En el procedimiento de anidado, se procura que las piezas se toquen unas al otras para ahorrar material, por lo tanto, basándose en el NFP, el problema de anidado de una pieza se puede simplificar al problema de escoger la posición óptima en el NFP.

Los métodos para generar el NFP se han enfocado en tres metodologías centrales: la suma de Minkowski (ver Sección 3.3.1.1), basada en gran parte en el teorema de la adición de la frontera de Ghosh (Bennell, 2001); el principio de colisión o el principio de órbitas, diseñado para simular el movimiento de los polígonos que deslizan; y la descomposición, donde los polígonos se descomponen en polígonos secundarios que pueden ser manejados más fácilmente.

3.4.2.3. Principio de anidado para piezas irregulares

El espacio de búsqueda del problema de nidificación irregular es infinito porque cualquier movimiento o rotación de las piezas puede llevar a una nueva pauta de nidificación. Para reducir el espacio de búsqueda, el principio de llenado inferior-izquierda (BL) se ha adoptado extensamente como principio de colocación. El principio

BL es colocar la pieza tan cerca como se pueda de la esquina inferior izquierda; trabaja con rectángulos, pero no con polígonos irregulares porque no considera irregularidades en las formas ni rotaciones arbitrarias en las piezas, de ahí que se tengan que desarrollar otros principios de colocación. Debido a que el NFP da todas las posiciones posibles de colocación dentro de la hoja, el principio de colocación se puede desarrollar basándose en el NFP. Dado un NFP, el principio de colocación para una pieza se simplifica encontrando la posición óptima de colocación en el NFP, como se ve en la Figura 3.20.

3.4.2.3.1 NFP del centro de gravedad

Para buscar para una posición conveniente de colocación en el NFP, algunos investigadores escogen el punto en el NFP donde la pieza y las piezas anidadas previamente forman un área mínima. Sin embargo, el método del área mínima apunta a encontrar una posición óptima local y no considera la optimización global, encontrar una posición de área mínima es algo complicado y añade carga computacional al algoritmo de nidificación.

Para conseguir un algoritmo de colocación rápido y efectivo para piezas, en este texto se propone el principio del mínimo centro de gravedad para encontrar rápidamente la posición de colocación en el NFP. En este algoritmo de colocación, se escoge como punto de colocación en el NFP aquel en el que la pieza tiene el centro de gravedad más bajo. Este algoritmo se propone "empujar" la pieza hacia las piezas anidadas tan de cerca como sea posible, y crear una frontera de nidificación relativamente plana para las piezas que todavía tienen que ser anidadas.

Para encontrar el centro de gravedad más bajo, primero se calcula el centro de gravedad de la pieza, y entonces se puede encontrar el NFP del centro de gravedad tomando el centro de gravedad como punto de referencia, como ilustra la Figura 3.21.

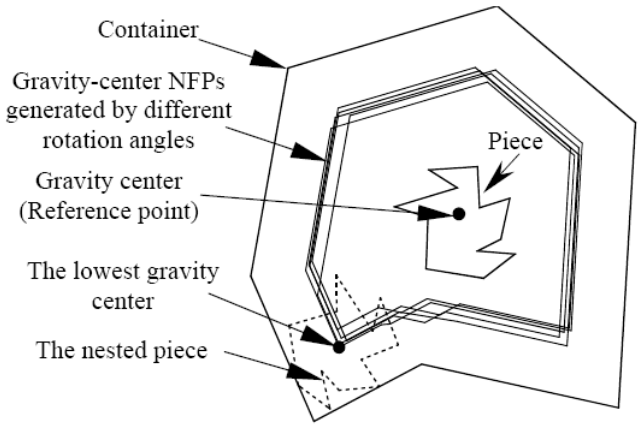


Figura 3.21. Búsqueda del mínimo centro de gravedad en el abanico de NFPs, con el centro de gravedad como punto de referencia, generados con diferentes ángulos de rotación de la pieza [4].

Nótese que ángulos diferentes de rotación de la pieza llevarán a NFPs del centro de gravedad diferentes, el centro de gravedad más bajo es el punto más bajo global entre todos los NFPs. Así si se determina primero el NFP con el punto global más bajo, se puede determinar el ángulo correspondiente de rotación la pieza. Por lo tanto se coloca la pieza en la posición donde el centro de gravedad es el más bajo y se gira el ángulo correspondiente de rotación, como se muestra en las líneas de puntos de la Figura 3.21.

Para encontrar el ángulo de rotación que lleva al centro de gravedad más bajo, se adopta un método de búsqueda de dos etapas. Primero, se divide el rango entero de rotación (360 grados) regularmente en varios ángulos de rotación, y se prueba cada uno de los ángulos para encontrar el que tiene el centro de gravedad más bajo. Después, se repite la división y se prueba dentro del ángulo encontrado en la etapa anterior y sus dos ángulos vecinos para lograr un ángulo más preciso de rotación.

3.4.2.3.2. Cálculo del centro de gravedad

Adoptamos un método de división en tiras para calcular el centro de gravedad de un polígono. Se proyecta y divide el polígono en varias tiras (Figura 3.22), y se calcula el centro de gravedad y el área de cada tira, finalmente se calcula el centro de gravedad global con la fórmula siguiente:

$x = \sum A_i x_i \Big/ \sum A_i$; $y = \sum A_i y_i \Big/ \sum A_i$, donde A_i es el área de la tira i y x_i, y_i las coordenadas x, y del centro de gravedad de la tira.

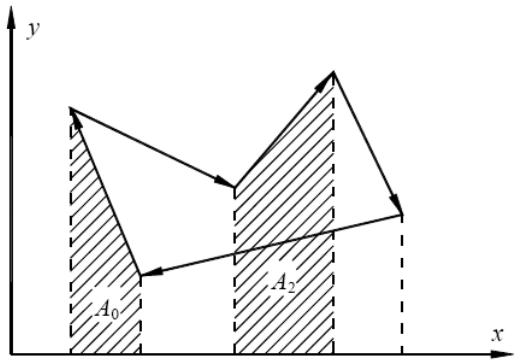


Figura 3.22. Cálculo del centro de gravedad dividiendo el polígono en bandas [4].

La Figura 3.22 muestra cómo se proyectan los segmentos del polígono en el eje x por líneas paralelas al eje y, de forma que a cada segmento le corresponde una tira como A_0 y A_2. Si la coordenada x del punto final del segmento del polígono es mayor que la del punto inicial, entonces el área es positiva, de otro modo el área es negativa. Por ejemplo, A_0 es un área negativa y A_2 es un área positiva en la Figura 3.22. Después de calcular el centro de gravedad y el área de todas tiras por separado, se puede calcular el centro de gravedad global del polígono según la fórmula de arriba.

3.4.2.4. Algoritmos para determinar la secuencia de anidado

El principio del mínimo centro de gravedad (LGC) determina la posición de nidificación y el ángulo de rotación para una pieza, y de la misma manera, se pueden anidar de una en una todas las piezas según una sucesión dada de nidificación. Es decir, el punto clave para resolver el problema de nidificación bidimensional e irregular es ahora encontrar una sucesión eficiente de nidificación. En este texto se propone un algoritmo recursivo y un algoritmo genético (GA) para buscar para una sucesión efectiva de nidificación.

3.4.2.4.1. Anidado recursivo de huecos

Si las piezas se colocaran por su área, habría varios huecos entre las piezas grandes si se anidaran juntas. Realmente estos huecos se pueden tapar con piezas más pequeñas. Desde este punto de vista, se propone el anidado de huecos recursivo (HRN) para conseguir una permutación efectiva.

Como se muestra en la Figura 3.23, primero se han clasificado las piezas según su área en orden decreciente, y después se han anidado en dicha sucesión. Cada vez que una pieza se anida, es posible que se formen algunos huecos que son suficiente grandes para acomodar una o más de las piezas restantes más pequeñas. Para maximizar el uso de tales huecos, se deben anidar primero las piezas pequeñas que se puedan colocar en tales huecos.

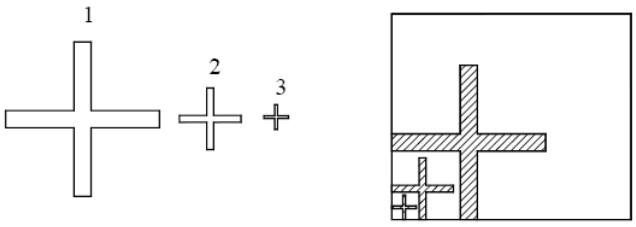


Figura 3.23. Anidado de huecos recursivo para eliminar huecos [4].

Por ejemplo, en la Figura 3.23, la sucesión original de nidificación clasificada según el área es 1 → 2 → 3, la pieza número 1 se coloca primero, entonces se comprueba si el hueco más grande formado entre la pieza 1 y la frontera de la hoja es suficiente grande para acomodar la pieza 2, si es así, se debe anidar la pieza 2 antes que la pieza 1 de manera que la sucesión de nidificación original se transforma en 2 → 1 → 3. De la misma forma, la pieza 3 se debe anidar antes que la pieza 2 y la sucesión de nidificación queda 3 → 2 → 1. Se sigue alterando la sucesión de anidado hasta que el hueco formado sea demasiado pequeño para contener cualquiera de las piezas restantes.

El principio del anidado de huecos recursivo no sólo requiere un tiempo mínimo, si no que en la práctica también logra una calidad moderada de la solución. Otro aspecto

importante es que se puede utilizar para generar soluciones iniciales (individuos) para el algoritmo genético de nidificación.

3.4.2.4.2. Algoritmo genético para la secuencia

Para lograr una sucesión de de nidificación gran calidad, se utiliza un algoritmo genético. Algunas de sus características más importantes se indican abajo.

- *Codificación genética de la solución*

Dado que el principio de colocación del mínimo centro de gravedad propuesto puede determinar el centro de gravedad y el ángulo de rotación de una pieza (Figura 3.21), no hay necesidad de codificar el ángulo de rotación en el cromosoma, a diferencia del método anterior, lo único que necesitamos codificar es la sucesión de nidificación (el orden de permutación).

- *Inicialización*

La primera generación de individuos tiene gran influencia en el resultado final del GA. Para conseguir un conjunto de individuos iniciales bien definidos, la permutación de nidificación clasificada por área se utiliza como primer individuo, y después se crean otros individuos mediante mutaciones del primer individuo. Todos estos individuos forman la primera generación.

- *Función de eficiencia y pendiente de eficiencia*

Se toma como función de eficiencia la altura de la hoja restante, se adopta una función lineal de pendiente de eficiencia para llevar el valor de eficiencia a un rango razonable:

$f' = \alpha f + \beta + \sigma$,

$\alpha = \frac{f_{avg}}{f_{avg} - f_{\min}}$, $\beta = \frac{-f_{avg} \cdot f_{\min}}{f_{avg} - f_{\min}}$, $\sigma = \frac{f_{avg}}{K}$, donde f es la eficiencia original, f' es la pendiente de eficiencia de salida y K es una constante.

Dado que en este GA se adopta el método proporcional de selección, se rechazarán algunos individuos de eficiencia muy baja demasiado pronto, lo que puede llevar a una convergencia prematura, así que se añade σ a la función de eficiencia para evitar esta clase de rechazo.

- *Operación de paso*

Se adopta la operación de paso OX (Davis, 1985)). Si se supone que un padre es (5, 2, 3, 7, 6, 1, 4) y el otro padre es (4, 6, 2, 1, 3, 5, 7), el procedimiento de paso OX se ilustra en la Figura 3.24.

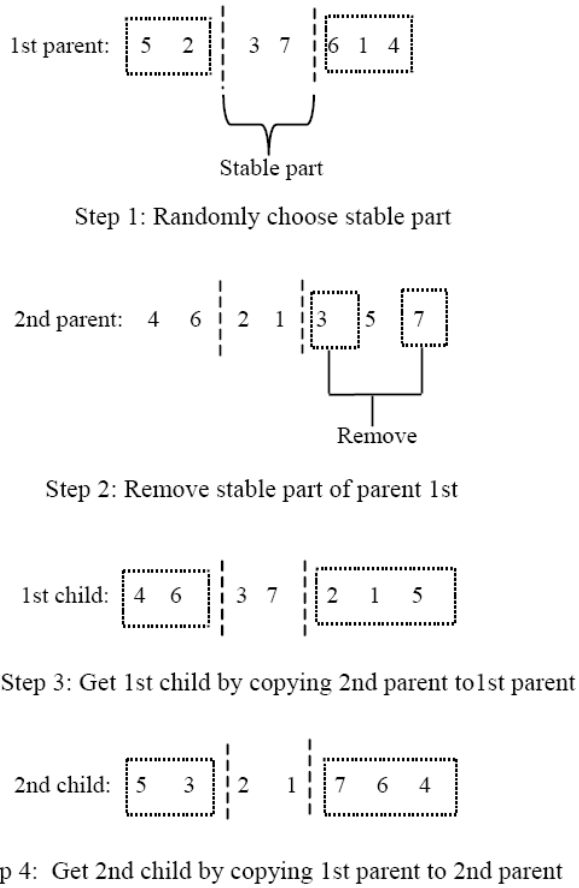


Figura 3.24. Generación de dos individuos mediante la operación de paso OX [4].

- *Mutación*

La mutación ocurre después de la operación de paso y se aplica a los niños engendrados. Aquí la mutación se utiliza para cambiar dos números en el cromosoma, la probabilidad de mutación es de 0.1 ~ 0.2. A veces también se pueden cambiar dos segmentos del cromosoma, en tal caso se debe disminuir la probabilidad de mutación desde el momento en el que el cromosoma cambia demasiado.

3.4.3. Algoritmo Heurístico de Llenado Inferior-Izquierdo para el Problema de Empaquetado Irregular Bidimensional [5]

El siguiente ejemplo de algoritmo genético de disposición, es de interés por el procedimiento que emplea para resolver las superposiciones, la parte genética del algoritmo es similar a las de los anteriores ejemplos por lo que solo expondremos el algoritmo de llenado inferior-izquierdo.

En los algoritmos genéticos vistos hasta ahora las piezas estaban siempre representadas mediante polígonos. Este método introduce la circunferencia en la representación de las formas.

3.4.3.1. Definiciones geométricas

Para ilustrar el método de empaquetado, se define primero qué constituye una “primitiva”, un “lazo” y una “forma”. En lo siguiente, una “primitiva” se define como un arco o una línea. Una línea se representa por sus puntos inicial y final, mientras que un arco es circular y se representa por su centro, su radio, su ángulo de inicio, y su distancia angular. Se define un “lazo” como una lista cerrada de primitivas en dirección antihoraria, donde cada punto final de una primitiva es el punto inicial de la próxima primitiva. Una “forma” se define como un lazo exterior y 0,…, *n* lazos internos que se pueden considerar agujeros en la forma. La mayor parte de los problemas de la literatura actual no incluyen formas con ni arcos ni agujeros y numerosos ejemplos sólo contienen formas convexas, donde es más fácil de discernir las superposiciones. Además, es necesario para los algoritmos trabajar con datos de punto flotante para establecer altos niveles de certeza y realismo en problemas del mundo real. La Figura 3.25 muestra un

caso de superposición entre dos formas, *A* y *B*. Podemos ver que el arco primitivo *a*2 se cruza con la línea primitiva *b*4 y que las líneas primitivas *a*3 y *b*3 se cruzan también.

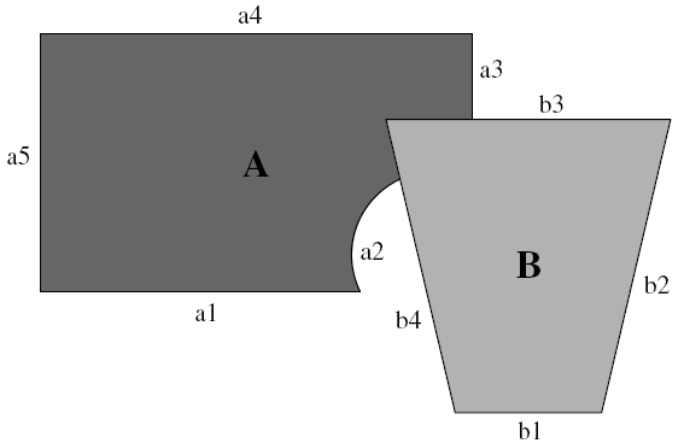


Figura 3.25. Formas solapadas [5].

3.4.3.2. Resolución del solapamiento de primitivas

Hay cuatro casos posibles que se deben manejar para resolver el cruce de primitivas. Uno de estas primitivas forma parte de la forma que se trata de colocar, llamada la “forma libre” y la otra forma parte de una forma que ya ha sido colocada en la hoja, llamada la “forma fija”. Las técnicas que se describen en las secciones siguientes implican el cálculo de la distancia vertical positiva requerida para trasladar la forma libre tanto que las dos primitivas ya no se crucen. Mientras esto se resuelve la superposición entre las dos primitivas que se cruzan, las dos formas todavía pueden no estar resueltas completamente ya que otras primitivas que pertenecen a las formas pueden cruzarse también. Sin embargo, la aplicación repetida de estas técnicas siempre resolverá las formas que se superponen con la distancia vertical positiva más pequeña requerida. Hay cuatro casos de cruce que deben ser considerados: dos líneas, línea y arco, arco y línea, y dos arcos. Se debe notar que en todos los casos, la primitiva de la forma fija que se interseca, llamada la “primitiva fija”, ya ha sido asignada a la hoja y a su posición no puede cambiar, mientras que la primitiva que se interseca que pertenece a la forma que se trata de colocar se llama “primitiva libre”. Se explicarán los pasos requeridos a la hora de resolver los cruces para cada uno de estos casos, pero se introduce primero alguna terminología. El espacio *x* de una primitiva se puede

considerar como el espacio horizontal de su rectángulo de borde. Otro concepto que se utiliza es la "línea vertical infinita". Esta es una línea vertical infinita por ambos lados y con la dirección del eje *y*. Las notaciones utilizadas dentro de los esquemas y las descripciones de las subdivisiones siguientes se presentan en la Tabla 3.1. Una vez establecida la terminología y la notación requeridas, se explica cada uno de los casos de cruce resumidos arriba.

Símbolo	Descripción
*A*1 → *A*2	Primitiva *A* (la primitiva libre)
*A*1, *A*2	Puntos inicial (*A*1) y final (*A*2) de la primitiva *A*
*B*1 → *B*2	Primitiva *B* (la primitiva fija)
*B*1, *B*2	Puntos inicial (*B*1) y final (*B*2) de la primitiva *B*
CP	Centro de un arco primitivo
*c*1,…, *cn*	Puntos de intersección
*t*1,*t*2	Puntos tangentes de una línea en un arco

Tabla 3.1. Notación [5].

3.4.3.2.1. Línea-línea (una línea libre moviéndose a través de una línea fija)

Para resolver cualquier par de líneas que se cruzan, se hayan los puntos finales de cada línea, *A* y *B*, que están dentro del espacio *x* de la otra. Estos puntos se conocen como puntos en rango (points in range, *pi*r). Se pasan líneas verticales infinitas por cada punto *pir* originario de la línea *A* y se encuentran los puntos de cruce de estas líneas con la línea *B*. Se calcula la distancia entre cada *pir* de la línea *A* y su punto correspondiente de cruce en la línea *B* utilizando la siguiente fórmula:

$distancepirA = intersectionPointB.y - pirA_y$.

También se necesita pasar líneas verticales infinitas por cada punto *pir* que originario de la línea *B* para encontrar sus puntos de cruce con la línea *A*. Se usa una fórmula distinta para calcular las distancias cuando el *pir* se origina en la primitiva fija (la línea *B*):

$distancepirB = pirB.y - intersectionPointA_y$.

Para el método de solución, la superposición siempre debe ser resuelta trasladando la línea *A* en la dirección vertical positiva. Las fórmulas de distancia pueden dar resultados negativos, por lo tanto, estos resultados no son movimientos verticales, positivos y válidos y son eliminados. Estas fórmulas de distancia forman también parte de las estrategias para resolver otros casos. Para el cruce de líneas, siempre existe un resultado positivo válido que puede ser utilizado para trasladar verticalmente la línea *A* y resolver la superposición. Un ejemplo de este enfoque se muestra en la Figura 3.26.

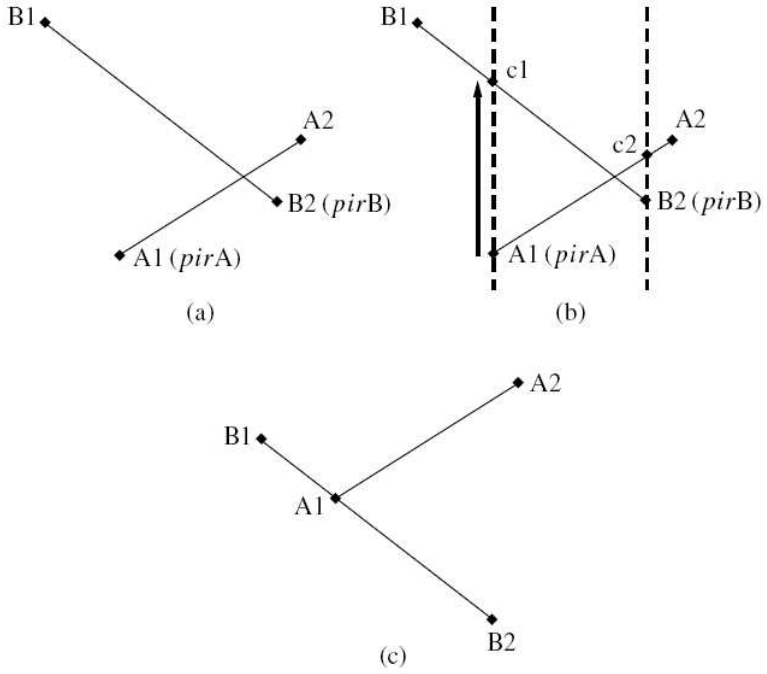


Figura 3.26. Resolución del caso línea-línea [5].

Aquí, el punto *A*1 está dentro del espacio *x* de *B*1 → *B*2 y *B*2 está dentro del espacio *x* de *A*1 → *A*2. Se marcan estos puntos como *pirA* y *pirB*, respectivamente (ver Figura 3.26). Se pasa una línea vertical infinita por *pirA* para crear el punto *c*1 de cruce y por *pirB* para crear el punto *c*2 de cruce (ver Figura 3.26). La distancia entre *pirA* y *c*1 y la distancia entre *pirB* y *c*2 se calculan utilizando las fórmulas anteriores. En este ejemplo, la primera distancia da un resultado positivo mientras la segunda da un resultado negativo. La siguiente fórmula muestra cómo combinar las fórmulas de distancia y en una función:

$yTraslation = \max(c1.y - pirA_y, pirB.y - c2_y)$.

El resultado se muestra gráficamente como una flecha gruesa en la Figura 3.26. La Figura 3.26 muestra cómo se ha resuelto la superposición trasladando verticalmente la línea *A* esa distancia positiva. En la práctica, todas las primitivas de la forma *A* se trasladan, no sólo la línea implicada en la superposición.

3.4.3.2.2. Línea-arco (una línea libre moviéndose a través de un arco fijo)

En este caso, en el que una línea se cruza con un arco, se debe encontrar la distancia vertical positiva que la línea debe ser trasladada para resolver completamente su cruce con el arco. Al igual que en el caso línea-línea se pueden utilizar los puntos en rango de cada primitiva. Sin embargo, dado que aparece un arco, también puede ser necesario utilizar los puntos tangentes entre la línea y el arco primitivos. La Figura 3.27 expone un ejemplo donde aplicar sólo el método de los puntos en rango no es suficiente para resolver la superposición.

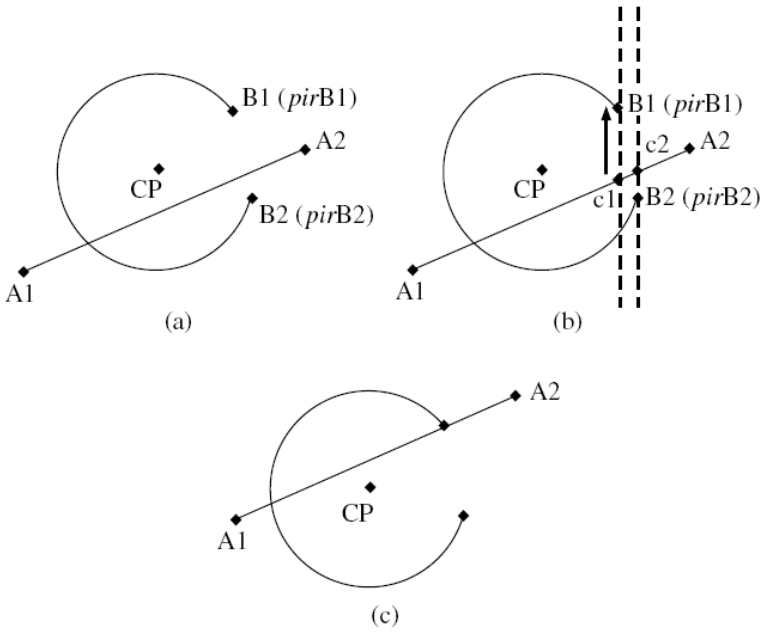


Figura 3.27. Ejemplo de situación en la que el método de los puntos en rango no es suficiente para resolver el problema [5].

La Figura 3.27 expone que los únicos puntos en rango son *B*1 (*pirB1*) y *B*2 (*pirB2*) originarios del arco (los puntos finales de la línea, *A*1 y *A*2, están fuera del

espacio x del arco y, por lo tanto, no están en rango). Una vez más, se pasa una línea vertical infinita por cada *pir* y se cruza con la línea. Esto crea los puntos de cruce *c*1 de *pirB1* y *c*2 de *pirB2* (ver Figura 3.27). La distancia entre cada *pir* y su cruce respectivo con la otra primitiva se calcula utilizando las fórmulas de distancia. En este ejemplo, ambos *pir* se originan en la primitiva fija (el arco *B*) y por lo tanto ambas distancias se calculan utilizando la segunda fórmula. Esto da un resultado positivo que se muestra con la flecha gruesa en la Figura 3.27. La Figura 3.27 expone que esta traslación vertical no es suficiente para resolver la superposición. La Figura 3.28 expone cómo se puede recurrir a los puntos tangentes para resolver completamente la superposición.

Los puntos tangentes se pueden encontrar trasladando la perpendicular (o "normal") de la línea hasta que pase por el punto central del arco, *CP*, como se muestra en la Figura 3.28. El cruce de la perpendicular con el arco da los puntos tangentes *t*1 y *t*2 (ver Figura 3.28). Estos puntos tangentes se utilizan de forma semejante a la técnica de puntos en rango, se pasan líneas verticales infinitas por cada punto tangente y se cruzan con la línea $A1 \rightarrow A2$ para dar los puntos *c*1 y *c*2. Las distancias de traslación se pueden calcular con la siguiente fórmula:
$distanceTangentB = tangentB.y - intersectionPointA_y$.

En el ejemplo, se puede ver que *t*1 daría una traslación positiva, mientras que *t*2 daría una distancia negativa de traslación. Por lo tanto, trasladando la línea según la distancia dada por *t*1 se resolverá la superposición (ver Figura 3.28). Se debe notar que si el cruce de la línea perpendicular con el arco no da puntos tangentes o si los puntos tangentes tienen como resultado distancias negativas de traslación usando la fórmula anterior, entonces la técnica de los puntos en rango debe poder resolver la superposición de primitivas.

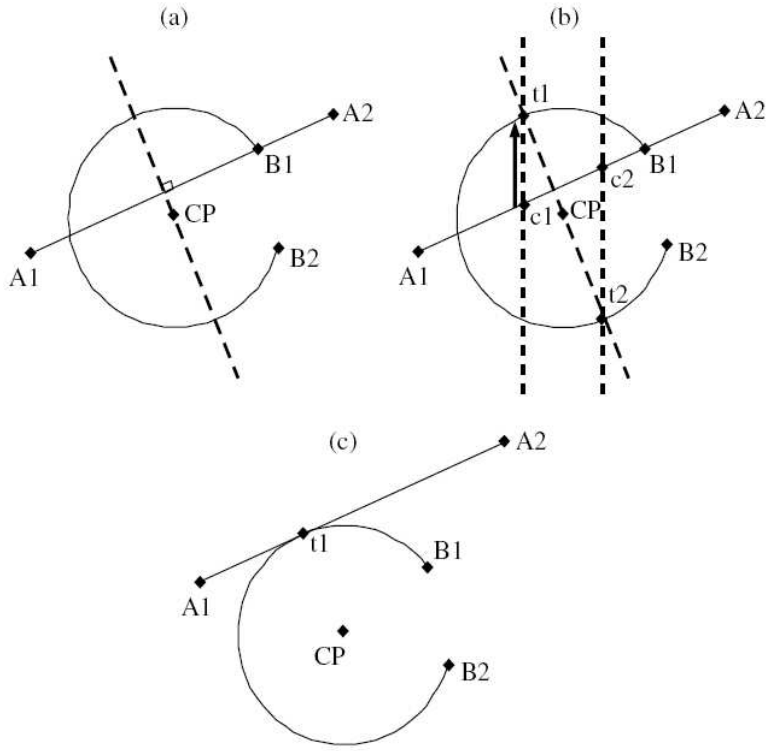


Figura 3.28. Resolución del caso línea-arco utilizando los puntos tangentes [5].

3.4.3.2.3. Arco-línea (un arco libre moviéndose a través de una línea fija)

Este caso, donde se tiene un arco que mueve a través de una línea fija, implica un enfoque semejante al caso de la línea libre y el arco fijo. Una vez más, se aplica la misma técnica de puntos en rango y, por lo tanto, no se repite aquí. Sin embargo, dado que el arco es ahora la primitiva libre (arco $A1 \rightarrow A2$) y la línea es ahora la primitiva fija (la línea $B1 \rightarrow B2$) debemos sustituir los puntos calculados de las tangentes y sus cruces en la primera fórmula de distancia en vez de en la segunda (como en el caso anterior). Se muestra un ejemplo en la Figura 3.29.

La Figura 3.29 muestra que los puntos *A*2 y *B*1 son los puntos en rango, *pirA* y *pirB.* Sin embargo, ambos puntos producen traslaciones negativas (utilizando las fórmulas de la distancia), así que no se pueden utilizar para resolver el cruce. Debemos recurrir al método de la tangente otra vez. En el ejemplo, sólo se encuentra un punto tangente ya que la línea perpendicular se cruza con el arco únicamente en un punto. La Figura 3.29 muestra cómo una línea vertical infinita se pasa por el punto tangente, *t*1, y

se cruza con la línea $B1 \rightarrow B2$ para producir el punto $c1$. Las distancias de la traslación se pueden calcular con la siguiente fórmula:
$distanceTangentA = intersectionPointB.y - tangentA_y$.

Utilizando esta fórmula en el ejemplo resulta una distancia positiva de traslación, como se muestra con la flecha gruesa en la Figura 3.29. El cruce se resuelve trasladando el arco $A1 \rightarrow A2$ dicha distancia vertical, como se muestra en la Figura 3.29. Una vez más, si el método de la tangente no encuentra puntos tangentes o no da un resultado positivo válido, entonces el método de los puntos en rango debe resolver completamente el cruce.

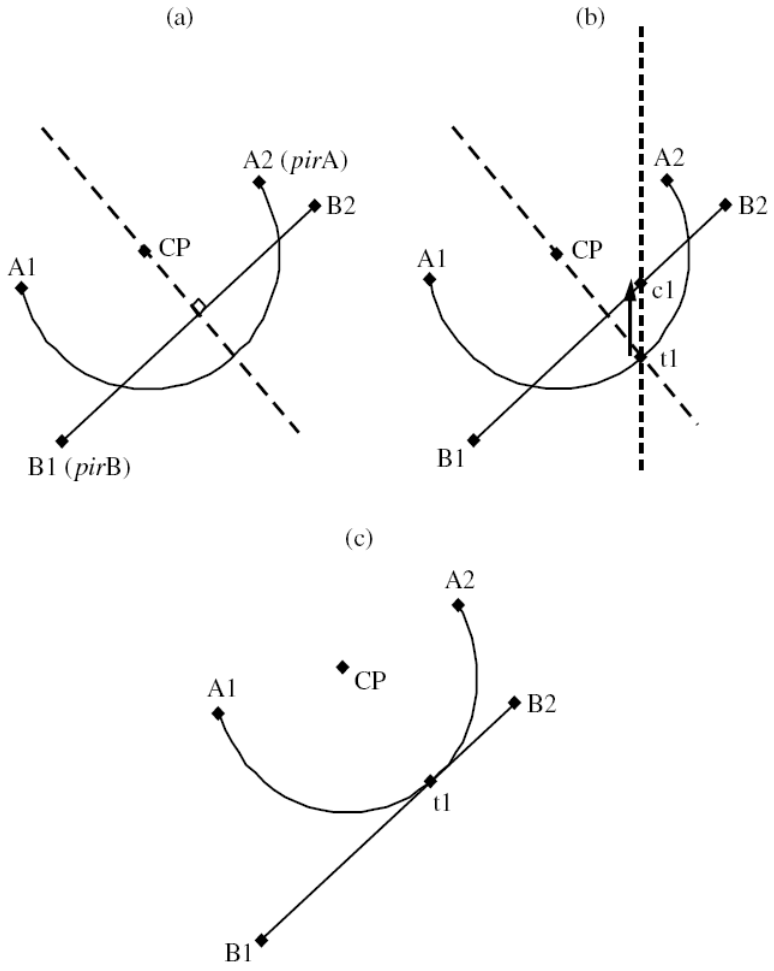


Figura 3.29. Resolución del caso arco-línea utilizando los puntos tangentes [5].

3.4.3.2.4. Arco-arco (un arco libre moviéndose a través de un arco fijo)

El caso de un arco a través de un arco utiliza inicialmente la técnica del punto en rango. No se explica con todo detalle ya que es idéntica a la técnica utilizada en los casos previos. Para las situaciones donde el método de los puntos en rango no puede resolver el cruce entre los dos arcos primitivos, se utilizarán dos métodos de círculos tangentes que utilizan los radios de los arcos y el teorema de Pitágoras.

En la Figura 3.30 se muestra un ejemplo de cruce de arcos con distinta orientación.

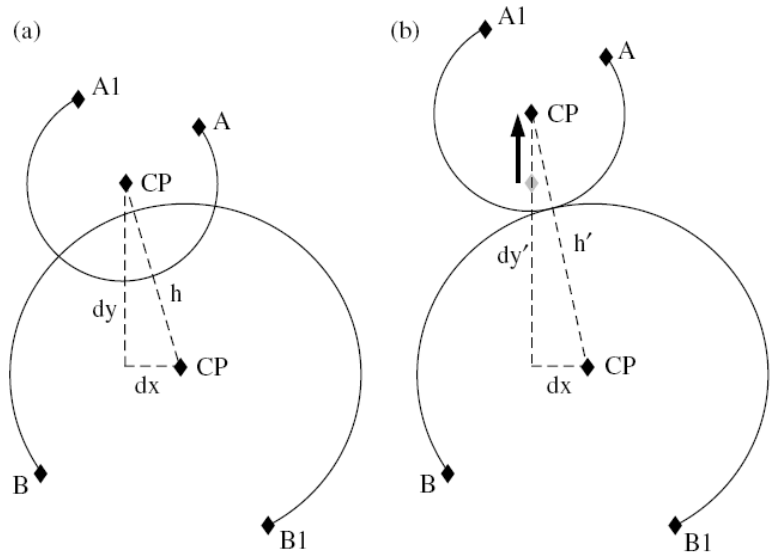


Figura 3.30. Resolución del caso arco-arco con el teorema de Pitágoras (método 1) [5].

Llamando *rA* al radio del arco libre *A*1 → *A*2 y *rB* al radio del arco fijo *B*1 → *B*2 se pueden hacer las siguientes observaciones:
De la Figura 3.30, cuando los arcos se cruzan, $rA < h < rA + rB$, y
De la Figura 3.30, cuando se haya resuelto el cruce, $h' = (rA + rB)$, por lo tanto:
$dy' = \text{sqrt}((h' * h') - (dx * dx))$,
$yTraslation = (dy' - dy)$.

Este cruce se puede resolver trasladando el arco *A*1 → *A*2 la distancia resultante de la fórmula anterior.

Un caso arco-arco adicional que hay que resolver implica dos arcos de orientación semejante como se muestra en la Figura 3.31.

Llamando *rA* al radio del arco libre *A*1 → *A*2 y *rB* al radio del arco fijo *B*1 → *B*2 se pueden hacer las siguientes observaciones:

De la Figura 3.31, cuando los arcos se cruzan, (*rB* – *rA*) < *h* < (*rB* + *rA*), y

De la Figura 3.31, cuando se haya resuelto el cruce, *h*’ = (*rB* – *rA*), por lo tanto:

dy’ = sqrt((*h*’ * *h*’) – (*dx* * *dx*)),

yTraslation = (*dy* – *dy*’).

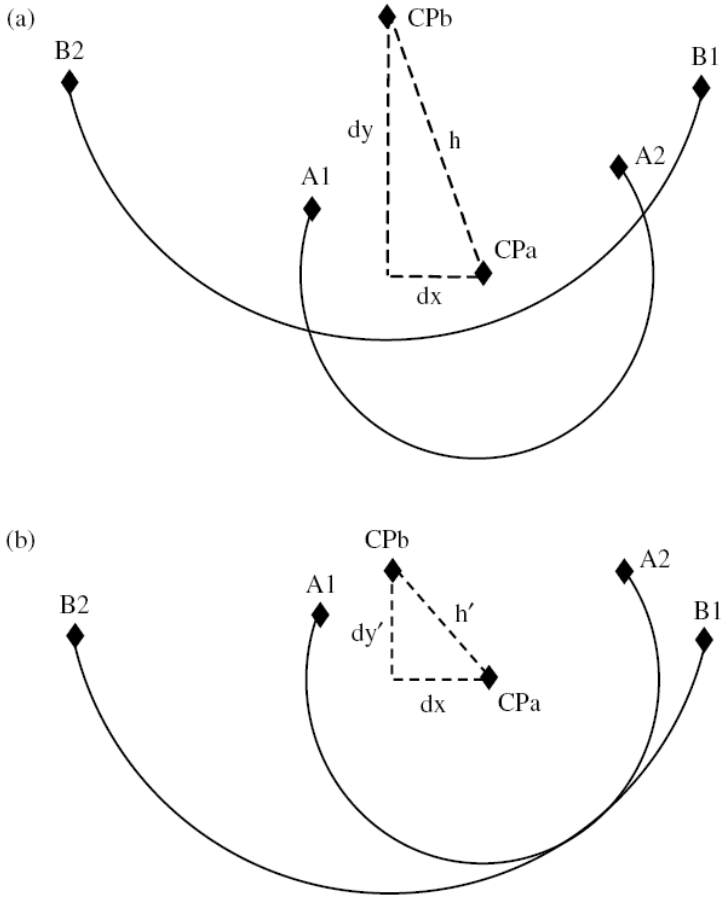


Figura 3.31. Resolución del caso arco-arco con el teorema de Pitágoras (método 2) [5].

Si el resultado de la fórmula anterior es positivo, aplicando esta traslación vertical al arco $A1 \rightarrow A2$ se resolvería la superposición. Se puede ver que, mientras el primer método de resolución de círculos tangentes traslada el arco libre al exterior del círculo del arco fijo, el segundo método traslada el arco libre dentro del círculo del arco fijo. Esto es imprescindible para la correcta manipulación de arcos convexos y cóncavos.

3.4.3.2.5. Resumen del método

Se han detallado los cuatro casos posibles de cruce y se ha mostrado que cada caso se puede resolver utilizando el método de los puntos en rango con las fórmulas de distancia. Este método siempre resolverá el cruce entre dos líneas. Si hay arcos implicados, el método de los puntos en rango puede no ser suficiente para resolver los cruces completamente, y se pueden emplear técnicas suplementarias basadas en las tangentes. Cuándo un arco y línea se cruzan, se utiliza la perpendicular de la línea primitiva, se desplaza hasta el punto central de arco, y se cruza con el arco para encontrar los puntos tangentes. Cuando el arco es la “primitiva fija” y la línea es el “primitiva libre”, se utiliza la fórmula de la Sección 3.4.3.2.2 para calcular la traslación vertical requerida. La fórmula de la Sección 3.4.3.2.3 se utiliza cuando el arco es el “primitiva libre” y la línea es la “primitiva fija”. El caso final, donde dos arcos se cruzan, introdujo dos métodos de círculos tangentes con el que se pueden resolver los cruces. El primer método hace que los círculos a los que pertenecen los arcos sean tangentes exteriormente. El segundo método hace que los círculos a los que pertenecen los arcos sean tangentes interiormente. El menos costoso de los casos es el que implica a dos líneas, ya que no se necesita el cálculo de tangentes. Esto presenta posibilidades de optimización; si hay muchos pares de primitivas que se cruzan entre dos formas, se resuelven primero los casos de línea-línea.

3.4.3.3. Formas completamente solapadas

Ahora se detalla el caso especial no válido en el que una forma está contenida completamente dentro de otra. Esto requiere otra estrategia de solución ya que no hay cruce de primitivas. Durante el proceso de la nidificación, es posible que una forma pueda caer enteramente dentro de otra forma ya colocada. En esta circunstancia, no hay

primitivas que se crucen y se requiere otro enfoque para resolver el solapamiento. Cuando la forma libre *A* está contenida por la forma fija *B*, se utiliza el punto más bajo en la forma *A*, *lpA*, por el que se pasa una línea vertical infinita. El cruce resultante de la línea vertical infinita y la forma *B* da los puntos *c*1,…, *cn*. La traslación que se realiza es definida por:

$(ci.y - lpA_y) > 0$ para $i = 1,\ldots, n$,

$yTraslation = \min(c1.y - lpA_y,\ldots, cn.y - lpA_y)$.

La Figura 3.32 muestra un caso donde empleando esta técnica no se resuelve completamente la superposición entre las formas. Sin embargo, la forma *A* ya no está contenida por la forma *B* y ahí se cruzan primitivas permitiendo emplear una vez más las técnicas para resolver cruces de primitivas. Este proceso continúa hasta que el cruce de las formas se haya resuelto completamente.

Al resolver cruces de forma, las traslaciones verticales empleadas pueden causar que la forma libre se cruce con otras formas ya colocadas. Estos cruces también se deben resolver hasta que la forma no se cruce y pueda ser colocada en la hoja, o hasta que la forma se haya movido por encima de la hoja. En este último caso, la forma debe ser trasladada abajo de la hoja a la próxima coordenada *x*, y el proceso continúa hasta que todas las formas hayan sido colocadas.

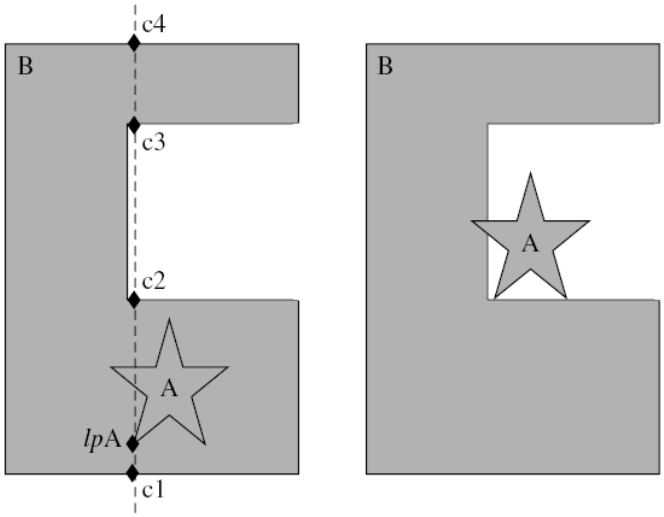


Figura 3.32. Ejemplo de pieza contenida en otra pieza [5].

3.5. RESUMEN

En este capítulo hemos comenzado exponiendo las diferencias entre los dos principales grupos de algoritmos heurísticos para el problema de corte bidimensional: los constructivos y los genéticos.

Aunque los algoritmos genéticos parecen inspeccionar un intervalo mayor de soluciones, no aseguran una solución eficiente. Sin embargo, los algoritmos constructivos colocan en cada paso la pieza que ofrece mejor disposición, sin alterar las formas ya colocadas, con lo que limitan el intervalo de soluciones alcanzables pero ofrecen mayores garantías de alcanzar una disposición eficiente.

En el tercer apartado se ha expuesto un ejemplo de algoritmo constructivo, a la vez que se ha enunciado la suma de Minkowski y se ha explicado porqué es provechosa para el problema de corte.

En el cuarto apartado se ha definido la estructura de un algoritmo genético, se han mostrado varios ejemplos de algoritmos genéticos y se han considerado algunos algoritmos de la literatura para resolver superposiciones (NFP, llenado inferior-izquierdo,…).