MODELADO Y SOLUCIÓN DEL PROBLEMA DEL CORTE IRREGULAR:

Aplicación en la Industria del cuero Colombiana

Trabajo de Grado para optar por el título de
Magister en Ingeniería Industrial

AUTOR
ING. LUIS FERNANDO PINILLA OLARTE

COAUTOR
ING. RAFAEL GUILLERMO GARCÍA CACERES PhD.

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA INGENIERÍA INDUSTRIAL
BOGOTÁ D.C.
2011

1 Contenido

2 Índice de Tablas

3 Índice de Figuras

4 Resumen

El problema del *Irregular Two Dimensional Cutting Stock Problem* (ITDCSP) está presente en diversas aplicaciones industriales que incluyen la confección, fabricación del calzado y la marroquinería. Debido a su naturaleza matemática NP completa, ha sido particularmente difícil de formular y resolver. Este Trabajo de Grado propone una Formulación Lineal Mixta base que sirve como base para emplear diversos procedimientos meta heurísticos para la búsqueda de soluciones cercanas al óptimo. Al respecto, se expone el uso de dos herramientas meta heurísticas, GRASP y Algoritmos Genéticos, para su solución. En estos casos se han encontrado soluciones de calidad aceptables en tiempos de ejecución razonable.

5 Glosario

Cuero: Material bidimensional de origen animal empleado en la fabricación de diversos productos. Se caracteriza por ser irregular en ambas dimensiones y estar plagado de defectos que enmarcan áreas de él no aprovechables.

Operario: Personas entrenadas para ejecutar la tarea del corte del cuero, que se realiza empleando maquinaria o herramientas manuales.

Open Source Es el término empleado en Informática para hacer referencia al software distribuido y desarrollado libremente. El código abierto tiende a estar dirigido y orientado a los beneficios prácticos de poder acceder a los algoritmos y soluciones antes que, a las cuestiones relacionadas con beneficios comerciales.

Parallel Computing: Palabras en inglés para hacer referencia a *Computación Paralela*. Es una forma de computación que consiste en emplear los múltiples núcleos de un CPU, para resolver simultáneamente las partes de un problema.

Piel: Hace referencia a una hoja de cuero y constituye la materia prima para el corte de las piezas que constituyen un producto en cuero. Su superficie geométrica puede ser representada o modelada como un polígono irregular en dos dimensiones.

Pieza: Parte constitutiva de un producto que requiere ser cortada de la piel. Su superficie geométrica puede ser representada o modelada como un polígono irregular en dos dimensiones.

Pixel: Entrada de una matriz que representa alguna geometría en dos dimensiones. Puede tomar valores de 1 cuando representa material aprovechable y 0 en cualquier otro caso.

Polígono: Figura geométrica cerrada, formada por segmentos rectos consecutivos y no alineados, llamados lados. Los polígonos pueden ser regulares, irregulares, convexos y no convexos.

6 Introducción

El presente Trabajo de Grado propone hacer uso de algunos procedimientos matemáticos para la solución del problema del corte irregular, propio de industrias de la confección y en particular la del cuero.

De manera introductoria, el problema atacado consiste en encontrar una forma de cortar un material bidimensional en el listado de piezas que serán empleadas para el ensamble de un conjunto determinado de productos. Dentro de las características del material y las piezas a cortar, la irregularidad dimensional presenta uno de los mayores obstáculos a la hora de minimizar el desperdicio de material durante la operación manual de corte. En el Planteamiento del Problema se exponen con mayores detalles especificaciones adicionales del problema y en el Estado del Arte se presentan documentos de autores que han trabajado sobre el mismo problema.

En la actualidad, las empresas suelen entrenar operarios para realizar esta tarea de manera eficiente. Sin embargo, cuando las presiones de trabajo por causa de la demanda son altas, los operarios tienden a incrementar el desperdicio. Dado que el cuero es el material base y generalmente más costoso en esta industria, resulta conveniente generar mecanismos de automatización para esta tarea que disminuyan los costos asociados. Al respecto, fabricantes de maquinaria dirigida a la marroquinería y el calzado han desarrollado herramientas dirigidas a automatizar el corte del cuero y hoy es posible aunque costoso, encontrar máquinas capaces de efectuar cortes usando tecnologías basadas en cuchillas, agua y laser. En todos los casos, las máquinas en la oferta requieren que un operario especialmente capacitado determine la acomodación de las piezas sobre las pieles, solo después de este procedimiento, la máquina efectúa rápidamente la tarea.

Para automatizar la acomodación de las piezas sobre las pieles minimizando el entrenamiento del recurso humano y aumentando la productividad de la tarea, proponemos generar un procedimiento de optimización que permite determinar una acomodación factible, de bajo desperdicio de material y tiempo razonable de ejecución, que en trabajos posteriores podrá ser integrado junto con la herramienta de corte, automatizando completamente la operación.

El procedimiento propuesto representa las geometrías de la piel y las piezas a cortar como polígonos irregulares, posteriormente emplea un procedimiento que transforma cada polígono en una retina de pixeles que puede ser procesada fácilmente empleando lenguajes de programación de propósito general para computadores. Para el proceso de acomodación, se hizo una formulación matemática empleando Programación Entera Mixta y basada en *Knapsack Problem*, donde las variables binarias determinan si una pieza en una ubicación en particular debe asignarse o no, y las restricciones están asociadas a prohibir que un pixel asociado a la piel sea ocupado por más de un pixel asociado a una pieza, esto evita el traslape de las piezas. La función objetivo está encaminada a maximizar el área remanente aprovechable, área que será demarcada por la ubicación en ella de piezas ficticias cuya suma total de áreas determinará el valor a maximizar. En resumen, el modelo está diseñado de modo tal que su solución maximiza la cantidad de piezas ficticias representando el área aprovechable remanente luego de acomodar la totalidad de piezas definidas.

Dado que el modelo presentado está basado en *Knapsak Problem* y éste es conocido entre otras cosas por su complejidad computacional correspondiente a un problema NP Completo, se puede inferir que su solución en instancias reales puede ser inaccesible en términos de optimalidad, siendo justificada la aplicación de métodos aproximados de solución.

En cuanto a la búsqueda de la solución, se empleó una estrategia conocida en la literatura como *Bottom Left Strategy* que consiste en desplazar de izquierda a derecha y arriba hacia abajo las retinas de pixeles asociadas a las piezas sobre la retina de pixeles asociada a la piel hasta encontrar una acomodación factible, de modo que la secuencia en

que se efectúe el procedimiento representa una solución codificable para el uso de procedimientos heutísticos y meta heurísticos.

Para el cálculo de la secuencia de acomodación, todas las meta heurísticas hacen uso del concepto en torno a la función objetivo del modelo de Programación Lineal Mixta para derivar y sustentar sus funciones de utilidad y, en este sentido, conducir el desarrollo e implementación de los procedimientos aproximativos de solución en un lenguaje de propósito general.

Definido el marco de trabajo, emplearon los procedimientos meta heurísticos *GRASP* y *Genetic Algorithm* con dos instancias de problemas, se realizaron comparaciones de las medias del desempeño de cada procedimiento en cuanto a la calidad de la solución y al tiempo de ejecución y, con base en estos resultados, se propusieron lineamientos para investigaciones futuras.

7 Planteamiento del problema

El cuero es el material básico y generalmente el más costoso empleado en la fabricación de diversos productos, que van desde llaveros y billeteras hasta tapizados para vehículos de lujo, pasando por calzado y prendas de vestir, entre otros. Por su origen animal, el cuero es un material irregular en casi todas sus dimensiones y en su superficie suele encontrarse diversos defectos atribuibles al proceso de crianza del ganado y al curtido. Durante el proceso de elaboración, dichos defectos deben ser evitados por los operadores de corte, quienes requieren de gran entrenamiento para adquirir maestría en el oficio. Pese a ello, cuando las demandas son altas y las presiones del trabajo aumentan los operarios suelen cometer más errores de los tradicionalmente cometidos desaprovechando más el material.

En general, el proceso del corte de cualquier piel inicia con un proceso de inspección visual y general sobre la hoja a cortar, el operario identifica los defectos superficiales en el cuero tales como rayas por alambre de púas, marcas dejadas por insectos, agujeros y manchas, y señala defectos en la carnaza o revés como variaciones en la consistencia que, si bien no representarán problemas para el corte, más adelante surgirán en procesos como el desbaste obligando a desechar la pieza. Con lo anterior, el operario basándose en su entrenamiento determinará la manera de localizar los moldes y/o troqueles para efectuar el despiece, tratando de aprovechar al máximo la piel. Por otro lado, cada producto final requiere para su elaboración de una lista de piezas específicas tanto en cantidad como en geometría. En algunos casos, las tonalidades del color del cuero a cortar varían entre hojas provenientes de un mismo lote, razón por la cual, el operario debe intentar cortar la lista de piezas de cada producto en la misma piel, de lo contrario, el productos final quedará con diferentes tonalidades en un mismo color y podría ser considerado defectuoso. La acomodación de los moldes y/o troqueles de corte sobre el cuero está sujeta a ciertas reglas relacionadas con la geometría de la piel a cortar y a sus defectos. El objetivo del operario cortador es aprovechar al máximo el material (o reducir el desperdicio), cumpliendo con la demanda de piezas. Los productos en cuero tienen demandas dominadas por las temporadas comerciales: productos como las maletas suelen tener incrementos en la demanda antes de épocas de vacaciones, las billeteras de mujer se venden mejor antes de días como el de la mujer y la madre, y el calzado colegial suele tener picos muy acentuados iniciando las temporadas escolares. Lo anterior ilustra a groso modo la complejidad y variabilidad de la tarea.

En el mercado, la oferta de maquinaria y equipo para el corte del material incluye diversas opciones para la ejecución del proceso: corte por punzón o cuchilla, corte con láser, incluso corte con agua. Independientemente de la forma de cortar, todos los equipos requieren el ingreso manual de las geometrías a cortar en una base de datos y el proceso de acomodación de los cortes puede ser hecho por un operador o ser calculadas por software basado en algoritmos y/o heurísticas de optimización a partir de la demarcación manual de las zonas no aprovechables. La herramienta de optimización de corte integrada por lo general se vende por separado al equipo de corte y tiene un costo considerable, comúnmente fuera del alcance para la mayoría de las organizaciones en la industria de los países en vía de desarrollo.

Por otro lado, el problema consiste fundamentalmente en encontrar una acomodación posible de varios elementos definidos por superficies en dos dimensiones dentro de otros, evitando traslapes. Las figuras asociadas a dichos elementos pueden ser irregulares y no convexas, lo que genera dificultades para la formulación y búsqueda de la solución óptima del problema. El objetivo suele ser la minimización del desperdicio del espacio o la maximización del área resultante aprovechable del elemento contenedor. Las restricciones están generalmente asociadas a la geometría de los elementos y a las zonas no aprovechables del contenedor. Para el caso del corte del cuero, tanto el contenedor como los elementos por anidar se asocian a la piel y a las piezas que deben ser cortadas respectivamente y, los defectos en el cuero pueden ser marcados como zonas del contenedor no aprovechables.

En la literatura revisada, se ha encontrado diversas formas de llamar el problema descrito, las cuales se pueden clasificar dentro de cuatro categorías:

- *Polygon packing* (Bennell & Oliveira, 2008), (Burke et al., 2007), (Dean et al., 2006), (Aras, 2005) (Bennell et al., 2001) (Babu & Babu, 2001).
- *Irregular Two-dimensional cutting stock problem* (Anand et al., 1999), (Cheng et al., 1994).
- *Two-dimensional irregular objects allocation problem* (Wong et al., 2009).
- *Nesting problems* (Lee et al., 2008), (Bennell & Oliveira, 2008), (Dowsland et al., 2002), (Ismail & Hon, 1995), (Prasad, 1994), (Tang & Rajesham, 1994).

Independiente de la categoría, autores como Bennell & Oliveira (2008) lo presentan como NP-COMPLETE razón por la cual es difícil resolver óptimamente, por lo que ha sido común la utilización de heurísticas y meta heurísticas para su solución. Posiblemente la mejor manera de denominar el problema en mención sea como el *"Irregular Two Dimensional Cutting Stock Problem"*, de modo que en adelante será nombrado por la sigla ITDCSP.

Hasta el momento no se ha encontrado evidencias de la existencia de formulaciones basadas en Programación Lineal, ni basadas en Programación Mixta, salvo en algunas aproximaciones relacionadas con los trabajos pioneros. Al respecto diversos autores han propuesto mecanismos de solución basados en heurísticas y meta heurísticas. La novedad del presente Trabajo de Grado radica en la propuesta de una formulación matemática basada en programación lineal mixta del ITDCSP que permite resolverlo por métodos exactos para instancias pequeñas, dada la complejidad asociada, y la implementación de meta heurísticas para instancias superiores. Lo anterior se justifica en la necesidad científica de avanzar en la solución del problema teórico e industrial de mejorar el desempeño del proceso asociado.

8 Planteamiento de la pregunta de Investigación

¿Cuál representación matemática del *Irregular Two Dimensional Cutting Stock Problem* produce buenas soluciones a partir del uso de meta heurísticas?

9 Estado del Arte

Los primeros intentos por resolver problemas de corte consistieron en asumir que las piezas a cortar eran rectángulos o aproximaciones a los mismos, donde dichos cortes se harían a modo guillotina, siendo efectuados entre los bordes opuestos de superficie rectangular (Gilmore & Gomory, 1961) (Dychof, 1981). Más adelante, se idearon métodos usando programación dinámica con modelos de tres etapas (Hahn, 1968) y de n etapas sin restricciones, donde cada etapa hace referencia al corte sobre el eje horizontal y vertical de la superficie a cortar. En otros estudios, Besley (1985) empleó procedimientos de búsqueda en árboles en conjunto con métodos de programación binaria para resolver el *orthogonal non-guillotine cutting stock problem*. Agrawal (1993) presentó soluciones obtenidas empleando una aproximación eficiente del algoritmo *branch and bound*.

Dentro de las heurísticas, se expusieron procedimientos para anidar figuras irregulares dentro de rectángulos usando algoritmos de *clustering* combinados con heurísticas simplificadas, desarrolladas a partir de enumeraciones parciales para la acomodación de figuras similares en patrones (Adamowicz & Albano, 1976). También se encontraron soluciones a partir de *Simulación de Monte Carlo* (Segenreich & Faria-Braga, 1986), procedimientos que tienen en cuenta simultáneamente el listado total de piezas a cortar y múltiples hojas a partir del anidamiento de figuras irregulares en rectángulos que no se sobreponen entre sí (Qu & Sanders, 1987), heurísticas para ubicación de partes irregulares con resultados no superiores a los logrados por operarios humanos (Albano & Sapuppo, 1980), procedimientos de anidamiento de piezas irregulares dentro de superficies irregulares usando secuencialmente procesos topológicos y reducción de datos (Heistermann & Lengauer, 1995), y se emplearon conjuntos de algoritmos de anidamiento inteligente para ubicar piezas irregulares en hojas de metal (Prasad, 1994). Wascher et al (2007) presenta una síntesis de la caracterización de la problemática, ver Figura 1. El ITDCSP está estrechamente relacionado con las familias *Bin Packing Problem*, y específicamente los relacionados con la acomodación de figuras irregulares dentro de superficies irregulares.

En cuanto a las revisiones consultadas, Bennell y Oliveira (2008) reúnen conceptos orientados a mejorar el entendimiento del problema de anidamiento de elementos geométricos, relacionando mecanismos de representación como Raster pixel method, Direct trigonometry, Nofit polygon y Phi-Functions, así como métodos aproximados y formando en conjunto un *Toolbox* para solucionar el problema.

Una síntesis de la revisión de literatura especificando el problema, objetivo buscado, restricciones relacionadas, supuestos, método de solución y herramientas de implementación se presenta en la Tabla 1. Se hace evidente que la mayoría de trabajos se relacionan con procedimientos aproximados y, el objetivo suele ser la maximización de la cantidad de piezas anidadas o la maximización del área resultante aprovechable. Adicionalmente, en todos los trabajos se incluyen mecanismos para evitar el traslape de las piezas, algunos permiten la rotación de las piezas y solo uno que algunas piezas contengan vacíos internos. La gran mayoría emplea la estrategia Bottom-Left o una análoga *step by step* (Lee et al., 2008). En cuanto a los mecanismos de solución, se destacan trabajos con los algoritmos genéticos y *Knapsack problem*.

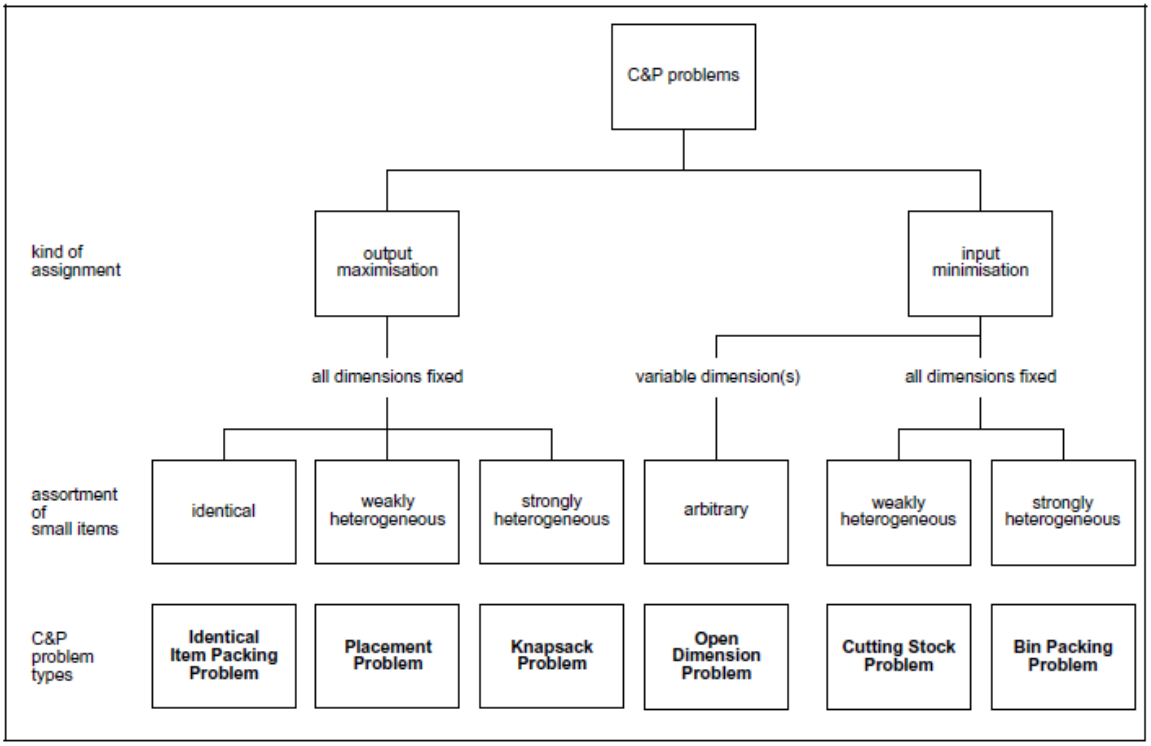


Figura 1: Tipos de problemas básicos (Wäscher, 2007)

Autor	Problema:	Objetivo:	Restricciones:	Supuestos:	Heurística o Método de solución	Herramienta de implementación
Costa et al., 2009	*Heuristic approaches to large-scale periodic packing of irregular shapes on a rectangular sheet*	Maximizar el porcentaje de la hoja a emplear o maximizar el número de piezas dentro de la hoja.	Cualquier rotación de las piezas es permitida, sin embargo una vez determinada no se puede cambiar. Las piezas no pueden traslaparse.	Aproximación de las geometrías de las piezas a polinomios. posicionamiento de las piezas sobre el blanco y evitando traslapes según concepto de *inner-fitpolygon* de Gomes y Oliveira (2006)	The parallel angles heuristic, The equally spaced angles heuristic y The free angles heuristic. Todas iniciando con soluciones aleatorias y optimizando a partir del mejor vecino.	No reportada
Wong et al., 2009	*Two-dimensional irregular objects allocation problems by using a two-stage packing approach*	Minimizar el espacio no utilizado.	Las piezas no pueden traslaparse. No se permite la rotación de las piezas	Representación de las piezas y superficie a cortar por medio del *raster pixel method*.La hoja a cortar es un rectángulo. Cada objeto tiene una orientación definida. Cada pieza puede ocupar cualquier lugar dentro de la hoja a cortar.	Metodología de dos etapas: *Genetic algorithm* para determinar la secuencia inicial. *Two-stage packing approach*, a "manera del juego de tetris" para mejorar la acomodación.	C++
Lee et al., 2008	*nesting problem*	Minimizar el espacio no utilizado.	Las piezas no pueden traslaparse.	Aproximación de las piezas y superficies a cortar a polígonos usando *cluster of straight lines.* Ubicación de las piezas por la regla step-by-step.	*Quick location and movement algorithm*, que emplea el radio de la circunferencia que circunscribe al polinomio para acelerar el posicionamiento de las piezas.	No reportada
Cui, 2007	*two-segment cutting patterns of strips*	Maximizar el número de piezas	Las piezas no pueden traslaparse.	Anidamiento de piezas en un patrón rectangular y eficiente, para luego acomodar cada patrón en la hoja. Búsqueda de la secuencia de localización a modo *knapsack problem.*	*knapsack problem.*	No reportada
Gomes & Oliveira, 2006	Irregular Strip Packing problems	Maximizar el número de piezas	Las piezas no pueden traslaparse.	Las piezas son polinomios (o son aproximadas a los mismos). Se emplea estrategia de acomodación *botton-left*.	Se empleó la meta heurística *simulated Annealing* junto con la heurística *neighbourhoods* para guiar la búsqueda de soluciones en el espacio generado por el modelo de programación lineal.	C and ILOG CPLEX 8.0

Cui, 2006	*Cutting stock problem and multi-segment cutting patterns*	Maximizar el número de circunferencias a cortar	Las piezas no pueden traslaparse.	Las circunferencias pueden tener diámetros variados. La hoja de corte es rectangular.	Ubicación de piezas circulares en elementos regtangulares, y solución por medio de algoritmos para la solución del *knapsack problem*	No reportada
Dowsland et al., 2002	*polygon placement/packing problema*	Localizar todas las piezas dentro de la hoja a cortar minimizando la longitud de la misma.	Los polígonos no tienen agujeros. No es posible la rotación.	La hoja de corte es rectangular y una de sus dimensiones es infinita.	Ubicación de la mayor cantidad de piezas irregulares en una superficie. Para ello usaron: *Botton-left. Heuristics* *The polygon placement algorithm*	No reportada
Babu & Babu, 2001	*Nesting if 2-D parts in 2-D sheets*			Representacion de las piezas a cortar segun raster pixel method. Acomodación usando *botton-left strategy*. Búsqueda de la secuencia de acomodación.	*Genetic Algorithm*	C++

Tabla 1: Listado de Artículos por tipo de problema y método de solución

10 Objetivos

General

Desarrollar un modelo matemático y procedimiento de solución para el *Irregular Two Dimensional Cutting Stock Problem.*

Específicos

- Ampliar la revisión de literatura relacionada con el *Irregular Two Dimensional Cutting Stock Problem* para encontrar lineamientos para su análisis, formulación y solución.
- Formular un modelo de programación matemática para la resolución exacta del problema en instancias pequeñas. Esto también ayudará a orientar los procesos de búsqueda y selección de soluciones de los procedimientos meta heurísticos.
- Implementar las meta heurísticas GRASP y Algoritmo Genético para la solución de instancias mayores del problema.
- Validar el modelo y el procedimiento de solución, garantizando su aplicabilidad en el contexto real de producción y evaluando el desempeño en términos de calidad de las soluciones y el tiempo para obtenerlas.

11 Metodología de la Investigación

Siguiendo los objetivos, la metodología se ejecutó de la siguiente manera:

1. Se amplió la revisión de literatura, se buscó evidencia de formulaciones matemáticas orientadas a soluciones por métodos exactos. Aunque se encontró evidencia de algunas formulaciones no lineales, aún se evalúa su potencial para resolver instancias prácticas del problema. Con todo, se construyó un planteamiento matemático-analítico del problema y la formulación MIP que sirvió como base para el desarrollo de las ideas que permitieron el desarrollo de los procedimientos de solución. Se encontraron procedimientos aproximados empleando conceptos como *Bottom Left* y *Pixel Raster Méthod* que condujeron hacia una formulación.
2. Para la formulación del problema, se asumió que las figuras (pieles y piezas de corte) son polígonos irregulares que a su vez serán representados en retinas o matrices de pixeles. Se determinó que la función objetivo más apropiada es la maximización del área resultante aprovechable y se plantearon sendas restricciones relacionadas con la contención de las piezas dentro de la piel y el no traslape entre piezas. La formulación fue basada en el *Knapsack Problem*.
3. Se desarrollaron rutinas en C# implementando Algoritmos Genéticos y GRASP.
4. Se emplearon técnicas del Diseño Experimental para calibrar los parámetros de las meta heurísticas y se empleó ANOVA para comparar las medias del desempeño en cuanto al tiempo de ejecución y calidad de cada meta heurística.

12 Marco Teórico

12.1 Metodología general

12.1.1 Programación lineal mixta

La Programación Lineal Mixta a diferencia de la Programación Lineal incluye además de expresiones de la forma ***AX,*** donde ***A*** y ***X*** son vectores de coeficientes y vectores de variables de primer orden [1] respectivamente, un conjunto de restricciones impide que al menos una de las variables pueda tomar valores no enteros. De este modo, la solución obtenida por Programación lineal mixta es un subconjunto del espacio factible del problema PL relacionado, convolución del problema ***P***. Pese a ello, los métodos de solución MIP distan en eficiencia de los algoritmos desarrollados para resolver problemas PL, aunque algunos se soportan en ellos.

12.1.2 Métodos de solución exactos

12.1.2.1 Método simplex

Es un algoritmo que busca la solución óptima entre los vértices (soluciones) de la región factible conformada por el espacio dada por la intersección de las restricciones del problema. Para ello emplea en cada iteración el gradiente de la función objetivo que le permite seleccionar una serie de soluciones factibles hasta alcanzar la solución óptima.

12.1.2.2 Algoritmo Branch and Bound

Este algoritmo encuentra soluciones a problemas con variables mixtas (enteras o binarias) con la ayuda del método simplex. En cada iteración se forman dos (pero podrían incluso llegar a ser más) problemas cuya variante es la adición de una (o más) restricción(es) que evitará que variables marcadas como enteras adquieran valores no enteros y etiquetando en cada paso el estatus de las variables involucradas. El procedimiento descrito se repite en cada iteración hasta encontrar la solución del problema mixto, en el estatus de infactible, óptimo o factible.

12.1.3 Métodos aproximados de solución

El problema del corte del cuero encaja en la familia de problemas de corte o anidamiento 2D de una colección de piezas irregulares en una colección de hojas irregulares con dimensiones finitas, razón por la cual los autores de trabajos relacionados con métodos heurísticos y meta heurísticos coinciden en realizar las siguientes actividades para solucionar la problemática de este trabajo:

1. Seleccionar el objetivo adecuado para cada caso. Generalmente reducir el desperdicio o área aprovechable no asignada, o maximizar el número de piezas a acomodar con su correspondiente función de utilidad.
2. Definir un mecanismo de representación de las piezas a cortar o anidar así como de las hojas sujetas a corte. Los mecanismos más usados son la aproximación a polígonos y el método de rastreo de pixeles.

[1] Toda variable está elevada a potencia 1, en el caso de estar elevada a la potencia 0 se entiende que el coeficiente ***A*** que acompaña a dicha variable es un término independiente. Todo esto mantiene los supuestos de linealidad dentro de los modelos de programación lineal.

3. Emplear una estrategia de acomodación que permita ubicar cada pieza sobre la hoja. La heurística o estrategia de solución conocida como "*Botton-Left*" que consiste en mover la pieza a acomodar de izquierda a derecha y de arriba hacia abajo hasta encontrar una ubicación factible, tiene un uso generalizado y su mecanismo de solución busca determinar la secuencia con la que cada pieza será acomodada. Algunos autores proponen estrategias para acelerar la acomodación, como en el caso del empleo del radio de la circunferencia que circunscribe al polígono para evitar recorrer áreas ya asignadas (Lee et al. 2008).
4. Según la heurística de solución, se generan las soluciones iniciales aleatoriamente (casos concretos basados en algoritmos genéticos) o con base a heurísticas voraces, por ejemplo donde la estrategia radica en acomodar las piezas ordenadas en áreas o dimensiones descendentes.
5. Implementar la meta heurística seleccionada con algún lenguaje de modelado como GAMS o AMPL o de propósito general tal como Fortran o JAVA o C++ y ejecutar una librería de instancias de problemas.
6. Documentar los resultados y si es el caso, comparar resultados en términos de calidad de soluciones y tiempos de ejecución, usando técnicas de Diseño de Experimentos.

Para efectos de búsqueda de soluciones en este Trabajo de Grado, se propondrá una función de utilidad, se seleccionará un mecanismo de representación, y se empleará librerías de algoritmos desarrollados por terceros que implementan meta heurísticas en C# como por ejemplo *OpenGenetic* desarrollada por Lapthorn (2003) para algoritmos genéticos. La meta heurística GRASP será implementada en su totalidad en el mismo lenguaje. Para el proceso de calibración se propondrá tres instancias del problema y se emplearán técnicas del diseño experimental para encontrar los valores de los parámetros que conducen sistemáticamente en cada meta heurística a mejores resultados.

12.1.3.1 Algoritmo Genético

Los algoritmos genéticos fueron inspirados en la evolución biológica de modo que su objetivo es utilizar operadores de entrecruzamiento y mutación para hacer evolucionar una población de individuos que representan cada uno una solución evaluable usando una expresión matemática denominada *función de utilidad*. Las soluciones son codificadas en vectores de valores binarios, enteros y reales. Estos vectores de valores se asimilan a cromosonmas (cadenas de genes) y las entradas de cada uno de ellos a los genes, de modo que pueden definirse operaciones entre cromosomas y genes según lo observado en la evolución biológica. En la literatura pueden encontrase diversas maneras de abstraer un problema particular en términos de genes y cromosomas dependiendo de la estructura misma del problema, cabe anotar que el éxito en este procedimiento meta heurístico depende en gran medida de la abstracción realizada. La principal ventaja del uso de los Algoritmos Genéticos radica en la posibilidad de hacer una exploración a partir de la generación inicial de individuos o soluciones de manera aleatoria, lo que supone no es necesario tener un conocimiento muy amplio del problema a solucionar. Su desventaja más notoria es la lentitud relativa con la que evoluciona, consecuencia directa de partir de soluciones aleatorias. Los algoritmos genéticos por lo general aplican durante sus iteraciones dos operadores conocidos como operadores de entrecruzamiento y mutación.

El operador de entrecruzamiento requiere típicamente dos individuos (o soluciones) representados en vectores de valores que pueden ser reales, enteros y binarios, para dividirlos en uno o varios puntos de corte elegidos aleatoriamente, para generar dos o más individuos nuevos conocidos como hijos. En la Figura 2 se muestran dos soluciones representadas por vectores a manera de individuos o soluciones al problema que son cortadas en dos sitios (podrían ser uno o varios puntos según sea la estrategia del investigador) e intercambiada su información en dos nuevos individuos, que serán sumados a la población.

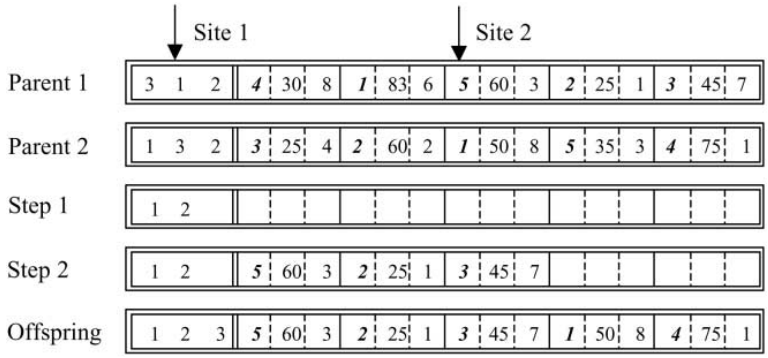


Figura 2: Ejemplo de Operador de Entrecruzamiento (Babu & Babu, 2001)

El operador de mutación consiste en seleccionar algunos valores del vector de individuos tomados al azar y modificarlos. La mutación es el concepto típicamente empleado para que el procedimiento pueda encontrar soluciones globales y escapar de las locales. En la Figura 3 se muestran dos ejemplos de la mutación, en el primero se detalla la elección de 4 puntos sobre la solución y donde, para el problema específico, se efectuará una pareja de intercambios; en el segundo ejemplo, se muestra la elección de 3 genes cuyos valores son reemplazados por valores aleatorios entre 4 (Babu & Babu, 2001).

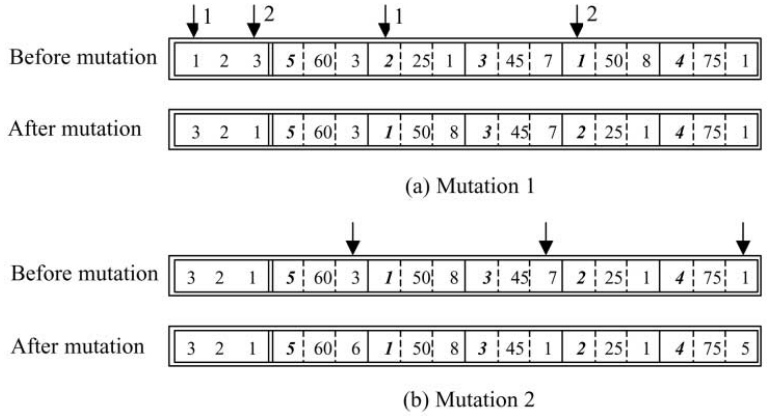


Figura 3: Ejemplos de Mutación. (Babu & Babu,2001)

Todo individuo de la población está sujeto a la probabilidad de ser elegido junto con otro para ser objeto de la aplicación de entrecruzamiento, así mismo todo individuo está sujeto a la probabilidad de ser elegido para ser objeto de mutación. Las probabilidades de entrecruzamiento y mutación son parámetros de la meta heurística así como la manera de evaluar a la población y la política de muerte y supervivencia de los individuos, mecánicas que dependen de la forma del problema particular en estudio.

12.1.3.2 Algoritmo GRASP

GRASP es una meta heurística caracterizada por tener dos fases: una constructiva en donde se arma una solución de inicio y otra de búsqueda local. Durante la primera fase, se emplean reglas o heurísticas voraces para generar una solución inicial que será luego mejorada durante la segunda fase a través de una búsqueda local. El proceso GRASP repite el algoritmo anterior varias veces con cada regla conocida para la generación de soluciones iniciales. Este procedimiento suele encontrar soluciones de buena calidad de manera rápida, aunque su principal desventaja radica en la necesidad de aplicar conocimientos previos (y posiblemente no

disponibles) sobre la estructura del problema para generar las reglas empleadas durante la fase inicial (Resende & Ribeiro, 2003), (Caballero & Alvarado, 2010).

12.1.3.3 Intercambios 2-Optimal

Los intercambios 2-Optimal fueron inicialmente propuestos por Croes (1958) inicialmente para aplicarlos a la solución del *Travelling Salesman Problem*. En esencia consiste en intercambiar arcos de la red y verificar si dicho intercambio produce mejoras al evaluar la nueva solución en la función objetivo. La fase de búsqueda local de GRASP emplea esta técnica para generar el vecindario de soluciones adyacentes a la solución obtenida durante la Fase de Construcción para obtener el óptimo local. En la Figura 4 se muestra el vecindario o conjunto de soluciones adyacentes a una solución actual luego de efectuar el procedimiento 2-óptimal.

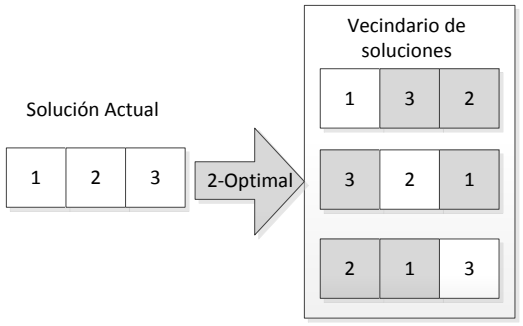


Figura 4: vecindario de soluciones adyacentes a una solución actual a partir de intercambios 2-óptimal.

12.1.4 Swath Method Algorithm:

Este es un algoritmo empleado para determinar si un punto *q* representado por sus coordenadas cartesianas está o no al interior de un polígono irregular, también representado por los segmentos de línea entre sus vértices p_i. *SMA* funciona trazando analíticamente una línea desde el punto *q* hacia el exterior del polígono irregular mientras se cuenta el número de segmentos del polígono intersecados por dicha línea. Si el número de intersecciones es impar, el punto está dentro del polígono, de lo contrario estará fuera de él, según se muestra en la Figura 5, donde cada línea horizontal es interceptada con los segmentos del polígono. Este algoritmo tiene una complejidad computacional polinomial calculada en O(NLogN) (Huang & Shih, 1997).

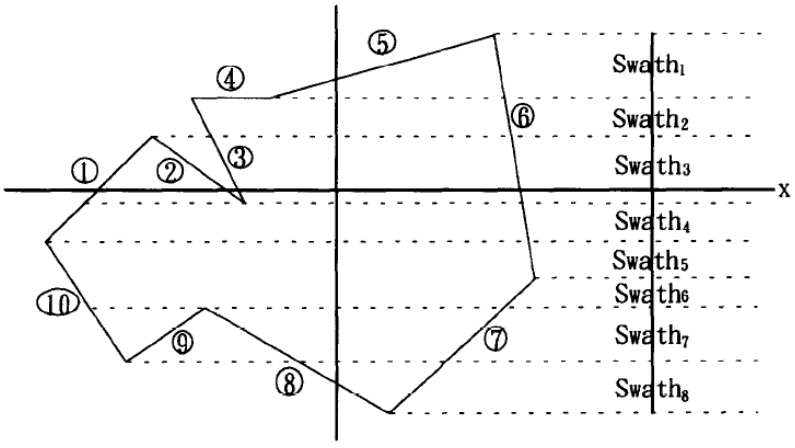


Figura 5: funcionamiento del Swath Method Algorithm (Huang & Shih, 1997).

Swath_Method_Algotihm(P,q)

Parámetros: **P**: Polígono irregular representado a partir de sus vértices p_1, $p_2 ... p_n$, y sus segmentos $e_1, e_2, ... e_n$. en la figura anterior n =10
q: un punto.

Retorno Interior?: variable binaria que tendrá el valor 1 si *q* esta al interior de *P*, 0 en cualquier otro caso.

Iniciar

(Pre procesamiento)
Transformar las coordenadas ***Y*** de ***P*** en valores enteros;
Ordenar las coordenadas ***Y*** de los vértices ***P*** en orden descendiente. Cada ***Y*** consecutivo forma un *Swath*;
Para todo segmento ***e*** **haga**
Determine cuantas *swaths* son interceptadas;
Fin Para Todo
Si Conteo es par
Retorne verdadero
Si No
Retorne falso.

Terminar

Figura 6: Implementación de *Swath Method Algoritm*. Adaptado a partir de Huang & Shih (1997).

12.1.5 Representación de las colecciones de Pieles y Piezas de corte

La representación de las piezas y pieles de corte tiene vital importancia en el proceso, debido a que edifican la base sobre la cual las funciones de utilidad y las restricciones modelan el problema. El problema actual consiste en encontrar una acomodación factible e idealmente óptima de las piezas sobre la piel, para ello se han empleado técnicas de representación como *Pixel Raster Method* y *Non-Fit-Polygon* que presentaremos a continuación:

12.1.5.1 Pixel Raster Method:

Consiste en generar una matriz de *m* x *n* a partir de la superposición sobre ella de una figura geométrica como un polígono, y asignar a continuación las entradas de la matriz (o pixeles). Usando *Swath Method Algorithm* se evaluará cada la ubicación de cada pixel. Los pixeles contendrán un valores de 1 si están en un lugar donde hay material aprovechable y 0 en caso contrario, ver Figura 7;

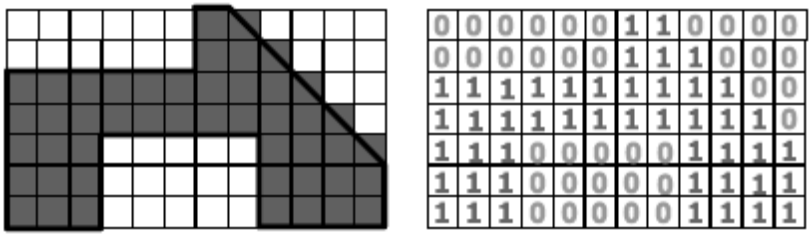


Figura 7: Obtención de una matriz a partir de una figura geométrica.Tomado de Bennell & Oliveira (2008).

Esta representación facilita el desarrollo de operaciones lógicas y matemáticas sobre la matriz que permitan determinar si hay o no superposiciones (Bennell & Oliveira, 2008). Tal como se nota de la Figura 3, al sumar dos matrices binarias pertenecientes a dos polígonos irregulares pueden generar matrices compuesta por valores mayores a 0 y 1, derivados de traslapes.

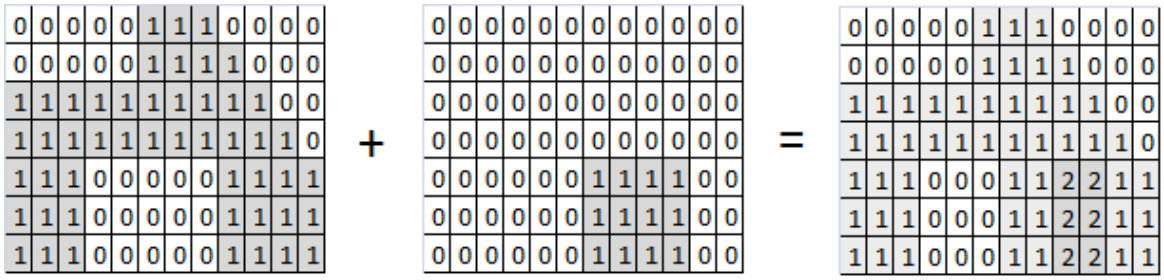


Figura 8: detección de traslapes a partir de Operaciones entre matrices obtenidas por *Pixel Raster Method* según la explicacion de Bennell & Oliveira (2008)

12.1.5.2 Non-Fit-Polygon:

Consiste en representar las formas de las piezas y pieles con polígonos, cuyos vértices son un conjunto ordenado de puntos cartesianos, las coordenadas de dichos puntos pueden ser empleados para efectuar ciertas operaciones matemáticas, encaminadas entre otras a determinar la existencia de traslapes entre polígonos (Costa et al., 2009), ver Figura 9.

Operaciones vectoriales, pueden ser desarrolladas en el plano cartesiano, al respecto, se ilustran dos ejemplos donde a través de la suma de vectores, relacionados con el punto de referencia de la pieza, se logran los efectos de traslación o rotación de la misma:

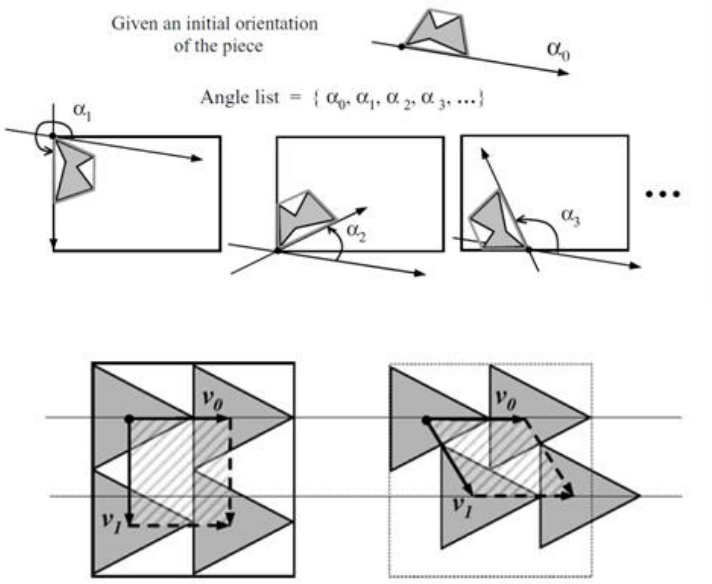


Figura 9: Ejemplos de rotación y traslación de polígonos (Costa et Al.,2009)

Tanto la representación por Pixel Raster Method y Non-Fit-Polygon permiten materializar las restricciones que evitan que las piezas queden traslapadas o se ubiquen sobre un área no adecuada de la piel.

12.2 Metodología particular de la investigación

Para el modelado de la programación matemática, se uso el *Raster Pixel Method* para representar en la piel las piezas a cortar, de manera similar a como Babu y Babu (2001) y Wong et al. (2009) lo hicieron para sus

procedimientos aproximados. Esto permitió pixel a pixel, la construcción de las restricciones que obligan a las piezas a quedar contenidas al interior de la piel y sin traslaparse entre sí. Más adelante, en el proceso de formulación lineal se encontró que el ITDCSP podría representarse usando *Knapsak Problem* adicionando dichas restricciones.

Para la industria, clientes finales de este trabajo, resulta provechosa la maximización del área resultante aprovechable de la piel. Se pensó en agregar piezas cuadradas ficticias que representarían dicha área, de modo que, la sumatoria de las áreas de las piezas ficticias ubicadas dentro de la piel constituye el objetivo a maximizar.

Si bien la *Formulación Lineal Mixta* presentada más adelante resulta novedosa, el hecho de estar basada en *Knapsak Problem* supone las dificultades conocidas para la búsqueda de soluciones óptimas. Por esta razón, el uso de meta heurísticas para la búsqueda de soluciones al ITDCSP resulta justificado.

La revisión de literatura sobre procedimientos aproximados de solución, encontró que autores como Babu y Babu (2001), Dowsland et al. (2002) y Gomes y Oliveira (2006) emplearon una estrategia de acomodación de las figuras conocida como *Bottom Left*, la cual permite al usarla en combinación con *Pixel Raster Method*, hacer acomodaciones de las piezas al interior de la piel que cumplen con las restricción de no traslape. Otra razón para emplear *Bottom Left* está en el hecho que todas las piezas ficticias, por ser localizadas luego de ubicar a las piezas reales, tenderán a ser acomodadas en el área inferior derecha, haciendo que el área resultante aprovechable esté concentrada, lo que es preferible en términos industriales. Se ha desarrollado este procedimiento para evaluar la función de utilidad, que usa este par de conceptos en las meta heurísticas escogidas, y de manera análoga a la función objetivo del modelo de programación lineal mixta propuesto.

La mecánica de ubicación implica que las piezas deben ser acomodadas siguiendo una secuencia y que esta permutación particular determina una solución particular al problema, lo que supone que cada secuencia de acomodación genera una solución alternativa. El problema de optimización surge de manera trivial ya que cada una de las alternativas de acomodación genera un desperdicio diferencial, la idea entonces es buscar aquella alternativa de acomodación que maximice el área aprovechable concentrada. Es necesario comentar que una desventaja del uso de la combinación *Botton Left* y *Pixel Raster Method* subyace en que la *pixelación* de las geometrías a representar y la mecánica de la estrategia de acomodación representan en sí mismas fuentes de desperdicio, hecho que es cuantificado por medio de ejemplos en el Análisis de Resultados y presentado en las conclusiones y recomendaciones para investigaciones futuras.

Hasta este punto, está definido un marco de trabajo para el planteamiento de las meta heurísticas GRASP y Algoritmos Genéticos, que permite bajo procedimientos de programación de computadoras, su implementación y ejecución.

Para el caso del *Algoritmo Genético*, se ideó una codificación de la secuencia de piezas, permutación, que representa una solución en un cromosoma de modo que las operaciones de entrecruzamiento y mutación propias de ésta meta heurística generan nuevas permutaciones que son también soluciones. Se buscó que se pudieran ejecutar con una librería de software libre conocida como *Simple C# Genetic Algorithm* escrita por Barry Lapthorn (2003), la cual reúne rutinas revisadas para los operadores de entrecruzamiento y mutación, permitiendo que nos concentremos en la definición de la función de utilidad y la estructura del cromosoma.

Para el Caso de *GRASP* se empleó como regla voraz ubicar primero en la secuencia Bottom-Left a las piezas según su potencial de generar las mejores soluciones, concepto con el que se construye la *función de utilidad* y la *Lista Restringida de Candidatos*, usada para la construcción de la solución durante la fase constructiva, que luego es mejorada durante la segunda fase usando intercambios 2-*optimal* para la búsqueda local.

Para el análisis de los resultados, se emplearon técnicas del diseño experimental para evaluar el desempeño de las meta heurísticas expuestas, tanto en el objetivo de la solución como en el tiempo empleado para su ejecución.

Para finalizar se proponen lineamientos para guiar investigaciones futuras.

13 Desarrollo del problema

13.1 Representación matemática del problema y complejidad algorítmica

El problema en general consiste en encontrar una permutación $\prod$ de una serie de *n* piezas irregulares acomodadas dentro de un área *P* correspondiente a la piel, que también es irregular, que maximice el área resultante s^*. La complejidad del problema esta acotada por *n!* soluciones posibles donde cada una de ellas Π_i, tiene un desperdicio d^{Π_i} asociado. El problema de optimización puede ser representado de la siguiente manera:

$$s^* = P - Min_{\Pi_i}\{d_1^{\Pi_i}, d_2^{\Pi_i}, \dots, d_{n!}^{\Pi_i}\}$$

$$sa: d^{\Pi_i} \geq 0 \quad \Pi_i \epsilon \{\Pi_1, \Pi_2, \dots, \Pi_{n!}\}$$

Donde $d_j^{\Pi_i}$ representa el desperdicio de ubicar la pieza *j* en el orden determinado para ella siguiendo el ordenamiento Π_i.

Las expresiones particulares de áreas sobrantes s^{Π_i}, se obtienen de la diferencia entre *P*, área total de la piel y la sumatoria de las áreas individuales, $a_j^{\Pi_i}$, usadas por cada una de las *n* piezas a cortar y que dependen de un ordenamiento específico Π_i, tenemos:

$$s^{\Pi_i} = P - \sum_{j=1}^{n} a_j^{\Pi_i}$$

Cada una de las áreas utilizadas por cada pieza está conformada por el área útil usada por la pieza y un área desperdiciada producto de su interacción con las piezas adyacentes:

$$a_j^{\Pi_i} = A_j + d_j^{\Pi_i}$$

Cada desperdicio global d^{Π_i} en una permutación específica Π_i se compone de los desperdicios individuales de cada una de las *n* piezas en ese ordenamiento particular:

$$d^{\Pi_i} = \sum_{j=1}^{n} d_j^{\Pi_i}$$

Resolviendo el algebra se obtiene:

$$s^{\Pi_i} = P - \sum_{j=1}^{n} \left(A_j + d_j^{\Pi_i}\right)$$

$$s^{\Pi_i} = P - \sum_{j=1}^{n} A_j - \sum_{j=1}^{n} d_j^{\Pi_i}$$

El área sobrante está compuesta por una parte que no depende la el ordenamiento, *S*, obtenida de la resta del área de la piel y las áreas de las piezas:

$$S = P - \sum_{j=1}^{n} A_j$$

A la cual se le resta el área del desperdicio relacionado con la permutación de las piezas:

$$s^{\Pi_i} = S - \sum_{j=1}^{n} d_j^{\Pi_i} = S - d^{\Pi_i}$$

En síntesis el problema de optimización se puede representar de una manera distinta centrándose en encontrar la permutación que genere la mayor área aprovechable:

$$s^* = Max_{\Pi_i}\{s^{\Pi_1}, s^{\Pi_2}, \dots, s^{\Pi_{n!}}\}$$

$$sa: 0 \leq s^{\Pi_i} \leq S \quad \Pi_i \epsilon \{\Pi_1, \Pi_2, \Pi_3, \dots, \Pi_{n!}\}$$

Para seguir avanzando en la solución del problema se define el concepto de *Pieza Ficticia*, como aquella pieza de forma cuadrada de lado *L* que no está en el conjunto de piezas a cortar y cuya área representa superficie resultante aprovechable. Áreas resultantes inferiores al de la pieza ficticia no se consideraran aprovechables y en la industria se estima como material de desecho.

La idea de la introducción de estas piezas en la solución del problema recae en medir el área aprovechable, de manera que esta sea cubierta por estas piezas ficticias de lado *L* que pueden ser escaladas por un valor entero k^{Π_i} asociado que depende de la permutación particular.

$$s^{\Pi_i} = k^{\Pi_i} L^2 = S - \sum_{j=1}^{n} d_j^{\Pi_i}$$

Entonces:

$$k^{\Pi_i} = \frac{S}{L^2} - \frac{S \sum_{j=1}^{n} d_j^{\Pi_i}}{L^2}$$

Visto desde un nuevo punto de vista, nuestro problema de optimización consiste en encontrar una permutación que maximice la variable *k* que escala el área aprovechable:

$$k^* = Max_{\Pi_i}\{k^{\Pi_1}, k^{\Pi_2}, \dots, k^{\Pi_{n!}}\}$$

$$sa: 0 \leq k^{\Pi_i} \leq \frac{S}{L^2} \quad \Pi_i \epsilon \{\Pi_1, \Pi_2, \Pi_3, \dots, \Pi_{n!}\}$$

A medida que el área de la pieza ficticia, L^2, disminuye, aumenta la precisión para medir los espacios de desperdicio (susceptibles de ser aprovechables) causado por el ordenamiento de las piezas. El desperdicio aumenta también a medida que aumenta el nivel de incompatibilidad entre las piezas y de estas con la piel, es decir, existe un menor acople simultaneo entre estas. Así, a medida que disminuye el área L^2 el valor de k^{Π_i} aumenta en similar proporción para poder cubrir el área aprovechable s^{Π_i}, ya que como se mostro previamente:

$$s^* = k^* L^2$$

En la practica el rango de k^{Π_i} oscilará entre 1, si la piel es cuadrada y la pieza demandada es idéntica a ella, hasta un número muy grande, teóricamente indefinidamente grande asociado a los Z^+ si tiende a 0.

13.2 Representación geométrica del problema

Para la solución del problema se requiere definir un sistema de coordenadas que permita la localización de cada pieza al interior de la piel y representar las irregularidades propias de cada figura. *Pixel Raster Method* presenta una solución a esta problemática (Bennell & Oliveira, 2008).

El algoritmo PRM tiene como entrada las coordenadas de los vértices de los polígonos irregulares y retorna una matriz de tamaño *M*x*N* de pixeles (que toman valores de 0 o 1 según sea el caso). El proceso de transformación de las figuras, que representan a la piel y a las piezas, requiere la definición de un parámetro *r* que determina el lado de un pixel cuadrado. Cuanto menor sea *r*, sin que llegue a ser 0, la calidad de la retina o matriz de pixeles será mejor en términos de aproximación a la figura y habrá menos desperdicio, pero el tiempo de ejecución será mayor porque requerirá de más iteraciones para abarcar el completo barrido del área, ver Figura 10. *M* y *N* se calculan usando las siguientes expresiones:

$$M = \left\lceil \frac{largo\ de\ la\ figura}{r} \right\rceil$$

$$N = \left\lceil \frac{ancho\ de\ la\ figura}{r} \right\rceil$$

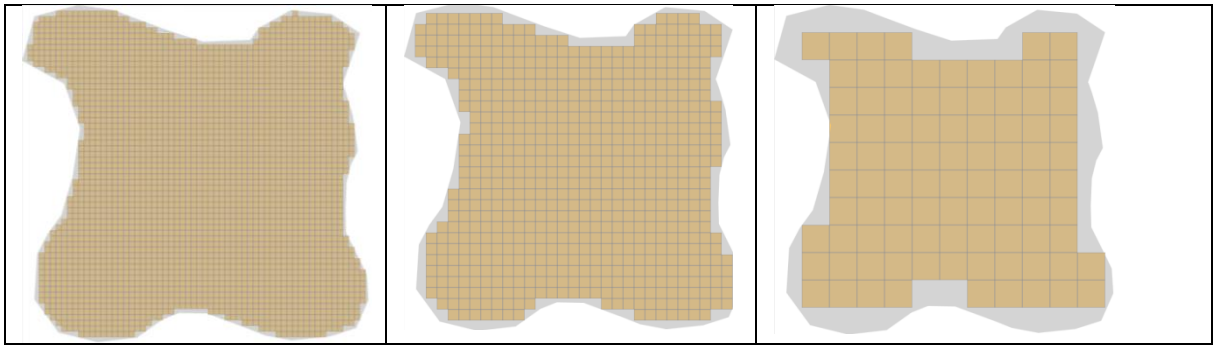


Figura 10: Retinas de matrices para una piel con parámetro *r*= 10, *r* = 20 y *r* = 50

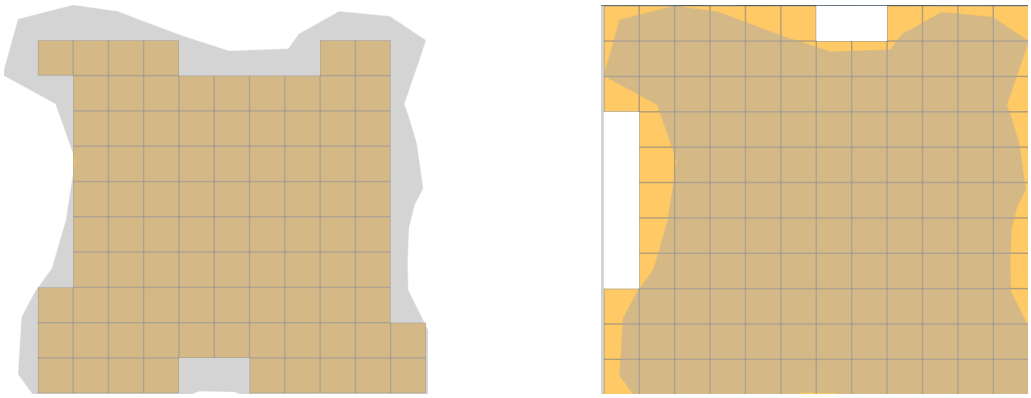
Adicionalmente, la aplicación de PRM en los pixeles en la frontera de las figuras debe ser diferente, dependiendo de si la figura a procesar representa a una piel o a una pieza real o ficticia.

Esto se justificará cuando, al encontrar una solución luego de emplear el procedimiento de acomodación propuesto y que trabaja sobre las matrices de pixeles, se proceda a representar las acomodaciones de las piezas reales sobre la piel según dicha solución, las piezas cercanas a los bordes de la piel no tengan vacíos. Los vacíos pueden presentarse por la pérdida de la representación que las matrices obtenidas por PRM que se hace más notoria para valores de *r* mayores.

Para evitar los posibles vacíos causados por valores superiores de *r*, se modificó el procedimiento PRM para evaluar el tratamiento en la frontera de la figura de la siguiente manera:

- Se dice que un pixel está en la frontera de una figura cuando no todos los puntos que lo conforman estén dentro de la figura.
- Los pixeles en la frontera de la piel serán marcados como no aprovechables, esto significa que tendrán valores de 0, ver Figura 11a.

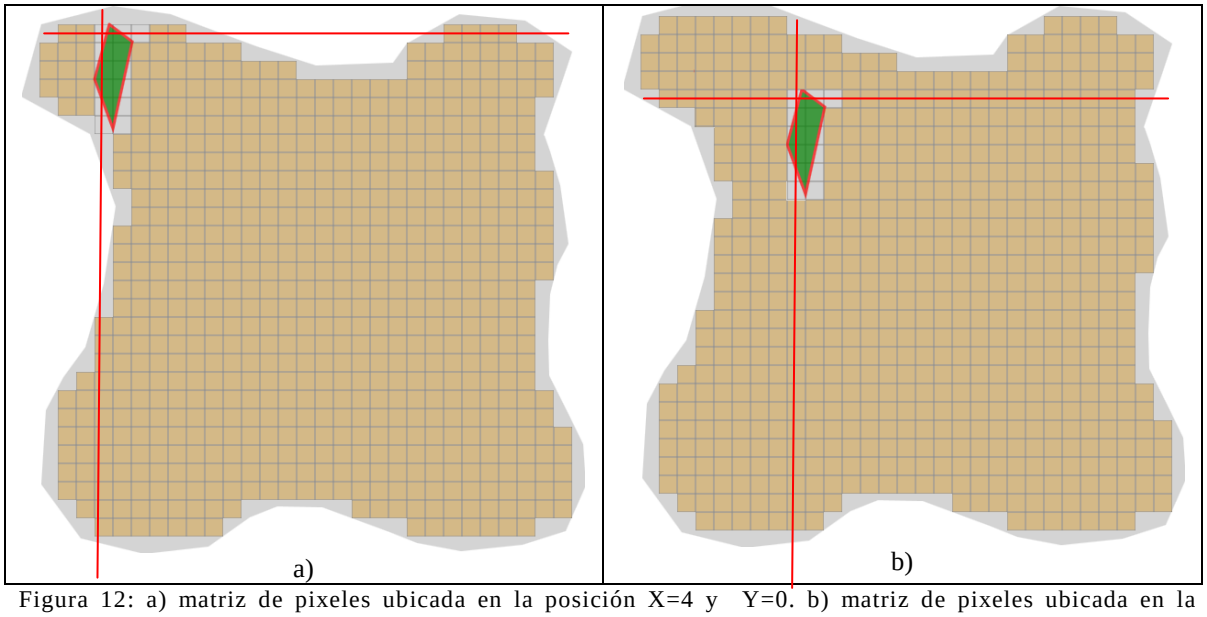
- Los pixeles en la frontera de las piezas (reales o ficticias) serán marcados con 1 mientras contengan un espacio de la pieza, amarillo oscuro, en caso contrario será etiquetados por 0, representado por los espacios blancos, ver Figura 11b.



a) b)

Figura 11 Aplicación de RPM sobre una figura en gris cuyo y cuyo resultado está superpuesto con cuadrados naranja representando a los pixeles donde hay material. a) RMP aplicado a la piel; b) RMP aplicado a piezas.

Dado que las dimensiones de las matrices de pixeles asociadas a las piezas son menores que las de las matrices asociadas a la piel, resulta posible desplazarlas y superponerlas en diferentes posiciones, horizontales, verticales y diagonales, determinando así una posible ubicación para cada pieza sobre la piel. La Figura 12 muestra dos posibles posiciones, referenciadas por la primera posición (horizontal, vertical) en el rastreo izquierda derecha-arriba abajo, que podría ocupar una pieza al interior de la piel, nótese como se activan o no los pixeles bajo la pieza al realizar un movimiento sobre la piel.



a) b)

Figura 12: a) matriz de pixeles ubicada en la posición X=4 y Y=0. b) matriz de pixeles ubicada en la posición X=9 y Y=5.

Explicada la representación de la piel y piezas por medio de pixeles, iniciamos con la descripción de una programación matemática. El modelo a desarrollar estará íntimamente relacionado a la representación geométrica del problema vista hasta aquí, de manera que, pixel a pixel se enmarca el espacio de soluciones, que es susceptible de ser explorado con procedimientos de búsqueda. El uso de *PRM* como base para definir

un modelo de programación entera mixta que delimite un espacio de soluciones resulta novedoso. Posteriormente se propondrán lineamientos para investigaciones futuras que busquen mejorar el modelo propuesto.

13.3 Formulación del Modelo de Programación Lineal Mixta

La Figura 13 presenta un nuevo esquema que determina la manera como fue ideada la programación matemática que se presentará más adelante. En la parte superior de la Figura se muestra las diferentes matrices *P* que definen las posibles posiciones en las que se ubicará una pieza *i*. Hacia el centro, se presentan agrupaciones de matrices *P* para 4 diferentes piezas y se hace un acercamiento de algunas de ellas evidenciando los pixeles o entradas de la matriz que contienen 0 o 1. En la parte inferior de la figura se muestra cómo la elección de algunas matrices *P* por medio de la variables binarias I asociadas, proyectan sus pixeles con valores de 1, sin traslapes, sobre la matriz de la Piel.

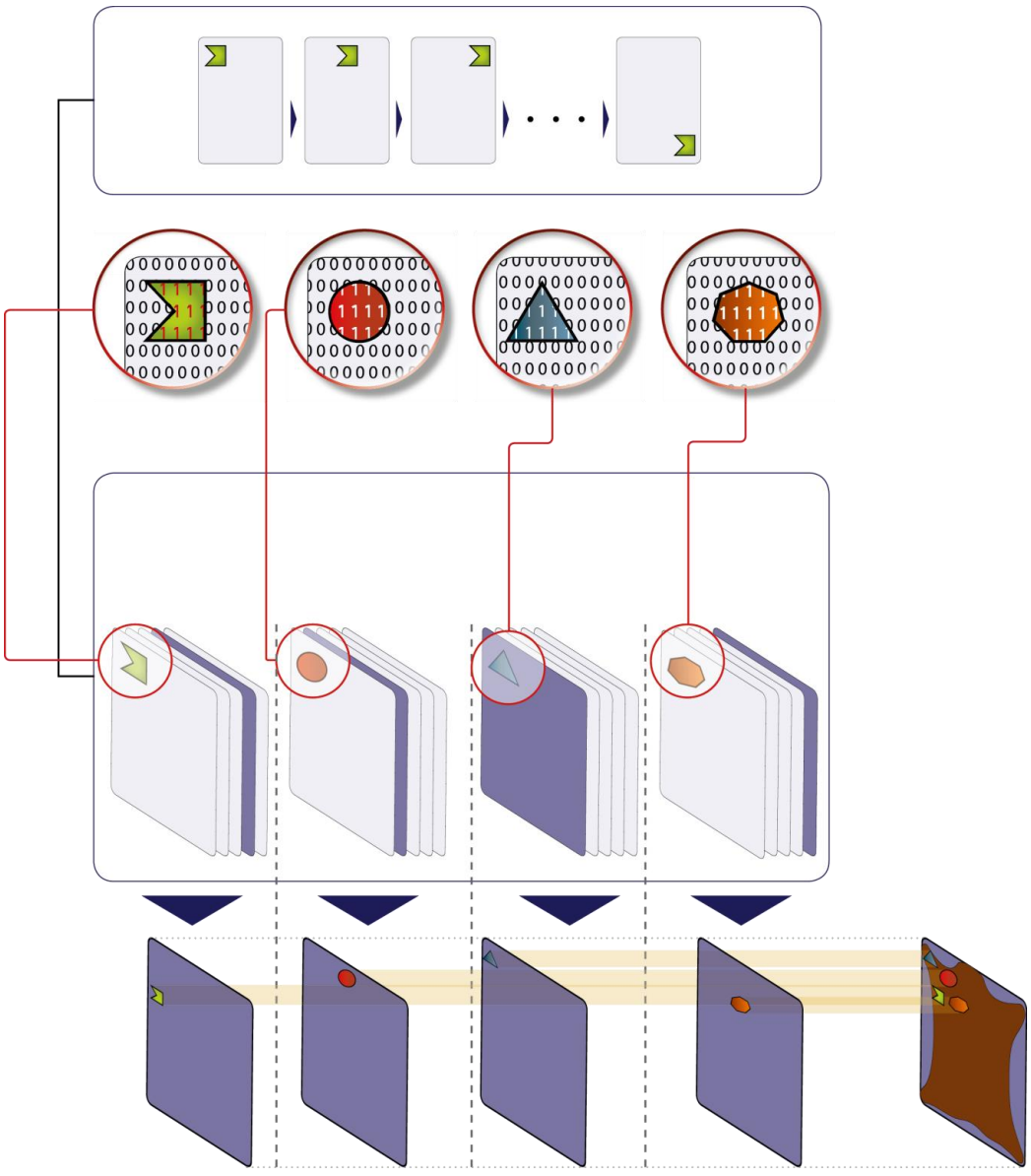


Figura 13: Representación grafica del Modelo de Programación Lineal Mixta

13.3.1 Usando Pixel Raster Method para generar los pixeles

Antes de formular el problema se requiere transformar todos los polígonos asociados a la piel y a las piezas en matrices binarias por el método *PRM*. La Figura 14 presenta la implementación del algoritmo.

Pixel_Raster_Method
Parámetros: **P**: Polígono irregular representado a partir de sus vértices $p_1,p_2 \ldots p_n$, y sus segmentos $e_1,e_2,\ldots e_n$.;
r: Resolución de la matriz;
a: amplitud o ancho horizontal del polígono;
h: amplitud o altura vertical del polígono;
z: Binaria. Toma valores de 1 si se está evaluando una pieza, 0 si es una piel;
Retorno: Matriz Binaria$_{MxN}$;
Variables **w**: Entero. Conteo de esquinas del pixel al interior del poligono a procesar, evitando vacios generados luego de aplicar RPM;
matriz$_{n,m}$: Matriz binaria;
q: Punto. Tomará los valores de las esquinas del pixel a evaluar;

```
Empezar
    (Pre procesamiento);
    M = ⌈a/r⌉;
    N = ⌈h/r⌉;
    Inicializar matriz con MxN entradas;

    Para i = 0 Hasta i<M Haga
        Para j = 0 Hasta j < N Haga

            w = 0;
            /* Contar los pixeles*/
            q = Esquina superior izquierda del pixel;
            Si (Swath_Method_Algotihm(P,q)) Entonces w = w + 1;
            q = Esquina superior derecha del pixel;
            Si (Swath_Method_Algotihm(P,q)) Entonces w = w + 1;
            q = Esquina inferior izquierda del pixel;
            Si (Swath_Method_Algotihm(P,q)) Entonces w = w + 1;
            q = Esquina inferior derecha del pixel;
            Si (Swath_Method_Algotihm(P,q)) Entonces w = w + 1;

            Si((z=1 Y w>0) O (z=0 Y W = 4)) Entonces
                matriz_i,j = 1;
            Si no Entonces
                matriz_i,j = 0;

        Fin Para
    Fin Para

    Retorne matriz;
Terminar
```

Figura 14: Algoritmo Pixel Raster Method. Recibe el polígono a evaluar, la resolución de la matriz, el ancho y alto del polígono y si se está evaluando una piel o una pieza, y retorna una matriz binaria.

13.3.1.1 Parámetros

P_{ixy}^{st}: $\begin{cases} 1 \text{ si hay material superpuesto en el pixel sobre la posición } (s,t) \text{ de la retícula que representa a la pieza } i \text{ y si tiene como posición extrema izquierda superior de la piel a } (x,y). \\ 0 \text{ en caso contrario.} \end{cases}$

δ_{st}: $\begin{cases} 1 \text{ si hay material superpuesto en el pixel sobre la posición } (s,t) \text{ de la retícula que representa a la piel} \\ 0 \text{ en caso contrario.} \end{cases}$

m: Número de piezas reales a ubicar

L: Lado de la pieza ficticia

L^2: Área de la pieza ficticia

r: Lado del pixel

r^2: Área del Pixel

A: Largo de la piel.

B: Ancho de la piel.

M: Numero de filas de la retícula = $\left\lfloor \frac{A}{r} \right\rfloor$

N: Número de columnas de la retícula = $\left\lfloor \frac{B}{r} \right\rfloor$

n: Cantidad de piezas ficticias = $\left\lfloor \frac{A}{L} \right\rfloor * \left\lfloor \frac{A}{L} \right\rfloor$

s: Columna donde se ubica un pixel sobre la retícula, $s=\{1,2,3,\ldots,M\}$

t: Fila donde se ubica un pixel sobre la retícula, $t=\{1,2,3,\ldots,N\}$

13.3.1.2 Variables

I_{xy}^{j} $\begin{cases} 1 & \text{Si la pieza ficticia } j \text{ tiene como pixel extremo izquierdo superior la posición } (x,y) \text{ de la retícula de la piel.} \\ 0 & \text{En caso contrario.} \end{cases}$

W_{xy}^{i} $\begin{cases} 1 & \text{Si la pieza real } i \text{ tiene como pixel extremo izquierdo superior la posición } (x,y) \text{ de la retícula de la piel.} \\ 0 & \text{En caso contrario.} \end{cases}$

13.3.1.3 Modelo

Función Objetivo: La función objetivo maximiza el área resultante aprovechable representada por el conjunto de piezas ficticias activadas. Esto lo realiza maximizando el producto del valor del área de una pieza ficticia cuadrada de lado *L* y la cantidad de piezas ficticias que se posan sobre la retícula de la piel. En otras palabras, maximizar el área resultante aprovechable.

$$Max: L^2 \sum_{x=1}^{M} \sum_{y=1}^{N} \sum_{j=1}^{n} I_{xy}^{j}$$

Sujeto a:

Restricción de superposición de piezas: Cada retícula que compone la piel podría ser ocupada o no por una sola pieza, real o ficticia. Esto evitará que las piezas se traslapen entre sí y que queden sobrepasen el área de la piel.

$$\sum_{j=1}^{n} \sum_{x=1}^{M} \sum_{y=1}^{N} P_{jxy}^{st} I_{xy}^{j} + \sum_{i=1}^{n} \sum_{x=1}^{M} \sum_{y=1}^{N} P_{ixy}^{st} W_{xy}^{i} \leq \delta_{st} \;\; \forall\, (s,t) \in Retícula$$

Restricción de demanda: La siguiente expresión obliga a incluir toda pieza real en una única posición *(x,y)*:

$$\sum_{x=1}^{M}\sum_{y=1}^{N} W_{xy}^{i} = 1 \ \forall \ i$$

En síntesis el modelo maximiza el espacio sobrante aprovechable y está sujeto por las restricciones de superficie de la piel y superposición y demanda de las piezas.

13.3.1.4 Implementación de meta heurísticas

Definida la Programación Lineal Mixta y la representación de las piezas a cortar, y caracterizadas las soluciones, corresponde a las meta heurísticas encontrar las secuencias, permutación, en las que las piezas serán acomodadas siguiendo la estrategia de acomodación sugerida, *Bottom-Left*.

13.3.2 Desarrollo de la función de utilidad y Estrategia de Acomodación

La estrategia de acomodación a emplear será la estrategia conocida como *Bottom-Left*, que consiste en ubicar las piezas de izquierda a derecha y de arriba hacia abajo (Bennell & Oliveira, 2008). A medida que se va encontrando la primera acomodación factible, en caso de encontrarla, se marca el área de la piel como aprovechable y se continua con la siguiente pieza en la secuencia; en caso de no encontrar una posición factible, la función terminará y retornará un valor negativo, por ejemplo -1000. El procedimiento termina cuando no hay más piezas en la lista para acomodar retornando el área aprovechable. Ver Figura 15.

Función_Utilidad
Parámetros: **Piezas**: VECTOR, listado de polígonos que aproximan a las piezas a acomodar;
Piel: MATRIZ Binaria, polígono que contendrá a las piezas;
n: ENTERO, tamaño horizontal de la piel;
m: ENTERO, tamaño vertical de la piel;
l: ENTERO, lado de la pieza ficticia;
Retorno: **Área**: Decimal,
Variables **conteo**: ENTERO, contendrá la cantidad de piezas ficticias agregadas;
mPiel: MATRIZ ENTERO[n,m], matriz de pixeles de la piel;
mPieza: MATRIZ ENTERO[,], matriz de pixeles de la pieza;
rpieza: PIEZA, representa una pieza real por acomodar;
ficticia: PIEZA, pieza que representará el área resultante aprovechable;
lleno: Binario, determina si la acomodación a evaluar resulta factible o no.

```
Empezar
    (Pre procesamiento);
    lleno = falso
    /* Acomodar Piezas Reales*/
    Para i=0; i<Piezas.Cantidad Haga
        rpieza = Piezas_i;
        mPieza = Pixel_Raster_Method(rpieza);
        infactible = 0;
        Aplique Bottom-Left a rPieza sobre mPiel;
        Si (mPieza no es contenida en mPiel) Entonces
            Lleno = Verdadero;
            Retornar -1000;
            Terminar;
    Fin Para
    /* Acomodar piezas Ficticias */
    Mientras que(lleno = falso)
        ficticia = Crear Pieza Ficticia de lado l
        lleno = VERDADERO
        mPieza = Pixel_Raster_Method(ficticia);
```

```
                    Aplique Bottom-Left a rPieza sobre mPiel;
                    Si (mPieza es contenida en mPiel) Entonces
                        conteo = conteo + 1;
                    Si no
                        lleno = Verdadero;
                    Fin si
                Fin mientras que

                RETORNAR conteo*l
Terminar
```

Figura 15: FUNCIÓN DE UTILIDAD EMPLEANDO LA ESTRATEGIA BOTTOM-LEFT

13.3.3 Algoritmo Genético

Se ha escogido un proyecto *Open Source* conocido como *Simple C# Genetic Algorithm* escrita por Barry Lapthorn (2003) y alojado por CodeProject.com. Este código implementa un algoritmo genético básico que permite de manera relativamente sencilla su adaptación a nuestro problema. Al respecto, se requiere la definición de un cromosoma que presenta la solución del problema y una función de utilidad que la evalúa en términos del objetivo de optimización. Por lo demás, dicha librería contiene ya implementados algoritmos de entrecruzamiento, mutación y selección de individuos por torneo o ruleta, ver Figura 16.

```
Algoritmo_Genético
Parámetros:     FuncionUtilidad fu: Función de utilidad.
                Entero t: Cantidad de iteraciones a realizar
                Decimal c: probabilidad de entrecruzamiento
                Decimal m: probabilidad de mutación.
Retorno:        Vector_Entero: solución;
Variables       Entero i, Iteración actual;
                Entero p1, p2, determinarán
                Vector_Entero: individuo;
                Lista Individuos: población;
Empezar
                poblacion = Generar población inicial
                        PARA i = 0 HASTA i <= t HAGA
                                /* Evaluar población*/
                                PARA j=0 HASTA j <= poblacion.Tamaño
                                        Individuo = poblacion[j];
                                        Evaluar Individuo usando fu;
                                FIN PARA
                                /* Aplicar operador de entrecruzamiento */
                                PARA j=0 HASTA j <= poblacion.Tamaño
                                        Individuo = población[j];
                                        SI(Aleatorio.Uniforme(0,1) < c) ENTONCES
                                                Aplicar entrecruzamiento entre individuo y
                                                población[Aleatorio(1,población.Tamaño);
                                                Agregar Hijos a población;
                                        FIN SI
                                        SI(Aleatorio.Uniforme(0,1) < m) ENTONCES
                                                Aplicar Mutación a individuo;
                                        FIN_SI
                                FIN PARA
                        FIN PARA
                RETORNE mejor Individuo;
Terminar
```

Figura 16: Pseudocódigo del algoritmo genético básico desarrollado por Lapthorn (2003)

13.3.3.1 Definición del Cromosoma y su Proceso de Decodificación

El cromosoma esencialmente representará una secuencia codificada con la que cada pieza será acomodada siguiendo la estrategia *Bottom-Left*. Al codificar la secuencia evitamos la aplicación de estrategias de reparación de cromosomas luego efectuar procedimientos de entrecruzamiento y mutación sobre cada individuo o cromosoma. La codificación empleada aquí es la propuesta por Wilkes en 1964 y denominada como Lista Circular. Usando este concepto, llenaremos el cromosoma con valores aleatorios uniformes entre 0 y $n - i$, donde n e i corresponden a la cantidad n de piezas a ubicar en la posición i del cromosoma, y que a su vez representaran posiciones relativas en la secuencia *Bottom-Left*.

Para ilustrar la codificación de la secuencia, llamemos $\boldsymbol{C}$ al vector de n genes que representa al cromosoma o individuo y $\boldsymbol{c_i}$ al valor del gen $\boldsymbol{i}$ de $\boldsymbol{C}$. los valores de $\boldsymbol{c_i}$ serán llenados con valores aleatorios uniformes entre 1 y $\boldsymbol{n\text{-}i}$. Ver Figura 17.

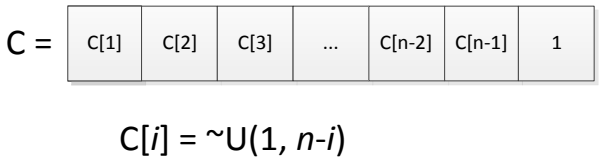
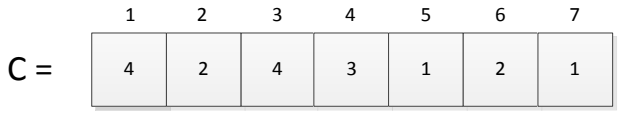
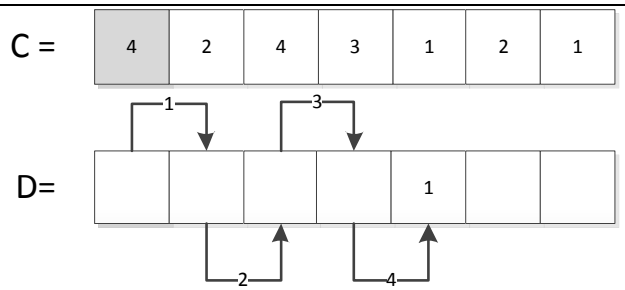


Figura 17: Esquema de generación de los Cromosomas

Dado que la codificación propuesta no representa de manera directa una secuencia de acomodación, el proceso de decodificación requerirá la generación de un nuevo vector de tamaño $\boldsymbol{n}$ al que llamaremos $\boldsymbol{D}$ y que contendrá la secuencia decodificada a evaluar usando *Bottom-Left*. Si $\boldsymbol{d_i}$ es el valor en el gen $\boldsymbol{i}$ en $\boldsymbol{D}$, $\boldsymbol{d_i}$ determina que la pieza correspondiente estará en la posición $\boldsymbol{i}$ de la secuencia *Botom-Left*.

El proceso de decodificación de $\boldsymbol{D}$ a partir de $\boldsymbol{C}$ se presenta a continuación en la Tabla 2 a modo de ejemplo:

Paso 0: Obtener el Cromosoma o Vector $\boldsymbol{C}$:	
Paso 1: Ubicar la pieza 1 luego de $\boldsymbol{c_1}$ posiciones en $\boldsymbol{D}$:	

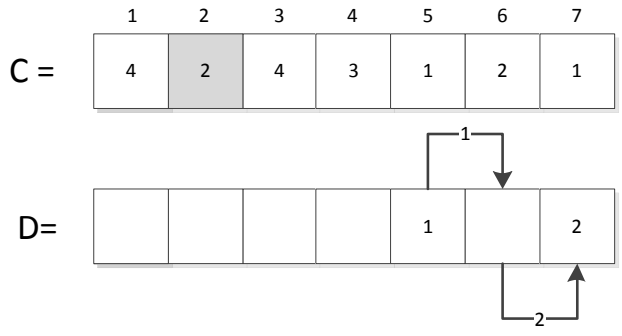
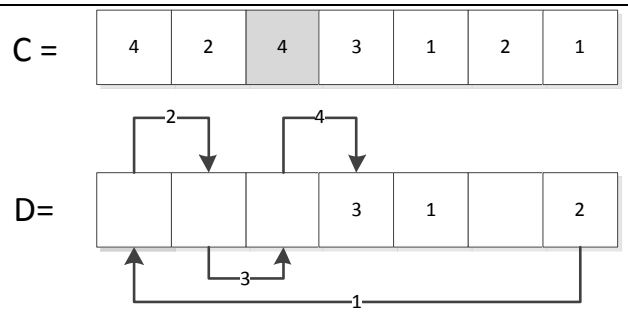
Paso 2: Ubicar la pieza 2 luego de c_2 posiciones a partir de la posición anterior.	
Paso 3: Ubicar la pieza 3 luego de c_3 posiciones a partir de la posición anterior.	
Paso k: Ubicar la pieza k luego de c_k posiciones a partir de la posición determinada en el paso $k-1$.	

Tabla 2: Ejemplo de decodificación del cromosoma en la secuencia de acomodación de piezas según el proceso de decodificación.

13.3.3.2 Definición de Operador de Entrecruzamiento

El operador de entrecruzamiento requiere dos individuos que actuarán a como padres y a los que llamaremos P_1 y P_2 y que serán copiados hasta cierto punto en los hijos H_1 y H_2 respectivamente. A partir de dicho punto, se copiará la información del padre contrario, así las entradas de H_1 y H_2 desde el punto de corte coincidirán con las entradas de P_2 y P_1 respectivamente, ver Figura 18. El pseudocódigo está contenido en la librería de código desarrollado por Lapthorn (2003).

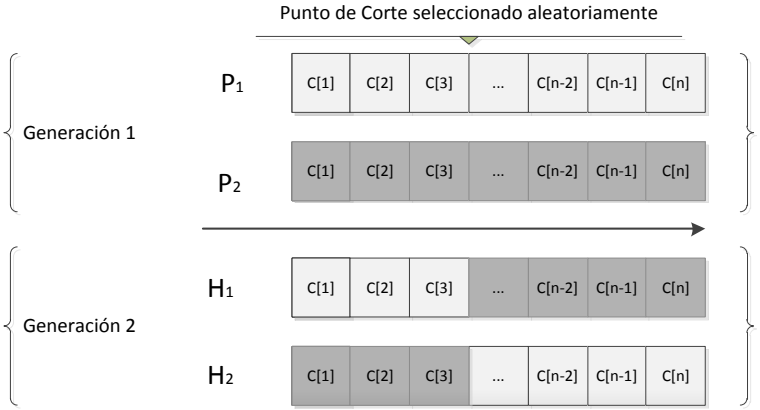


Figura 18: Funcionamiento del Operador de entrecruzamiento

13.3.3.3 Definición de Operador de Mutación

El operador de mutación trabaja seleccionando un conjunto de individuos según un parámetro de probabilidad a los que se les aplica una modificación de una entrada seleccionada aleatoriamente así: sea ***C*** un individuo del conjunto de individuos seleccionado, sea c_i el valor que contiene el gen que pertenece al vector ***C***, e ***i*** un numero aleatorio distribuido uniformemente entre 1 y la cantidad de elementos en ***C***. El procedimiento consiste en hacer c_i igual a un número aleatorio uniforme distribuido entre 0 y n, ver Figura 19.

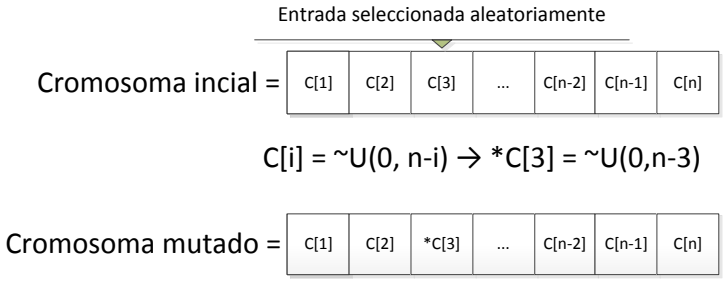


Figura 19: Ejemplo del funcionamiento del Operador de Mutación

13.3.3.4 Selección de individuos

La selección de individuos candidatos para pasar a la siguiente generación se realiza por ruleta según el trabajo de Lapthorn (2003).

La regla de selección por ruleta consiste en asignar un valor de probabilidad a cada individuo de la población, dicha probabilidad es directamente proporcional a la adaptación del individuo según la función de utilidad.

13.3.4 GRASP

El procedimiento GRASP, durante la Fase de Construcción, fue desarrollado completamente en este trabajo, al no encontrarse evidencia de librerías *Open Source*. En general, GRASP ejecuta iterativamente la fase constructiva y de búsqueda local, hasta terminar el número de repeticiones definido, momento en el cual retorna la mejor solución encontrada. Ver Figura 20.

El procedimiento GRASP requiere del uso de una regla o heurística voraz para la generación de la solución inicial, en este caso, un orden particular con la que las piezas deben ser ordenadas en la secuencia *Bottom-Left*.

Durante la fase de búsqueda local se generará de manera iterativa el vecindario de soluciones adyacentes a la solución actual, la generada por la etapa de construcción, con el objeto de encontrar un mejoramiento local de la función objetivo. El procedimiento de búsqueda local termina cuando ningún intercambio produzca mejores soluciones, se ha "alcanzado" el óptimo local.

Algoritmo_GRASP	
Parámetros:	**FuncionUtilidad** fu: Función de utilidad; **LISTA PIEZA** Elementos; **ENTERO** numElementos, Cantidad de Piezas a acomodar según secuencia;
Retorno:	**Vector_Entero**: mSolución, representa la secuencia por los índices de los trabajos en sus entradas;
Variables	**VECTOR[**numElementos**] ENTERO** mSolucion: contiene la secuencia solución
Empezar	

	Ejecutar FaseConstructiva(Elementos) **MIENTRAS QUE** CriterioParada = Falso **Ejecutar** BusquedaLocal **FIN MIENTRAS QUE** **RETORNE** mSolucion
Terminar	

Figura 20: Algoritmo GRASP

13.3.4.1 Codificación de la solución

La solución será representada por un vector de entradas enteras e indexadas. Así y a diferencia de lo propuesto para el Algoritmo Genético, la solución corresponde a un ordenamiento particular del conjunto de las etiquetas de las piezas en los campos del vector. En la Figura 21 se muestra una solución, el vector tiene en el primer campo, utilizando la estrategia *Bottom-Left*, la pieza 1, a continuación la pieza 3 y así sucesivamente.

C =

1	3	4	2	5	6	7

Figura 21: Representación de una secuencia solución, donde cada entrada corresponde a la etiqueta de la pieza en la secuencia.

13.3.4.2 Cálculo del área de las piezas:

El área de las piezas a cortar es criterio para la generación de la solución inicial durante la fase constructiva. Dicha área se calculó a partir de producto entre el área del pixel y la sumatoria de los valores 1 presentes en la retina o matriz de pixeles asociada a la pieza en evaluación, según se presenta en la siguiente ecuación:

$$A_p = r^2 \sum_{i=1}^{n} \sum_{j}^{m} X_{ij}$$

Donde A_p es el área de la pieza *p*, *r* el lado del pixel, *n* y *m* el numero de filas y columnas de la retina y X_{ij} es 1 si el pixel ubicado en la posición *i,j* representa material aprovechable y 0 en cualquier otro caso

13.3.4.3 Fase constructiva

La fase constructiva propuesta inicia con el vector de valores enteros que representa una secuencia de acomodación de las piezas reales. Con cada iteración, se selecciona aleatoriamente una pieza de la Lista Restringida de Candidatos (*LRC*) que es generada al inicio de la iteración. La generación de *LRC* requiere de un parámetro *alpha* (α) y una Función de Utilidad (*FU*). El parámetro α corresponde al tamaño de *LRC* y puede ser expresado como un número entero menor o igual al tamaño de la lista de piezas reales a ubicar, o en términos porcentuales del tamaño de la misma lista. *FU* debe permitir la evaluación del decremento del área resultante aprovechable de la piel al agregar una pieza adicional, teniendo en cuenta las piezas ya acomodadas en iteraciones anteriores. El pseudo algoritmo para *FU* se presenta en la Figura 22 y la Figura 23, que explica cómo se ejecuta la fase constructiva de GRASP a partir de una lista de piezas reales por ubicar y el parámetro α.

```
Función Utilidad
Parámetros:     LISTA PiezasAcomodadas;
                PIEZA piezaPorAcomodar
                PIEL piel;
                DECIMAL AreaPiezaFicticia;
Retorno:        DECIMAL AreaRemanente dada la acomodación
Variables       ENTERO contador;
Empezar
                Ubicar las piezas de la lista PiezasAcomodadas en la Piel siguiendo Bottom-Left
                Ubicar la piezaPorAcomodar  siguiendo Bottom-Left
                Contador = 0;

                MENTRAS (pueda ubicarse una pieza ficticia adicional) HAGA
                        Ubicar pieza ficticia adicional sobre Piel empleando Bottom-Left
                        Contador = Contador + 1
                FIN MIENTRAS


                Retornar Contador*AreaPiezaFicticia;
Terminar
```

Figura 22: Función de Utilidad GRASP

```
Fase Constructiva
Parámetros:     LISTA P, lista de piezas reales a acomodar.
                ENTERO α
Retorno:        VECTOR ENTERO S, Solución inicial;
Variables
                LISTA LRC, Lista restringida de candidatos;
Empezar
                S =  vacio

                MIENTRAS QUE Existan piezas en la lista P

                        Evaluar FuncionUtilidad con todas las piezas en P y
                        Tomar las α piezas mejores para alimentar con ellas a LRC
                        Seleccione aleatoriamente una pieza en LRC
                        Remueva la pieza seleccionada de la lista P
                        Adicione la pieza seleccionada a la soluciónInicial

                FIN MIENTRAS

                Retornar solucionInicial;
Terminar
```

Figura 23: Fase Constructiva de GRASP

13.3.4.4 Fase de Búsqueda Local

La fase de Búsqueda local, requiere de la solución inicial obtenida en la fase previa, se soporta en intercambios 2-optimal cuyo efecto es evaluado con la función de utilidad a través de un procedimiento iterativo. Ver Figura 24 y Figura 25.

```
Búsqueda_Local
Parámetros:     VECTOR ENTERO solActual, obtenida por durante fase contructiva;
Retorno:        VECTOR ENTERO mejorSol, mejor solución encontrada;
Variables       VECTOR ENTERO vecino, solucion adyasente a la solucion actual;
                Elemento elemento;
                BOOLEANO mejora;
Empezar
                solActual = new Solucion(Elementos.Count);
                        //agregar el orden actual a la solucion;
                        PARA ENTERO i = 0 HASTA Elementos.Count
                                elemento = Elementos[i];
                                solActual.secuencia[solucion.IndexOf(elemento)] = i
                        FIN PARA
                        solActual.fitness = FitnessFunction(solActual.secuencia)
                        //compare los vecinos hasta que no haya un vecino mejor
                        HACER
                                mejora = false;
                                vecino = ObtenerMejorVecino(solActual);
                                        SI (vecino.fitness > solActual.fitness) ENTONCES
                                                solActual = vecino;
                                                mejora = true;
                                        FIN SI
                        FIN HAGA
                MIENTRAS NO mejora;
Terminar
```

Figura 24: Fase de búsqueda local

```
ObtenerMejorVecino
Parámetros:     VECTOR ENTERO solucion
Retorno:        VECTOR ENTERO mejorSol, mejor solución encontrada;
Variables       VECTOR ENTERO vecino
Empezar
                Inicializar vecino
                        vecino.AJUSTE = -100000;
                        PARA ENTERO i = 0; HASTA solucion.TAMAÑOVECTOR - 1
                                PARA ENTERO j = i + 1 HASTA solucion.TAMAÑOVECTOR
                                        vecino.secuencia = solucion.Clone();
                                        intercambiarElementos(vecino, i, j);
                                        vecino.fitness = FitnessFunction(vecino.secuencia);
                                        SI (vecino.fitness > solucion.fitness)ENTONCES
                                                RETORNE vecino;
                                                FIN FUNCION
                                        FIN SI
                                FIN PARA
                        FIN PARA
                RETORNE vecino;
Terminar
```

Figura 25: Función Mejor Vecino. Apoya la búsqueda local.

13.3.4.5 Instancias del Problema

El proceso de validación tiene como propósito determinar el nivel de utilidad práctica de los procedimientos de solución desarrollados en este trabajo. Se crearon 3 instancias del problema cuya diferencia radica en la cantidad de piezas a cortar. Hay 5 tipos de piezas para los cuales las instancias A, B, y C requerían cantidades

a cortar de 1, 2 y 5 unidades respectivamente, el tamaño de la piel y su geometría se determinaron de modo tal que cada instancia tuviera una solución factible, ver Tabla 3,Tabla 4, Tabla 5.

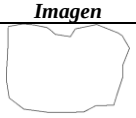
Figura	*Cantidad*	*Dimensiones*		*Imagen*	*Puntos en los vértices del polinomio (x,y)*
Piel		Largo	190		(6,8)(5,73)(0,162)(9,219)(53,252)(129,262)(187,262)(233,260) (268,242)(301,241)(322,240)(331,211)(349,157)(349,122)(361,92) (369,72)(347,35)(335,27)(294,11)(271,2)(207,15)(192,38)(147,31) (131,17)(122,11)(87,4)(52,0)
		Ancho	370		
		Área	81271		
Pieza 1	1	Largo	114		(87,94)(71,154)(91,208)(112,113)
		Ancho	42		
		Área	2419		
Pieza 2	1	Largo	172		(78,80)(74,138)(73,211)(130,218)(147,179)(141,125)(144,76) (188,132)(179,65)(125,47)(80,45)
		Ancho	115		
		Área	12850		
Pieza 3	1	Largo	41		(19,17)(17,36)(26,57)(52,52)(58,26)(44,14)
		Ancho	42		
		Área	1349		
Pieza 4	1	Largo	83		(94,86)(86,115)(115,140)(119,116)(133,93)(159,91)(170,108) (194,114)(205,88)(197,70)(171,57)
		Ancho	120		
		Área	4341		
Pieza 5	1	Largo	70		(208,329)(232,328)(237,300)(249,277)(245,269)(233,287) (232,264)(228,260)(222,282)(216,260)(210,260)(215,285) (204,269)(198,269)(205,289)(205,303)(190,293)(185,298)
		Ancho	65		
		Área	2270		

Tabla 3: Instancia A del problema

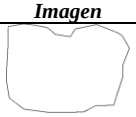
Figura	*Cantidad*	*Dimensiones*		*Imagen*	*Puntos en los vértices del polinomio (x,y)*
Piel		Largo	190		(6,8)(5,73)(0,162)(9,219)(53,252)(129,262)(187,262)(233,260) (268,242)(301,241)(322,240)(331,211)(349,157)(349,122)(361,92) (369,72)(347,35)(335,27)(294,11)(271,2)(207,15)(192,38)(147,31) (131,17)(122,11)(87,4)(52,0)
		Ancho	370		
		Área	81271		
Pieza 1	2	Largo	114		(87,94)(71,154)(91,208)(112,113)
		Ancho	42		
		Área	2419		
Pieza 2	2	Largo	172		(78,80)(74,138)(73,211)(130,218)(147,179)(141,125)(144,76) (188,132)(179,65)(125,47)(80,45)
		Ancho	115		
		Área	12850		
Pieza 3	2	Largo	41		(19,17)(17,36)(26,57)(52,52)(58,26)(44,14)
		Ancho	42		
		Área	1349		
Pieza 4	2	Largo	83		(94,86)(86,115)(115,140)(119,116)(133,93)(159,91)(170,108) (194,114)(205,88)(197,70)(171,57)
		Ancho	120		
		Área	4341		
Pieza 5	2	Largo	70		(208,329)(232,328)(237,300)(249,277)(245,269)(233,287) (232,264)(228,260)(222,282)(216,260)(210,260)(215,285) (204,269)(198,269)(205,289)(205,303)(190,293)(185,298)
		Ancho	65		
		Área	2270		

Tabla 4: Instancia B del Problema

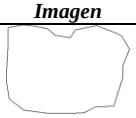
Figura	*Cantidad*	*Dimensiones*		*Imagen*	*Puntos en los vértices del polinomio (x,y)*
Piel		Largo	190		(6,8)(5,73)(0,162)(9,219)(53,252)(129,262)(187,262)(233,260) (268,242)(301,241)(322,240)(331,211)(349,157)(349,122)(361,92) (369,72)(347,35)(335,27)(294,11)(271,2)(207,15)(192,38)(147,31) (131,17)(122,11)(87,4)(52,0)
		Ancho	370		
		Área	81271		
Pieza 1	5	Largo	114		(87,94)(71,154)(91,208)(112,113)
		Ancho	42		
		Área	2419		
Pieza 2	5	Largo	172		(78,80)(74,138)(73,211)(130,218)(147,179)(141,125)(144,76) (188,132)(179,65)(125,47)(80,45)
		Ancho	115		
		Área	12850		
Pieza 3	5	Largo	41		(19,17)(17,36)(26,57)(52,52)(58,26)(44,14)
		Ancho	42		
		Área	1349		
Pieza 4	5	Largo	83		(94,86)(86,115)(115,140)(119,116)(133,93)(159,91)(170,108) (194,114)(205,88)(197,70)(171,57)
		Ancho	120		
		Área	4341		
Pieza 5	5	Largo	70		(208,329)(232,328)(237,300)(249,277)(245,269)(233,287) (232,264)(228,260)(222,282)(216,260)(210,260)(215,285) (204,269)(198,269)(205,289)(205,303)(190,293)(185,298)
		Ancho	65		
		Área	2270		

Tabla 5: Instancia C del Problema

13.4 Calibración de parámetros para GA

Para determinar cuáles niveles de los parámetros resultan adecuados para la ejecución de la mayoría de las ejecuciones del algoritmo genético, planteamos un diseño factorial 2^k que usa k factores con 2 niveles. Los factores controlables que se presume afectan a las variables de respuesta definidas como el rendimiento en términos de tiempo y la eficacia en términos de la función objetivo son 4: Probabilidad de entrecruzamiento, Probabilidad de Mutación, tamaño de la población y el número máximo de generaciones a ejecutar.

Probabilidad de Entrecruzamiento: Es la probabilidad que tiene un individuo en la población para ser seleccionado junto con otro para la aplicación del operador de entrecruzamiento que da origen a dos nuevos individuos con rasgos de ambos padres y la posibilidad de tener una mejor adaptación. Para el diseño factorial se seleccionaron arbitrariamente 2 niveles: 0.7 para el nivel bajo y 0.9 para el nivel alto.

Probabilidad de Mutación: Es la probabilidad que tiene un individuo en la población para ser seleccionado y sobre él aplicado el operador de mutación. El material genético del individuo mutado agregará diversidad a la población. Para el diseño factorial se seleccionaron arbitrariamente 2 niveles: 0.01 y 0.05 para los niveles bajo y alto respectivamente.

Tamaño de la población: En consecuencia de la aplicación del operador de entrecruzamiento, el tamaño de la población tiende a incrementarse de generación en generación. Este incremento no es deseable por varias razones dentro de las cuales está la limitación en el tamaño de la memoria de la computadora que ejecuta el procedimiento. Por esta razón, luego de aplicarse los operadores se realiza un proceso de selección de individuos que determina cuáles de ellos pasarán a la siguiente generación y cuales serán eliminados. El tamaño de la población para efectos del diseño factorial tendrá 2 niveles: 100 para el nivel bajo y 200 para el nivel alto.

Máximo número de generaciones: hace referencia a la cantidad de generaciones, cada incluyendo su proceso de entrecruzamiento, mutación y selección respectivos, que se ejecutarán determinando el criterio de parada. Por la experiencia de otros autores se sabe que entre más generaciones se ejecuten calidad de las soluciones aumenta asintóticamente hasta la solución óptima. Sin embargo, dado que la ejecución toma cierto tiempo del CPU, resulta conveniente limitar este parámetro. Para efectos del diseño factorial, los niveles fueron arbitrariamente ajustados en 200 para el bajo y 400 para el alto.

A continuación se presenta en resumen los resultados para la instancia B empleando los 4 factores en sus niveles.

Tamaño de la Población			100				200			
Máxima cantidad de Generaciones			200		400		200		400	
Entrecruzamiento	Mutación	Replica	Tiempo [s]	Función objetivo	Tiempo [s]	Función objetivo	Tiempo [s]	Función objetivo	Tiempo [s]	Función objetivo
0.7	0.01	1	772	101150	1346	100950	809	101050	1534	101000
		2	771	100950	1342	101200	813	101150	1540	101000
		3	803	101150	1350	101100	777	101000	1557	101100
		4	773	101000	1332	101000	798	101100	1567	101050
		5	800	101050	1361	100900	789	101150	1568	100800
		6	816	101000	1378	100950	818	100950	1548	101000
		7	780	101050	1323	101100	795	101100	1548	101000
		8	769	100900	1354	101150	808	100850	1530	101100
		9	806	101100	1332	101050	820	101100	1549	101250
		10	772	101050	1358	100900	772	101150	1539	101150
	0.05	1	779	97900	1328	98200	781	98100	1525	97950
		2	769	98350	1359	97900	778	98100	1574	97850
		3	799	98150	1367	97900	827	98050	1539	97950
		4	796	98050	1324	97900	821	98000	1570	98150
		5	791	97900	1357	98100	802	98150	1558	97900
		6	783	98050	1376	97900	794	98150	1530	98000
		7	760	98000	1346	98000	817	97950	1532	98000
		8	806	98000	1378	98100	791	98100	1534	98150
		9	778	97800	1344	98100	779	97800	1549	98150
		10	779	98050	1341	97800	776	98150	1527	98000
0.9	0.01	1	769	102550	1350	102400	837	102550	1551	102400
		2	775	102550	1355	102550	774	102650	1560	102750
		3	785	102400	1330	102550	770	102600	1542	102400
		4	757	102500	1337	102350	808	102550	1524	102450
		5	786	102300	1348	102400	792	102700	1546	102550
		6	803	102600	1327	102450	824	102550	1571	102500
		7	773	102600	1342	102550	798	102550	1562	102450
		8	772	102400	1346	102450	780	102550	1568	102650

		9	819	102600	1376	102450	807	102500	1546	102550
		10	819	102600	1364	102600	799	102450	1549	102300
	0.05	1	781	99900	1367	99800	817	100050	1540	100000
		2	773	100000	1344	99900	789	100050	1593	100150
		3	773	99900	1358	100050	786	99900	1551	100000
		4	782	99950	1349	99950	795	100100	1563	100000
		5	764	99900	1380	100200	793	100050	1570	100000
		6	770	100050	1343	99950	805	100100	1569	100050
		7	773	100100	1363	100050	813	100100	1530	100100
		8	777	100050	1376	100100	809	100100	1535	100000
		9	796	100000	1354	100100	826	100150	1555	100000
		10	797	100000	1344	100000	786	100000	1554	99900

Tabla 6: Resumen de tiempos y valores para la función de utilidad en términos de los Factores con sus niveles y réplicas, empleados en un diseño factorial 2^k completo.

Antes de realizar la ejecución del análisis factorial, se requiere probar los supuestos de normalidad y homogeneidad de varianzas. Para esto, se generó un índice que agrupó las diez réplicas ejecutadas con cada combinación de los niveles de los factores elegidos.

La Tabla 7 presenta los resultados de los Test de Normalidad Kolmogorov-Smirnov y Shapiro-Wilk donde para todas las agrupaciones salvo la primera, no se encuentra evidencia estadística para afirmar que los datos provenientes de cada agrupación no sigan una distribución normal, al 95% de confiabilidad.

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Agrupación	Statistic	df	Sig.	Statistic	df	Sig.
Tiempo	1	.269	10	.039	.824	10	.028
	2	.131	10	.200*	.978	10	.956
	3	.186	10	.200*	.932	10	.472
	4	.169	10	.200*	.939	10	.537
	5	.139	10	.200*	.974	10	.924
	6	.126	10	.200*	.953	10	.701
	7	.191	10	.200*	.885	10	.148
	8	.207	10	.200*	.878	10	.122
	9	.196	10	.200*	.901	10	.225
	10	.134	10	.200*	.970	10	.887

	11	.135	10	.200*	.963	10	.821
	12	.139	10	.200*	.950	10	.673
	13	.199	10	.200*	.896	10	.197
	14	.148	10	.200*	.914	10	.308
	15	.188	10	.200*	.923	10	.382
	16	.128	10	.200*	.963	10	.818
Funcion Objetivo	1	.151	10	.200*	.950	10	.673
	2	.175	10	.200*	.933	10	.475
	3	.256	10	.062	.857	10	.071
	4	.253	10	.070	.926	10	.408
	5	.234	10	.127	.931	10	.453
	6	.258	10	.058	.903	10	.234
	7	.257	10	.061	.828	10	.032
	8	.237	10	.118	.885	10	.149
	9	.245	10	.090	.826	10	.030
	10	.219	10	.191	.919	10	.346
	11	.284	10	.022	.915	10	.318
	12	.152	10	.200*	.966	10	.846
	13	.185	10	.200*	.905	10	.249
	14	.136	10	.200*	.981	10	.972
	15	.243	10	.096	.866	10	.089
	16	.317	10	.005	.853	10	.064

a. Lilliefors Significance Correction

*. This is a lower bound of the true significance.

Tabla 7: Test de Normalidad para el tiempo de ejecución y función objetivo, agrupado las observaciones según los niveles de los factores. Estos test parte de la hipótesis nula que afirma los datos están distribuidos normalmente.

Adicionalmente, la prueba de Homogeneidad de Levene cuya hipótesis nula es la igualdad de las varianzas entre todas las agrupaciones encuentra que, con un 95% de confiabilidad, no hay evidencia para afirmar que las varianzas entre las agrupaciones sean diferentes, ver Tabla 8.

Test of Homogeneity of Variance

		Levene Statistic	df1	df2	Sig.
Tiempo	Based on Mean	.880	15	144	.588
	Based on Median	.615	15	144	.859
	Based on Median and with adjusted df	.615	15	119.421	.858
	Based on trimmed mean	.843	15	144	.629
FuncionObjetivo	Based on Mean	1.080	15	144	.380
	Based on Median	.948	15	144	.514
	Based on Median and with adjusted df	.948	15	114.894	.515
	Based on trimmed mean	1.060	15	144	.398

Tabla 8: Prueba de Homogeneidad de Varianzas para el tiempo y función objetivo en términos de las agrupaciones según los niveles de los factores seleccionados.

La Tabla 9 presenta los resultados del ANOVA para los factores definidos en el algoritmo genético. Se encuentra que el *tamaño de la población*, el *máximo número de ejecuciones* y su interacción resultan significativos. No se encuentra evidencia estadística para afirmar que la *probabilidad de entrecruzamiento*, *mutación* y su interacción resulten significativas a la hora de afectar el tiempo de ejecución, pero sí en términos de la calidad de las soluciones, según se evidencia en Análisis de Varianza para la calidad de las soluciones en la Tabla 10. Posteriormente, los gráficos de Pareto que priorizan los efectos principales que llevan a establecer los niveles 0.9, 0.01, 200, 400 para los factores *Probabilidad de entrecruzamiento, Probabilidad de Mutación, Tamaño de la Población y Máximo número de generaciones* respectivamente, cómo los más adecuados en términos de calidad de la solución y tiempo de respuesta, ver Figura 26 y Figura 27. La Figura 28y Figura 29 presentan las pruebas Anderson-Darling para los residuales. La hipótesis nula de Anderson-Darling es que los datos se distribuyen normalmente y la alternativa, el caso contrario. Para el caso, no hay evidencia estadística para afirmar que los residuales no están normal mente distribuidos.

Source	DF	Seq	SS	MS	Adj	F	P
Main	Effects	4	17768345	17768345	4442086	16999.22	0.000
	Entrecruzamiento	1	108	108	108	0.41	0.521
	Mutacion	1	319	319	319	1.22	0.271
	TamPoblación	1	430530	430530	430530	1647.57	0.000
	MaxGeneraciones	1	17337389	17337389	17337389	66347.65	0.000
2-Way	Interactions	6	332247	332247	55375	211.91	0.000
	Entrecruzamiento*Mutacion	1	175	175	175	0.67	0.415
	Entrecruzamiento*TamPoblación	1	40	40	40	0.15	0.695
	Entrecruzamiento*MaxGeneraciones	1	194	194	194	0.74	0.390
	Mutacion*TamPoblación	1	7	7	7	0.03	0.870
	Mutacion*MaxGeneraciones	1	867	867	867	3.32	0.071

	TamPoblación*MaxGeneraciones	1	330964	330964	330964	1266.55	0.000
3-Way	Interactions	4	736	736	184	0.7	0.591
	Entrecruzamiento*Mutacion*TamPoblación	1	183	183	183	0.7	0.405
	Entrecruzamiento*Mutacion*MaxGeneraciones	1	32	32	32	0.12	0.727
	Entrecruzamiento*TamPoblación*MaxGeneraciones	1	492	492	492	1.88	0.172
	Mutacion*TamPoblación*MaxGeneraciones	1	29	29	29	0.11	0.741
4-Way	Interactions	1	149	149	149	0.57	0.451
	Entrecruzamiento*Mutacion*TamPoblación*MaxGeneraciones	1	149	149	149	0.57	0.451
Residual	Error	144	37629	37629	261		
	Pure Error	144	37629	37629	261		
Total		159	18139107				

Tabla 9: ANOVA para los efectos de los factores en el tiempo de ejecución del Algoritmo Genético

Source	DF	Seq	SS	MS	Adj	F	P
	Effects	4	428788455	428788455	107197114	10668.8	0
	Entrecruzamiento	1	122441302	122441302	122441302	12185.97	0
	Mutacion	1	306336416	306336416	306336416	30488.14	0
	TamPoblación	1	18	18	18	0	0.967
Main	MaxGeneraciones	1	10719	10719	10719	1.07	0.303
	Interactions	6	1888098	1888098	314683	31.32	0
	Entrecruzamiento*Mutacion	1	1830654	1830654	1830654	182.2	0
	Entrecruzamiento*TamPoblación	1	4	4	4	0	0.985
	Entrecruzamiento*MaxGeneraciones	1	34692	34692	34692	3.45	0.065
	Mutacion*TamPoblación	1	7511	7511	7511	0.75	0.389
	Mutacion*MaxGeneraciones	1	7040	7040	7040	0.7	0.404
2-Way	TamPoblación*MaxGeneraciones	1	8197	8197	8197	0.82	0.368
	Interactions	4	15756	15756	3939	0.39	0.814
	Entrecruzamiento*Mutacion*TamPoblación	1	6989	6989	6989	0.7	0.406
	Entrecruzamiento*Mutacion*MaxGeneraciones	1	202	202	202	0.02	0.887
	Entrecruzamiento*TamPoblación*MaxGeneraciones	1	4488	4488	4488	0.45	0.505
3-Way	Mutacion*TamPoblación*MaxGeneraciones	1	4077	4077	4077	0.41	0.525
	Interactions	1	8067	8067	8067	0.8	0.372
4-Way	Entrecruzamiento*Mutacion*TamPoblación*MaxGeneraciones	1	8067	8067	8067	0.8	0.372
Residual	Error	144	1446872	1446872	10048		
Pure		14	1446872				

Source	DF	Seq	SS	MS	Adj	F	P
Error		4					
Total		159	432147248				

Tabla 10: ANOVA para los efectos de los factores en la calidad de las respuetas del Algoritmo Genético

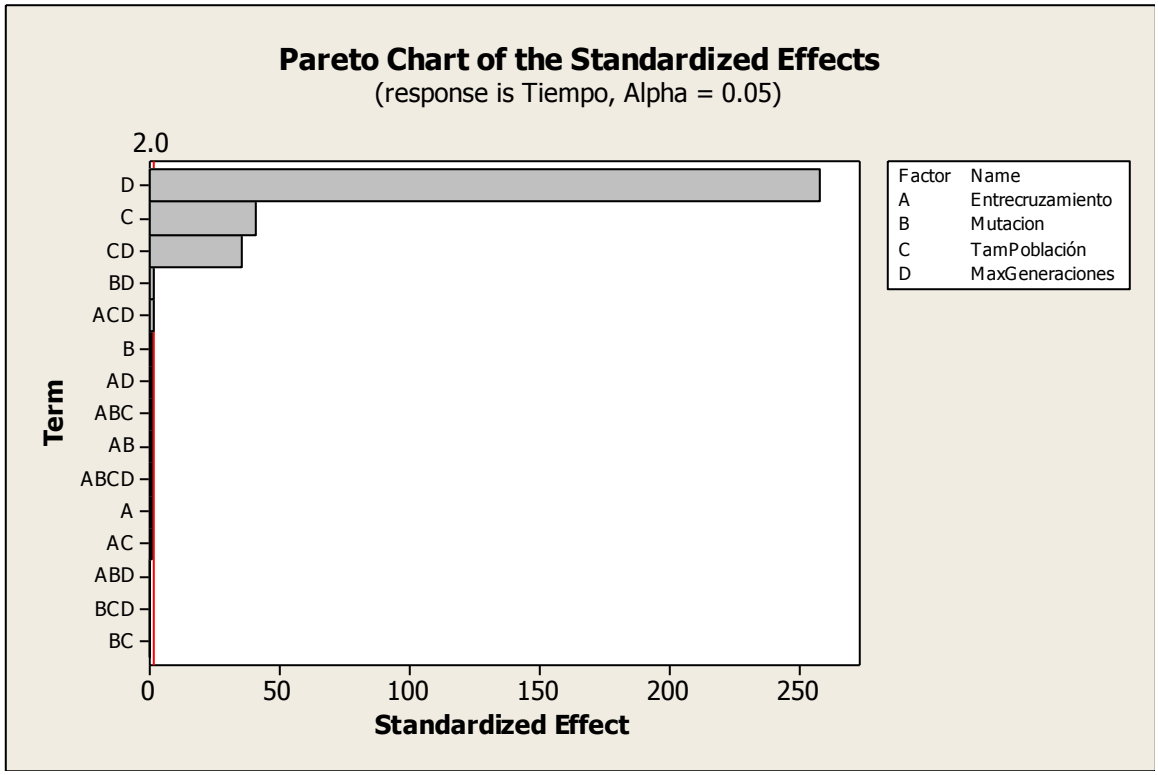


Figura 26: Gráfico de Pareto para los efectos estandarizados de los factores sobre el tiempo de ejecución. De la línea vertical roja hacia la derecha se observan los efectos significativos.

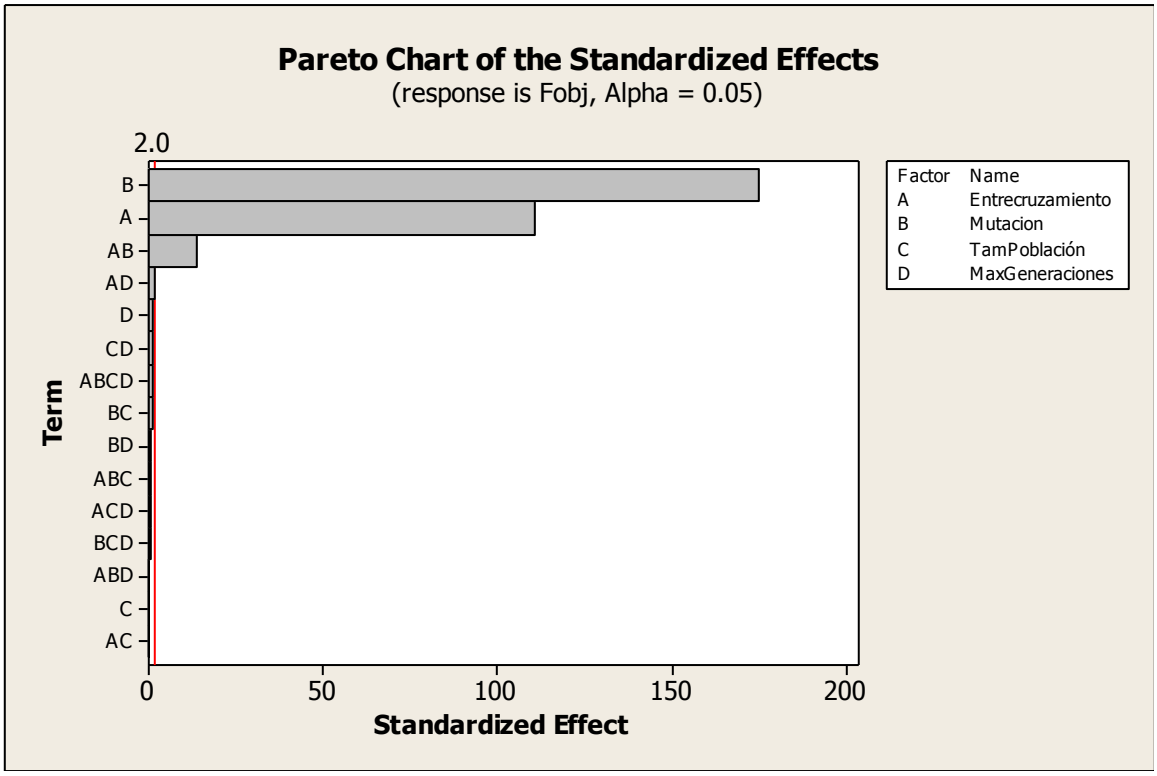


Figura 27: Gráfico de Pareto para los efectos estandarizados de los factores sobre la calidad de las soluciones. De la línea vertical roja hacia la derecha se observan los efectos significativos.

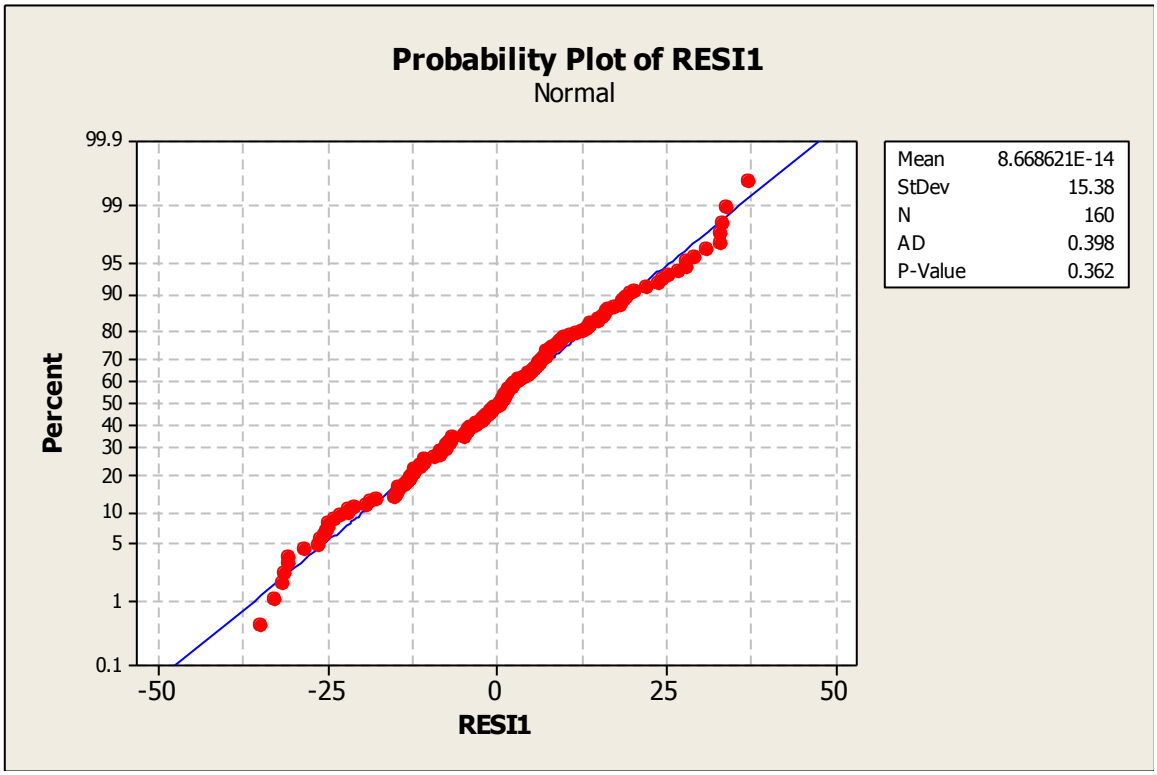


Figura 28: Prueba Anderson-Darling para Normalidad de los residuales del tiempo.

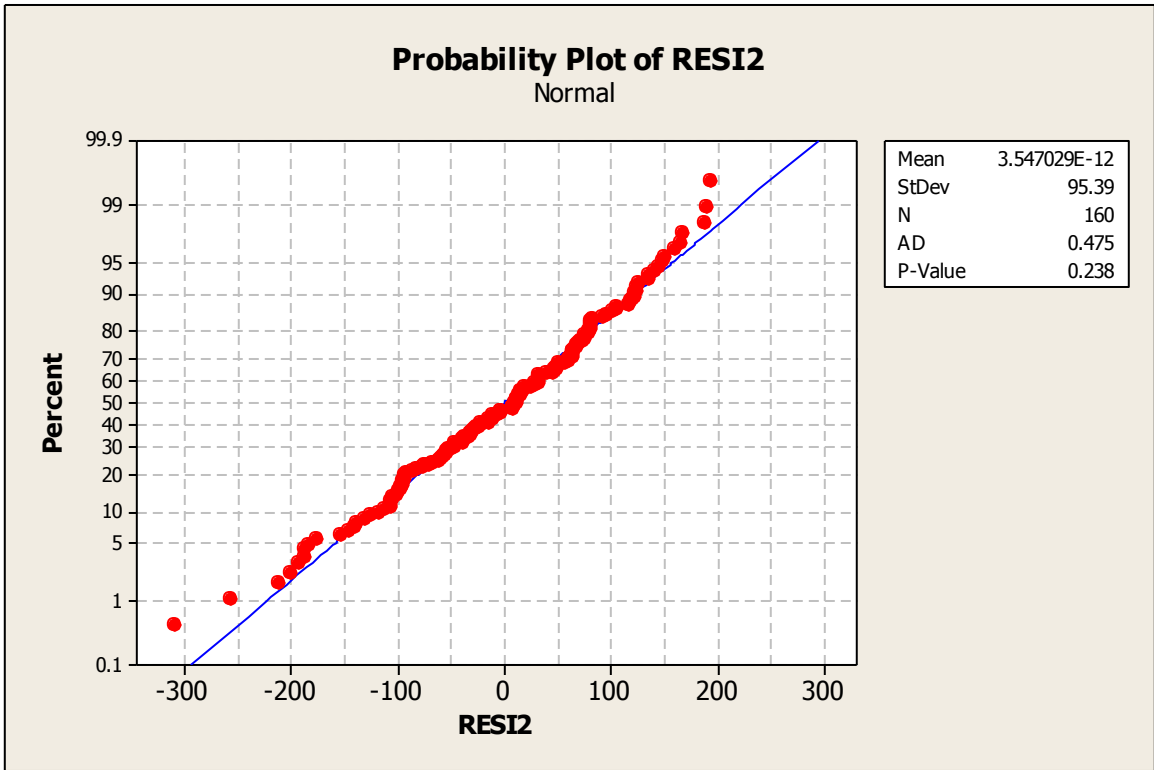


Figura 29: Prueba Anderson-Darling para Normalidad de los residuales de la calidad de las soluciones.

13.5 Calibración de parámetros para GRASP

Para GRASP, como primera medida se probaron los supuestos de normalidad y de homogeneidad de varianzas y se encontró que en general no se encuentra evidencia estadística para afirmar que los datos no sigan distribución normal al 95% de confianza, especialmente para la función objetivo, ver Tabla 11 .

Tests of Normality

		Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Alpha	Statistic	df	Sig.	Statistic	df	Sig.
FuncionObjetivo	0.2	.206	10	.200*	.901	10	.225
	0.4	.193	10	.200*	.959	10	.771
	0.6	.160	10	.200*	.942	10	.575
Tiempo	0.2	.269	10	.039	.824	10	.028
	0.4	.169	10	.200*	.939	10	.537
	0.6	.199	10	.200*	.896	10	.197

a. Lilliefors Significance Correction

*. This is a lower bound of the true significance.

Tabla 11: Test de normalidad entre réplicas a niveles de alpha iguales a 0.2, 0.4 y 0.6.

La prueba de homogeneidad de varianzas en la Tabla 12, no reveló evidencia para afirmar que las varianzas de las observaciones entre niveles de alpha de 0.2, 0.4 y 0.6 difieran significativamente al 95% de confibilidad.

Test of Homogeneity of Variance

		Levene Statistic	df1	df2	Sig.
FuncionObjetivo	Based on Mean	.977	2	27	.389
	Based on Median	.826	2	27	.448
	Based on Median and with adjusted df	.826	2	20.332	.452
	Based on trimmed mean	.989	2	27	.385
Tiempo	Based on Mean	3.714	2	27	.038
	Based on Median	1.230	2	27	.308
	Based on Median and with adjusted df	1.230	2	20.308	.313
	Based on trimmed mean	3.443	2	27	.047

Tabla 12: Prueba de Homogeneidad de varianzas entre réplicas a niveles de alpha iguales a 0.2, 0.4 y 0.6.

Dado que los supuestos de normalidad y homogeneidad en general se cumplen, podremos determinar cuál valor del parámetro *Alpha* resulta adecuado en términos del tiempo de ejecución y calidad de la solución, en primer lugar se ejecutó un ANOVA de una vía que determinó que existe evidencia estadística para afimrar que las medias del tiempo y calidad de respuesta difieren significativamente en términos del parámetro *Aplha*, ver Tabla 13. Posteriormente empleamos la prueba por Intervalos de Duncan para determinar el mejor nivel de los factores, y el mejor nivel en términos de la función objetivo es 0.4 para el parámetro *Alpha*, ver Figura 30.

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Tiempo	Between Groups	2132977.867	2	1066488.933	5287.404	.000
	Within Groups	5446.000	27	201.704		
	Total	2138423.867	29			
FObj	Between Groups	7103898.600	2	3551949.300	532.184	.000
	Within Groups	180205.700	27	6674.285		
	Total	7284104.300	29			

Tabla 13: Anova de Una vía para el tiempo de respuesta y la calidad de la solución en función del parámetro Alpha

Tiempo

Duncan

%Alpha	N	Subset for alpha = 0.05	
		1	2
0.6	10	778.6000	
0.2	10	786.2000	
0.4	10		1348.0000
Sig.		.242	1.000

Means for groups in homogeneous subsets are displayed.

FObj

Duncan

%Alpha	N	Subset for alpha = 0.05	
		1	2
0.6	10	99980.6000	
0.2	10		101003.6000
0.4	10		101021.9000
Sig.		1.000	.621

Means for groups in homogeneous subsets are displayed.

Figura 30: Prueba por intervalos de Duncan para las variables de respuesta tiempo y calidad de las soluciones, en términos del parámetro Alpha

14 Análisis de Resultados

Todas las ejecuciones de Software se realizaron en una CPU Dual 2.16 GHz y 4GB RAM sobre Ms Windows 7®. Las Meta Heurísticas fueron escritas en C# sobre Visual Studio 2008® sin emplear programación para múltiples hilos.

La Programación Lineal Mixta propuesta en el presente trabajo fue ejecutada en LpSolve para la Instancia A durante varias horas sin obtener resultados, y confirmándose la condición NP-Completa, la Figura 31 presenta el registro de LpSolve.

```
Model name:  'LPSolver' - run #1
Objective:   Maximize(R0)
SUBMITTED
Model size:     3169 constraints,    9674 variables,      1304189 non-zeros.
Sets:                                   0 GUB,                  0 SOS.
Using DUAL simplex for phase 1 and PRIMAL simplex for phase 2.
The primal and dual simplex pricing strategy set to 'Devex'.
Found feasibility by dual simplex after        332033 iter.
Relaxed solution       43050.9921925 after     339699 iter is B&B base.
spx_run: Lost feasibility 10 times - iter     490272 and        11 nodes.
bfp_factorize: Resolving 1 singularity at refact 1429, iter 613726
bfp_factorize: Resolving 1 singularity at refact 1786, iter 740419
lp_solve optimization was stopped by the user.
lp_solve unsuccessful after 3639094 iter and a last best value of -1e+030
lp_solve explored 78 nodes before termination
MEMO: lp_solve version 5.5.2.0 for 32 bit OS, with 64 bit REAL variables.
In the total iteration count 3639094, 3107533 (85.4%) were bound flips.
There were 2406 refactorizations, 0 triggered by time and 539 by density.
... on average 220.9 major pivots per refactorization.
The largest [LUSOL v2.2.1.0] fact(B) had 3618575 NZ entries, 6.3x largest basis.
The maximum B&B level was 47, 0.0x MIP order, with 78 nodes explored.
The constraint matrix inf-norm is 1, with a dynamic range of 1.
Time to load data was 2.428 seconds, presolve used 0.334 seconds,
... 55135.192 seconds in simplex solver, in total 55137.954 seconds.
```

Figura 31: Log de LpSolve

A Manera de comprobación se ejecutó la instancia C del problema definidas anteriormente usando las implementaciones en C# de las meta heurísticas seleccionadas. Se planteó un ANOVA de una Vía para comprar los rendimientos de ambos algoritmos, que fueron empleados como factor. Las tablas 11 y 12 presentan los análisis de varianza para la calidad de las soluciones y tiempo de ejecución, en ambas se observa evidencia para afirmar que el desempeño de las Meta Heurísticas en ambos criterios difieren significativamente al 95%. En este sentido, el análisis hecho usando el Método de Tukey sugiere que el Algoritmo Genético tiende a generar soluciones de mejor calidad; y GRASP tiende a ejecutarse más rápidamente, ver Tabla 13.

Source	DF	SS	MS	F	P
Algoritmo	1	308761250	308761250	56.92	0.000
Error	198	1074027500	5424381		
Total	199	1382788750			

Tabla 14: ANOVA para la calidad en unidades de área entre Algoritmos.

Source	DF	SS	MS	F	P
Algoritmo	1	556008.6	556008.6	25257.59	0.000
Error	198	4358.7	22.0		
Total	199	560367.3			

Tabla 15: ANOVA para el tiempo de respuesta en segundos entre Algoritmos

Algoritmo	N	Mean	Grouping
GA	100	264400	A
GRASP	100	261915	B

Algoritmo	N	Mean	Grouping
GA	100	449.55	A
GRASP	100	344.10	B

Tabla 16: Agrupaciones para la calidad (izquierda) y el tiempo de ejecución (derecha), usando el Método de Tukey.

La Figura 32 muestra una imagen de la instancia C ejecutada, donde se observa la acomodación de todas las piezas en la región superior de la piel, producto del uso de *Bottom-Left*.

Figura 32: Instancia Mejor solución encontrada usando Algoritmos Genéticos. En naranja los pixeles que pertenecen las piezas ficticias y representan área aprovechable; en azul pixeles en regiones no aprovechables; otros colores las figuras que representan las pieza

15 Conclusiones

- Luego de realizar la revisión de literatura, no se encontró evidencia de la existencia de programaciones lineales o lineales mixtas que resuelvan o conduzcan la solución del ITDCSP.
- Se desarrolló una formulación capaz de representar el problema demarcando un espacio de búsqueda que, si bien presenta dificultades en la búsqueda de soluciones óptimas incluso en instancias pequeñas, permite la exploración dirigida usando procedimientos aproximados.
- Las Meta Heurísticas GRASP y Algoritmos Genéticos encontraron soluciones de calidad aceptable en tiempos razonables. Los AG tienden a dar mejores soluciones en términos de calidad pero GRASP se ejecuta con mayor velocidad.
- Si bien las Meta Heurísticas probadas en las instancias propuestas funcionaron, es altamente recomendable encaminar futuros trabajos para encontrar la combinación de parámetros más adecuada para acelerar el proceso de búsqueda e incrementar la calidad de las soluciones.
- Aunque el Algoritmo Genético implementado tardó mayor tiempo en generar las soluciones, la calidad de las mismas siempre fue mejor que las obtenidas usando GRASP. Esto puede indicar que la función de utilidad empleada durante la etapa constructiva de GRASP, que consiste en evaluar iterativamente el decremento del área resultante aprovechable, no resulta adecuada para encontrar las secuencias óptimas.

16 Investigaciones futuras

A Partir de la formulación MIP encontrada, pueden formularse trabajos futuros orientados hacia la ampliación de la formulación del problema para permitir la rotación de las piezas al interior de las pieles, y la optimización del desperdicio con múltiples hojas y extender el problema de 2D a 3D para aplicaciones como *Bin Packing Problem* en aplicaciones de contenedores, distribución de planta, etc. Estudios sobre la morfología del politopo o híper espacio de soluciones limitado por las restricciones, así como cambios en la función objetivo, podrían conducir a la reescritura de las restricciones o incluso a relajaciones disminuyendo la complejidad computacional y habilitando la posibilidad de encontrar soluciones óptimas en tiempos razonables.

A Partir de la mecánica propuesta con meta heurísticas, pueden emplearse otras puras como *Particle Swarm Optimization*, *Simulated annealing*, entre otras, y algunas híbridas como por ejemplo implantando soluciones obtenidas por GRASP en la población inicial de GA. Adicionalmente pueden probarse algoritmos más eficientes que *Bottom Left*, tal vez basados en algoritmos de inteligencia colectiva, que permitan encontrar posicionamientos más compactos y/o rápidos, así como emplear otras maneras más eficientes de representar las figuras y desarrollar algoritmos de post ejecución para compactar las soluciones encontradas. En cuanto a la forma de desarrollar el código fuente, se propone explotar las características de las diferentes metaheurísticas con programación por hilos, también conocida como *Parallel Computing*.

Otras líneas de investigación pueden explorar posibilidades de desarrollar programaciones matemáticas lineales, no necesariamente mixtas, basadas en la posibilidad de fraccionar un polígono irregular en triángulos aprovechando sus propiedades geométricas.

17 Bibliografía

Adamowicz, M. & Albano, A., (1976). Nesting two-dimensional shapes in rectangular modules. Computer-Aided Design, 8(1), págs.27-33.

Agrawal, P., (1993). Minimizing trim loss in cutting rectangular blanks of a single size from a rectangular sheet using orthogonal guillotine cuts. European Journal of Operational Research, 64(3), págs.410-422.

Alvarez-Valdes, R. & Parreño, F. & Tamarit, J.M., (2007). A tabu search algorithm for a two-dimensional non-guillotine cutting problem. European Journal of Operational Research, 183(3), págs.1167-1182.

Anand, S. & McCord, C. & Sharma, R., (1999),. An integrated machine vision based system for solving the nonconvex cutting stock problem using genetic algorithms. Journal of Manufacturing Systems, 18(6), págs.396-415.

Aras, N., (2005). Erratun to "The irregular cutting-stock problem- a new procedure for deriving the no-fit polygon". Computers & Operations Research, 32(5), pág.1353.

Babu, R.A., & Babu, R.N., (2001). A generic approach for nesting of 2-D parts in 2-D sheets using genetic and heuristic algorithms. Computer-Aided Design, 33(12), págs.879-891.

Beasley, J.E., (1985). An Exact Two-Dimensional Non-Guillotine Cutting Tree Search Procedure. Operations Research, 33(1), págs.49-64.

Bennell, J.A. & Song, X., (2008). A comprehensive and robust procedure for obtaining the nofit polygon using Minkowski sums. Computers & Operations Research, 35(1), págs.267-281.

Bennell, J.A., & Dowsland, K.A. & Dowsland, W.B., (2001). The irregular cutting-stock problem -- a new procedure for deriving the no-fit polygon. Computers & Operations Research, 28(3), págs.271-287.

Bennell, J.A. & Oliveira, J.F., (2008). The geometry of nesting problems: A tutorial. European Journal of Operational Research, 184(2), págs.397-415.

Burke, E.K., & Hellier, R.S.R., & G. Kendall, G. Whitwell, (2007). Complete and robust no-fit polygon generation for the irregular stock cutting problem. European Journal of Operational Research, 179(1), págs.27-49.

Caballero, J.P., & Alvarado, J.A., (2010). Greedy Randomized Adaptive SearchProcedure (GRASP), una alternative valiosa en la minimización de la tardanza total ponderada en una máquina. IyU, 14(2), págs. 275-296.

Cheng, C.H., & Feiring, B.R. & Cheng, T.C.E., (1994). The cutting stock problem -- a survey. International Journal of Production Economics, 36(3), págs.291-305.

Costa, M.T., & Gomes, A.M. & Oliveira, J.F., (2009). Heuristic approaches to large-scale periodic packing of irregular shapes on a rectangular sheet. European Journal of Operational Research, 192(1), págs.29-40.

Croes GA (1958). A Method for Solving Traveling-Salesman Problems. Operations Research, 6(6), 791{812.

Cui, Y., (2007). Exact algorithm for generating two-segment cutting patterns of punched strips. Applied Mathematical Modelling, 31(9), págs.1865-1873.

Cui, Y., (2006). Generating optimal multi-segment cutting patterns for circular blanks in the manufacturing of electric motors. European Journal of Operational Research, 169(1), págs. 30-40.

Dean, H.T., Tu, Y. & Raffensperger, J.F., (2006). An improved method for calculating the no-fit polygon. Computers & Operations Research, 33(6), págs.1521-1539.

Dowsland, K.A. & Dowsland, W.B., (1995). Solution approaches to irregular nesting problems. European Journal of Operational Research, 84(3), págs.506-521.

Dowsland, K.A., & Vaid, S. & Dowsland, W.B., (2002). An algorithm for polygon placement using a bottom-left strategy. European Journal of Operational Research, 141(2), págs.371-381.

Dyckhoff, H., (1981). A New Linear Programming Approach to the Cutting Stock Problem. Operations Research, 29(6), págs.1092-1104.

Gilmore, P.C. & Gomory, R.E., (1961). A Linear Programming Approach to the Cutting-Stock Problem. Operations research, 9(6), págs.849-859.

Hahn, S.G., (1968). On the Optimal Cutting of Defective Sheets. Operations Research, 16(6), págs.1100-1114.

Ismail, H.S. & Hon, K.K.B., (1995). The nesting of two-dimensional shapes using genetic algorithms. Archive: Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 1989-1996 (vols 203-210), 209(22), págs.115-124.

Lapthorn, Barry, (2003). A Simple C# Genetic Algorithm. The Code Project. http://www.codeproject.com/KB/recipes/btl_ga.aspx.

Lee, W.C., & Ma, H. & Cheng, B.W., (2008). A heuristic for nesting problems of irregular shapes. Computer-Aided Design, 40(5), págs.625-633.

Lutfiyya, H., & McMillin, B. & Poshyanonda, P., & Dagli, C., (1992). Composite stock cutting through simulated annealing. Mathematical and Computer Modelling, 16(1), págs.57-74.

Prasad, Y.K.D.V., (1994). A set of heuristic algorithms for optimal nesting of two-dimensional irregularly shaped sheet-metal blanks. Computers in Industry, 24(1), págs.55-70.

Resende, M., Ribeiro, C. (2003). Greedy Randomized Adaptive Search Procedures. In Glover, F., Kochenberger, G. (eds), Handbook of MetaHeuristics, Kluwer, págs 335-368.

Qu, W. & Sanders, J.L., (1987). A nesting algorithm for irregular parts and factors affecting trim losses. International Journal of Production Research, 25(3), pág.381.

Segenreich, S.A. & Faria-Braga, L.M.P., (1986). Optimal nesting of general plane figures: A Monte Carlo heuristical approach. Computers & Graphics, 10(3), págs.229-237.

Tang, C.H. & Rajesham, S., (1994). Computer aided nesting in sheet metal for pressworking operations involving bending. Journal of Materials Processing Technology, 44(3-4), págs.319-326.

Wong, W.K., & Wang, X.X, & Leung, S.Y.S, & Kwong, C.K., (2009). Solving the two-dimensional irregular objects allocation problems by using a two-stage packing approach. Expert Systems with Applications, 36(2, Part 2), págs.3489-3496.