UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

ALGORITMO GENÉTICO PARA EL PROBLEMA DE EMPAQUETAMIENTO DE CONTENEDORES DE DOS DIMENSIONES, APLICADO PARA LA OPTIMIZACIÓN DE PAPEL EN IMPRENTAS DIGITALES

TESIS PROFESIONAL PARA OBTENER EL GRADO DE:

DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS CON OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

PRESENTA:

M.I.C.A. YAINIER LABRADA NUEVA

DIRECTOR: DR. MARCO ANTONIO CRUZ CHÁVEZ

CUERNAVACA, MORELOS ENERO, 2024

Resumen

En el presente trabajo de investigación se aplica un algoritmo genético para resolver el problema de empaquetamiento de contenedores de dos dimensiones aplicado a la optimización de papel en imprentas digitales. Para dar solución a este problema, se realizó un estudio sobre geometría computacional para evitar el traslape entre figuras amorfas que se inserten en la hoja de papel, y así poder optimizar la función objetivo cumpliendo con las restricciones del problema. Además, se diseñó y se implementó una estructura híbrida de vecindad con movimientos bidimensionales, con el objetivo de mejorar el desempeño del algoritmo genético implementado.

Se aplicó una metodología de sintonización, la cual permitió realizar el análisis de sensibilidad de los parámetros de control del algoritmo genético, lo que favorece que el algoritmo trabaje con el mejor desempeño en eficiencia y eficacia.

La aportación de este trabajo de investigación es la implementación de un mapeo del problema de empaquetamiento de contenedores de dos dimensiones aplicado a la optimización del desperdicio de papel en imprentas digitales tratado con un algoritmo genético y una estructura híbrida de vecindad con movimientos bidimensionales (Rotación, Traslación, Inserción e Intercambio).

Las pruebas experimentales realizadas al algoritmo genético implementado mostraron que es posible obtener buenas soluciones para figuras amorfas.

Abstract

In the present research work, a genetic algorithm is applied to solve the two-dimensional container packaging problem applied to the optimization of paper in digital printing presses. To solve this problem, a study on computational geometry was carried out to avoid the overlap between amorphous figures that are inserted in the sheet of paper, and thus be able to optimize the objective function complying with the restrictions of the problem, it was also studied, designed and a hybrid neighborhood structure with two-dimensional movements was implemented, to improve the performance of the implemented genetic algorithm.

A tuning methodology was applied, which allowed to performance of the sensitivity analysis of the control parameters of the genetic algorithm, which allowed the algorithm to work with the best performance in efficiency and effectiveness.

The contribution of this research work is the implementation of a mapping of the two-dimensional container packaging problem applied to the optimization of paper waste in digital printing presses treated with a genetic algorithm and a hybrid neighborhood structure with two-dimensional movements (Rotation, Translation, Insertion, and Exchange).

The experimental tests carried out on the implemented genetic algorithm showed that it is possible to obtain good solutions for amorphous figures.

Agradecimientos

A CONACYT por brindarme el apoyo económico para la realización de mis estudios de posgrado.

Al Dr. Marco Antonio Cruz Chávez, director de este trabajo de investigación, por la orientación y consejos para la realización y culminación del mismo.

A los integrantes del comité tutoral y revisores de tesis: Dr. Marco Antonio Cruz Chávez, Dr. Martin Heriberto Cruz Rosales, Dr. Alvaro Zamudio Lara, Dr. Juan Manuel Rendón Mancha, Dra. Margarita Tecpoyotl Torres, Dra. Alina Martínez Oropeza, Dra. Marta Lilia Eraña Díaz por sus comentarios, orientación y consejos para la realización de este trabajo de investigación.

Dedicatorias

A mi familia, especialmente a mi mamá, a mi hermana, a mi esposa, a mi hijo, a mis primos y a mis abuelos, por estar presente siempre en todos mis logros.

Al Dr. Marco A. Cruz Chávez por ser un excelente y ejemplar profesional, por su apoyo comprensión, y consejos que han sido útiles en mi evolución personal y profesional.

A mi esposa Elena y a mi hijo Joel por estar conmigo todo este tiempo.

Al Dr. Diego Seuret Jimenez, a su esposa Rossy y a sus hijos Dieguito y Halis por adoptarme como un miembro de su familia.

A mi suegra Rosa Elena, a mis cuñados Vanesa y Hugo por estar presentes en estos años.

A todos mis amigos que hemos compartido tanto acá en México como en Cuba, entre ellos están Ismar, Prevot, Ayixon, Jvdith, Liz y Gerardo.

A Fernando, Adriana, y a Don Victor y su Familia por su apoyo.

A Lalo, Percy, Mary, Aime, Margarita y Soto por todos sus consejos.

A mis compañeras y compañeros de posgrado por la amistad, apoyo y confianza que nos permite tener un equipo de trabajo en constante evolución personal y profesional, especialmente a Alina, Gerardo, Carmen, Pedro, Luis Dévora, Marco, Juanita, Alfonso, Jazmin, Betty, Ariadna, Mara, Alán, Mireya, Erika y a otros que no aparecen porque la lista sería interminable. A todos los aprecio y estimo.

A los Doctores Juan Manuel Rendón Mancha, Lorena Díaz González y Mauricio Rosales del CINC, así como a todos mis alumnos del CINC, por su apoyo en todo momento.

Nomenclatura

SA Siglas en inglés para recocido simulado (simulated annealing)

TS Siglas en inglés para búsqueda tabú (tabu search)

ILS Siglas en inglés para búsqueda local iterada (iterated local search)

VNS Siglas en inglés para búsqueda local variable (variable local search)

EAs Siglas en inglés para algoritmos evolutivos (evolutionary algorithms)

ACO Siglas en inglés para optimización de colonia de hormigas (ant colony optimization)

PSO Siglas en inglés para optimización por enjambre de partículas (particle swarm optimization)

SS Siglas en inglés para búsqueda dispersa (scatter search)

ES Siglas en inglés para estrategias evolutivas (evolutionary strategies)

AG-PEC2DAOPID Algoritmo Genético para el Problema de Empaquetamiento de Contenedores de 2 Dimensiones Aplicado a la Optimización de Papel en Imprentas Digitales

Algoritmo genético

P_s	Porcentaje de selección
P_m	Porcentaje de mutación
T_{pob}	Tamaño de población
P_c	Probabilidad de cruzamiento
P_{ini}	Población inicial
P_{mut}	Probabilidad de mutación
N_g	Número de generaciones
N_{inds}	Número de individuos

Índice

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Capítulo 1 Introducción

Los Problemas de empaquetamiento consisten, por lo general, en el corte de materias primas para obtener un conjunto de elementos minimizando el desperdicio de material generado o en el empaquetado de un conjunto de artículos en el menor número de contenedores. Esta clase de problemas cae dentro de la categoría de problemas de optimización combinatoria. Usualmente, se presentan en muchas aplicaciones industriales, tales como: vidrio, papel y corte de acero, carga de contenedores y camiones, diseño de circuitos integrados, optimización de portafolio, entre otras.

La mayoría de los problemas de optimización combinatoria, y por consiguiente los problemas de empaquetamiento, son, en general, difíciles de resolver en la práctica. Estos problemas están incluidos en la clase de problemas NP-Completo [1], ya que no se conocen algoritmos exactos con complejidad polinómica que permitan resolverlos. Debido a su intratabilidad, se han diseñado una gran cantidad de métodos aproximados, los cuales encuentran buenas soluciones en tiempo computacional razonable. En esta clase de problemas, la búsqueda de una solución requiere una exploración organizada a través del espacio de búsqueda: una búsqueda sin guía es extremadamente ineficiente.

La optimización combinatoria es una rama de la optimización en matemáticas aplicadas y en ciencias de la computación, relacionada a la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional. La misma estudia el tratamiento de problemas considerados difíciles de resolver [2]. El objetivo por seguir en esta área es el desarrollo o mejora de nuevos métodos que

permitan abordar problemas de optimización con el menor esfuerzo computacional posible.

La característica básica y principal de un problema de optimización es la búsqueda de la mejor solución mediante el desarrollo de métodos que permitan reducir la complejidad para encontrar buenas soluciones al problema. Para representar un problema de optimización es necesario utilizar modelos matemáticos. Cada modelo cuenta con cierta particularidad como es el caso de la función objetivo y las restricciones propias del problema. La función objetivo dependerá del problema tratado, de modo que se puede minimizar o maximizar [3].

1.1 Métodos de Optimización Combinatoria

La solución de problemas de optimización consiste en encontrar el mejor valor de la función objetivo de modo que se satisfagan las restricciones propias de cada problema para una instancia dada. Una instancia es un problema de prueba que puede ser definido como el tamaño de entrada de los datos del problema. Para obtener una buena solución a un problema de optimización clasificado como NP, es necesaria la implementación de un algoritmo no determinístico de tipo heurístico [2], debido a que este tipo de algoritmos permiten obtener soluciones cercanas al óptimo para instancias medianas y grandes en un tiempo computacional razonable.

Una heurística es un método bien estructurado, desarrollado en base a la experiencia que permite obtener soluciones aproximadas de un problema sin garantizar la optimalidad [4]. Este tipo de métodos se aplican cuando no es posible encontrar una solución óptima por un método exacto, debido a la naturaleza y

complejidad del problema. En la Figura 1.1 se muestra una clasificación de los principales métodos de optimización.

El objetivo que persigue cada uno de los métodos mostrados en la Figura 1.1, es encontrar la mejor solución a un problema combinatorio. En el caso de métodos deterministas, se puede mencionar que el Simplex Clásico [3] es probablemente el más conocido, este método permite obtener el valor óptimo a un problema dado, aunque cuenta con ciertas limitaciones, debido a que en el caso del Simplex Clásico, está comprobado que su complejidad es de orden exponencial [5], lo que significa que los tamaños de problemas que pueden ser resueltos por este método, dependerá directamente del número de variables manejadas por el problema, además de que deberá ser un problema de programación lineal.

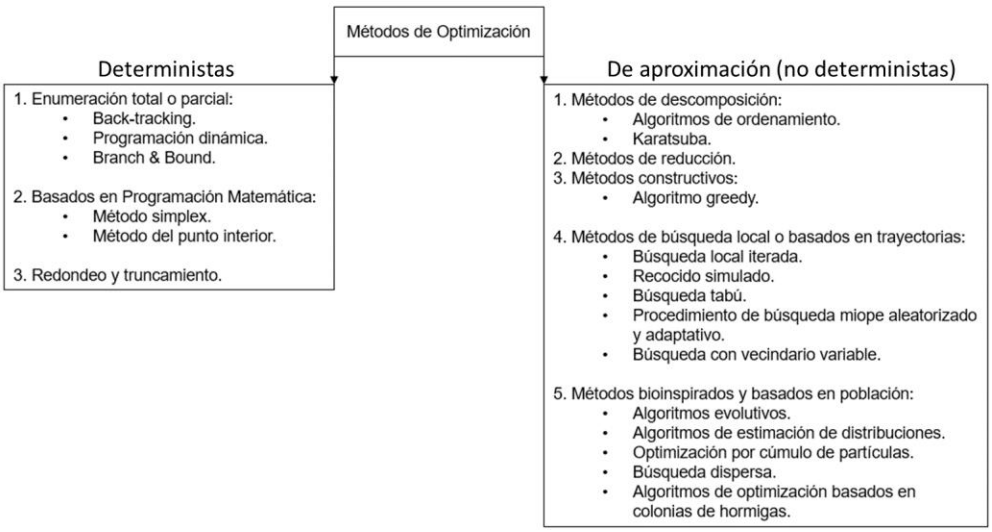

Figura 1.1. *Clasificación de los métodos de optimización* [6] [7] *.*

Muchos investigadores estudian la mejora de métodos heurísticos para ser aplicados a problemas combinatorios, para los cuales no se conoce un algoritmo determinístico con comportamiento polinomial que los resuelva, puesto que los

tiempos requeridos para encontrar una solución a instancias consideradas de tamaño grande son extremadamente largos.

Las heurísticas son procedimientos para resolver problemas de optimización bien definidos, mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución en un tiempo computacional razonable [8][3] para instancias medianas y grandes de problemas clasificados como difíciles. Derivado de ello surgen las denominadas metaheurísticas, que son métodos para obtener soluciones aproximadas y diseñados para resolver problemas complejos de optimización combinatoria.

Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos [9]. Dentro de esta clasificación se encuentran una gran variedad de métodos de naturaleza muy diferente, por lo que se complica el dar una clasificación completa. Seguidamente se realiza una clasificación para ubicar a los métodos heurísticos más conocidos [8] [10].

Métodos de Descomposición (Divide y Vencerás): Son métodos donde el problema original es descompuesto en sub-problemas más pequeños y sencillos de resolver, teniendo en cuenta que todos pertenecen al mismo problema. Algunos ejemplos de métodos de este tipo son los algoritmos de ordenamiento [11][12].

Métodos de Reducción: Métodos que consisten en analizar e identificar las propiedades que cumplen generalmente las buenas soluciones e introducirlas como restricciones del problema. El objetivo es acotar el espacio de soluciones,

simplificando el problema, aunque existe el riesgo de dejar fuera la solución óptima del problema.

Métodos Constructivos: Métodos que consisten en ir construyendo poco a poco una solución a un problema dado. Normalmente son métodos deterministas y suelen estar basados en la elección de la mejor solución en cada iteración. Estos métodos han sido muy utilizados en problemas de optimización clásicos, como es el caso del problema del Agente Viajero. Un ejemplo de este tipo de métodos son los algoritmos voraces (Greedy)[12].

Métodos de Búsqueda Local: A diferencia de los métodos mencionados anteriormente, los procedimientos de búsqueda local comienzan con una solución inicial y la van mejorando progresivamente. El procedimiento se lleva a cabo realizando en cada iteración, un movimiento de una solución a otra que mejore la solución anterior. Este movimiento puede ser una permutación, inserción o eliminación. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore. La diferencia con los métodos deterministas es que estos no necesariamente encontraran una solución óptima.

Métodos Bioinspirados: Esta clasificación engloba el conjunto de algoritmos que simulan un proceso “inteligente” de los animales que viven en sociedad, como las abejas, las termitas, las hormigas, entre otros. En estas heurísticas no necesariamente se simula un proceso de evolución [13], sino que pueden simular el comportamiento de dichas comunidades al realizar ciertas actividades, como es el caso de búsqueda de alimento. Ejemplos de estos métodos son Colonia de Hormigas, Algoritmos Evolutivos, Algoritmos Genéticos, entre otros.

1.2 Teoría de la Complejidad de Algoritmos

La *Complejidad Algorítmica* estudia la eficiencia de los algoritmos basado en el tiempo y espacio necesarios para resolver un problema computacional, conocidos como Complejidad Temporal y Complejidad Espacial [14].

La complejidad temporal se define como una función *T(n)*, misma que es dependiente de la implementación del algoritmo. Estos requerimientos pueden ser expresados en función del tamaño de la instancia, la cual refleja la cantidad de datos de entrada requeridos para abordar el problema.

El cálculo de la *complejidad temporal* de un algoritmo *T(n)* es medida asintóticamente. *T(n)* corresponde al número de pasos requeridos por un algoritmo para resolver un problema [15]; por lo que, a mayor complejidad, mayor será el tiempo que requiera el algoritmo para obtener una solución a un problema dado, de modo que mientras incrementa la complejidad de un algoritmo, su eficiencia disminuye.

La complejidad temporal permite expresar matemáticamente la relación existente entre la cantidad de datos (tamaño de la entrada) y el tiempo de ejecución del algoritmo. De modo que, a mayor tamaño de la entrada, mayor será el tiempo de ejecución requerido para su procesamiento. Independientemente del problema abordado, siempre existe el mejor y peor caso, por lo que el intervalo existente se puede definir como $Tmin(n)<T(n)< Tmax(n)$, tomando $Tmin(n)$ como el *mejor de los casos*, es decir que se cumplen las características para el algoritmo requiera la menor cantidad de instrucciones para devolver una solución, por el contrario $Tmax(n)$ corresponde a aquellas características que implican que el algoritmo debe

realizar todos los cálculos considerados por el algoritmo para obtener una solución, lo que se denomina el *peor de los casos*.

Por otro lado, para el cálculo de la *complejidad espacial*, se toma en cuenta la cantidad de memoria requerida para almacenar los datos utilizados por el algoritmo, debido a que la cantidad de datos que pueden ser almacenados está limitada por la cantidad de memoria con que cuenta el equipo. Otros de los factores que comúnmente afectan la eficiencia de un algoritmo son principalmente la programación, la velocidad del procesador y el tipo de compilador.

Un algoritmo eficiente se centra fundamentalmente en la simplicidad y en el manejo adecuado de los recursos, siendo la sencillez una característica fundamental para el diseño de algoritmos, debido a que facilita el cálculo de la función temporal, además de optimizar el número de instrucciones evaluadas por el procesador, lo cual se ve reflejado en el tiempo de ejecución.

En la Tabla 1.1 se muestra la clasificación correspondiente al comportamiento de los algoritmos de acuerdo con el tiempo computacional en función al número de instrucciones con base a una entrada n.

De acuerdo con la clasificación mostrada en la Tabla 1.1, el algoritmo más eficiente corresponde a aquel que presenta una complejidad constante $T(n) = k$, lo que significa que, si el tamaño de la entrada aumenta, el tiempo de ejecución del algoritmo se mantiene constante. En el caso de una complejidad logarítmica $T(n) = log(n)$, si el tamaño de la entrada se incrementa 100 veces, el tiempo de ejecución del algoritmo únicamente se duplicará, por lo que se considera un algoritmo eficiente. Por el contrario, si un algoritmo presenta una complejidad exponencial $T(n) = 2^n$, este se considera sumamente ineficiente debido a que su tiempo de

ejecución aumenta exponencialmente conforme se incrementa el tamaño de la entrada.

Tabla 1.1 Tipos de complejidad para el estudio de eficiencia de los algoritmos [11] [6][16].

Tiempo computacional en función del número de instrucciones	**Nombre**
K	Constante
$\log(n)$	Logarítmica
n	Lineal
$\mathrm{nlog}(n)$	Cuasi-lineal
n^2	Cuadrática
n^K	Polinómica
$n!$	Factorial
2^n	Exponencial

En la Figura 1.2 se muestra el comportamiento de las funciones de complejidad de la Tabla 1.1.

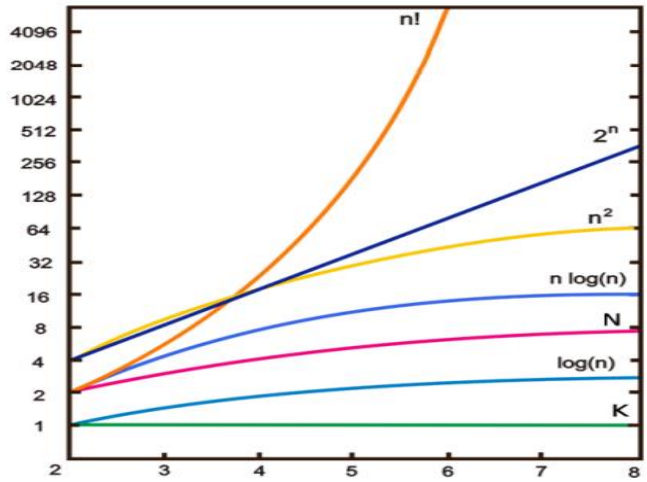

Figura 1.2 *Representación del crecimiento de funciones utilizadas comúnmente en las estimaciones con notación O* [17].

La importancia de conocer los tipos de complejidad algorítmica y su comportamiento con respecto al tamaño de la entrada radica en que permite diseñar

algoritmos que, en la medida de lo posible, cuenten con un desempeño eficiente que permitan obtener soluciones a un problema dado en un tiempo computacional logarítmico o polinomial, lo que implica que independientemente del equipo utilizado, el tiempo computacional requerido por el algoritmo será razonable. Por el contrario, un algoritmo con una complejidad superior a 2^n implica que el algoritmo no es aplicable al problema dado, debido al gran incremento en el tiempo computacional con respecto a un pequeño cambio en el tamaño de la entrada.

1.3 Teoría de la Complejidad de los Problemas

La teoría de la complejidad [2] [1] estudia la forma de clasificar los problemas de acuerdo con su complejidad para resolverlos. Esto involucra la cantidad de recursos computacionales requeridos por un algoritmo para resolver un problema determinado (explicado en el punto 1.2). De acuerdo con esto, se pueden determinar los recursos necesarios para diversos problemas de optimización y de esta forma poder clasificarlos dentro de las clases de complejidad existentes [18], para así poder desarrollar algoritmos eficientes que puedan resolverlos. Esto es de vital importancia debido a que aún el algoritmo más eficiente para un problema determinado, bajo cierto tamaño de instancia puede resultar sumamente ineficiente para cantidades de datos mayores, ya que los recursos computacionales pueden llegar a sobrepasar las capacidades de un equipo de cómputo o bien, el tiempo requerido para encontrar una solución puede ser extremadamente largo.

Los problemas se encuentran divididos dentro de la clasificación más importante en la actualidad, correspondiente a tres clases de complejidad,

identificadas como P, NP y NP-Completo, donde la clase P y NP-Completo son subconjuntos de la clase NP, como se muestra en la Figura 1.3.

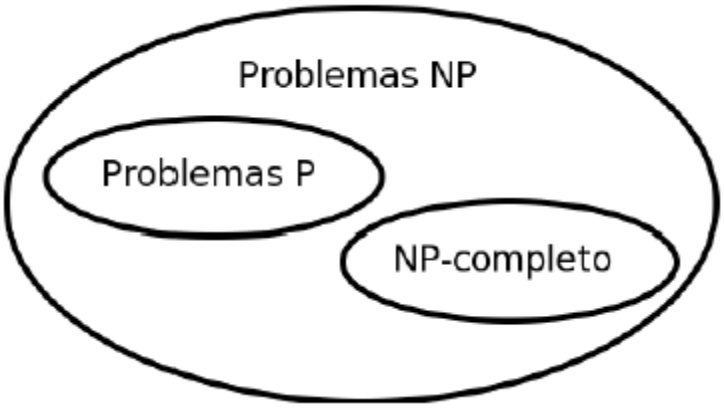

Figura 1.3 *Clasificación de los problemas acorde a la Teoría de la Complejidad* [19] [20].

Es importante tomar en cuenta la complejidad de un problema antes de proponer un método de solución, ya que de acuerdo con su clasificación se puede identificar la cantidad de recursos requeridos para obtener una solución bajo cierto tamaño de instancia y con base en esto, proponer un método de solución que puede ser determinístico o no-determinístico. De modo que se puede decir que un algoritmo resuelve un problema, si este funciona de forma eficiente para cualquiera de sus instancias [1].

De acuerdo con su clasificación [1] [14], los problemas P son el conjunto de problemas que pueden ser resueltos en todas sus instancias en el peor de los casos por un algoritmo determinístico en tiempo polinomial. Los problemas NP o problemas intratables, son aquellos que únicamente pueden ser tratados por algoritmos no-determinísticos acotados en tiempo polinomial, debido a que resulta muy difícil y costoso obtener una solución óptima para todos los tamaños de

instancia de un problema dentro de esta clasificación. Finalmente, la clase correspondiente a los NP-Completos engloba a los problemas más difíciles de resolver y que de acuerdo a sus características solo pueden ser abordados por algoritmos no-determinísticos.

Los algoritmos determinísticos son aquellos que a una misma entrada de datos devuelve la misma salida. Por otro lado, existen problemas tan difíciles, que se dice que encontrar un algoritmo que lo resuelva de forma eficiente en todas sus instancias es tan complicado como encontrar algoritmos eficientes para todos los problemas existentes dentro de la clase NP. Para este tipo de problemas, se desarrollaron los algoritmos no-determinísticos, los cuales son métodos estocásticos mejor conocidos como *heurísticos* [21], generalmente basados en la experiencia, mientras que los *metaheurísticos*[22], sustentan su funcionamiento con base a fenómenos naturales. Este tipo de métodos no tiene prueba de optimalidad. Estas heurísticas no llegan a encontrar soluciones óptimas a estos problemas para las instancias clasificadas como grandes.

1.4 Problema de Empaquetamiento

El problema de empaquetamiento de contenedores surge en las industrias de vidrio, papel, acero, imprentas, etc., este problema es un problema de optimización combinatoria que desde el punto de vista computacional se clasifica como un problema NP-Completo [1]. Esto significa que no se alcanza una solución exacta en un tiempo polinomial, y para encontrar soluciones cercanas al óptimo se hace uso de métodos heurísticos, donde las reglas empíricas para encontrar soluciones a problemas están basadas en la experiencia, y no tienen pruebas de optimalidad. En

este tipo de problemas se trata de encontrar la mejor solución entre un conjunto de soluciones factibles. Este problema cuando se analiza en dos dimensiones también es un problema geométrico porque para resolverlo, se trata de empaquetar un conjunto de piezas tanto regulares como irregulares en una pieza regular grande, en forma rectangular, además se debe verificar que cada una de las piezas a empaquetar no se traslapen entre ellas y que la suma de cada una de sus áreas sea menor que el área de la pieza regular rectangular grande.

En este trabajo de investigación se aplica el problema de empaquetamiento de dos dimensiones con figuras amorfas a un problema que surge en una imprenta. En la cual se procesa X toneladas de papel al año y tiene un desperdicio en merma de papel de Y %, esto representa Z toneladas de papel, Z=X*Y/100. Lo que se desea es lograr disminuir el porcentaje de la merma de papel.

Por lo tanto, **el objeto de este trabajo** es el estudio del problema de empaquetamiento en dos dimensiones para reducir la merma del papel en imprentas, derivándose como **campo de acción** la optimización del desperdicio de papel en imprentas a partir del desarrollo de una Metaheurística específicamente un Algoritmo Genético.

Con el propósito de brindarle una solución efectiva al problema de investigación anteriormente planteado, se define los objetivos de la investigación.

1.5 Estado del Arte

Algunos investigadores han abordado este problema, entre ellos Bennell y Oliveira en el 2008 [23], que proporcionan un tutorial con las metodologías geométricas básicas que actualmente emplean los investigadores en el corte y empaquetado de

formas irregulares. La inserción de figuras en el área de trabajo de estas metodologías es con la técnica not-fit-polygon [24], la cual recorre todo el contorno de la figura irregular insertada y trata de insertar otras figuras cerca de su contorno tomando en cuenta sus concavidades. También en el año 2009, exploraron diferentes formas de representar el problema y los mecanismos para avanzar entre las soluciones [25]. Además, evalúan los enfoques básicos para darle solución a este problema. Bennell y Song logran representar el problema como una lista ordenada de piezas que se empaquetan a partir de una heurística de colocación utilizando Not-Fit-Polygon [26].

Además, Han y colaboradores generan diseños de corte de guillotina de formas irregulares utilizando estrategias que consiste en aproximar las piezas a rectángulos, y utilizan una heurística de grupo para agrupar la mayor cantidad de piezas irregulares posibles con el objetivo de obtener buenas soluciones [27]. También López y colaboradores proponen una adaptación de una heurística que combina la selección con la ubicación de figuras [28]. Las pruebas que realizan son con una gran variedad de polígonos de tipo convexo y aplican varias técnicas de inserción como por ejemplo First Fit (FF), First Fit Decreasing (FFD), First Fit Increasing (FFI), Filler y Best Fit (BF). Estas técnicas de inserción de figuras no recorren el contorno de una figura como lo hace Not-Fit-Poligon, sino que insertan una figura en el área de inserción a partir de las coordenadas (0,0), a continuación de otra sin tomar en cuenta la existencia de concavidades, la inserción la hacen ordenando las figuras en orden creciente o decreciente. Martínez-Sykora y colaboradores en el año 2017 proponen varios modelos de programación para determinar la asociación entre figuras irregulares y contenedores. Aquí las figuras

pueden rotar libremente con el fin de hacer una mejor colocación en el contenedor, también aplican la técnica de Not-Fit-Polygon [29]. López y colaboradores describen una metodología hiperheurística al problema de empaquetamiento en donde obtienen resultados que se pueden comparar con una heurística simple, y en varios casos la hiperheurística da mejores resultados [30]. Las pruebas las realizan con polígonos cóncavos irregulares, aplican técnicas de inserción como First Fit (FF), First Fit Decreasing (FFD), First Fit Increasing (FFI), Filler y Best Fit (BF). También, Chernov y colaboradores presentan modelos matemáticos para el problema de corte y empaquetamiento, explican varias técnicas para la construcción de figuras en 2D y 3D, pero no resuelven los modelos propuestos [31]. Utilizan la técnica de insertar cada figura abajo a la izquierda de una figura ya insertada, el procedimiento tiene en cuenta las figuras ya insertadas, para evitar traslapes entre figuras y para poder llenar los espacios que quedaron vacíos en etapas anteriores de inserción. Por otra parte Wascher [32] presentan una tipología mejorada, que se basa en las ideas originales de Dyckhoff, pero introducen nuevos criterios, que definen categorías de problemas diferentes de las de Dyckhoff [33]. Demuestran la viabilidad del nuevo esquema al usarlo como base para una categorización de la literatura de corte y empaquetamiento entre los años 1995 y 2004, aplican la técnica de Not-Fit-Polygon. Los estudios más recientes realizados sobre el tema del Problema de Empaquetamiento de Contenedores de Dos Dimensiones Aplicado a Imprentas Digitales han sido realizados por nuestro grupo [34]. En esta investigación presentan un Algoritmo de Búsqueda Local Iterada con una Estructura de Vecindad Simple con el fin de encontrar traslapes entre figuras amorfas. Los resultados muestran que es posible optimizar el desperdicio de papel en imprentas digitales.

En el presente trabajo se presenta una técnica para la inserción de figuras, distinta a las técnicas mencionadas antes y que generalmente se aplican en la literatura.

1.6 Objetivos de la Investigación

Objetivo General

Implementar un Algoritmo Genético, con el diseño de un mecanismo de vecindad, para el Problema de Empaquetamiento de Contenedores de dos Dimensiones Aplicado a Imprentas Digitales.

Objetivos Específicos

- Implementar una estructura híbrida de vecindad eficaz basada en movimientos bidimensionales (Inserción, Traslación, Rotación e Intercambio) para obtener buenas soluciones.
- Implementar un Algoritmo Genético que permita darle solución al problema de empaquetamiento de contenedores de dos dimensiones aplicado a imprentas digitales.
- Evaluar la eficacia y la eficiencia del Algoritmo Genético implementado.
- Desarrollo de una herramienta con interfaz visual que permita interpretar de forma simple las soluciones obtenidas.

1.7 Alcance de la Tesis

- Implementación de un algoritmo genético con una estructura de vecindad híbrida aplicado al Problema de Empaquetamiento de Contenedores de dos Dimensiones.

- Diseño de una herramienta con interfaz visual que permitirá interpretar la solución del problema en forma sencilla.
- Los tipos de figuras pueden ser una combinación de regulares e irregulares de diferentes tamaños y formas.

1.8 Contribución de la Tesis

- Implementación de una estructura de vecindad híbrida con diversidad en las perturbaciones bidimensionales.
- Implementación de un Algoritmo Genético con una estructura de vecindad híbrida con diversidad en las perturbaciones bidimensionales.
- Disminución del área residual, que representa la merma en la hoja de papel de imprenta.

1.9 Organización de la Tesis

Este trabajo de investigación está conformado por 6 capítulos. En el Capítulo 1 se da una introducción del problema a tratar; se da una breve explicación del estado del arte y se enumeran los objetivos y alcances de la investigación, así como la organización general de la tesis.

En el Capítulo 2 se da una introducción al Problema de Empaquetamiento de Contenedores de dos dimensiones, así como su descripción conceptual y el modelo matemático que representa al problema que trata el presente trabajo.

En el Capítulo 3 se habla acerca de las Metaheurísticas existentes y se da una descripción de cada una de ellas, y por último se justifica porque se selecciona la Metaheurística Algoritmo Genético.

En el Capítulo 4 se expone la Metodología de Solución. En este capítulo se describe el Algoritmo Genético y sus componentes.

En el Capítulo 5 se exponen los Resultados experimentales. En este capítulo se presentan los resultados del Algoritmo Genético. La metodología de sintonización de los parámetros de control del Algoritmo Genético y el Análisis de la Complejidad del Algoritmo. Un análisis de eficacia y eficiencia.

En el Capítulo 6 se exponen las Conclusiones y Trabajos Futuros. En este capítulo se presentan las conclusiones obtenidas de este trabajo de investigación, así como el trabajo futuro.

Capítulo 2. Problemas de Empaquetamiento de Contenedores de Dos Dimensiones

En este capítulo, se presenta una descripción de los problemas de empaquetamiento existentes, se hace énfasis en sus semejanzas y sus diferencias. Luego se realiza una clasificación de estos problemas teniendo en cuenta sus principales características. Más adelante se centra en el problema de empaquetamiento en dos dimensiones en forma rectangular, se expone su modelo matemático y por último se concluye con la explicación del problema de empaquetamiento en dos dimensiones aplicado a las imprentas.

Los problemas de empaquetamiento son problemas de la optimización combinatoria y a su vez son problemas geométricos de dos dimensiones. Los mismos parten de modelos básicos, y se encuentran en una amplia gama de aplicaciones prácticas existentes, en dependencia de quien lo esté tratando.

2.1 Descripción de los Problemas de Empaquetamiento

Los problemas de empaquetado consisten en colocar un conjunto de elementos, por lo general pequeños, en uno o más objetos, por lo general de grandes dimensiones, sin que los elementos pequeños a posicionar se traslapen, a modo de minimizar o maximizar una función objetivo. El problema de empaquetamiento de dos dimensiones es un problema de optimización combinatoria con muchas aplicaciones importantes en las industrias de la madera, del vidrio, del aluminio, del cuero, y del papel, además en el paginado de periódicos y en la distribución de la

carga de contenedores y camiones. Por muchas décadas, el problema de empaquetado ha atraído la atención de muchos investigadores en varias áreas.

Los problemas de empaquetamiento se pueden clasificar usando distintos criterios. La dimensionalidad del problema es uno de los criterios y la mayoría de los problemas se definen en una, dos o tres dimensiones. En este trabajo de investigación se considera un problema en dos dimensiones. El próximo criterio para clasificar los problemas de empaquetamiento en dos dimensiones es la geometría de los elementos pequeños. Este trabajo de investigación está enfocado principalmente a los problemas de geometrías irregulares. El problema de empaquetamiento con geometrías irregulares ha sido estudiado en las últimas décadas. Cuando las figuras de los elementos son polígonos o tienen figuras de distinto tipo de geometría, el problema es denominado irregular.

Estos problemas describen patrones que consisten en combinaciones geométricas de grandes objetos y pequeños elementos. En el caso de los problemas de empaquetado, los objetos grandes (contenedores) necesitan ser llenados con pequeños elementos (por ejemplo, cajas). Por su parte, los problemas de corte están caracterizados por grandes objetos (por ejemplo, planchas o rollos) que deben ser cortados en pequeños elementos (por ejemplo, figuras de dos dimensiones). El objetivo de los procesos de corte y empaquetado es maximizar la utilización del material, es decir, asignar todos los elementos sin traslapes en un mínimo número de contenedores o planchas.

Los problemas de empaquetamiento, por un lado, se denominan problemas geométricos porque dentro de cada objeto grande se deben ordenar uno o más objetos pequeños de tal forma que se eviten la traslapes o solapamientos entre los

objetos, y de posicionarse en los límites del objeto geométrico, es decir, cómo cortar los pequeños objetos. También se denominan combinatorios debido a que se deben tomar decisiones de cuáles serán los elementos a producir y de qué objeto se obtendrán.

Como se puede observar, a cada uno de los grandes objetos se le asigna una serie de pequeños objetos o elementos y además cada elemento se asigna como máximo a un único objeto grande. Estos dos problemas se combinan y las soluciones deben ser posibles, tanto desde el punto de vista “cuantitativo” (es decir, del número mínimo o máximo de pequeños elementos que producir y la disponibilidad de los grandes objetos), como desde el punto de vista geométrico (es decir, no ocasionar traslapes entre los pequeños elementos y la contención de los pequeños elementos dentro de los límites de los grandes objetos).

2.2 Clasificación de los Problemas de Empaquetamiento de Contenedores de dos Dimensiones

En las últimas décadas, muchos investigadores de la comunidad científica han estudiado los problemas de empaquetamiento. Estos problemas surgen en muchas industrias y no están restringidos al sector de la manufactura. Por ejemplo, se pueden encontrar problemas de empaquetado de una manera más abstracta en la investigación de operaciones y en el sector financiero.

Teniendo en cuenta lo anterior, se puede mencionar que los problemas de empaquetamiento aparecen bajo diferentes nombres en la literatura [35]. Estos problemas pueden aparecer con el nombre de Dickhoff [33], en donde se presenta una caracterización integrando todas las clases de problemas de corte y

empaquetado, la cual generaliza la clasificación presentada por Hinxman [35] a principios de los años 80. La taxonomía propuesta por Dickhoff permite identificar propiedades comunes de ciertos problemas, los cuales parecieran no tener relación a primera vista. Por otra parte, las diferencias entre problemas que aparentan ser similares surgen al analizar sus principales características. Dickhoff distingue entre problemas de corte y empaquetamiento involucrando dimensiones espaciales y aquellos involucrando dimensiones no espaciales. El primer grupo consiste que los problemas de corte y empaquetamiento o carga definidos en un máximo de tres dimensiones en el espacio euclídeo (por ejemplo, problemas de corte, carga de vehículos y carga de paletas). El otro grupo abarca a los problemas de corte y empaquetamiento definidos en un espacio de dimensiones no espaciales, tales como peso, tiempo o dinero (por ejemplo, asignación de memoria, presupuestación del capital, cambio de moneda y balance en línea).

Esta taxonomía se basa en la estructura lógica básica de los problemas de corte y empaquetamiento, la cual se puede determinar de la siguiente forma:

1. Existen dos grupos de datos básicos cuyos elementos definen cuerpos geométricos de formas fijas (figuras) en un espacio de una o más dimensiones de números reales:

- Materia prima, también llamada objeto.
- Lista u orden de elementos.

2. El proceso de empaquetamiento se realiza en base a la generación óptima de patrones, los cuales son combinaciones geométricas de pequeños elementos asignados a grandes objetos, que determinan la posición de cada elemento en los

grandes objetos. Los espacios residuales, es decir, figuras que ocupan lugar en los patrones pero que no pertenecen al conjunto de pequeños elementos, se tratan por lo general como desperdicio.

La clasificación de Dickhoff describe cuatro características de las más importantes de los problemas de corte y empaquetamiento [33]:

La característica más importante es la dimensión, la cual define la cantidad mínima de dimensiones necesarias para describir la geometría de los patrones (una dimensión, dos dimensiones, tres dimensiones o más dimensiones). Problemas con más de tres dimensiones se obtienen cuando se expanden a dimensiones no espaciales, como por ejemplo tiempo o peso.

La clase de asignación describe si se deben asignar todos los objetos y elementos o sólo una parte de ellos.

La variedad de los objetos distingue entre problemas que tienen los objetos de idéntica forma o diferente.

La variedad de los elementos se refiere a la forma y cantidad de los elementos. Los problemas pueden consistir en pocos elementos, elementos congruentes, muchos elementos de muchas formas diferentes y numerosos elementos de una cantidad relativamente reducida de formas diferentes.

La tipología de Dyckoff ha presentado algunas deficiencias, las cuales han creado problemas para tratar con recientes desarrollos e impide que sean aceptados de forma más general. En el año 2007, Wascher y colaboradores presentaron una nueva tipología para los problemas de corte y empaquetamiento, basada parcialmente en las ideas originales de Dyckhoff, pero introduce nuevos

criterios para la categorización, lo que permite definir categorías de problemas en forma diferente a lo que hace Dyckoff [32].

Los problemas de empaquetamiento regulares están relacionados con el empaquetamiento de un conjunto de rectángulos en un objeto también rectangular. El empaquetado irregular (ver ejemplo en Figura 2.1) también es conocido como de anidación, y uno de los ejemplos de aplicación es en la industria naval, y como problema de diseño de marcaciones en la industria textil. Este trabajo de investigación se enfocará en una clase de problemas irregulares que se describirá en detalles más adelante.

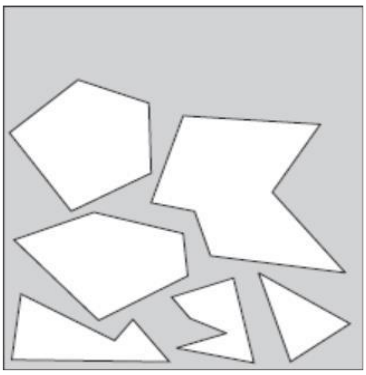

Figura 2.1. *Ejemplo de patrones con figuras irregulares.* [7]

En función de la forma que tengan los elementos o piezas, o a la geometría de estos, al ser empaquetados se pueden distinguir dos tipos de distribuciones. En el caso de elementos regulares, los patrones pueden ser ortogonales (cortes que deben ser paralelos a los lados del objeto) y no ortogonales.

En las Figuras 2.2 y 2.3 se muestran ejemplos de distintos tipos de cortes. Sin embargo, los cortes ortogonales pueden ser de guillotina o no guillotina. En los cortes de guillotina, se deben realizar una serie de cortes paralelos a los ejes del gran objeto que atraviesen el mismo de lado a lado; mientras que los cortes “no

guillotina", no se aplica esta restricción: un elemento se puede colocar en cualquier posición disponible, siempre que no resulte una superposición con otro elemento cualquiera. Algunos patrones deben respetar cortes de guillotina, pero en niveles de n-etapas, es decir por una sucesión de cortes horizontales o verticales considerando el ancho de la plancha, que en este trabajo no es más que la hoja de papel que se quiere optimizar el uso de esta.

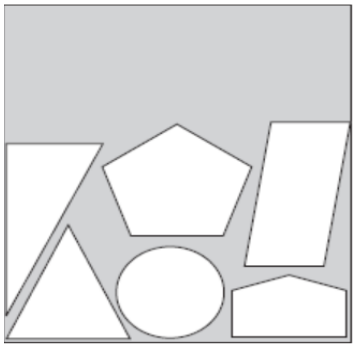

Figura 2.2. *Ejemplos de patrones con cortes no ortogonales.* [7]

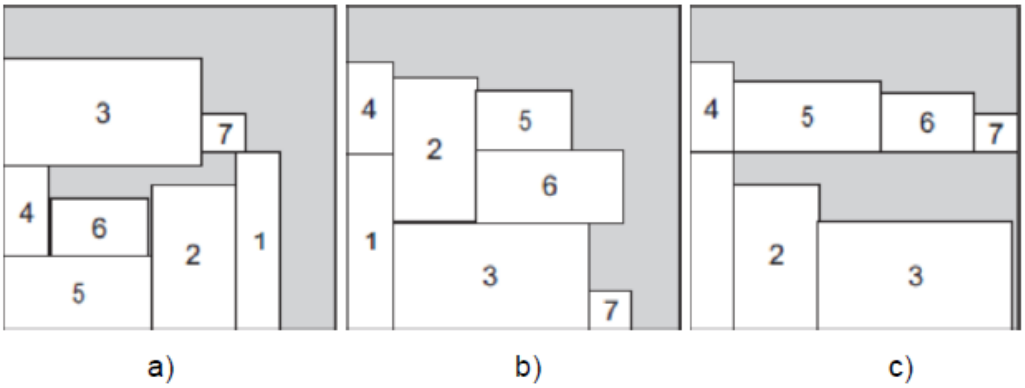

Figura 2.3. *Ejemplos de patrones con cortes ortogonales: (a) corte no guillotina; (b) corte guillotina; (c) corte guillotina en niveles.* [7]

La restricción del corte de la guillotina se aplica generalmente, por las características tecnológicas de las máquinas automatizadas que realizan el corte, mientras que, por lo general, no está presente en aplicaciones de empaquetado.

En la Figura 2.3 (b) se muestra un ejemplo de corte guillotina. En un primer corte vertical se divide la plancha en dos partes: en una de ellas, se realizan dos cortes horizontales, se obtienen las piezas 1 y 4 y el material de desperdicio. La otra parte necesita de más cortes, realizando un corte horizontal por el ancho se obtienen nuevamente dos partes: una con las piezas 3 y 7 (que se separan por un corte vertical) y con las piezas 2, 6 y 5 que para separarlas se realiza primero un corte vertical y luego uno horizontal, además, son necesarios otros cortes para separar el desperdicio.

En la Figura 2.3 (c) se presenta un ejemplo también de cortes guillotina, pero en este caso las piezas están distribuidas formando niveles, primeramente, estas piezas se obtienen realizando cortes horizontales de lado a lado de la plancha para obtener los distintos niveles, luego se aplican los cortes verticales para obtener las distintas piezas y material de desperdicio.

En algunos problemas se asume que los elementos tienen orientación fija (es decir, no se pueden rotar) y no se aplica la restricción sobre el patrón del corte. En ciertos contextos reales, se permite la rotación de elementos, generalmente en 90^0, a fin de producir mejores asignaciones. Por ejemplo, la rotación no está permitida cuando los elementos son artículos para acomodar en la página de un periódico o son piezas para cortar de planchas corrugadas o decoradas; mientras que sí se permite en el corte de materiales lisos y en la mayoría de los contextos de empaquetado. En este trabajo de investigación al ser piezas que son de geometría irregular, las mismas van a poder rotar en cualquier dirección para buscar la mejor posición en la hoja de papel.

2.3 Problemas de Empaquetamiento de Forma Irregular

Se considera el siguiente problema de empaquetado rectangular en dos dimensiones.

Dados ***n*** elementos (rectángulos pequeños) ***I = {1, 2,. . ., n}***, cada uno caracterizado por su ancho ***wi*** y su altura ***hi,*** y uno o más objetos grandes (rectángulos). Los elementos se deben colocar en forma ortogonal sin provocar superposición entre los objetos (los lados de cada elemento son paralelos a los lados del objeto) de forma tal de minimizar o maximizar una función objetivo determinada.

El problema de empaquetado rectangular surge en muchas aplicaciones industriales, frecuentemente con pequeñas variaciones en las restricciones impuestas. Muchas son las variantes de este problema que se han considerado en la literatura. Las siguientes características son importantes para clasificar al problema: tipo de elementos seleccionados (figura) a colocar en el objeto (hoja de papel), ordenamiento de los objetos y ordenamiento de los elementos.

En este trabajo de investigación científica, se presentarán los seis tipos de problemas de empaquetamiento rectangular más estudiados. Para simplificar, se definen los problemas asumiendo que cada elemento tiene una orientación fija y no se asigna la restricción de corte de la guillotina. Es simple extender la definición para otros casos donde cada elemento puede ser rotado en 90^0, donde se apliquen cortes de la guillotina. Lo primero es considerar dos tipos de problemas de empaquetamiento, un gran objeto rectangular grande y alto, el cual puede crecer en una o más dimensiones, donde se deben colocar los elementos sin superposición. Los problemas son llamados *strip packing y minimización de área.*

Problema de strip packing. Dados ***n*** elementos (pequeños rectángulos), cada uno caracterizado por su ancho ***wi*** y su altura ***hi***, y un gran objeto (llamado tira o strip), cuyo ancho ***W*** es fijo, pero su altura ***H*** es variable. El objetivo es reducir al mínimo la altura ***H*** usada de la tira, de manera que todos los elementos puedan ser empaquetados.

Problema de minimización de área. Dados ***n*** elementos, cada uno definido por su ancho ***wi*** y su altura ***hi***, y un gran objeto cuyo ancho ***W*** y altura ***H*** son variables. El objetivo es minimizar el área ***W*** × ***H*** del objeto de manera que todos los artículos pueden ser empaquetados en la tira.

Otros dos problemas de empaquetamiento son el *problema de empaquetamiento en dos dimensiones y problema de la mochila en dos dimensiones,* en los cuales los objetos grandes tienen dimensiones fijas y se puede tener uno único o varios de ellos.

Problema de empaquetamiento en dos dimensiones. Dado un conjunto de elementos, donde cada elemento ***i*** tiene un ancho ***wi*** y una altura ***hi***, y una cantidad ilimitada de grandes objetos (cajas rectangulares) con idéntico ancho y alto. El objetivo es reducir al mínimo el número de cajas rectangulares utilizadas para colocar todos los elementos. En este trabajo de investigación resolveremos este problema en dos dimensiones, haciendo una analogía en donde el número de cajas rectangulares es el número de hojas de papel y donde cada elemento es una figura con geometría irregular, y el objetivo es minimizar el desperdicio del papel, evitando la superposición entre las figuras.

Problema de la mochila en dos dimensiones. Dado un conjunto ***I*** de elementos, donde cada elemento ***i*** ∈ ***I*** tiene un ancho ***wi***, una altura ***hi*** y un valor ***ci***.

También se cuenta con una mochila rectangular con ancho ***W*** y alto ***H***. El objetivo es hallar un subconjunto ***I'*** ⊆ ***I*** con valor máximo total Σ***i***∈***I'ci*** tal que todos los elementos ***i*** ∈ ***I'*** puedan ser empaquetados en la mochila.

Para el problema de empaquetamiento en dos dimensiones. Lodi y colaboradores proponen métodos heurísticos y algoritmos metaheurísticos y realizan experimentos sobre instancias de varios casos de prueba [36]. Para el problema de la mochila en dos dimensiones Wu y colaboradores proponen algoritmos heurísticos que son efectivos para muchas instancias de prueba [37]. También es importante mencionar otros dos problemas**: *patrón de corte en dos dimensiones y carga de paletas.***

Problema del patrón de corte en dos dimensiones. Dado un conjunto de elementos, donde cada elemento ***i*** tiene un ancho ***wi***, una altura ***hi*** y una demanda ***di***, y una cantidad ilimitada de objetos de idéntico ancho y alto. El objetivo es reducir al mínimo la cantidad de objetos utilizados para colocar todos los elementos (es decir, para cada elemento ***i*** se colocan ***di*** copias en los distintos objetos).

Carga de paletas. Dada una cantidad suficientemente grande de elementos de idéntico tamaño **(*w, h)***, misma área y dimensiones, y un gran objeto rectangular de tamaño ***(W, H),*** el objetivo es colocar la máxima cantidad de elementos en el objeto rectangular, en el que cada elemento se puede girar 90^0.

2.4 Modelo Matemático

El problema que se está analizando en este trabajo se puede representar matemáticamente como un problema de minimización de áreas como se muestra en la Figura 2.4. (***Problema de empaquetamiento en dos dimensiones***) en donde

dado ***n*** elementos, cada uno definido por su ancho ***wi*** y su altura ***hi,*** y un gran objeto grande cuyo ancho y altura son fijos. El objetivo es minimizar el área ***W x H*** del objeto de manera que todos los elementos puedan ser empaquetados en ese objeto rectangular grande**.** El mismo se puede representar formalmente como el siguiente modelo matemático:

$$Min\ f(W, H, w, h) = ((W*H) - \sum (wi * hi) - A) \qquad (2.1.1)$$

Subject to:

$$0 \leq x_i \leq W - wi \qquad (2.1.2)$$

$$0 \leq yi \leq H - hi \qquad (2.1.3)$$

$$xi + wi \leq xj \qquad (2.1.4)$$

$$xj + wj \leq xi \qquad (2.1.5)$$

$$yi + hi \leq yj \qquad (2.1.6)$$

$$yj + hj \leq yi \qquad (2.1.7)$$

$$xi, xj, yi, yj, wi, wj, hi, hj, W, H, \in N \qquad (2.1.8)$$

Figura 2.4. *Modelo Matemático del problema de empaquetamiento en contenedores en dos dimensiones* [34].

La función objetivo está representada por la Ecuación (2.1.1) consiste en minimizar el área de desperdicio de la hoja de papel o área residual, la cual realiza una resta del área de la hoja de papel ***(W*H)*** y la suma de las áreas de cada una de las figuras. Aquí ***W*** y ***H*** representan el ancho y alto de la hoja de papel, ***w*** y ***h*** representan las dimensiones de un polígono regular que tiene embebida a una figura F***, A*** representa el área residual en pixeles del polígono regular sin tomar en cuenta el área de la figura embebida F, ***xi***, ***yi*** son las coordenadas del borde superior derecho de la figura F que se inserta en la hoja de papel.

Las restricciones de las Ecuaciones 2.1.2 y 2.1.3, indican que cada figura F, debe de estar dentro de la hoja de papel, mientras que las Ecuaciones de las

restricciones 2.1.4 a 2.1.7, indican que cada figura F, no debe de tener traslape, es decir, cada desigualdad en cada restricción significa la ubicación relativa de cada figura F, la cual puede estar a la izquierda una con otra o bien a la derecha una con otra, o en cualquier dirección con respecto a cada figura F. Se debe de aclarar que el modelo matemático no acepta valores negativos en las variables.

2.5 Problema de Empaquetamiento Irregular. Representación.

La Figura 2.5 muestra una representación gráfica del problema de empaquetamiento de dos dimensiones con figuras irregulares aplicado a imprentas. Se observa una maqueta que se desea construir para una producción en serie sobre papel, utilizando una imprenta digital. El papel se forma de un área rectangular grande, representado aquí por un fondo color negro, se tienen varias figuras de distintos tamaños insertadas en el área rectangular del papel. El área residual es la resta del área de la hoja de papel y la suma total de las áreas de cada una de las figuras, para que se entienda mejor, se mostraran tres variables y por último se formará una ecuación, la primera variable se llamará área hoja de papel y se representará por AHP, la segunda variable se llamará suma total de áreas de todas las figuras STAF y la tercera variable se llamará área residual y se representará como RA entonces al formar la ecuación seria RA = AHP – STAF, esta ecuación representa la función objetivo que se va a minimizar en este trabajo de investigación. El objetivo es minimizar el área residual sin que existan traslapes en las figuras, mientras más figuras y menos fondo negro exista en la maqueta, el desperdicio de papel será menor de acuerdo con el diseño óptimo de la maqueta, aplicada a una producción en serie sobre papel utilizando de una imprenta digital. El diseño óptimo

de la maqueta, que es igual a la inserción óptima de figuras en la hoja de papel que minimiza el área residual.

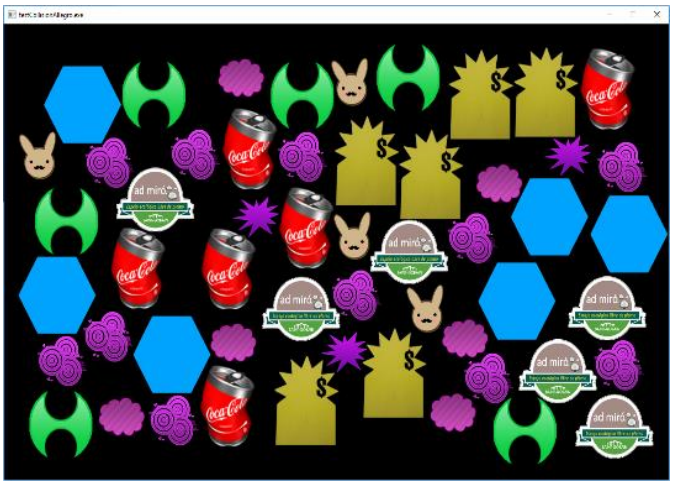

Figura 2.5. *Representación de una maqueta para el problema de empaquetamiento de dos dimensiones* [34].

2.6 Conclusiones Parciales del Capítulo 2

Se concluye que el Problema de Empaquetamiento de Contenedores de dos dimensiones, al ser un problema que surge en la industria, puede ser aplicado a problemáticas que aparecen en las imprentas. En la actualidad existen imprentas que anualmente procesan alrededor de 350 toneladas de papel al año y tienen un desperdicio promedio del 10 % que representa 35 toneladas de papel. En este trabajo de investigación se resuelve el Problema de Empaquetamiento de Contenedores de dos Dimensiones aplicado a imprentas Digitales, el mismo se analiza con un Algoritmo Genético y una Estructura Híbrida de Vecindad con movimientos bidimensionales (Inserción, Traslación, Rotación e Intercambio) embebida en el operador genético de la mutación con el fin de encontrar mejores soluciones a las que ya se han obtenido y tiempos de respuestas razonables.

Capítulo 3 Metaheurísticas

3.1 Problemas de optimización

La optimización combinatoria es una rama de las matemáticas que estudia problemas caracterizados por un número finito de soluciones factibles. Un área de aplicación importante se refiere al uso eficiente de recursos con el fin de incrementar la productividad. Como ejemplo de esta clase de problemas se pueden mencionar: El Problemas de Empaquetamiento de Contenedores de dos Dimensiones, el Problema del Viajante de Comercio, los Problemas de Planificación de Tareas, de Asignación, entre otros problemas.

Según Papadimitriou y Steiglitz, un problema de optimización se define de la siguiente manera [2]:

Problema de optimización: un problema de optimización se formaliza como un par (S, f) donde $S \neq 0$ representa el espacio de soluciones (o de búsqueda) del problema, mientras que f es una función denominada función objetivo o función de fitness, que se define como: $f: S \rightarrow R$, donde R representa el conjunto de los números reales. Así, resolver un problema de optimización consiste en encontrar una solución, $i^* \in S$, que satisfaga la siguiente desigualdad:

$$f(i^*) \leq f(i), \forall i \in S \qquad (3.1.1)$$

Asumir el caso de maximización o minimización no restringe la generalidad de los resultados, puesto que se puede establecer una igualdad entre tipos de problemas de maximización y minimización de la siguiente forma [38] [39]:

$$Max\{f(i)/i \in S\} \equiv min\{-f(i)/i \in S\}. \quad (3.1.2)$$

En función del dominio al que pertenezca ***S***, podemos definir problemas de optimización binaria $(S \subseteq B*)$, entera $(S \subseteq N*)$, continua $(S \subseteq R*)$ o mixtas $(S \subseteq (B \cup N \cup R)*)$. Aunque en principio la solución óptima para tales problemas se puede hallar por una simple enumeración, en la práctica esto es con frecuencia imposible, especialmente para problemas con tamaños reales, donde el número de soluciones factibles puede ser muy alto. Para problemas que presentan un alto grado combinatorio es difícil generar un modelo basado en programación matemática que represente exactamente una situación real. Durante la década de los setenta se han diseñado diversos métodos, conocidos como heurísticos ad hoc, capaces de encontrar soluciones de buena calidad y, que, en muchos casos, son muy cercanas a la solución óptima.

Gran parte de estos métodos fueron concebidos inspirándose en la resolución de problemas de fácil representación, pero de muy difícil solución, como lo son el problema del vendedor viajero (TSP), el problema de la mochila, entre otros problemas más. Debido a la variada naturaleza de estos problemas, los métodos eran útiles apenas para el problema en el cual habían sido inspirados.

Con el desarrollo de la teoría de complejidad a principios de los años setenta, se vuelve claro que, como la mayoría de los problemas de optimización eran NP-duros, había poca esperanza de hallar procedimientos eficientes de solución exacta. Esto enfatizó el rol de las heurísticas para resolver problemas de optimización que fueran encontrados en aplicaciones de la vida real, fueran o no NP-duros.

Mientras se desarrollaban y se experimentaba con muchas opciones, las más populares fueron las técnicas de búsqueda local. Estas técnicas son un procedimiento de búsqueda iterativo que, comenzando con una solución inicial factible, progresivamente la va mejorando al aplicar una serie de modificaciones locales o movimientos. En cada iteración, la búsqueda se mueve a una solución factible mejorada que difiere sólo un poco de la solución actual. La búsqueda finaliza cuando encuentra un óptimo local con respecto de las transformaciones que considera. Por lo general, este óptimo local es una solución bastante trivial, transformándose en una limitación del método. En búsqueda local, la calidad de las soluciones obtenidas y los tiempos de computación empleados son altamente dependientes de la "bondad" del conjunto de transformaciones consideradas en cada iteración de la heurística ad hoc.

Un Problema de Optimización Combinatoria ***P*** para el Problema de Empaquetamiento de Contenedores de Dos Dimensiones aplicado a Imprentas Digitales se puede representar a través de una tupla de 3 elementos: ***P = (f, S, F)*** donde:

- ***S:*** Conjunto finito de soluciones.
  - ✓ s: Configuración de figuras en una hoja de papel.
- ***F:*** Conjunto de soluciones factibles, $s \in F \subseteq S$, que está incluido en el conjunto finito de soluciones.
- $f: F \rightarrow R$, función del valor de aptitud, evalúa área residual.
  - ✓ Donde f(s) = valor de aptitud (Área residual)
  - ✓ Si $s^* \in F \subseteq S$, y $\forall s \in F$ entonces se cumple que $f(s^*) \leq f(s)$
  - ✓ Problema que busca minimizar el área residual en una hoja de papel.

Otro de los conceptos importantes que no se pueden dejar de ver cuando se habla de los problemas de optimización combinatoria, en la búsqueda local, es el **Concepto de Vecindad.**

Una **Vecindad** se puede definir a partir de una solución factible $s \in F$, la vecindad $N(s)$ de esa solución es un subconjunto del espacio de soluciones que están próximas a s, como muestra la Figura 3.1.

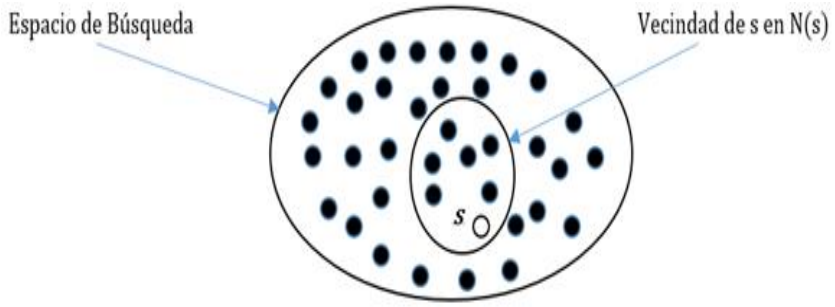

Figura 3.1. *Vecindad de una solución* ***s*** *marcado por el área interior de una circunferencia.*

La cercanía entre dos soluciones en el espacio de búsqueda se puede definir a partir de una función distancia, y con la función distancia se puede definir la vecindad $N(s) \subseteq F$ de la solución s como:

$$N(s) = \{s' \in F: Distancia\_Hamming(s, s') \leq \varepsilon\} \text{ para algún } \varepsilon > 0. \quad (3.1.3)$$

Donde ε es la distancia máxima de un vecino de s. Si β es el número máximo de perturbaciones en las coordenadas de la pieza, se puede crear s' a partir de s. En consecuencia, la vecindad de s está definida por la Ecuación 3.1.4, donde s' se alcanza desde s, aplicando un movimiento que puede ser de Inserción, Traslación, Rotación e Intercambio.

$$N(s) = \{s' \in S' \; V \; s \; \beta \; s'\} \qquad (3.1.4)$$

3.2 Características de las Metaheurísticas

En los años ochenta surgen las técnicas metaheurísticas y éstas plantean un cambio importante en el desarrollo de técnicas alternativas frente a heurísticas ad hoc. La idea básica de las metaheurísticas era combinar diferentes métodos heurísticos a un nivel más alto, para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. El término metaheurísticas fue introducido por primera vez por Glover en 1986 [40]. Antes de que el término fuese aceptado completamente por la comunidad científica, estas técnicas eran denominadas heurísticas modernas [41]. Esta clase de algoritmos incluye técnicas como colonias de hormigas, algoritmos evolutivos, búsqueda local iterada, recocido simulado y búsqueda tabú. Se pueden encontrar revisiones de metaheurísticas en los trabajos de Glover y Blum en el año 2003 [42] [43].

De las diferentes descripciones de metaheurísticas que se encuentran en la literatura se pueden destacar ciertas propiedades fundamentales que caracterizan a este tipo de métodos:

- Las metaheurísticas son estrategias o plantillas generales que guían el proceso de búsqueda.
- El objetivo es una exploración eficiente del espacio de búsqueda para encontrar soluciones óptimas o cercanas a la óptima.
- Las metaheurísticas son algoritmos no deterministas y generalmente son no deterministas.

- Las metaheurísticas pueden incorporar mecanismos para evitar regiones no prometedoras del espacio de búsqueda.
- El esquema básico de cualquier metaheurísticas tiene una estructura predefinida.
- Las metaheurísticas pueden usar conocimiento del problema que se trata de resolver en forma de heurísticos específicos que son controlados por la estrategia de más alto nivel.

Teniendo en cuenta todos estos puntos, se puede exponer que una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar y explotar el espacio de búsqueda. Resulta de especial importancia el correcto equilibrio (generalmente dinámico) entre diversificación e intensificación. El término diversificación se refiere a la evaluación de soluciones en regiones distantes del espacio de búsqueda (de acuerdo con una distancia previamente definida entre soluciones); también se conoce como exploración del espacio de búsqueda. Por otro lado, el término intensificación, se refiere a la evaluación de soluciones en regiones acotadas y pequeñas con respecto al espacio de búsqueda centradas en la vecindad de soluciones concretas (explotación del espacio de búsqueda).

El equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente las regiones prometedoras del espacio de búsqueda global y, por otro lado, no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Dentro de las metaheurísticas podemos distinguir dos tipos de estrategias de búsqueda. Por un lado, tenemos las extensiones “inteligentes” de los métodos de

búsqueda local (metaheurísticas basadas en trayectoria). El objetivo de estas estrategias es evitar de alguna forma los mínimos locales y moverse a otras regiones prometedoras del espacio de búsqueda. Este tipo de estrategia es seguida por la búsqueda tabú, la búsqueda local iterada, la búsqueda con vecindario variable y el recocido simulado. Estas metaheurísticas trabajan sobre una o varias estructuras de vecindario impuestas por el espacio de búsqueda. Una estrategia distinta es la seguida por los algoritmos basados en colonias de hormigas o los algoritmos evolutivos; éstos incorporan un componente de aprendizaje en el sentido de que, de forma implícita o explícita, intentan aprender la correlación entre las variables del problema para identificar las regiones del espacio de búsqueda con soluciones de alta calidad. Estos métodos realizan, en este sentido, un muestreo parcial del espacio de búsqueda. Estas metaheurísticas se denominan basadas en población.

3.3 Clasificación de las Metaheurísticas.

Las técnicas metaheurísticas se pueden clasificar y describir de distintas maneras, dependiendo de las características seleccionadas para diferenciarlas[43]. Es posible clasificarlas en metaheurísticas inspiradas en la naturaleza versus las no inspiradas, basadas en memoria o sin ella, o en métodos que usen funciones objetivo estáticas o dinámicas, etc. Una de las clasificaciones más populares, hoy en día, las describen teniendo en cuenta si utilizan en cada paso un único elemento del espacio de búsqueda, denominadas metaheurísticas basadas en trayectoria, o si trabajan con un conjunto de puntos o población, en este caso referidas como

metaheurísticas basadas en población. La Figura 3.2 presenta la taxonomía adoptada y además las distintas metaheurísticas dentro de cada clase. En las siguientes secciones se presentarán los aspectos más destacados de cada una de las metaheurísticas.

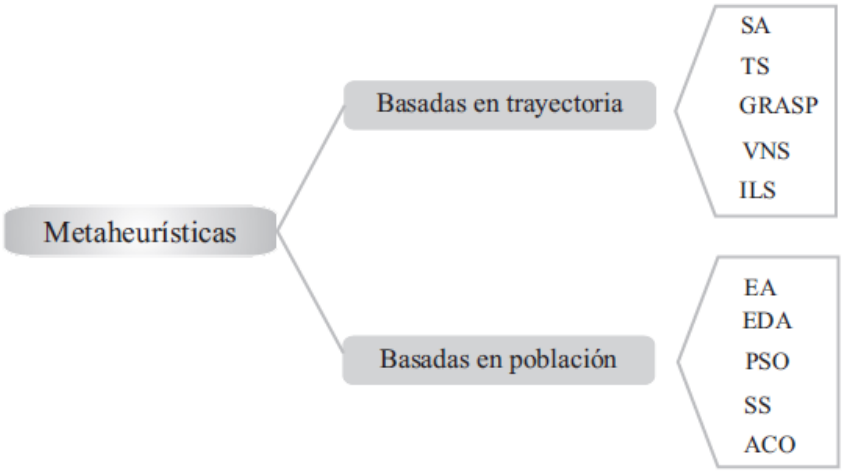

Figura 3.2. *Clasificación de las metaheurísticas.*

3.4 Métodos Basados en Trayectoria.

Esta sección está dedicada a distinguir las metaheurísticas más importantes basadas en trayectoria y a explicar en forma sintética su funcionamiento. En general las distintas metaheurísticas que caen bajo esta clasificación se caracterizan por partir de una única solución. La actualización de esta solución se realiza examinando el vecindario, de esta manera se genera una trayectoria. La mayoría de estos algoritmos surgen como extensiones de los métodos simples de búsqueda local a los que se les añade algún mecanismo para escapar de los mínimos locales, debido a que su rendimiento es poco satisfactorio. Los criterios de parada deben ser más complejos que alcanzar un mínimo local; lo que habitualmente se utiliza es alcanzar un tiempo máximo de procesamiento o un número máximo de iteraciones,

encontrar una solución s de determinada calidad o detectar que no se produjeron mejoras en las últimas iteraciones.

3.4.1 Recocido Simulado (SA).

La técnica de enfriamiento simulado o Simulated Annealing (SA) se considera como una de las pioneras entre los métodos metaheurísticas y una de las primeras en explicitar una estrategia para escapar de los óptimos locales. SA fue presentado como un algoritmo de búsqueda por Kirkpatrick y colaboradores en 1983 [44] y Cerney en 1985 [45] , en forma independiente.

El algoritmo se origina de un mecanismo estadístico, denominado “metrópolis” [46]. Los conceptos de SA fueron inspirados en el proceso de enfriamiento físico de metales. En cada iteración del algoritmo, se comparan los valores objetivos de la solución actual s y una solución s′ del vecindario N(s). Se acepta s′ si su valor de función de fitness es mejor que el fitness de s, pasando a ser ahora la solución actual para la nueva iteración. Por otro lado, si s′ no tiene valor de fitness inferior al valor de s, tiene probabilidad de ser aceptada con la esperanza de escapar de un mínimo local en la búsqueda del óptimo global. La probabilidad de aceptar una solución de inferior calidad depende del parámetro temperatura T y de la diferencia fitness f(s)−f(s′) entre ambas soluciones; típicamente T decrementa con cada iteración del algoritmo. Este algoritmo ha sido aplicado a una gran variedad de problemas de optimización combinatoria, tales como problemas de asignación cuadrática (QAP) [47], problemas de asignación de tareas [48], problema del

vendedor viajero, diseño de circuitos a gran escala de integración [49], [50], [51], entre otros.

3.4.2 Búsqueda Tabú (TS)

La búsqueda tabú o Tabú Search (TS, *por sus siglas en inglés*) es una de las metaheurísticas más citadas y usadas para resolver problemas de optimización combinatoria. La idea básica fue introducida por Glover [52], posteriormente describe el método de forma formal [53]. En cada iteración, TS explora el espacio de búsqueda al moverse desde la solución actual s a una vecina, donde el vecindario de la solución actual N(s) son transformaciones permitidas de la solución s realizadas por una heurística ad hoc especifica (también denominada operador de vecindario). Cuando se encuentra un óptimo local, la búsqueda se mueve a la mejor solución del vecindario, la cual no es necesariamente mejor que s. Para evitar ciclos, se mantiene una historia de la búsqueda en una memoria de corto plazo (lista tabú). Esto se realiza con el objetivo de no quedar atrapado en un óptimo local y explorar mejor el espacio de búsqueda.

Los movimientos que podrían llevar a la trayectoria de búsqueda a soluciones recientemente visitadas se denominan movimientos tabúes y no son permitidos, a menos que reúnan ciertas condiciones (criterio de aspiración). El criterio de aspiración más ampliamente usado es permitir soluciones cuyo fitness sea mejor que el de la mejor solución encontrada hasta el momento. La lista tabú, por lo general, almacena sólo una cantidad limitada y fija de información. Una opción rara vez usada es registrar soluciones completas, porque requiere una cantidad importante de almacenamiento y es costoso chequear si un movimiento potencial es tabú o no. La opción más usada consiste en registrar las últimas

transformaciones realizadas de la solución actual y prohibir transformaciones en reversa; otras están basadas en características claves de las soluciones o de los movimientos. Este algoritmo se ha utilizado para resolver el QAP [54], el problema MAXSAT [55], el problema de ruteo (VRP) [56] entre otros.

3.4.3 Búsqueda miope aleatorizado y adaptativo

El procedimiento de búsqueda miope aleatorizado y adaptativo o Greedy Randomized Adaptive Search Procedure (GRASP, por sus siglas en inglés) presenta un diseño relativamente simple [57], el cual combina una heurística ad hoc constructiva aleatoria-voraz con una búsqueda local. Un algoritmo GRASP es un procedimiento de multi-reinicio. La fase de construcción genera una solución factible sp, añadiendo paso a paso diferentes componentes c a la solución parcial sp, inicialmente vacía. El conjunto de elementos candidatos está formado por aquellos elementos c que se pueden agregar a sp en construcción sin alterar su factibilidad. Una función voraz mide el beneficio local de incluir un elemento candidato en la solución parcial sp. Los elementos candidatos con un valor de función greedy al menos tan bueno que un umbral especificado se agregan a la lista restringida de candidatos (RCL). El próximo elemento que incluir en la solución se selecciona de RCL en forma aleatoria. Esta inclusión en la solución altera la función greedy y el conjunto de elementos candidatos para determinar la próxima RCL. El proceso de construcción finaliza cuando el conjunto de elementos candidatos está vacío.

Luego se investiga su vecindario hasta hallar un mínimo local durante la fase de búsqueda local. La mejor solución es el resultado de la búsqueda. Esta metaheurísticas ha sido aplicada a una amplia variedad de problemas de

optimización como, por ejemplo: job shop scheduling [58], problemas de asignación , problema MAX-CUT [59], VRP [60], etc.

3.4.4 Búsqueda con vecindario variable

La búsqueda con vecindario variable o Variable Neighborhood Search (VNS, por sus siglas en inglés), propuesta por Hansen y colaboradores [61], consiste en cambiar de forma sistemática la estructura de entorno durante la búsqueda. En primer lugar, se debe definir la estructura del conjunto de vecindarios. La elección de la estructura se puede realizar de distintas maneras. Luego se genera una solución inicial s, se inicia el índice de vecindario k, y el algoritmo itera hasta que se alcance la condición de terminación. Cada iteración consiste en tres fases: la elección del candidato, la búsqueda local y el movimiento. En la primera fase, se elige aleatoriamente un vecino s´ de s usando el k−ésimo vecindario. En la segunda fase, esta solución s´ es el punto de partida de la búsqueda local. Cuando termina el proceso de mejora, se compara la nueva solución s´´ con la original s. Si es mejor, s´´ se convierte en la solución actual y se reinicializa el contador de vecindarios (k ←1); si no es mejor, se repite el proceso, pero utilizando el siguiente vecindario (k ← k+1). La búsqueda local es el paso de intensificación del método y el cambio de vecindario puede considerarse como el paso de diversificación. Con VNS se han abordado distintos problemas entre los que se encuentran los problemas de empaquetado [62], de localización [63], [64] y VRP [65] [66] [67].

3.4.5 Búsqueda Local Iterada (ILS)

La Búsqueda local Iterada o Iterated Local Search (ILS, *por* sus siglas en inglés) es una técnica potente, simple de implementar, robusta y altamente eficiente [68]. Propone un esquema en el que se incluye una heurística ad hoc base para mejorar los resultados de la repetición de dicha heurística. En cada iteración, se perturba la solución actual y, a esta nueva solución, se le aplica un método de búsqueda local para mejorarla. El mínimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa una prueba de aceptación. La importancia del proceso de perturbación es obvia: si es demasiado pequeño puede que el algoritmo no sea capaz de escapar del mínimo local; por otro lado, si es demasiado grande, la perturbación puede hacer que el algoritmo sea como un método de búsqueda local con un reinicio aleatorio. Por lo tanto, el método de perturbación debe generar una nueva solución que sirva como inicio a la búsqueda local, pero que no debe estar muy lejos de la actual para que no sea una solución aleatoria. El criterio de aceptación actúa como contra-equilibrio, ya que filtra la aceptación de nuevas soluciones dependiendo de la historia de búsqueda y de las características del nuevo mínimo local. La historia del proceso de búsqueda puede ser explotada en la forma de memoria de corto o largo término. Esta metaheurísticas ha sido usada para resolver el problema del viajante de comercio [69], entre otros.

3.5 Métodos Basados en Población

Los métodos basados en población trabajan en cada iteración con un conjunto de soluciones, usualmente denominado población. De esta forma, los algoritmos

basados en población proveen una forma natural e intrínseca de explorar el espacio de búsqueda.

3.5.1 Algoritmos Evolutivos (EAs)

Los algoritmos evolutivos o Evolutionary Algorithms (EAs) están inspirados en la capacidad natural de evolucionar de los seres vivos bien adaptados a su ambiente. Los algoritmos evolutivos son algoritmos estocásticos cuyos métodos de búsqueda modelan un fenómeno natural: la herencia genética y la rivalidad darwiniana para la supervivencia. Operan sobre una población de soluciones, denominadas individuos. Por lo general, la población inicialmente se genera en forma aleatoria, pero también se puede usar alguna heurística ad hoc de construcción. El esquema general de un EA está formado por tres pasos: seleccionar, alterar (reproducción) y reemplazar; los cuales se repiten hasta que se alcanza un punto de terminación, normalmente después de un número dado de iteraciones. Cada solución se evalúa para dar alguna medida de su aptitud (fitness).

Se seleccionan algunos miembros de la población (paso seleccionar) para someterlos a transformaciones (paso alterar) por medio de operadores genéticos para formar nuevas soluciones. Hay transformaciones de orden superior (tipo crossover), las cuales crean nuevos individuos al combinar varias partes de dos o más individuos y transformaciones unarias (tipo mutación), las cuales crean nuevos individuos al producir pequeños cambios en un único individuo. Luego, se forma una nueva población al seleccionar los individuos más idóneos (paso reemplazo) de la población actual y/o los mejores individuos generados, dando lugar a la siguiente generación del algoritmo. Tras algún número de generaciones, el algoritmo converge; es de esperar que el mejor individuo represente una solución cercana a

la óptima. Estos algoritmos establecen un equilibrio entre la explotación de buenas soluciones (fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (fase de reproducción), basados sobre el hecho que la política de reemplazo permite la aceptación de nuevas soluciones que no mejoran necesariamente las existentes. En la literatura se han propuesto diferentes algoritmos basados en este esquema general. Básicamente, estas propuestas se pueden clasificar en tres categorías que fueron desarrolladas de forma independiente. Estas categorías son: la programación evolutiva o Evolutionary Programming (EP) desarrollada por Fogel [70], las estrategias evolutivas o Evolution Strategies (ES) propuestas por Rechenberg en [71] y los algoritmos genéticos o Genetic Algorithms (GA) introducidos por Holland [72] [73].

3.5.2 Algoritmos de Estimación de Distribuciones (EDAs)

Los algoritmos de estimación de distribuciones o Estimation of Distribution Algorithms (EDAs) [74] fueron creados para reducir las desventajas de los operadores de recombinación de los EAs, los cuales tienden romper buenos bloques constructivos. Estos algoritmos tienen una fundamentación teórica en la teoría de probabilidad. Como los EAs, trabajan con una población que evoluciona a medida que progresa la búsqueda. Una vez generada la población inicial, se repite el siguiente ciclo hasta alcanzar el punto de parada. Se selecciona una parte de las mejores soluciones pertenecientes a la población actual, de las cuales se calcula una distribución de probabilidades del espacio de búsqueda. Esta distribución de probabilidades es la muestra para producir la población de la próxima iteración. El campo de EDAs es aún muy nuevo. Se pueden encontrar aplicaciones al problema

de la mochila, problema de planificación de tareas y otros problemas de optimización [75].

3.5.3 Optimización por Cúmulo de Partículas (PSO)

La optimización por cúmulo de partículas o Particle Swarm Optimization (PSO) [76] está inspirada en el comportamiento social del vuelo de aves o el movimiento de los bancos de peces. El algoritmo inicia con una población de soluciones aleatorias y busca el óptimo al actualizar generaciones. Las soluciones potenciales del problema, denominadas partículas vuelan a través del espacio del problema al seguir las partículas óptimas actuales. PSO tiene dos operadores primarios: actualización de velocidad y actualización de posiciones. Durante cada generación cada partícula es acelerada hacia la partícula mejor posicionada en la iteración previa y hacia la mejor posicionada global. En cada iteración se calcula un nuevo valor de velocidad para cada partícula en función de su velocidad actual, la distancia entre la posición actual y la mejor conocida por esa partícula (componente cognitivo), y distancia entre la posición actual y la mejor posición del vecindario (componente social). El valor de la nueva velocidad se usa para calcular la próxima posición de la partícula en el espacio de búsqueda. Este proceso es repetido una cantidad de veces o hasta que se logre un error mínimo. PSO se ha aplicado con éxito en diferentes campos de investigación. Algunos ejemplos son: optimización de funciones numéricas [76], entrenamiento de redes neuronales [77], problema del viajante de comercio , entre otros.

3.5.4 Búsqueda Dispersa (SS)

La búsqueda dispersa o Scatter Search (SS) [78], opera sobre un conjunto de referencia de soluciones (S_{ref}) formado por soluciones factibles al problema bajo

consideración. Este conjunto de referencia es generado e iterativamente actualizado, intentando intensificar y diversificar la búsqueda. Tras combinar las soluciones en S_{ref}, se aplica un procedimiento de búsqueda local para mejorar las soluciones resultantes. El S_{ref} se actualiza para incorporar tanto buenas soluciones como dispersas. Estos pasos se repiten hasta que se alcanza algún criterio de parada. SS ha despertado gran interés en los últimos años, entre otros problemas, ha sido aplicado a problemas de asignación multi-objetivo [79] y a problemas de ordenamiento lineal [80].

3.5.5 Optimización Basada en Colonias de Hormigas (ACO)

Los algoritmos de optimización basados en colonias de hormigas o Ant Colony Optimization (ACO) fue propuesta por Dorigo [81] [82]. Esta metaheurísticas emplea estrategias inspiradas en el comportamiento de las colonias de hormigas para descubrir fuentes de alimentación. Mientras que una hormiga realiza su camino va depositando una sustancia química, denominada feromona. Esta sustancia ayudará encontrar la comida al resto de las hormigas. Como resultado de la exploración y cooperación entre los individuos de la colonia se establece, en general, el camino más corto entre dicha fuente de alimentación y el hormiguero. Este comportamiento es el que intenta simular este método para resolver problemas de optimización. La técnica se basa en dos pasos principales: construcción de una solución basada en el comportamiento de una hormiga y actualización de los rastros de feromonas artificiales. La metaheurísticas ACO ha sido exitosamente aplicada a problemas de planificación [83], de asignación cuadrática [84], del viajante de comercio [85], etc.

3.6 Metaheurística usada en este trabajo

En la Sección 3.3 se realizó una descripción muy general de las distintas metaheurísticas. Por lo tanto, en esta sección se centrará en detallar más exhaustivamente las metaheurísticas que se han utilizado como mecanismo de búsqueda para aplicarlas al problema de empaquetamiento de contenedores de dos dimensiones aplicado a imprentas digitales. En este trabajo de investigación se utilizó una metaheurísticas basada en población: específicamente algoritmos genéticos. Es importante mencionar aquí que, debido a la complejidad del problema abordado, tanto la representación como los operadores genéticos utilizados dependen del problema que se está resolviendo, por lo que sólo se incluye el esquema general del algoritmo genético. Los detalles de implementación serán introducidos en los capítulos siguientes.

3.6.1 Algoritmos evolutivos

Alrededor de los años 60, varias corrientes de investigación comenzaron en forma independiente, formando lo que ahora se conoce como computación evolutiva, tomando su inspiración en la evolución de las especies. La idea era implementar algoritmos basados en el modelo de la evolución de las especies, denominados algoritmos evolutivos (EAs, *por sus siglas en inglés*), como un intento de resolver tareas complejas de optimización en computadoras. Hoy en día, debido a su robustez, a su amplia aplicabilidad y a la disponibilidad de cada vez mayor poder computacional, la computación evolutiva recibe una atención creciente por parte de los investigadores de un gran número de disciplinas. Los algoritmos evolutivos constituyen una técnica general de resolución de problemas de búsqueda y optimización inspirada en los procesos biológicos que se pueden apreciar en la

naturaleza, como la selección natural [86] y la herencia genética [87]. Estos algoritmos permiten abordar problemas complejos que surgen en las ingenierías y los campos científicos: problemas de planificación de tareas, horarios, tráfico aéreo y ferroviario, búsqueda de caminos óptimos, optimización de funciones, etc.

Los algoritmos evolutivos se pueden caracterizar como modelos computacionales del proceso evolutivo. En cada iteración, se aplica un cierto conjunto de operadores a los individuos de la población actual para generar los individuos de la población de la próxima generación (número determinado de iteraciones). Generalmente, los EAs usan operadores denominados recombinación o cruce para recombinar dos o más individuos para producir nuevos individuos. Además, usan un operador de mutación, es decir, modificaciones que causan una "alteración" de los individuos. La fuerza motriz de los algoritmos evolutivos es la selección de los individuos sobre la base de su aptitud (que puede basarse en la función objetivo, el resultado de un experimento de simulación o algún otro tipo de medida de calidad). Los individuos con una mayor aptitud tienen mayor probabilidad de ser elegidos miembros de la población de la siguiente iteración (o como padres para la generación de nuevos individuos). Esto corresponde al principio de la supervivencia del más apto en la evolución natural. Es la capacidad de la naturaleza para adaptarse a un entorno cambiante lo que dio la inspiración para los algoritmos evolutivos.

Se analiza a continuación en forma detalla el funcionamiento de un EA, cuyo pseudocódigo se muestra en el Algoritmo 3-1[7]. Los algoritmos evolutivos mantienen una población de individuos, $\boldsymbol{P(t) = x^t_1, x^t_2, \ldots x^t_\mu}$ en la iteración $\boldsymbol{t}$, donde $\boldsymbol{\mu}$ es el tamaño de la población. Cada individuo representa una solución potencial al

problema y se implementa como una estructura de datos ***S***. En cada iteración del algoritmo se selecciona un conjunto de individuos (paso *seleccionar Padres*) a los que se denomina padres. A estos miembros se les aplica ciertas transformaciones (paso *alterar*) por medio de operadores "genéticos" para formar nuevas soluciones ***λ***, denominadas hijos. Hay transformaciones de orden superior c_j (tipo cruzamiento), las cuales crean nuevos individuos al combinar varias partes de dos o más individuos ($m_i: Sx \cdots xS \rightarrow S$) y transformaciones unarias mi (tipo mutación), las cuales crean nuevos individuos al producir pequeños cambios en un único individuo ($m_i: S \rightarrow S$). Eiben y colaboradores han investigado los méritos del uso de más de dos padres en el proceso de reproducción [88]. Cada solución x^t_i se evalúa (paso *evaluar*) para dar alguna medida de su aptitud o "fitness". Entonces, se forma una nueva población con un número predeterminado de iteraciones. Se seleccionan los individuos más idóneos (paso *seleccionar Nueva Población*). Tras una cierta cantidad de generaciones, el algoritmo converge; es de esperar que el mejor individuo represente una solución cercana a la óptima.

Algoritmo 3-1 Pseudocódigo de un EA

```
1: t←0;
2: P(t) ←generarPoblaciónInicial();
3: evaluar(P(t));
4: while not(condición de terminación) do
5:  P´(t) ← seleccionarPadres(P(t));
6:  P´(t) ← alterar P´(t);
7: evaluar(P(t));
8:  P´(t) ← seleccionarNuevaPoblación(P(t), P´(t));
9:  t ← t + 1;
10: end while
11: return la mejor solución encontrada
```

Hay varias variantes de algoritmos evolutivos, las cuales incluyen:

- **Programación evolutiva:** se hace evolucionar una población de máquinas de estados finitos sometiéndolas a transformaciones unitarias.
- **Estrategias evolutivas:** se hace evolucionar una población de estructuras compuestas por un vector real y una variable aleatoria (parámetro) que codifican las posibles soluciones de un problema numérico y las dimensiones de los cambios o saltos. La selección es implícita.
- **Programación genética:** se hace evolucionar una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.
- **Algoritmos genéticos:** se hace evolucionar una población de individuos sometiéndolos a transformaciones genéticas y a un proceso de selección.

A pesar de las similitudes entre los algoritmos evolutivos, hay también muchas diferencias entre ellos (a menudo ocultas a un bajo nivel de abstracción).

Frecuentemente usan estructuras de datos ***S*** diferentes para la representación de los cromosomas, en consecuencia, los operadores genéticos también son diferentes. Pueden incorporar o no alguna otra información (para controlar el proceso de búsqueda) en sus genes.

Hay muchos métodos para seleccionar individuos para supervivencia y reproducción. Esos métodos incluyen:

- **Selección proporcional:** la probabilidad de selección es proporcional al fitness del individuo.

- **Métodos de ranking:** todos los individuos de la población se ordenan de mejor a peor y las probabilidades de selección son fijas durante todo el proceso de evolución.
- **Selección por torneo:** algunos individuos (usualmente dos) compiten por ser seleccionados para la nueva generación; este paso de competencia (torneo) se repite $\boldsymbol{\mu}$ veces.

La selección proporcional puede necesitar el uso de métodos de truncamiento o de escalamiento. Hay diferentes formas para asignar probabilidades en métodos de ranking (distribuciones lineales o no lineales). El tamaño de un torneo juega un rol significativo en el método de selección por torneo. Es importante determinar las políticas generacionales, es decir, especificar la forma en la cual quedará conformada la próxima generación. Por ejemplo, es posible reemplazar la población entera de tamaño $\boldsymbol{\mu}$ por la población de $\boldsymbol{\lambda}$ hijos, o seleccionar los mejores individuos de las dos poblaciones (padres e hijos), esta selección se puede hacer de una manera determinística o no determinística. Es también posible producir pocos hijos (por lo general $\boldsymbol{\lambda = 1 \text{ o } \lambda = 2}$), con los cuales reemplazar algunos individuos, consecuentemente coexisten con sus padres. Sistemas basados en tales políticas son llamados de estado estacionario. En un modelo generacional, el conjunto de los padres se mantiene fijo mientras se generan todos los hijos que formarán parte de la próxima generación. También, se puede usar un modelo elitista, el cual pasa el mejor individuo de una generación a otra. Esto significa que, si el mejor individuo de la generación actual corre el riesgo de perderse, ya sea por la selección o los operadores genéticos, el sistema fuerza a que quede en la próxima generación; tal modelo es muy útil para resolver muchas clases de problemas de optimización.

Existe un modelo intermedio (***1 < λ < μ***) en el cual se determina un porcentaje de la población para participar en la creación de la nueva generación. El modelo se denomina de gap generacional. El gap es el porcentaje de la población que participa en la aplicación de los operadores evolutivos. La propuesta constituye una generalización de los modelos de evolución, que incluye los dos enfoques opuestos ya presentados (modelo generacional y de estado estacionario).

Las representaciones de las soluciones potenciales para un problema particular, junto con el conjunto de operadores genéticos, constituyen los componentes esenciales de cualquier algoritmo evolutivo. Esos son los elementos claves que permiten distinguir varios paradigmas de métodos evolutivos. Una de las principales dificultades de los algoritmos evolutivos (en particular cuando se aplica búsqueda local) es la convergencia prematura hacia soluciones sub-óptimas. El mecanismo más simple para diversificar el proceso de búsqueda es el uso de un operador de mutación aleatorio. Con el fin de evitar la convergencia prematura, existen otras formas de mantener la diversidad de la población. Probablemente, la más antigua es crowding [89]. Estrategias más recientes son fitness sharing [90] y niching [91] en el cual la aptitud para la reproducción asignada a un individuo en una población se reduce proporcionalmente al número de otros individuos que comparten la misma región del espacio de búsqueda. Los algoritmos evolutivos basan parte de sus buenos resultados en el equilibrio entre una eficiente exploración y explotación cuando se resuelve un problema difícil. La explotación consiste en utilizar el conocimiento adquirido por la exploración para llegar a mejores posiciones en el espacio de búsqueda. Por otra parte, la exploración es investigar las zonas nuevas y desconocidas del espacio de búsqueda. Por consiguiente, tanto la

explotación y exploración deben ser debidamente controladas durante el proceso de búsqueda genética a fin de lograr la convergencia a la solución óptima. La exploración es útil para evitar estancarse en óptimos locales mientras que la explotación se usa para obtener el óptimo global una vez que se ha aproximado a él lo suficiente.

3.6.2 Algoritmos genéticos

Una de las ramas más desarrolladas dentro de algoritmos evolutivos son los algoritmos genéticos (GAs, por sus siglas en inglés). Los Gas fueron inventados por John Holland (ref72) e introducidos por Goldberg (ref96 y 97). Estos algoritmos, son un tipo de algoritmo evolutivo pensados inicialmente para trabajar con soluciones representadas mediante cadenas binarias, denominadas cromosomas. No obstante, a lo largo de los años, se han usado otras representaciones no binarias, como permutaciones [92], vectores de enteros [93],números reales [94], entre otros.

La principal característica de los GAs es el uso de un operador de recombinación o cruce como mecanismo principal de búsqueda: construye descendientes que poseen características de los cromosomas que se cruzan. Su utilidad viene dada por la suposición de que diferentes partes de la solución óptima pueden ser descubiertas independientemente y luego ser combinadas para formar mejores soluciones. Adicionalmente, se emplea un operador de mutación cuyo uso se considera importante como responsable del mantenimiento de la diversidad. El principal problema de los GAs se debe a que no usan información sobre el problema que están intentando resolver.

Este trabajo de investigación se centra en la implementación de un algoritmo genético (Algoritmo 3-2) para resolver el problema de empaquetamiento de

contenedores de dos dimensiones, aplicado a la optimización de papel en imprentas digitales. Esta decisión tiene que ver con una característica fundamental de este tipo de algoritmos que los hacen más adecuados que los GAs generacionales. Es bien conocido dentro del campo de los GAs que la selección por estado estacionario permite al algoritmo converger más rápidamente que el esquema generacional [95].

De hecho, se ha demostrado que el algoritmo genético de estado estacionario (ssGA, *de sus siglas en inglés*) evoluciona con la misma dinámica que un GA generacional, utilizando la mitad del costo computacional [95]. Aunque, esto tiene como inconveniente que la exploración sea pobre y no siempre de como resultado la solución óptima.

Sin embargo, como se está abordando problemas del mundo real que conllevan tiempos de evaluación muy altos para la función de fitness, este esquema de selección permitirá reducir considerablemente los tiempos de cómputo. Los algoritmos genéticos tienen varias ventajas en comparación con otros métodos. Estos algoritmos se pueden aplicar a una amplia área de problemas, obtienen buenos resultados en problemas del mundo real, tienen un gran potencial para combinarse con otros métodos de búsqueda y optimización, pueden adecuarse fácilmente al ambiente paralelo y distribuido, además sus operadores pueden adaptarse para cierto tipo de problemas [98].

El criterio de reemplazo en un ssGA cobra una mayor importancia que en un algoritmo evolutivo generacional. El criterio de reemplazo puede ser aleatorio, proporcional al fitness o elitista. Normalmente se trabaja en hipótesis de competencia entre el descendiente generado y el individuo a reemplazar. Este mecanismo de reproducción paso a paso fue inicialmente propuesto como un modo

de evitar inconvenientes en la resolución de problemas de entrenamiento de redes neuronales. Luego se extendió su uso para una variedad de problemas y áreas de aplicación.

El modelo de evolución de estado estacionario consiste en una interesante propuesta para mantener el equilibrio entre los mecanismos de exploración y de explotación del algoritmo evolutivo. El algoritmo evolutivo mantiene las características poblacionales (garantizando la exploración) y a la vez realiza una explotación local del tipo hill-climbing que presiona hacia los individuos mejor adaptados. Las técnicas de selección elitistas (elitismo propiamente dicho o selección por ranking) son usualmente utilizadas en los algoritmos evolutivos de estado estacionario. La estrategia de reemplazar al peor individuo usualmente conduce a convergencia prematura. La estrategia de reemplazo aleatorio funciona correctamente en general, aunque la convergencia es lenta. Para disminuir los efectos adversos puede utilizarse reemplazo probabilístico, inversamente proporcional a los valores de fitness. La mayor diferencia entre el modelo de estado estacionario y el modelo generacional consiste en que para cada x miembros de la población creados, el modelo de estado estacionario necesita realizar 2x selecciones. Como consecuencia, la presión de selección y la “deriva genética” (pérdida de información) será el doble en un algoritmo evolutivo de estado estacionario. El algoritmo de estado estacionario será como mínimo mucho más veloz para converger que el algoritmo generacional [95]. El GA de estado estacionario evoluciona con la misma dinámica que un GA generacional utilizando la mitad del coste computacional [95].

Algoritmo 3-2 Pseudocódigo de un GA

```
1: t←0;
2: P(t) ←generarPoblaciónInicial();
3: evaluar(P(t));
4: while not(condición de terminación) do
5:  padres ← seleccionarPadres(P(t));
6:  hijo ← recombinar(padres);
7:  hijo ← mutar(hijo);
8:  evaluar(hijo);
9:  P´(t) ← seleccionarNuevaPoblación(P(t), hijo);
10:  t ← t + 1;
11: end while
12: return la mejor solución encontrada
```

Componentes del algoritmo genético

Los algoritmos genéticos se componen de varios elementos como población inicial, evaluación de población, operador de selección, cruzamiento y mutación, estos componentes hacen que el algoritmo tenga un buen funcionamiento, además de que se pueden adaptar a diferentes problemas de optimización combinatoria [99][100].

3.7 Conclusiones Parciales del Capítulo 3

En este capítulo se realizó una introducción al campo de las metaheurísticas. Inicialmente se comenzó presentado la definición de problemas de optimización, luego se abordó todo lo relacionado a estructura de vecindad y por último se presentó el concepto y las propiedades de las metaheurísticas. A continuación, se ofreció un repaso por las técnicas más importantes dentro de este campo, las cuales han sido clasificadas en dos categorías: metaheurísticas basadas en trayectoria y en población, en función del número de soluciones tentativas con las que trabajan

en cada iteración. Finalmente, se detallaron los algoritmos metaheurísticos utilizados en este trabajo de investigación. Detalles de su adecuación para tratar al problema en estudio serán presentados en los Capítulos 4 y 5.

Capítulo 4 Metodología de solución

En esta sección se describe la metodología de solución propuesta para resolver el Problema de Empaquetamiento de Contenedores de dos Dimensiones aplicado a Imprentas Digitales con la finalidad de encontrar buenas soluciones y el diseño de una maqueta que represente de forma simple una solución al problema que se está analizando en esta investigación. Se consideran los siguientes pasos para la metodología de solución.

1. **Modelo matemático.** Se propone una función objetivo y un conjunto de restricciones para solucionar el Problema de Empaquetamiento de Contenedores de dos Dimensiones Aplicado a Imprentas (ver capítulo 2 sección 2.5).
2. **Estrategia de solución.** De acuerdo con el análisis realizado se plantea una estrategia para solucionar el problema que se está analizando. Como primer punto se van a posicionar las figuras de forma aleatoria en la hoja de papel, cada figura tiene un valor de forma aleatoria de x, un valor de forma aleatoria de y, un ángulo aleatorio de rotación, este último es entre 0^0-359^0. Tanto los valores aleatorios de ***x*** y ***y*** son valores que no deben de exceder las dimensiones de la hoja de papel. Cada vez que se inserta una nueva figura se calculan las distancia entre esta nueva figura y las que ya están en la hoja de papel, si estas distancias exceden un valor determinado entonces se verifica el traslape entre figuras. Si una figura se traslapa con la otra entonces se vuelven a generar los datos aleatorios anteriormente mencionados. Todo este proceso se hace un número finito de veces con el fin de obtener una solución no optimizada del problema.

3. **Algoritmo genético (AG-PEC2DAOPID).** Se diseña y desarrolla un algoritmo genético que resuelva el Problema de Empaquetamiento de Contenedores de Dos Dimensiones Aplicado a Imprentas Digitales con el fin de disminuir la merma del papel. La configuración resultante es obtenida por el algoritmo genético.
4. **Sintonización de parámetros.** Se realiza el análisis de sensibilidad de los parámetros de control del algoritmo AG-PEC2DAOPID, con el objetivo de encontrar los mejores valores para cada parámetro del algoritmo que permitan mejorar su desempeño. La sintonización de los parámetros se muestra en el capítulo 5.
5. **Pruebas Experimentales.** Se realizan las pruebas experimentales utilizando los parámetros de control sintonizados. Por cada prueba se realizan 30 ejecuciones para obtener un buen análisis estadístico del desempeño del algoritmo genético. Las pruebas experimentales se presentan en el capítulo 5.

4.1 Estrategia de solución

De acuerdo con el análisis realizado se plantea una estrategia para solucionar el problema que se está analizando. Como primer punto se van a posicionar las figuras de forma aleatoria en la hoja de papel. Cada figura tiene un valor de forma aleatoria de x, un valor de forma aleatoria de y, un ángulo aleatorio de rotación, este último es entre 0^0-359^0. Tanto los valores aleatorios de ***x*** y ***y*** son valores que no deben de exceder las dimensiones de la hoja de papel. Cada vez que se inserta una nueva

figura se calculan las distancia entre esta nueva figura y las que ya están en la hoja de papel, si estas distancias exceden un valor determinado entonces se verifica el traslape entre figuras. Si una figura se traslapa con la otra entonces se vuelven a generar los datos aleatorios anteriormente mencionados. Todo este proceso se hace un número finito de veces con el fin de obtener una solución no optimizada del problema. Luego se genera una población inicial aleatoria y por último se ejecuta la segunda parte del Algoritmo Genético secuencial.

4.2 Representación de los individuos

Una solución para el Problema de Empaquetamiento de Contenedores de Dos Dimensiones se representa a partir de un conjunto de figuras de varios tipos según su geometría representada como un individuo. Los individuos son potenciales soluciones del problema que pueden ser expresados como un conjunto de características (llamadas genes), que a su vez constituyen una serie de valores que se denominan cromosoma. Es esencial contar con una forma de representar estos genes para su uso en etapas posteriores del algoritmo genético y asignarles valores correspondientes. Un cromosoma es un conjunto de genes. Un gen representa un valor (binario, entero, real, etc) de la solución.

En este trabajo de investigación un individuo se puede representar como se muestra en la Figura 4.1. Este individuo tiene 7 parámetros. El parámetro ***x*** y ***y*** son las coordenadas que se generan de forma aleatoria, ***w*** y ***h*** son las dimensiones de las imágenes que contienen las figuras, ***img*** representa la máscara que está formada por ceros y unos y por último el parámetro **Angle** es el ángulo de rotación de cada

figura. Este individuo que se genera de forma aleatoria siempre es factible. Para que un individuo sea factible no debe existir traslape entre los objetos contenidos en la imagen, como se muestra en la Figura 4.2. Es importante remarcar, que en la Figura 4.1 cada columna representa una Figura del individuo (Figura 4.2).

x	1108	104	297	831	30	868	637	875	727	1070
y	609	600	147	582	401	399	128	177	13	197
w	150	150	150	150	150	150	150	150	150	150
h	150	150	150	150	150	150	150	150	150	150
img	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38
mask	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}	0x08074630 {w= 150 h= 150 bits= 0x080abff8{0}}
Angle	0	0	0	0	0	0	0	0	0	0

Figura 4.1. *Representación de la estructura de un individuo.*

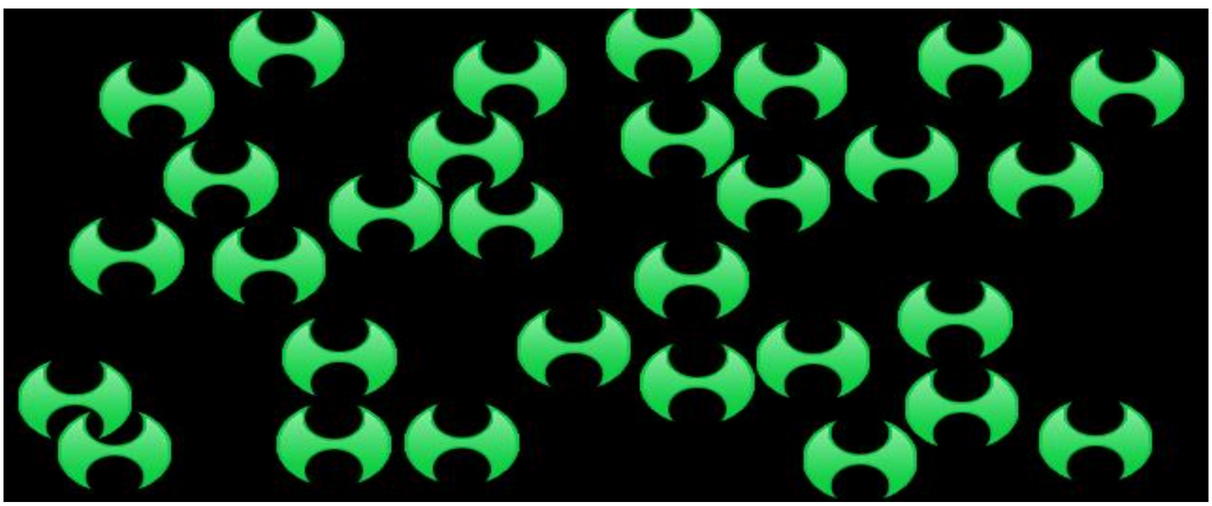

Figura 4.2. *Representación en forma gráfica de un individuo.*

4.3 Generación de la población inicial

La población inicial se puede generar mediante alguna técnica heurística o aleatoriamente, en este trabajo se genera de forma aleatoria. Los individuos de la población son evaluados para verificar su factibilidad. ¿Cómo saber si una solución es factible?, si cumple con las restricciones de que no exista traslapes entre las figuras que se están posicionando en la hoja de papel, ¿Por qué trabajar con una

población de individuos factibles?, porque se ha comprobado que cuando se trabaja con soluciones factibles para este tipo de problemas el comportamiento del algoritmo mejora considerablemente [101].

La Figura 4.3. Muestra el algoritmo que genera la población inicial de soluciones factibles.

- Primeramente, se carga la instancia (*cargar figuras*) del problema, se inicializan las variables P_{ini} y N_{ind}.
- Luego se genera un individuo de forma aleatoria, donde se genera cada coordenada de forma aleatoria y se verifican los traslapes entre cada una de las figuras que conforman al individuo.
- Posteriormente se incrementa el contador de individuos factibles. El mismo procedimiento se genera hasta que la población inicial P_{ini} sea completada.

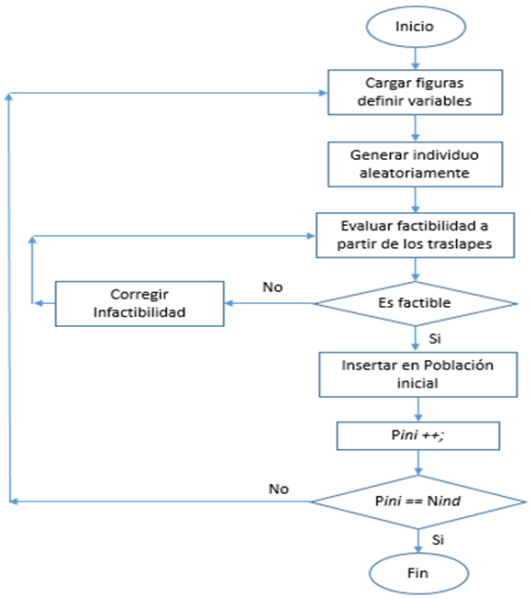

Figura 4.3. *Diagrama de flujo del algoritmo para generar la población inicial.*

4.4 Evaluación de la factibilidad de la población

La evaluación de la población es muy importante porque determina si un individuo cumple con las características adecuadas para formar parte de la población. Un individuo es factible cuando no existe traslapes entre las figuras. Un individuo para que sea factible no puede tener figuras que estén traslapadas.

4.5 Evaluación de población

El valor de aptitud se calcula a partir de la resta de área de la hoja de papel y la suma de las áreas de cada una de las figuras. La función objetivo es minimizar el valor de aptitud que se genera cuando se inserta cada figura. La función objetivo está sujeta a un conjunto de restricciones que se deben cumplir al contemplar cualquiera de los componentes. Estas restricciones toman en cuenta que cada figura esté posicionada dentro de la hoja de papel, es decir las coordenadas se generan dentro de las dimensiones de la hoja de papel, más específicamente dentro de la pantalla negra que se mostraron en las figuras en Figura 4.2. También se tiene en cuenta que no existan traslapes entre las figuras posicionadas en la hoja de papel.

Los algoritmos genéticos trabajan con tres operadores básicos como: selección, cruzamiento y mutación. La implementación de cada uno de ellos para este trabajo se describe a continuación.

4.6 Selección

El método de selección utilizado en este algoritmo es por torneo, ya que en diferentes problemas de optimización ha dado buenos resultados [102] [103]. El método de selección por torneo consiste en elegir un subconjunto de n individuos de una población, donde n puede tomar valores de dos o más individuos [104]

[105][106]. Es un método eficiente y fácil de implementar, con complejidad computacional del algoritmo de O(n).

Esta técnica presenta como ventaja respecto a otras técnicas de selección su eficiencia computacional ya que no requiere escalamiento de la función de aptitud (usa comparaciones directas) por lo cual no requiere la ordenación de los individuos para poder seleccionarlos con base en su aptitud. Además, este método puede implementarse en paralelo [104].

Este método selecciona un conjunto de n individuos de forma aleatoria, posteriormente se comparan de acuerdo con su valor de aptitud, el más apto es elegido como un individuo padre. Se realiza el mismo procedimiento para seleccionar al individuo madre. Comúnmente, la muestra de selección por torneo es n = 2 [99][106].

El Algoritmo 4-1 describe el proceso del operador de selección por torneo. Se generan dos individuos de forma aleatoria (línea 8 y 11). Y con un bucle *while* se verifica que ambos siempre sean diferentes para posteriormente aplicar el operador de cruzamiento.

Algoritmo 4-1. Pseudocódigo del operador de selección por torneo.

```
1: Entrada: indice_padre1, indice_padre2, Shapes hijo [GEN_NUMSHAPES].
2: for i=0 to Generaciones do
3: begin
4:     for j = 0 to Tam_Torneo do
5:     begin
4:         printf("Calcula la probabilidad de cruzamiento")
5:           prob = ((double)rand()/((double)RAND_MAX + 1)
6:           if (prob < PROB_CRUZ)
7            begin
8:               printf("Escoge dos padres")
9:               indice_padre1 = rand()%POP_SIZE
10:              do
10:                begin
11:                    indice_padre2 = rand()%POP_SIZE
```

```
12:                 end
13              while(indice_padre1 == indice_padre2)
14:         end
15:     end
16: end
```

4.7 Cruzamiento

Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación. Él operador de cruzamiento consiste en intercambiar características de dos padres para generar dos descendientes. En este trabajo se propuso un operador de cruzamiento de un punto. Tal y como se ha indicado anteriormente, el cruce es una estrategia de reproducción sexual.

Su importancia para la transición entre generaciones es elevada puesto que las tasas de cruce con las que se suele trabajar rondan el 90%. Los diferentes métodos de cruce podrán operar de dos formas diferentes. Si se opta por una estrategia destructiva los descendientes se insertarán en la población temporal, aunque sus padres tengan mejor ajuste (trabajando con una única población esta comparación se realizará con los individuos a reemplazar). Por el contrario, utilizando una estrategia no destructiva la descendencia pasará a la siguiente generación únicamente si superan la bondad del ajuste de los padres (o de los individuos a reemplazar). La idea principal del cruce se basa en que, si se toman dos individuos correctamente adaptados al medio y se obtiene una descendencia que comparta genes de ambos, existe la posibilidad de que los genes heredados

sean precisamente los causantes de la bondad de los padres. Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, debería tener una bondad mayor que cada uno de los padres por separado. Si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene un peor ajuste que los padres no significan que se esté dando un paso atrás. Optando por una estrategia de cruce no destructiva garantizamos que pasen a la siguiente generación los mejores individuos. Si, aún con un ajuste peor, se opta por insertar a la descendencia, y puesto que los genes de los padres continuarán en la población, aunque dispersos y posiblemente levemente modificados por la mutación, en posteriores cruces se podrán volver a obtener estos padres, recuperando así la bondad previamente pérdida. Existen multitud de algoritmos de cruce. Sin embargo, para este trabajo de investigación se empleó el cruzamiento de un punto de cruce. Este tipo de cruzamiento es la técnica más sencilla de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes (Figura 4.4). De esta manera ambos descendientes heredan información genética de los padres.

Padre 1:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	1021	1063	40	611	920	841	819	14	731	297	976	461	473	212	327
y	168	21	322	117	33	195	314	98	62	188	442	157	414	65	333
w	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
h	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
img	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38
mask	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}
Angle	23	32	56	51	3	4	8	77	96	101	39	42	12	92	9

Padre 2:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	807	194	913	1079	196	619	835	697	750	18	1054	455	398	520	94
y	99	277	452	386	21	166	217	291	436	300	41	223	472	57	97
w	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
h	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
img	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38
mask	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}
Angle	62	65	28	351	144	235	301	43	107	101	33	220	107	89	99

Hijo 1:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	1021	1063	40	611	920	841	835	697	750	18	1054	455	398	520	94
y	168	21	322	117	33	195	217	291	436	300	41	223	472	57	97
w	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
h	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
img	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38
mask	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}
Angle	23	32	56	51	3	4	301	43	107	101	33	220	107	89	99

Hijo 2:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x	807	194	913	1079	196	619	819	14	731	297	976	461	473	212	327
y	99	277	452	386	21	166	314	98	62	188	442	157	414	65	333
w	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
h	150	150	150	150	150	150	150	150	150	150	150	150	150	150	150
img	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38	0x0807ae38
mask	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}	0x08074630 {w=150 h=150 bits=0x080abff8{0}}
Angle	62	65	28	351	144	235	8	77	96	101	39	42	12	92	9

Figura 4.4. *Operador de cruzamiento de un punto de cruce.*

Como se puede observar en el ejemplo de la Figura 4.4, se tienen dos individuos, en este ejemplo se quiso simular como es el proceso del cruce entre ellos. Se puede observar que ambos padres tienen el parámetro ***x*** y ***y***, ***w*** y ***h***, ***img***, ***mask*** y ***Angle***. Los parámetros ***x*** y ***y*** responden a las coordenadas ubicadas en la hoja de papel, los parámetros de ***w*** y ***h***, es el largo y el ancho de la imagen que por defecto se está trabajando con el mismo largo y ancho de la imagen, ambos son de ***150x150 pixeles***. El parámetro ***mask*** responde a la máscara que se crea a partir de cargar el parámetro ***img, la mask,*** es decir la máscara también es de ***150x150***

pixeles. El último parámetro es ***Angle, que como su nombre lo indica es el ángulo de rotación de la imagen.*** Se debe de aclarar que las rotaciones de la imagen solo se hacen en el mismo sentido de las manecillas del reloj y que su eje de rotación es desde el centro de la imagen ***img.***

En la Figura 4.4, se puede observar que ambos padres tienen la misma cantidad de elementos y estos elementos son las figuras posicionadas que van desde cero a N-elementos, donde en este caso N=14. El punto de cruce es aleatorio, en la Figura 4.4 se utilizó la posición 5.

El Algoritmo 4-2. Es el pseudocódigo del Algoritmo de Cruzamiento utilizado en la implementación del Algoritmo Genético de esta investigación.

Algoritmo 4-2. Pseudocódigo del operador de cruzamiento por un punto de cruce.

```
1: Entrada: indice_padre1, indice_padre2, Shapes hijo [GEN_NUMSHAPES].
2:  punto = rand()%GEN_NUMSHAPES
3: for i=0 to punto do
4:     begin
5:         hijo[i].x = Pob[indice_padre1].shapes_Individuos[i].x;
6:         hijo[i].y = Pob[indice_padre1].shapes_Individuos[i].y;
7:         hijo[i].w = Pob[indice_padre1].shapes_Individuos[i].w;
8:         hijo[i].h = Pob[indice_padre1].shapes_Individuos[i].h;
9:         hijo[i].angle = Pob[indice_padre1].shapes_Individuos[i].angle;
10:        hijo[i].image = Pob[indice_padre1].shapes_Individuos[i].image;
11:        hijo[i].mask = Pob [indice_padre1].shapes_Individuos[i]. mask;
10:    end
12: for i=punto + 1 to GEN_NUMSHAPES do
13:     begin
14:         hijo[i].x = Pob[indice_padre2].shapes_Individuos[i].x;
15:         hijo[i].y = Pob[indice_padre2].shapes_Individuos[i].y;
16:         hijo[i].w = Pob[indice_padre2].shapes_Individuos[i].w;
17:         hijo[i].h = Pob[indice_padre2].shapes_Individuos[i].h;
18:         hijo[i].angle = Pob[indice_padre2].shapes_Individuos[i].angle;
19:         hijo[i].image = Pob[indice_padre2].shapes_Individuos[i].image;
20:         hijo[i].mask = Pob [indice_padre2].shapes_Individuos[i]. mask;
21:     end
```

Una vez cruzados los padres se obtienen los hijos, pero con estos hijos obtenidos se generan soluciones no factibles, entonces con el Algoritmo 4-3. *Reparación del individuo,* se reparan los individuos obtenidos.

Algoritmo 4-3. Pseudocódigo del Algoritmo de Reparación de los Individuos.

```
1: Entrada: Shapes hijo [GEN_NUMSHAPES], Shapes hijo_reparado [GEN_NUMSHAPES], SHAPES, i.
2: LABEL = 0
3: ALLEGRO_BITMAP F1 = al_load_bitmap("1.png")
4: ALLEGRO_BITMAP F2 = al_load_bitmap("2.png")
5: ALLEGRO_BITMAP F3 = al_load_bitmap("3.png")
6: ALLEGRO_BITMAP F4 = al_load_bitmap("4.png")
7: hijo_reparado [0].image = F1
8: hijo_reparado [1].image = F2
9: hijo_reparado [2].image = F3
10: hijo_reparado [3].image = F4
11: for i = 0 to 4
12:   begin
13:     hijo_reparado[i].mask = Mask_New(hijo_reparado[i].image)
14:   end
15: for i = 0 to GEN_NUMSHAPES
16:   begin
17:     bool overlap = false
18:     do
19:       begin
20:         hijo[SHAPES] = clone()
21:         LABEL = rand()%4
22:         hijo[SHAPES] = hijo_reparado[LABEL]
23:         hijo[SHAPES].w = al_get_bitmap_width(hijo_reparado[LABEL].image)
24:         hijo[SHAPES].h = al_get_bitmap_height(hijo_reparado[LABEL].image)
25:         hijo[SHAPES].x = getNewPositionX(hijo_reparado[LABEL].w)
26:         hijo[SHAPES].y = getNewPositionY(hijo_reparado[LABEL].h)
27:         hijo[SHAPES].angle= getAngle()
27:         hijo[SHAPES].image = hijo_reparado[LABEL].image
28:         hijo[SHAPES].mask = hijo_reparado[LABEL].mask
29:         for j = 0 to SHAPES
30:         begin
31:           if (Mask_Collide(hijo[SHAPES].mask, hijo[j].mask, hijo[SHAPES].x - hijo[j].x, hijo[SHAPES].y - hijo[j].y))
32:             begin
33:               overlap = true
34:               break
35:             end
36:           overlap = false
37:         end
38:   end while(overlap)
39: end
```

4.8 Mutación

El operador de mutación propuesto consiste en hacer una pequeña alteración aleatoria de los genes a un porcentaje de individuos Pm. Para este trabajo de investigación, la mutación que se implementó es la Estructura Híbrida de Vecindad con distintos tipos de movimientos bidimensionales (Inserción, Traslación, Rotación e Intercambio de figuras).

4.9 Estructura Híbrida de Vecindad

El algoritmo genético tiene embebida una estructura híbrida de vecindad. Esta estructura híbrida de vecindad consta de 4 movimientos importantes. Estos movimientos son las perturbaciones que se le aplican a las soluciones para su respectiva mejora. El primero movimiento es la inserción, este permite insertar una figura a la hoja de papel. El segundo movimiento es la traslación, la traslación funciona cuando cambiamos de coordenadas una figura y aquí la figura puede moverse en cualquier sentido, teniendo en cuenta la coordenada. El tercer movimiento es la rotación y este movimiento es el más importante porque permite la rotación de la figura entre los ángulos de 0 a 360 grados con el fin de ocupar el mayor espacio posible con los tipos de figuras. El cuarto movimiento es el intercambio, este movimiento es similar a la traslación, solo que acá se toma una figura y esa figura se cambia por la otra y una vez que se cambia, se puede cambiar su coordenada, todos estos movimientos se hacen teniendo en cuenta que no haya traslapes entre las figuras porque si no la solución es una solución no factible. En la

Figura 4.5 se puede observar el diagrama de flujo de la Estructura Híbrida de Vecindad.

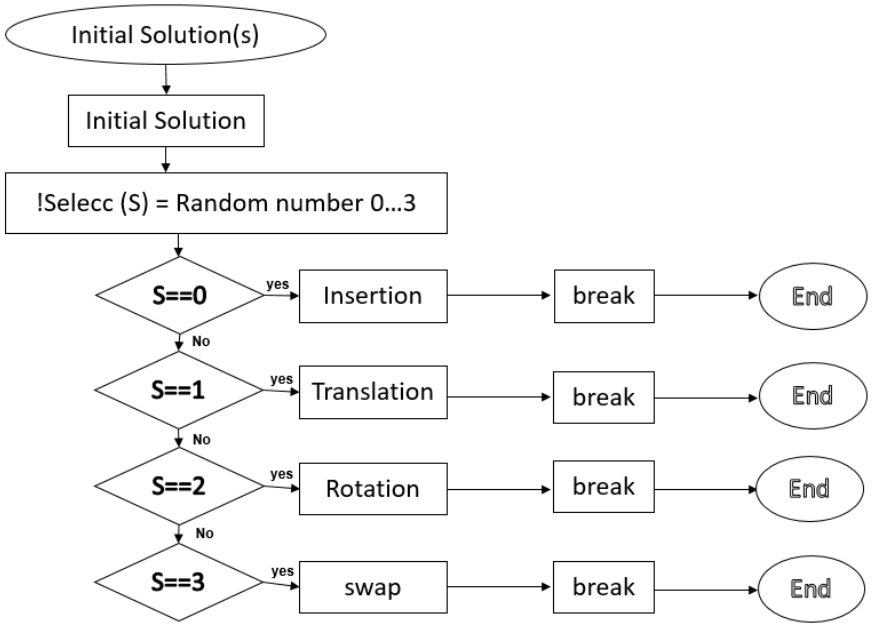

Figura 4.5. *Diagrama de Flujo de la Estructura Híbrida de Vecindad.*

El algoritmo *Mask_New_Rotates* (Algoritmo 4-4) es el algoritmo encargado de la rotación de la figura a partir de su máscara. Lo primero que se hace es cargar la figura y una vez cargada, esta se convierte en una máscara de ceros y unos, que no es más que una matriz de ceros y unos. Luego se aplica el clásico algoritmo de la matriz de rotación para rotar la matriz de ceros y uno. Por último, una vez que se tiene la matriz de ceros y uno rotada se vuelve a pintar la figura a partir de la máscara rotada y es así como queda la figura rotada.

Algoritmo 4-4 Pseudocódigo del Algoritmo Mask_New_Rotates

```
1: function mask_t *Mask_New_Rotates (ALLEGRO_BITMAP *bmp, float theta)
2:    mask_t *temp
3:    h <- al_get_bitmap_height(bmp)
4:    w <- al_get_bitmap_width(bmp)
5:    ALLEGRO_COLOR transColor <- al_map_rgb (255, 0, 255)
6:    ALLEGRO_COLOR pixel
7:    J2, I2
8:    temp <- Mask_Create (w, h)
9:    if (! temp)
10:       return 0
11:   Endif
12:   Mask_Clear(temp)
13:    for i <- 0 to h
```

```
14:      for j <- 0 to w
15:        J2 <- j - w/2
16:        I2 <- i - h/2
17:        J2 <- J2 * cos(theta) - I2 * sin(theta)
18:         I2 <- J2 * sin(theta) + I2 * cos(theta)
19:        J2 <- J2 + w/2
20:         I2 <- I2 + h/2
21:        pixel <- al_get_pixel (bmp, j, i)
22:         if (! Color_Equiv (pixel, transColor) &&! Transparent(pixel))
23:              Mask_SetBit (temp, J2, I2)
24:        Endif
25:     Endfor
26:   Endfor
28:   return temp
29: Endfuntion
```

4.10 Algoritmo genético secuencial

El Algoritmo 4-5 muestra el pseudocódigo del algoritmo genético secuencial. En primer lugar, el algoritmo inicializa los parámetros de entrada (línea 1), tales como probabilidad de cruce (*Pc*), probabilidad de mutación (*Pmut*), tamaño de la población (Tpob), número de generaciones (Ng), porcentaje de selección (Ps) y porcentaje de mutación (Pm), carga las imágenes (línea 2). PNG que contienen las figuras que el Algoritmo Genético va a posicionar en la hoja de papel quedando así la configuración inicial (línea 3). Posteriormente, la generación G se inicializa en cero (línea 4). Los valores que toma esta configuración inicial dependen del análisis de sensibilidad que se realiza para los parámetros de entrada. Después genera una población inicial aleatoria de individuos (línea 5). Los individuos de la población son evaluados para verificar su factibilidad (línea 6), aquí se verifica que no existan traslapes entre las figuras amorfas, y si existen se eliminan estos traslapes, también que las figuras amorfas se posicionen exactamente dentro de las dimensiones de la hoja de papel. Posteriormente se evalúa el valor de aptitud de cada individuo (línea 7). El criterio de paro del Algoritmo Genético es G < Ng (línea 8). Los individuos evolucionan durante generaciones a través de tres operadores como: selección, cruzamiento y mutación. El método de selección que se utiliza es por torneo (línea

12), donde selecciona a los individuos más aptos. Después los padres seleccionados son recombinados para formar individuos descendientes que heredan las características de sus padres utilizando el operador de cruce (línea 13), debido a un factor de probabilidad, el cual determina si el cruce se lleva a cabo o no. Después del cruzamiento se evalúa la factibilidad de los individuos, la cual se realiza exactamente como se comentó anteriormente, por ejemplo, se verifica que no existan traslapes entre las figuras amorfas, y si existen se eliminan estos traslapes, también que las figuras amorfas se posicionen exactamente dentro de las dimensiones de la hoja de papel. Más tarde, se realiza la mutación a un porcentaje de individuos de la población total (línea 15), aquí se realizan la combinación de los movimientos descritos en la Estructura Híbrida de Vecindad. Después de la mutación, nuevamente se evalúa la factibilidad de los individuos. Se agregan a la nueva población los individuos factibles (línea 17) hasta completar el tamaño de la población. Se evalúa el valor de aptitud de la nueva población (línea 20). Se reemplaza la nueva población e incrementa el número de generación. La Figura 4.6 presenta el diagrama de flujo del algoritmo genético secuencial, en el cual se sigue el flujo de ejecución de acuerdo con los pasos presentados en el Algoritmo 4-5.

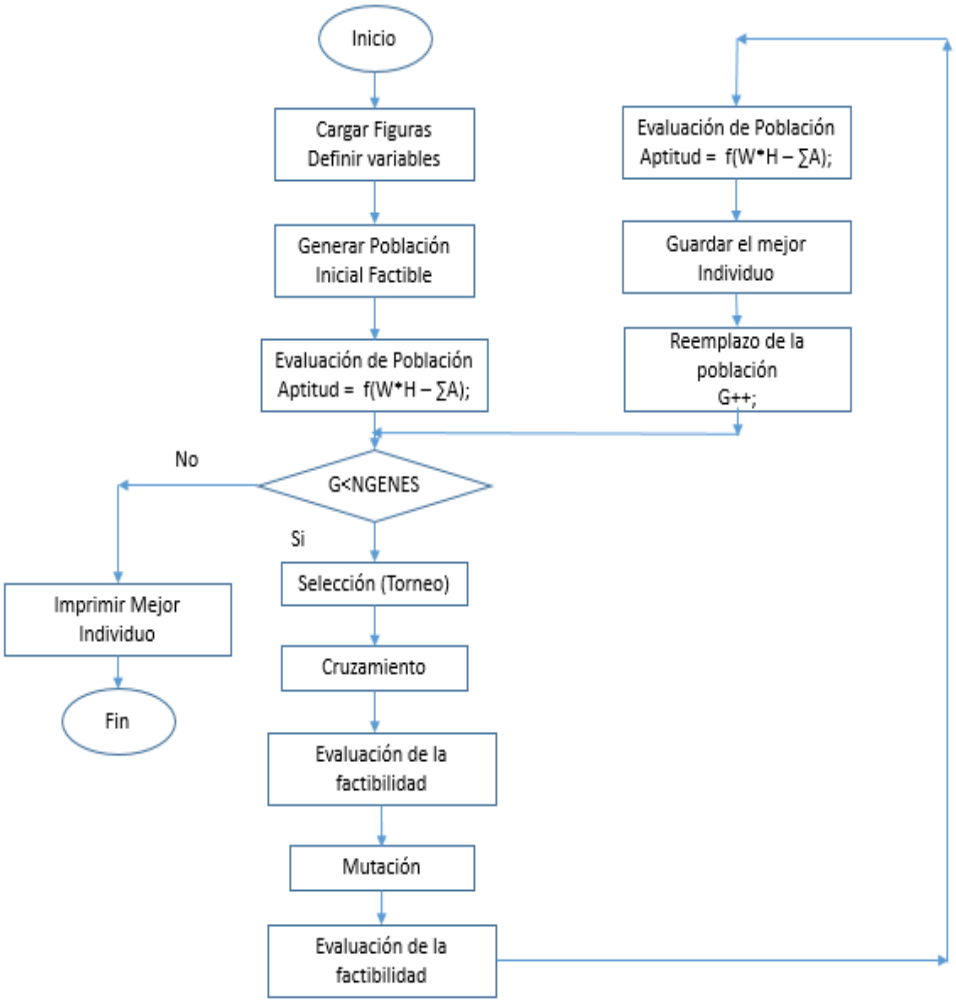

Figura 4.6. *Diagrama de Flujo del Algoritmo Genético.*

Algoritmo 4-5. Pseudocódigo del Algoritmo Genético Secuencial.
1: Parámetros (Pc, Pm, Ptam, Ng); //Inicializar los parámetros de entrada
2: CargarImagenesFiguras(); //Acá se cargan los datos de las imágenes y las máscaras
3: Configuración inicial
4: G=0; //Número de pasos
5: Generar(P);//Población inicial
6: EvaluaciónFactibilidad(P);//Se verifica que no hayan traslapes entre figuras
7: EvaluarFitnees(P)//Evaluación de la población(Resta del Área de la hoja de papel y la suma de las Áreas de las figuras)
8: Mientras G<Ng Hacer
9: P´=0;Inicializar Población P´
10: MejorIndividuo(MejorIND, P´)//Se guarda el mejor individuo
11: Mientras Ninds<Ptam Hacer
12: Selección(P, pi, p2); Seleccionar padres p1, p2 de la Población P
13: {h1, h2}=Cruzamiento(p1,p2, Pc);//Cruzamiento de los padres p1, p2
14: EvaluaciónFactibilidad(h1, h2);//Reparación de los hijos, quitar las figuras traslapadas

15: Mutación(h1, h2);//Mutación de los descendientes, aplicación de la estructura híbrida de vecindad

16: EvaluaciónFactibilidad(h1, h2);//Reparación de los hijos, quitar las figuras traslapadas

17: Agregar(P´, h1, h2); //Insertar los descendientes h1, h2 a la población P´

18: Ninds +=2;

19: Fin Mientras

20: EvaluarFitness(P); //Evaluación de la población(Resta del Área de la hoja de papel y la suma de las Áreas de las figuras)

21: ReemplazoPoblación(P, P´);

22: G+=1; Incremento de generación

23: Fin Mientras

Capítulo 5 Resultados experimentales

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al algoritmo genético. Para ello se realizó un análisis de sensibilidad con el objetivo de mejorar el desempeño del algoritmo. La sintonización de los parámetros de control se realizó en el clúster CIICAp. Se presenta el análisis estadístico para el algoritmo genético que soluciona el Problema de Empaquetamiento de Contenedores de Dos Dimensiones Aplicado a Imprentas Digitales. Se utilizó la librería Allegro 5 para representar visualmente las soluciones que se muestran en este capítulo.

5.1 Descripción del equipo utilizado

Para las pruebas se utilizó una computadora Laptop Asus ROG ubicada en el Centro de Investigación en Ingeniería y Ciencias Aplicadas (CIICAp) perteneciente a la Universidad Autónoma del Estado de Morelos (UAEM). Las características del equipo utilizado se presentan en la Tabla 5.1

Tabla 5.1 Características del Equipo Utilizado.

Motherboard	Procesador	Memoria	Tarjeta gráfica
• BaseBoard Manufacturer: ASUSTek COMPUTER INC. • BaseBoard Product: GL50GE. • BaseBoard Version: 1.0.	• Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz	• 8192MB RAM	• NVIDIA GeForce GTX 1050 Ti. • Manufacturer: NVIDIA.

5.2 Análisis de Sensibilidad

Para la sintonización de los parámetros de control de un algoritmo, es necesario llevar a cabo un análisis de sensibilidad. El análisis de sensibilidad es un

componente importante en la construcción de modelos matemáticos, computacionales y de simulación. La finalidad del análisis de sensibilidad es encontrar la mejor sintonización de los parámetros de control del algoritmo de manera que este tenga el mejor funcionamiento posible en cuanto a la eficiencia y eficacia.

1. Selección de los parámetros de control

En este paso hay que seleccionar los parámetros de control de acuerdo al algoritmo, para ello es necesario realizar una revisión bibliográfica de los autores que hayan implementado dicho algoritmo y así determinar los parámetros adecuados:

- Porcentaje de selección (P_s)
- Porcentaje de mutación (P_m)
- Tamaño de población (T_{pob})
- Probabilidad de cruzamiento (P_c)
- Probabilidad de mutación (P_{mut})
- Número de generaciones (N_g)

2. Establecer rangos de evaluación

Después de haber identificado los parámetros de control, es necesario determinar los rangos de cada parámetro, lo cual permitirá realizar el análisis de sensibilidad de dicho algoritmo, en la Tabla 5.2 se presentan los rangos de cada parámetro de control.

Tabla 5.2 Rangos de los parámetros de control.

Parámetro de control	Límite inferior	Límite superior
P_s	10%	50%
P_m	10%	50%
T_{pob}	100	500
P_c	N/A	0.7
P_{mut}	N/A	0.2
N_g	N/A	200

3. Pruebas para rangos de evaluación

Una vez establecidos los rangos para cada parámetro de control, se realizan las pruebas experimentales, ejecutando 30 pruebas por cada una de las muestras calculadas. Para la adecuada sintonización de los parámetros de control, es necesario realizar las pruebas de los valores correspondientes a una de las variables, pero manteniendo fijos los demás, hasta encontrar el valor que mejore la calidad de las soluciones. En cuanto se obtenga el mejor valor de dicha variable, se fija el valor y se comienza con la variación de otra variable, llevando a cabo el mismo proceso, hasta obtener el conjunto de valores que permita obtener el mejor desempeño del algoritmo en cuanto a eficacia y eficiencia.

En la Tabla 5.3 se muestra los resultados obtenidos de la sintonización de parámetros de control del algoritmo genético secuencial. La metodología utilizada se basa en determinar un porcentaje de individuos seleccionados del 10% al 50%, dejando fijo el porcentaje de mutación a un 10%. Se realizaron 30 ejecuciones con una población de 500 individuos y 200 generaciones. Se presenta el mejor resultado, peor y promedio de un total de 30 ejecuciones, teniendo en cuenta el área

residual (AR) y el número de figuras (No. F) insertadas en la hoja de papel, se destaca que estas pruebas están hechas para un tipo de figura. De acuerdo a estos se determina el porcentaje de selección a un 50% debido a que se encuentra el mejor promedio del área residual y de figuras insertadas. Se puede observar que el mejor valor de la función objetivo teniendo en cuenta el área residual y el número de figuras insertadas en la hoja de papel, fue de 630089 pixeles cuadrados para un total de 48 figuras insertadas. El peor valor obtenido por la función objetivo teniendo en cuenta el área residual fue de 635424 pixeles cuadrados para un total de 47 figuras y por último el valor promedio obtenido por la función objetivo teniendo en cuenta el área residual fue de 635424 pixeles cuadrados para un total de 47 figuras insertadas en la hoja de papel.

Tabla 5.3 Resultados del porcentaje de selección.

Selección (%)	Mejor AR (px^2)	Mejor No. F	Promedio AR (px^2)	Promedio No. F	Peor AR (px^2)	Peor No. F
10	686118	41	693360	40	693360	40
20	657150	45	657150	45	657150	45
30	649908	46	657150	45	657150	45
40	642666	47	649908	46	649908	46
50	**630089**	**48**	**635424**	**47**	**635424**	**47**

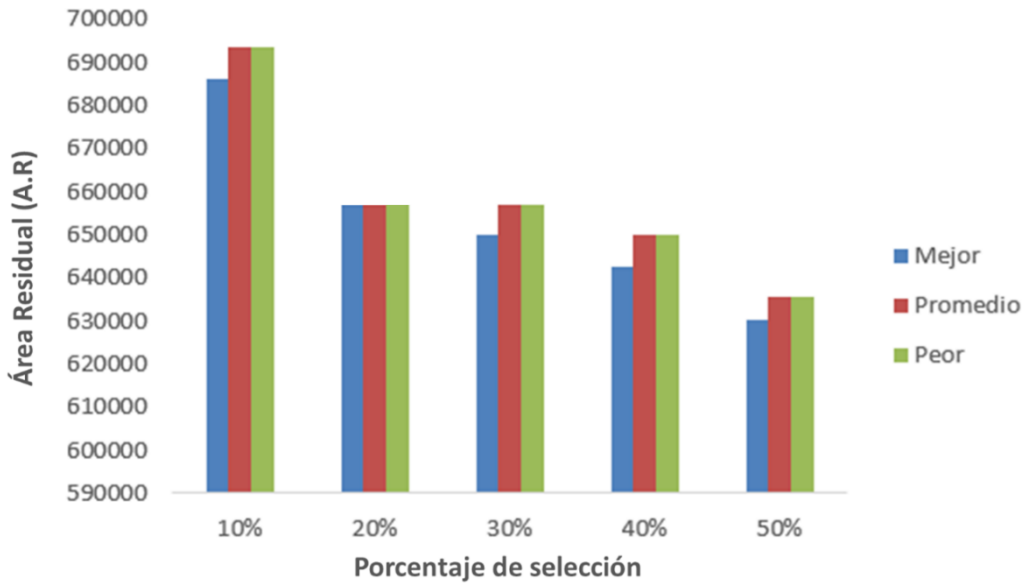

Figura 5.1 Comportamiento del mejor valor de aptitud, peor y promedio con respeto al porcentaje de selección.

El comportamiento de las soluciones de acuerdo con la mejor solución encontrada por la función objetivo, correspondiente al porcentaje de selección que se muestra en la Figura 5.1. Se observa que el mejor valor aptitud se presenta en un 50%.

La Tabla 5.5 muestra los resultados obtenidos del porcentaje de mutación del 10% hasta el 50%, dejando fijo el porcentaje de selección a un 50% debido a los resultados presentados en la Tabla 5.3. Se realizaron 30 ejecuciones con una población de 500 individuos y 200 generaciones. Se presenta el mejor, peor y promedio de un total de 30 ejecuciones.

Tabla 5.4 Resultados del porcentaje de mutación.

Mutación (%)	Mejor AR (px^2)	Mejor No. F	Promedio AR (px^2)	Promedio No. F	Peor AR (px^2)	Peor No. F
10	671634	43	678876	42	686118	41
20	671634	43	686118	42	693360	40
30	664392	44	671634	43	686118	41
40	**620940**	**50**	**649908**	**46**	**657150**	**45**
50	664392	44	671634	43	693360	40

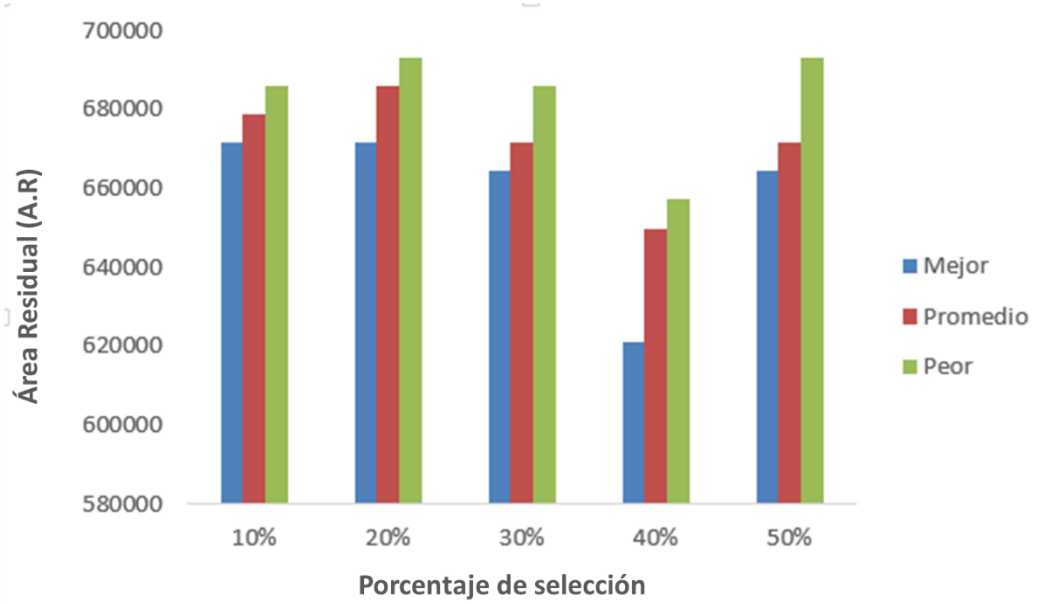

Figura 5.2 Comportamiento del mejor valor de aptitud, peor y promedio con respecto al porcentaje de mutación.

La Tabla 5.5 muestra los resultados obtenidos del tamaño de población, dejando fijo el porcentaje de selección a un 50%, el porcentaje de mutación a un 40% y 200 generaciones. Se realizaron 30 ejecuciones con un tamaño de población de 100, 200, 300, 400 y 500.

Tabla 5.5 Resultados del tamaño de la población.

Número Individuos	Mejor AR (px^2)	Mejor No. F	Promedio AR (px^2)	Promedio No. F	Peor AR (px^2)	Peor No. F
100	657150	45	671634	43	42	686118
200	649908	46	664392	44	40	693360
300	664392	44	686118	41	41	686118
400	671634	43	686118	41	40	693360
500	**637331**	**49**	**664392**	**44**	**45**	**657150**

Sintonización de Parámetros

Una vez terminadas las pruebas experimentales para obtener los valores de cada uno de los parámetros de control definidos para todas las muestras, estos valores son definidos como los valores de sintonización, el valor de la probabilidad de cruzamiento se determinó de acuerdo con la literatura (ver Tabla 5.7).

Tabla 5.6 Valores de sintonización para los parámetros de control.

Parámetro de control	**Valor sintonizado**
P_s	**50%**
P_m	**40%**
T_{pob}	**500**
P_c	**0.7**
P_{mut}	**0.2**
N_g	**200**

La sintonización de parámetros se lleva a cabo con el objetivo de identificar los valores correspondientes a los parámetros de control, los cuales permitan obtener una mejora en el desempeño del algoritmo tanto en eficacia como en eficiencia.

5.3 Algoritmo Genético secuencial AG-PEC2DAOPID

Después del análisis de sensibilidad, se realizaron las pruebas finales para obtener los resultados del algoritmo genético secuencial **AG-PEC2DAOPID** para el Problema de Empaquetamiento de Contenedores de Dos Dimensiones Aplicado a la Optimización de Papel en Imprentas Digitales. Se muestra la convergencia del algoritmo. También se presenta el análisis estadístico de los resultados y por último se define el cálculo de la complejidad del algoritmo.

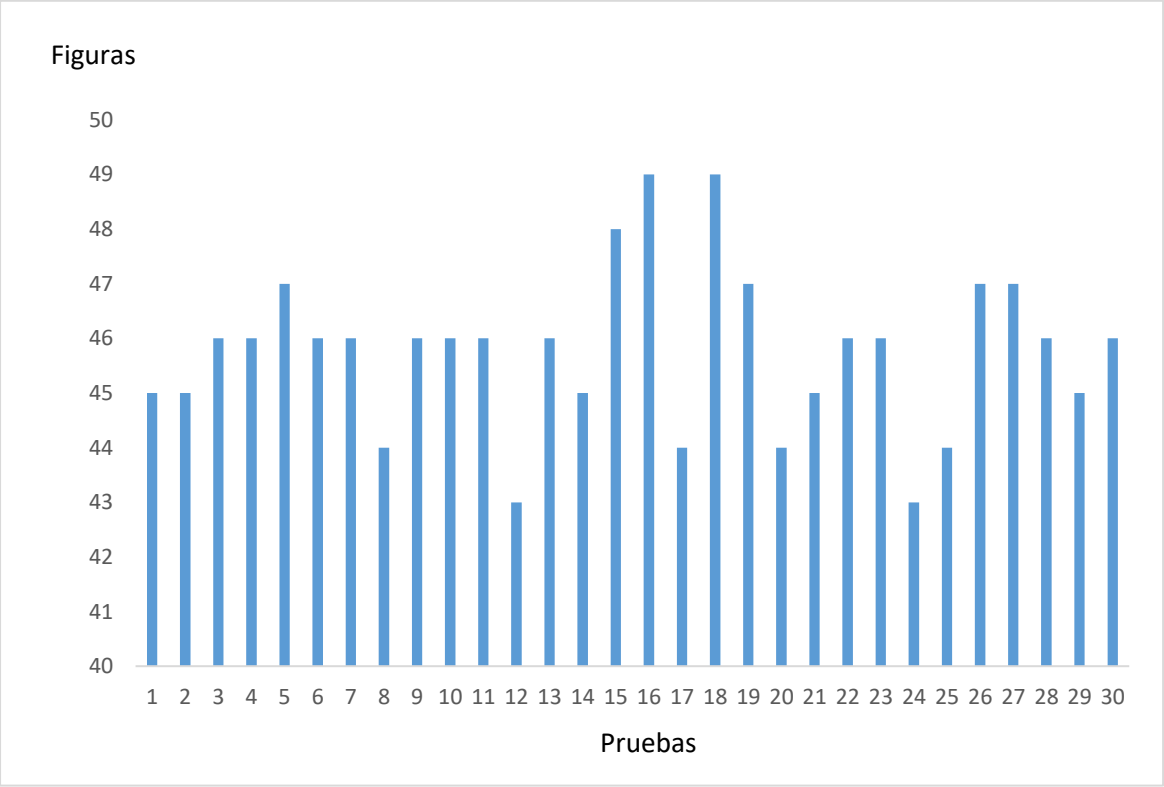

Figura 5.3 Comportamiento del Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio a partir del número de figuras para un solo tipo de figuras.

En la Figura 5.3 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e

intercambio, teniendo en cuenta el número de figuras insertadas en la hoja de papel para un mismo tipo de figuras. Los parámetros que se sintonizan en el Algoritmo Genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.3, se puede decir que la mejor solución encontrada fue para 49 figuras, y la peor solución fue de 43 figuras ambas posicionadas en la hoja de papel. La Tabla 5.7 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al algoritmo genético. Cabe resaltar que cada prueba tiene una duración de 8 horas en promedio.

Tabla 5.7 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Número de figuras.

Mejor solución	Peor solución	Mediana	Moda
49	43	46	46

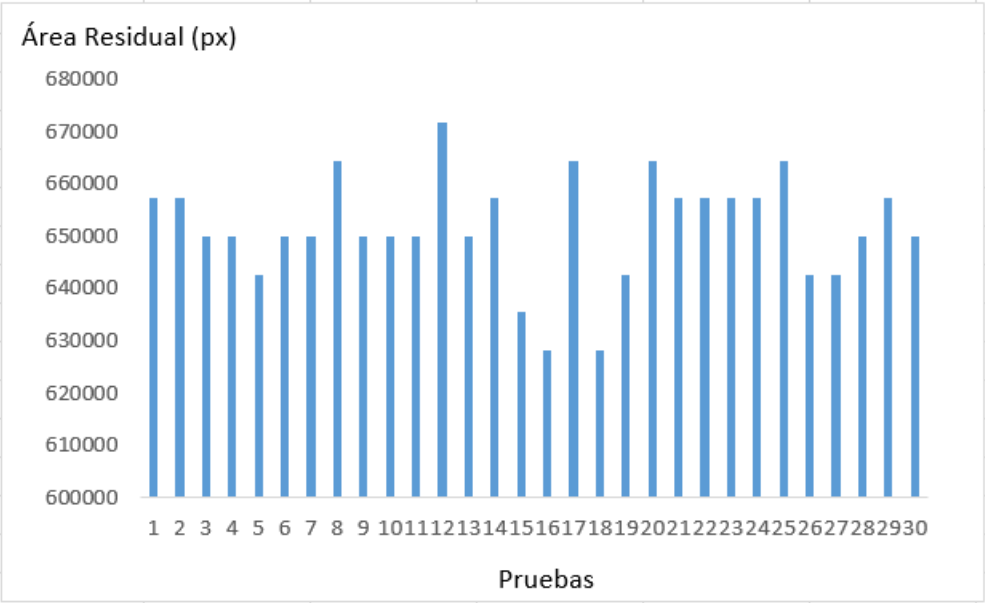

Figura 5.4 Comportamiento del Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio a partir del Área residual para un solo tipo de figuras.

En la Figura 5.4 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e

intercambio, teniendo en cuenta el Área residual en la hoja de papel para un mismo tipo de figuras. Los parámetros que se sintonizan en el Algoritmo Genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.4, se puede decir que la mejor solución encontrada fue para 628182 pixeles cuadrados, y la peor solución fue de 671634 pixeles cuadrados en la hoja de papel. La Tabla 5.8 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al algoritmo genético. Cabe resaltar que cada prueba tiene una duración de 8 horas en promedio.

Tabla 5.8 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Área residual.

Mejor solución	Peor solución	Mediana	Moda
628182	671634	649908	649908

Figura 5.5 Mejor solución obtenida por el Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio a partir del número de figuras para un solo tipo de figuras.

Como se puede observar en la Figura 5.5 se muestran la mejor solución obtenida de forma gráfica de las 30 pruebas experimentales realizadas al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, teniendo en cuenta el Número de figuras posicionadas en la hoja de papel y Área residual obtenida y todo esto para un mismo tipo de figuras. Es importante destacar que en todas las pruebas realizadas y presentados de forma visual se utilizó la librería Allegro5

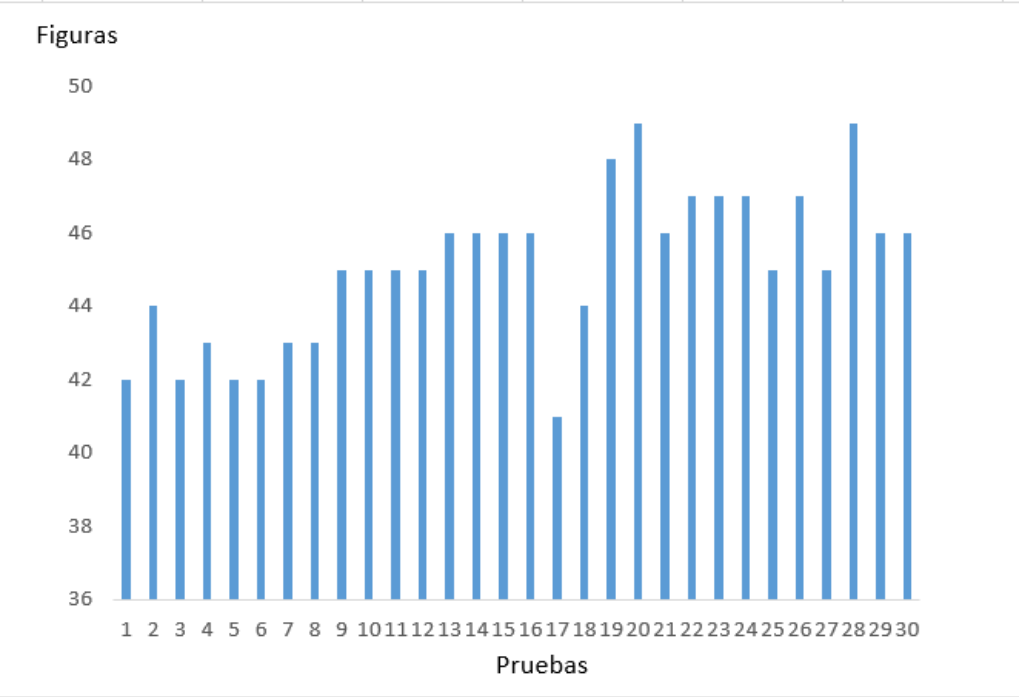

Figura 5.6 Comportamiento del Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio a partir del número de figuras para un solo tipo de figuras.

Como se puede observar en la Figura 5.6 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, además en esta prueba se agregó el movimiento de

rotación. En todos los casos se tiene en cuenta el número de figuras insertadas en la hoja de papel para un mismo tipo de figuras. Los parámetros que se sintonizan en el Algoritmo Genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.6, se puede decir que la mejor solución encontrada fue para 49 figuras, y la peor solución fue de 41 figuras ambas posicionadas en la hoja de papel. La Tabla 5.9 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al algoritmo genético. Cabe resaltar que cada prueba tiene una duración de 11 horas en promedio.

Tabla 5.9 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Número de Figuras con Rotación.

Mejor solución	Peor solución	Mediana	Moda
49	41	46	46

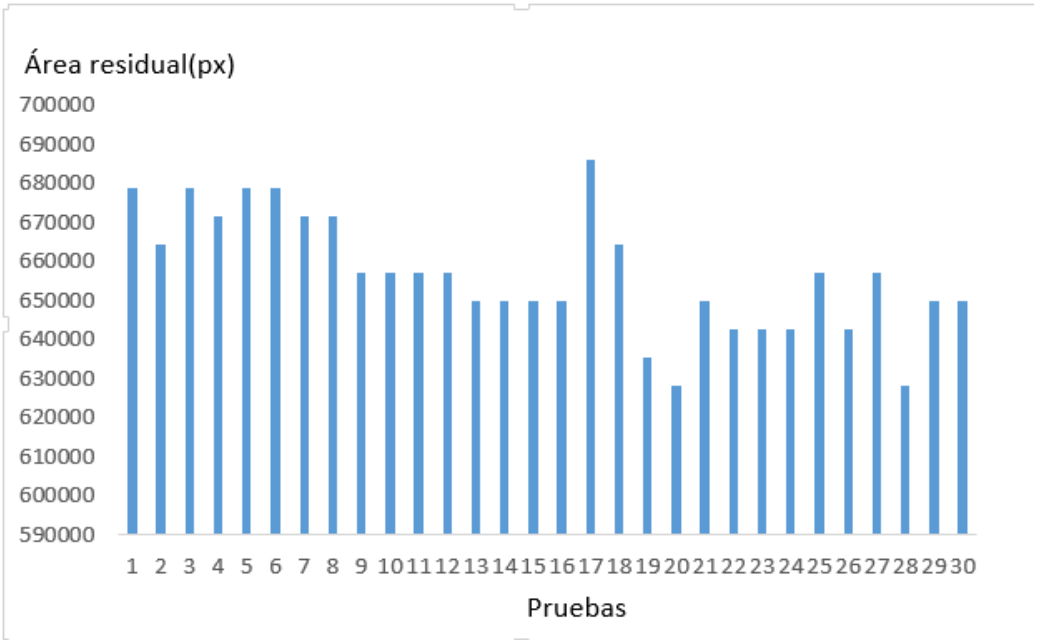

Figura 5.7 Comportamiento del Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio a partir del Área residual para un solo tipo de figuras.

Como se puede observar en la Figura 5.7 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación, intercambio y rotación, teniendo en cuenta el Área residual en la hoja de papel para un mismo tipo de figuras. Los parámetros que se sintonizan en el Algoritmo Genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.7, se puede decir que la mejor solución encontrada fue para 628182 pixeles cuadrados, y la peor solución fue de 686118 pixeles cuadrados en la hoja de papel. La Tabla 5.10 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al algoritmo genético. Cabe resaltar que cada prueba tiene una duración de 11 horas en promedio.

Tabla 5.10 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Área residual con Rotación.

Mejor solución	Peor solución	Mediana	Moda
628182	686118	657150	649908

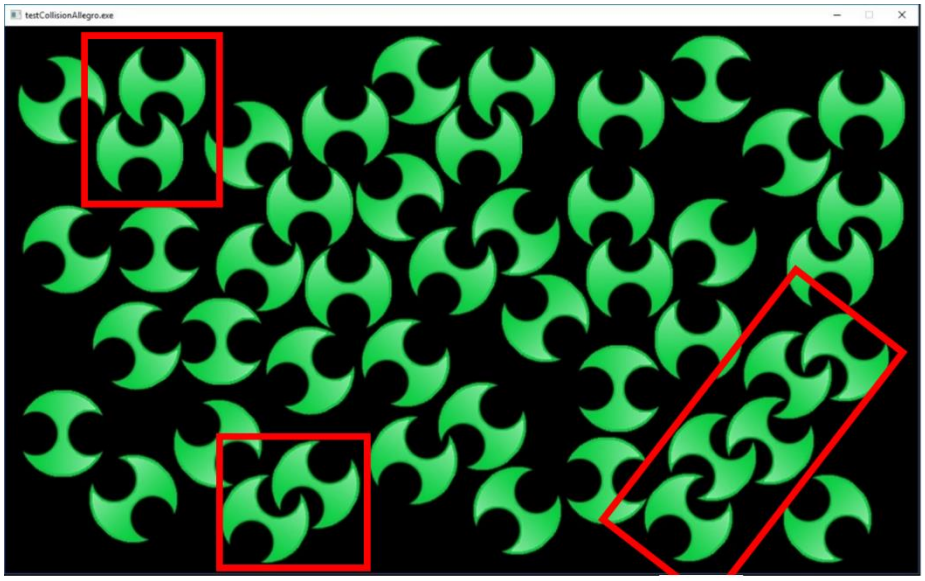

Figura 5.8 Mejor solución obtenida por el Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio a partir del número de figuras para un solo tipo de figuras.

En la Figura 5.8 se muestra la mejor solución obtenida de forma gráfica de las 30 pruebas experimentales realizadas al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio, teniendo en cuenta el Número de figuras posicionadas en la hoja de papel y Área residual obtenida y todo esto para un mismo tipo de figuras. Como se aprecia, en las regiones señaladas en color rojo, se obtiene concatenaciones de las partes curvas de las figuras. Estos resultados se deben a la estructura híbrida de vecindad, específicamente al movimiento de rotación.

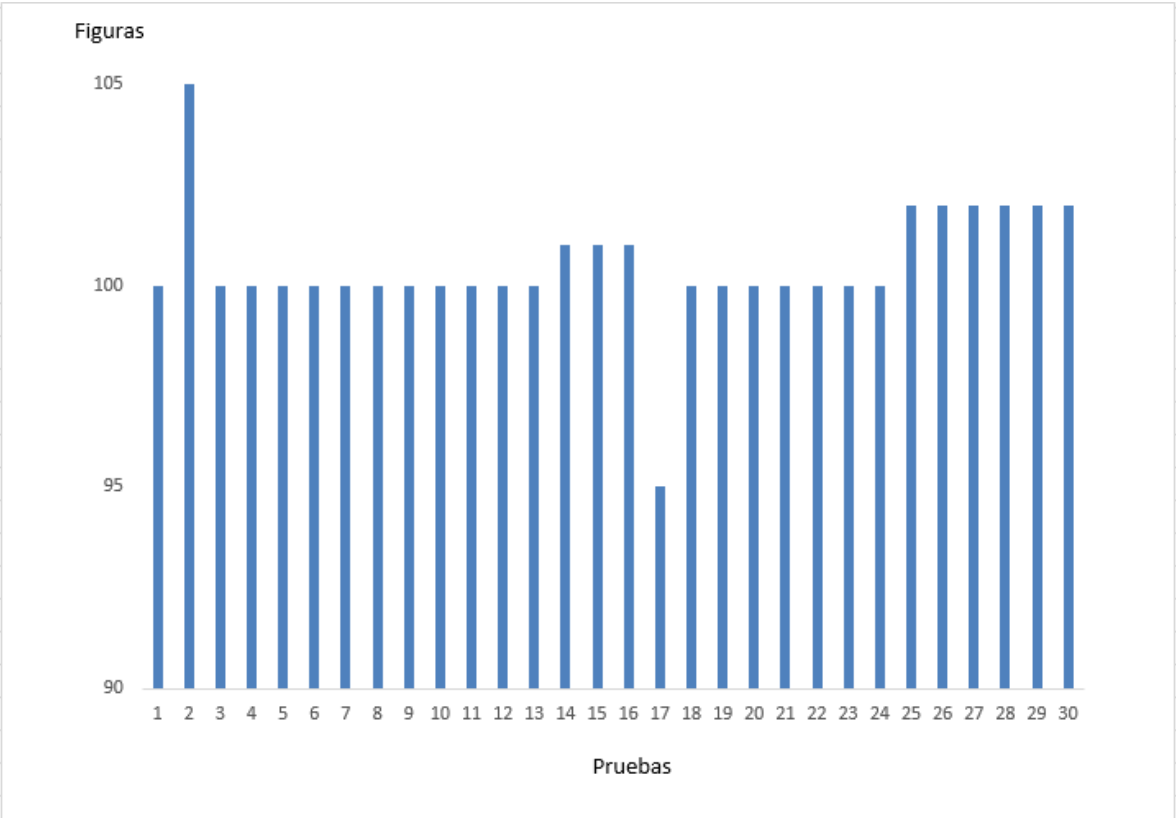

Figura 5.9 Comportamiento del Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio a partir del número de figuras para varias figuras.

Como se puede observar en la Figura 5.9 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, teniendo en cuenta el número de figuras insertadas en la hoja de papel. En estas 30 pruebas se comenzaron a trabajar con diferentes tipos de figuras (conejos, latas, nubes, entre otras). Los parámetros que se sintonizaron en el Algoritmo Genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.9, se puede decir que la mejor solución encontrada fue para 105 figuras, y la peor solución fue de 95 figuras ambas posicionadas en la hoja de papel. La Tabla 5.11 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al algoritmo genético. Cabe resaltar que cada prueba tiene una duración de 13 horas en promedio.

Tabla 5.11 Parámetros estadísticos extraídos de las 30 pruebas experimentales para varias figuras teniendo en cuenta el Número de Figuras.

Mejor solución	Peor solución	Mediana	Moda
105	95	100	100

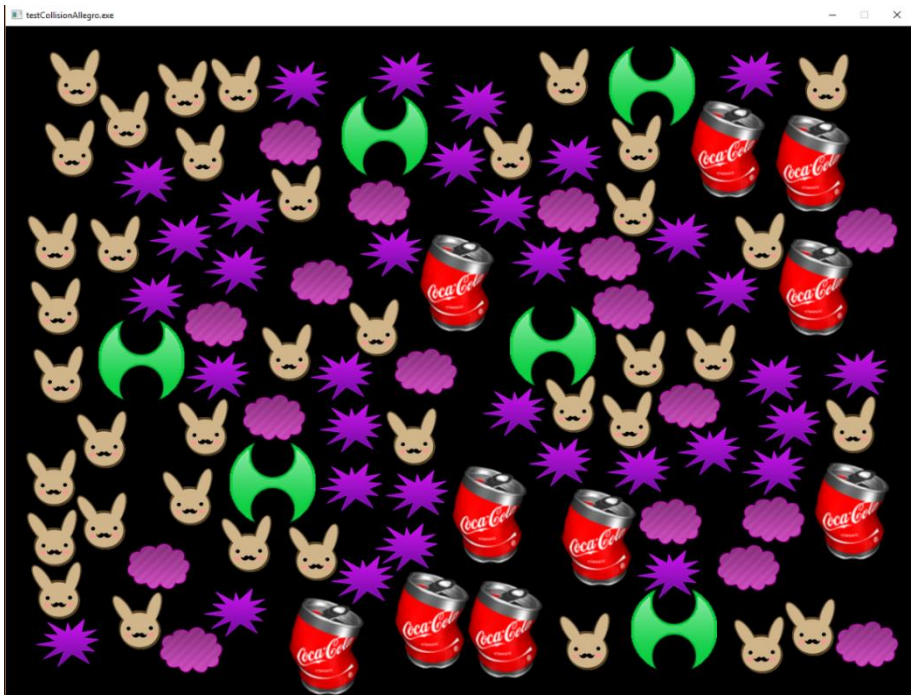

Figura 5.10 Representación de la mejor solución obtenida por el Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, teniendo en cuenta el número de figuras posicionadas en la hoja de papel para varios tipos de figuras, 105 figuras.

La Figura 5.10 muestra la mejor solución obtenida de forma gráfica de las 30 pruebas experimentales realizadas al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, teniendo en cuenta el Número de figuras posicionadas en la hoja de papel para varios tipos de figuras. En este caso, se insertan distintos tipos de figuras: como conejos, latas, estrellas, entre otros. Lo que representa una solución más heterogénea respecto al tipo de figuras. Nuevamente, se aprecia una concatenación de varias figuras. Principalmente, de las figuras verdes y los conejos.

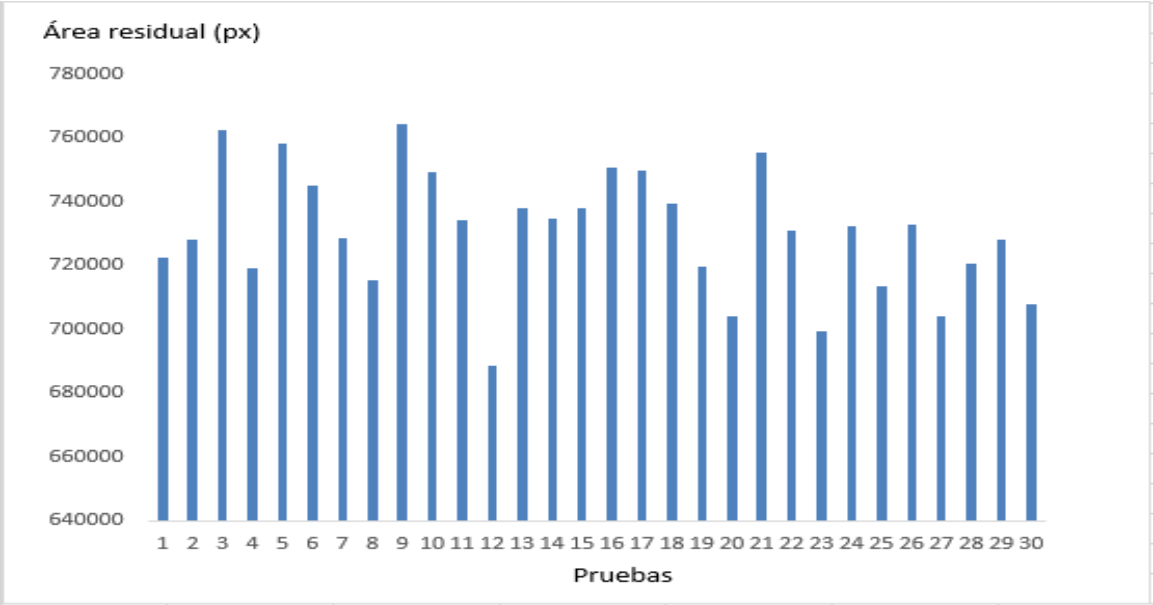

Figura 5.11 Comportamiento del Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio a partir del Área Residual medida en pixeles cuadrados.

Como se puede observar en la Figura 5.11 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e intercambio. En este caso lo que se está teniendo en cuenta es el área residual. Los parámetros que se sintonización en el Algoritmo Genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.11, se puede decir que la mejor solución encontrada fue de 688353, y la peor solución fue de 764459 ambas medidas en pixeles cuadrados. La Tabla 5.12 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al algoritmo genético. Cabe resaltar que cada prueba tiene una duración de 13 horas en promedio.

Tabla 5.12 Parámetros estadísticos extraídos de las 30 pruebas experimentales para varias figuras teniendo en cuenta el Área residual.

Mejor solución	Peor solución	Mediana	Moda
688353	764459	731446	-

Figura 5.12 Representación de la mejor solución obtenida por el Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, teniendo en cuenta el Área residual medida en pixeles cuadrados, el Área Residual es de 688353 pixeles cuadrados.

En la Figura 5.12 se muestra la mejor solución obtenida de forma gráfica de las 30 pruebas experimentales realizadas al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación e intercambio, teniendo en cuenta el Área residual a partir Número de figuras posicionadas en la hoja de papel para varios tipos de figuras. En este caso el algoritmo fue capaz de insertar un mayor número de figuras sin que exista traslape, y logrando concatenación entre varias imágenes. Lo que refleja la eficacia del mismo. Sin embargo, se insertan pocas figuras de mayor área, como las latas de soda, a que el algoritmo selecciona las figuras para insertar en la hoja de papel de forma aleatoria. También se debe tener en cuenta, que solo se han aplicado tres movimientos en este caso, omitiendo la rotación.

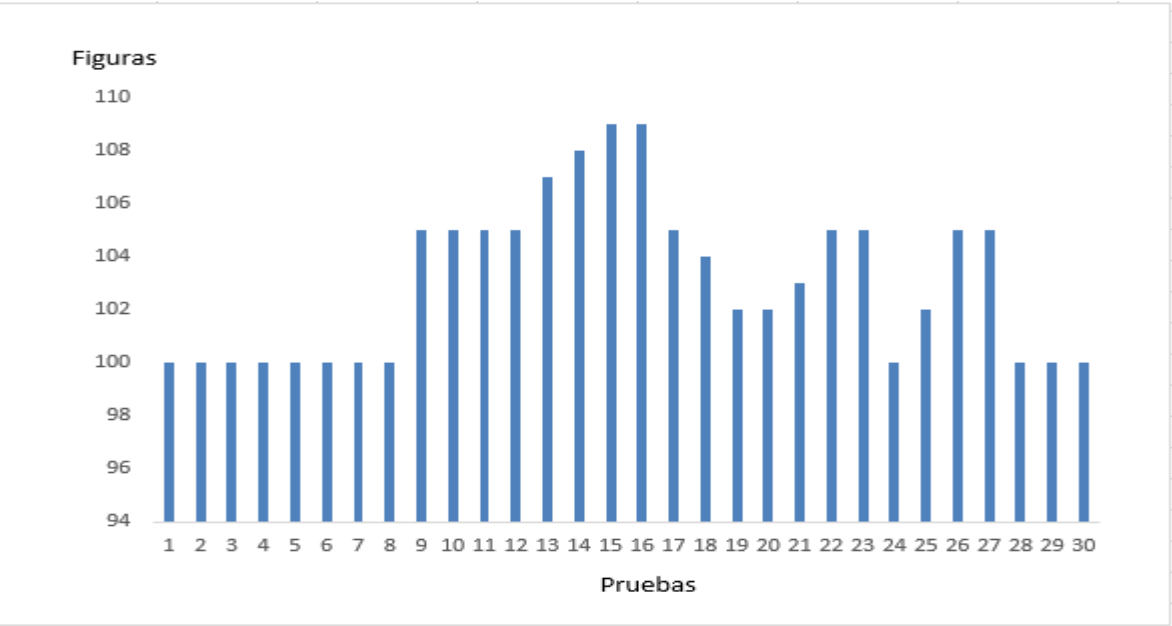

Figura 5.13 Comportamiento del Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación, rotación e intercambio a partir del número de figuras.

Como se puede observar en la Figura 5.13 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación, rotación e intercambio, teniendo en cuenta el número de figuras insertadas en la hoja de papel. Los parámetros que se sintonización en el algoritmo genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.13, se puede decir que la mejor solución encontrada fue para 109 figuras, y la peor solución fue de 100 figuras ambas posicionadas en la hoja de papel. La Tabla 5.13 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al Algoritmo Genético. Cabe resaltar que cada prueba tiene una duración de 18 horas en promedio.

Tabla 5.13 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Número de Figuras con Rotación.

Mejor solución	Peor solución	Mediana	Moda
109	100	102	100

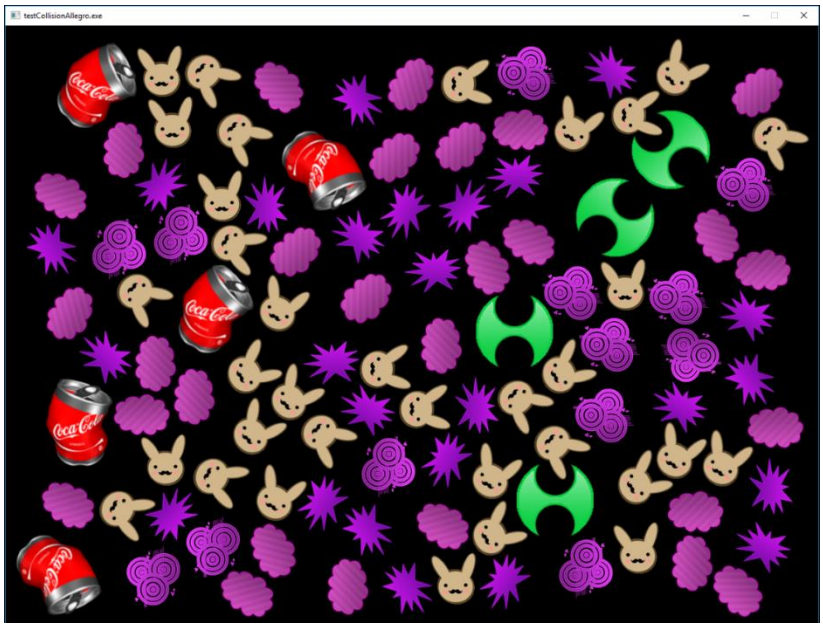

Figura 5.14 Representación de la mejor solución obtenida por el Algoritmo Genético con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio, teniendo en cuenta el número de figuras posicionadas en la hoja de papel para varios tipos de figuras, 109 figuras.

Como se puede observar en la Figura 5.14 se muestran la mejor solución obtenida en forma de gráfica de las 30 pruebas experimentales realizadas al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio, teniendo en cuenta el Número de figuras posicionadas en la hoja de papel para varios tipos de figuras. La concatenación de figuras es mayor, debido a que se aplican los 4 movimientos de la estructura híbrida de vecindad.

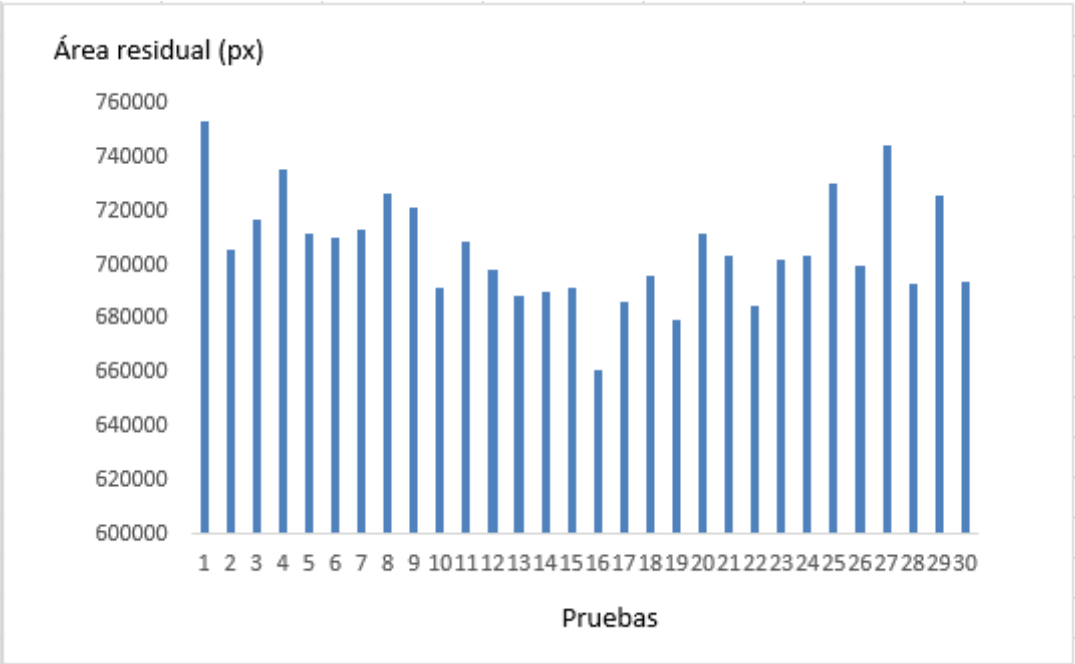

Figura 5.15 Comportamiento del Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación, rotación e intercambio a partir del Área Residual.

En la Figura 5.15 se muestran 30 pruebas experimentales al Algoritmo Genético con la Estructura Híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio, teniendo en cuenta el Área Residual medida en pixeles cuadrados. Los parámetros que se sintonizaron en el algoritmo genético fueron los que aparecen en la Tabla 5.6. De la Figura 5.15, se puede decir que la mejor solución encontrada fue 660660, y la peor solución fue 752933 ambas medidas en pixeles cuadrados. La Tabla 5.14 muestra los parámetros estadísticos extraídos de las 30 pruebas experimentales hechas al Algoritmo Genético. Cabe resaltar que cada prueba tiene una duración de 18 horas en promedio.

Tabla 5.14 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Área residual con Rotación.

Mejor solución	Peor solución	Mediana	Moda
660660	752933	703561	-

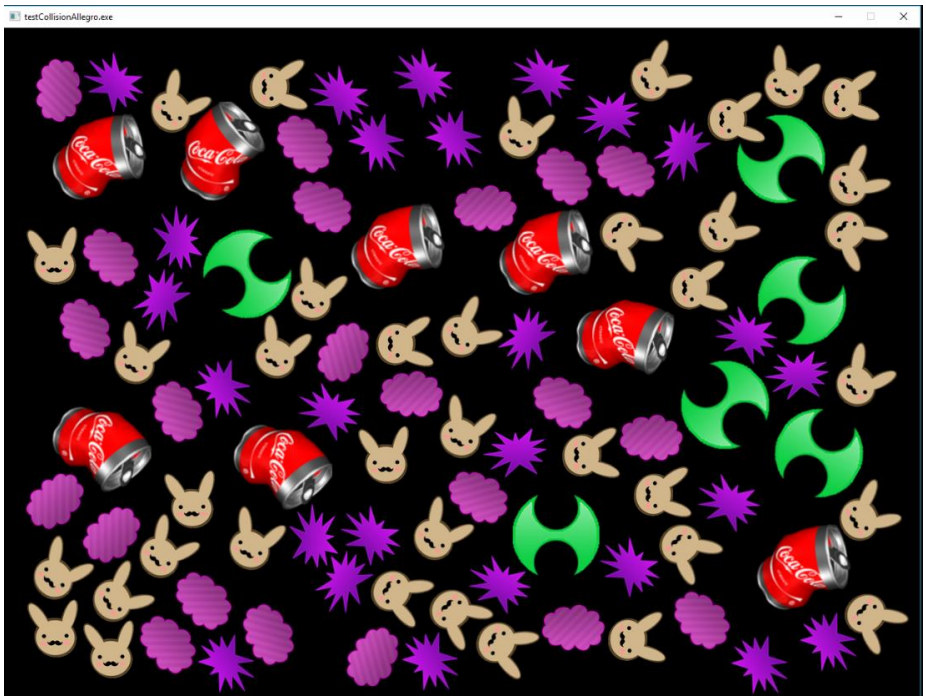

Figura 5.16 Representación de la mejor solución obtenida por el Algoritmo Genético, con la Estructura híbrida de Vecindad con los movimientos de inserción, traslación rotación e intercambio teniendo en cuenta el Área residual medida en pixeles cuadrados, el Área Residual es de 660660 pixeles cuadrados.

Finalmente, en la Figura 5.16 se muestra la mejor solución para las 30 pruebas experimentales realizadas al Algoritmo Genético con la Estructura Híbrida de Vecindad. En este caso aplicaron 4 movimientos: inserción, traslación rotación e intercambio, teniendo en cuenta el Área residual a partir del Número de figuras posicionadas en la hoja de papel. Aplicar los 4 movimientos favorece que se puedan insertar de manera aleatoria figuras de mayor área en la hoja de papel y que exista una concatenación entre varias de ellas.

5.3.1 Convergencia del algoritmo genético

La convergencia del algoritmo permite establecer en qué punto el algoritmo alcanza la estabilidad. En la Figura 5.17 se presenta el comportamiento del algoritmo genético con base al número de generaciones y al valor aptitud. Se utilizó un tamaño de población de 500 individuos, 200 generaciones, 40% de porcentaje de mutación y 50% porcentaje de selección. Se observa que se alcanza un valor estable del valor de aptitud y por lo tanto la convergencia del algoritmo. Esto significa que el valor de aptitud de la solución ya no mejora más. Cabe resaltar que cada prueba tiene una duración de 21 horas en promedio.

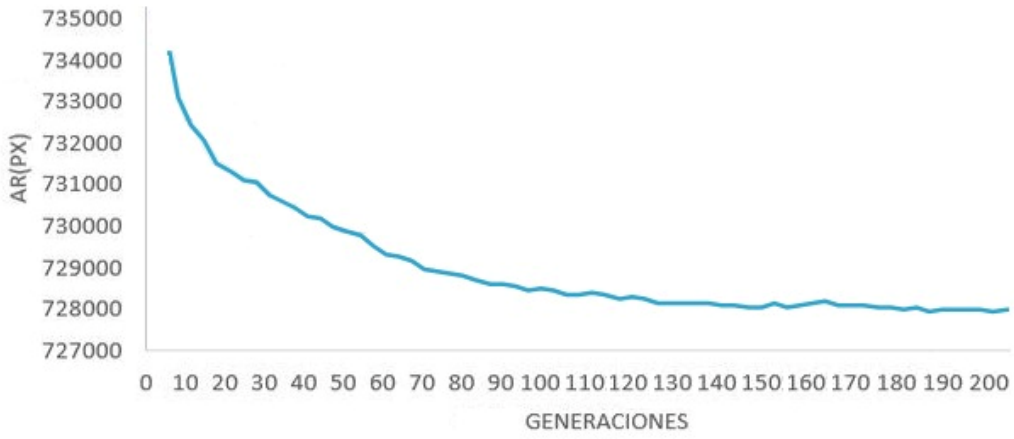

Figura 5.17 Comportamiento de la convergencia del Algoritmo Genético con la Estructura Híbrida de Vecindad con los Movimientos de Inserción, Traslación, Rotación e Intercambio.

5.3.2 Análisis Estadístico

El análisis estadístico es elemental para evaluar el comportamiento del algoritmo con respecto a la instancia o instancias evaluadas y así determinar el desempeño del mismo. En este trabajo se evalúa una instancia de un caso real, por lo que se trabaja con

varias instancias, por ejemplo, se trabajan con un tipo de figuras y con la combinación de varias figuras. A continuación, se definirán los parámetros estadísticos analizados:

Media: La media aritmética o promedio de un conjunto de n mediciones es igual a la suma de las mediciones dividida entre n [107].

Moda: La moda es la categoría que se presenta con más frecuencia o el valor de x que se presenta con más frecuencia [107].

Mediana: La mediana m de un conjunto de n mediciones es el valor de x que cae en la posición media cuando las mediciones son ordenadas de menor a mayor [107].

Después de realizar el análisis de sensibilidad y determinar los parámetros de control sintonizados, se realizaron 30 ejecuciones para cada una de las instancias. Es decir, para un tipo de figuras y para la combinación de distintos tipos de figuras con un tamaño de población de 500 individuos, 50% de selección, 40% de mutación y 200 generaciones.

En las Tablas 5.15 y 5.16 se presentan las mejores soluciones encontradas con respecto a la función objetivo, la mejor solución, la peor solución, la mediana y la moda. De acuerdo con el resultado presentado para la moda, se observa que el valor de aptitud se repite con mayor frecuencia para la distribución de datos.

Tabla 5.15 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Número de Figuras con los movimientos de Inserción, Traslación, Rotación e Intercambio.

Medida	Valor de aptitud (Para el No. de Figuras)
Mejor solución	109
Peor solución	100
Mediana	102
Moda	100

Tabla 5.16 Parámetros estadísticos extraídos de las 30 pruebas experimentales respecto al Área residual con los movimientos de Inserción, Traslación, Rotación e Intercambio.

Medida	Valor Aptitud (Para el Área residual)
Mejor solución	660660
Peor solución	752933
Mediana	703561
Moda	-

En varios casos se aprecia que no existe moda, lo cual se debe a que el algoritmo posiciona distintas figuras en la hoja de papel con diferentes áreas y al momento de encontrar el área residual, no se repite ningún valor. Puede que el número de figuras sea el mismo en distintas soluciones, sin embargo, como las figuras tienen diferentes áreas el área residual no tiene por qué ser siempre la misma.

5.3.3 Análisis de la complejidad del algoritmo genético

Para obtener la complejidad de un algoritmo, es necesario realizar un estudio para conocer su comportamiento, medir su rendimiento y el uso de los recursos.

Un algoritmo se enfoca en dos recursos importantes para resolver un problema, tiempo y espacio. El tiempo de un algoritmo es el número máximo de pasos necesarios para resolver un problema de tamaño N, siendo cualquier número natural arbitrario pero fijo. La complejidad se define generalmente en términos del análisis del peor caso [108].

La función temporal del algoritmo propuesto se presenta en la ecuación (5-1):

$$\boldsymbol{T(N,n,g) = N(56g + 43gn^3 + gn^2(209 + 83m^2) + gn^4(148 + 24m^2))}$$

donde ***N*** es el número de figuras, ***g*** es el número de generaciones, ***n*** es el número de individuos y ***m*** es el tamaño de las figuras, el algoritmo implementado tiene una complejidad temporal para el peor de los casos de $O(N*gn^4*m^2)$.

Capítulo 6 Conclusiones y Trabajos Futuros

6.1 Conclusiones

Se implementó un algoritmo genético con una estructura de vecindad híbrida para resolver el problema de empaquetamiento de contenedores en dos dimensiones aplicado a la optimización de papel en imprentas digitales. La estructura de vecindad se diseñó e implementó con el fin de encontrar espacios disponibles para insertar más figuras en la hoja de papel. Se percibió que la estructura híbrida de vecindad propuesta funcionó con respecto al algoritmo genético.

El algoritmo genético implementado resuelve el problema de empaquetamiento de contenedores de dos dimensiones aplicado al problema de desperdicio de papel en imprentas digitales, teniendo en cuenta el número de figuras insertadas en la hoja de papel, en cuanto al cálculo de áreas de figuras amorfas y del área residual, además teniendo en cuenta la eficaz detección de traslapes entre las figuras amorfas rotadas y no rotadas.

La interfaz visual generada con la librería Allegro 5, sirve para validar de forma simple la solución encontrada al problema de empaquetamiento de contenedores de dos dimensiones aplicado a imprentas digitales, debido a que se puede observar la existencia o no de traslapes entre figuras amorfas y que todas las figuras insertadas se encuentran en la hoja de papel.

La aportación de este trabajo de investigación es la implementación de un algoritmo genético para resolver el problema de empaquetamiento de contenedores de dos dimensiones aplicado a imprentas digitales, así como también el diseño e implementación de una estructura híbrida de vecindad con el fin de insertar figuras

amorfas en la hoja de papel. Además, se logró mapear el problema de empaquetamiento de contenedores de dos dimensiones, al problema del desperdicio de papel en una imprenta digital.

6.2 Trabajos Futuros

La propuesta de solución desarrollada durante este trabajo de investigación servirá como base para las siguientes actividades a realizar como trabajo futuro:

Realizar una aplicación ya sea web o móvil para la disminución del desperdicio de papel en imprentas digitales, por ejemplo, una interfaz visual que contenga embebido el algoritmo implementado y que se pueda acceder desde internet.

Implementar un algoritmo genético de forma distribuida en MPI y de forma paralela con CUDA mediante un ambiente GRID para el problema de empaquetamiento de contenedores de dos dimensiones aplicado a la optimización de imprentas digitales con el fin de disminuir los tiempos de respuestas de este.

Implementar de forma paralela en CUDA un algoritmo de recocido simulado con el fin de hacer una comparación entre los comportamientos del Algoritmo Genético en Paralelo y el Recocido Simulado en Paralelo.

Para mejorar la eficacia se buscará realizar una cooperación entre proceso.

Ambos algoritmos se proponen de forma paralela porque actualmente el procesamiento de las imágenes es a partir del uso de los pixeles y este proceso es muy tardado ya que depende del tamaño en pixeles de las imágenes que contienen las figuras.

Referencias

1. Garey, M.R.; Johnson, D.S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*; W. H. Freeman & Co.: New York, NY, USA, 1990; ISBN 0716710455.
2. Papadimitriou, C.H.; Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1982; ISBN 0-13-152462-3.
3. Hillier, F.S.; Lieberman, G.J.; Pérez Castellanos, J.H.; González Osuna, M. *Introduction to Operations Research*; Libros McGraw-Hill de Mexico, 1988;
4. Wetzel, L.; Resnik, M.D. Frege and the Philosophy of Mathematics. *Philos. Rev.* **1983**, *92*, 114.
5. Klee, V.; Minty, G.; Pach, J.; Reed, B.; Yuditsky, Y. How Good Is the Simplex Algorithm? Almost All String Graphs Are Intersection Graphs of Plane Convex Sets. *34th Int. Symp. Comput. Geom.* **1970**.
6. Martinez-Oropeza, A. Solución Al Problema de Máquinas En Paralelo No Relacionadas Mediante Un Algoritmo de Colonia de Hormigas, 2010.
7. Salto, Carolina; Alba, Enrique; Leguizamón, G. Metaheurísticas Híbridas Paralelas Para Problemas Industriales De Corte, 2009.
8. Alicia Riojas Cañari Conceptos, Algoritmo y Aplicación Al Problema de Las N Reinas, 2005.
9. Osman, I.H.; Kelly, J.P. Meta-Heuristics Theory and Applications. *J. Oper. Res. Soc.* **1997**, *48*, 657–657.
10. Luque, G.; Alba, E. *Parallel Genetic Algorithms*; Studies in Computational Intelligence; Springer Berlin Heidelberg: Berlin, Heidelberg, 2011; Vol. 367;

ISBN 978-3-642-22083-8.

11. Aho, A. V.; Hopcroft, J.E.; Ullman, J.D. *The Design and Analysis of Computer Algorithms*; 1974.
12. Michalewicz, Z.; Fogel, D.B. How to Solve It: Modern Heuristics. *How to Solve It Mod. Heuristics* **2004**.
13. Téllez-Enríquez, E.; Mezura-Montes, E.; Coello, C.A.C. An Ant System with Steps Counter for the Job Shop Scheduling Problem. *2007 IEEE Congr. Evol. Comput.* **2007**, 477–484.
14. Sipser, M. *Introduction-To-The-Theory-Of-Computation-Michael-Sipser*; 1997.
15. Hartmanis, J.; Hopcroft, J.E. An Overview of the Theory of Computational Complexity. *J. Assoc. Comput. Mach.* **1971**.
16. Bisbal Riera, J. *Manual de Algorítmica : Recursividad, Complejidad y Diseño de Algoritmos*; 2009; ISBN 8497880277.
17. Rosen, K.H. *Discrete Mathematics and Its Applications*; McGraw-Hill, 2003;
18. Lovász, L. Complexity of Algorithms Lecture Notes. **2020**.
19. Papadinitrou, C. *Computational Complexity*; Addison-We.; Addison-Wesley, 1994.
20. Oropeza, A.M. Algoritmo Genético Cooperativo Paralelizado En Ambiente Grid Para El Problema de Ruteo Vehicular Con Ventanas de Tiempo, 2015.
21. Colorni, A.; Dorigo, M.; Maniezzo, V. Metaheuristics for High School Timetabling. *Comput. Optim. Appl.* **1998**, *9*, 275–298.
22. Amaro Junior, B.; Santos, M.C.; de Carvalho, G.N.; de Araújo, L.J.P.; Pinheiro, P.R. Metaheuristics for the Minimum Time Cut Path Problem with Different

Cutting and Sliding Speeds. *Algorithms 2021, Vol. 14, Page 305* **2021**, *14*, 305.

23. Bennell, J.A.; Oliveira, J.F. The Geometry of Nesting Problems: A Tutorial. *Eur. J. Oper. Res.* **2008**, *184*, 397–415.
24. Luo, Q.; Rao, Y. Improved Sliding Algorithm for Generating No-Fit Polygon in the 2D Irregular Packing Problem. *Math. 2022, Vol. 10, Page 2941* **2022**, *10*, 2941.
25. Bennell, J.A.; Oliveira, J.F. A Tutorial in Irregular Shape Packing Problems. *J. Oper. Res. Soc.* **2009**, *60*.
26. Bennell, J.A.; Song, X. A Beam Search Implementation for the Irregular Shape Packing Problem. *J. Heuristics* **2010**, *16*, 167–188.
27. Han, W.; Bennell, J.A.; Zhao, X.; Song, X. Construction Heuristics for Two-Dimensional Irregular Shape Bin Packing with Guillotine Constraints. *Eur. J. Oper. Res.* **2013**, *230*, 495–504.
28. López-Camacho, E.; Ochoa, G.; Terashima-Marín, H.; Burke, E.K. An Effective Heuristic for the Two-Dimensional Irregular Bin Packing Problem. *Ann. Oper. Res.* **2013**, *206*, 241–264, doi:10.1007/s10479-013-1341-4.
29. Martinez-Sykora, A.; Alvarez-Valdes, R.; Bennell, J.A.; Ruiz, R.; Tamarit, J.M. Matheuristics for the Irregular Bin Packing Problem with Free Rotations. *Eur. J. Oper. Res.* **2017**, *258*, 440–455, doi:10.1016/j.ejor.2016.09.043.
30. López-Camacho, E.; Terashima-Marin, H.; Ross, P.; Ochoa, G. A Unified Hyper-Heuristic Framework for Solving Bin Packing Problems. *Expert Syst. Appl.* **2014**, *41*, 6876–6889, doi:10.1016/j.eswa.2014.04.043.
31. Chernov, N.; Stoyan, Y.; Romanova, T. Mathematical Model and Efficient

Algorithms for Object Packing Problem. *Comput. Geom. Theory Appl.* **2010**, *43*, 535–553, doi:10.1016/j.comgeo.2009.12.003.

32. Wäscher, G.; Haußner, H.; Schumann, H. An Improved Typology of Cutting and Packing Problems. *Eur. J. Oper. Res.* **2007**, *183*, 1109–1130, doi:10.1016/j.ejor.2005.12.047.
33. Dyckhoff, H. A Typology of Cutting and Packing Problems. *Eur. J. Oper. Res.* **1990**, *44*, 145–159, doi:10.1016/0377-2217(90)90350-K.
34. Labrada-Nueva, Y.; Cruz-Rosales, M.H.; Rendón-Mancha, J.M.; Rivera-López, R.; Eraña-Díaz, M.L.; Cruz-Chávez, M.A. Overlap Detection in 2D Amorphous Shapes for Paper Optimization in Digital Printing Presses. *Mathematics* **2021**, *9*, 1–22, doi:10.3390/math9091033.
35. Hinxman, A.I. The Trim-Loss and Assortment Problems: A Survey. *Eur. J. Oper. Res.* **1980**, *5*, 8–18, doi:10.1016/0377-2217(80)90068-5.
36. Lodi, A.; Martello, S.; Vigo, D. Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. *INFORMS J. Comput.* **1999**, *11*, 345–357, doi:10.1287/ijoc.11.4.345.
37. Wu, Y.L.; Huang, W.; Lau, S.C.; Wong, C.K.; Young, G.H. An Effective Quasi-Human Based Heuristic for Solving the Rectangle Packing Problem. *Eur. J. Oper. Res.* **2002**, *141*, 341–358.
38. Bäck, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*; Oxford University Press, Inc.: USA, 1996; ISBN 0195099710.
39. Goldberg, D.E. Genetic Algorithms in Search Optimization and Machine Learning.; 1988.

40. Glover, F. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. Oper. Res.* **1986**, *13*, 533–549, doi:10.1016/0305-0548(86)90048-1.

41. Reeves, C.R. Modern Heuristic Techniques for Combinatorial Problems.; 1993.

42. Glover, F. *GLOVER, Fred. Handbook of Metaheuristics*; Fred W. Glover, G.A.K., Ed.; 2003; ISBN 978-0-306-48056-0.

43. Blum, C.; Roli, A. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.* **2003**, *35*, 268–308, doi:10.1145/937503.937505.

44. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science (80-.).* **1983**, *220*, 671–680.

45. Cerney, M.S. Countertransference Revisited. *J. Couns. Dev.* **1985**, *63*, 362–364.

46. Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **2004**, *21*, 1087.

47. Jessup, L.M.; Connolly, T.; Galegher, J. The Effects of Anonymity on GDSS Group Process with an Idea-Generating Task. *MIS Q. Manag. Inf. Syst.* **1990**, *14*, 313–321, doi:10.2307/248893

48. Van Laarhoven, P.J.M.; Aarts, E.H.L.; Lenstra, J.K. Job Shop Scheduling by Simulated Annealing. *Oper. Res.* **1992**, *40*, 113–125.

49. Leung, T.W.; Yung, C.H.; Troutt, M.D. Applications of Genetic Search and Simulated Annealing to the Two-Dimensional Non-Guillotine Cutting Stock

Problem. *Comput. Ind. Eng.* **2001**.

50. Rayward-Smith, V.; Osman, C.; Reeves, C.; Smith, G.D. Modern Heuristic Search Methods. **1996**.
51. Schwefel, H.P.; Rudolph, G. Contemporary Evolution Strategies. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* **1995**, *929*, 893–907.
52. Glover, F. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. Oper. Res.* **1986**, *13*, 533–549.
53. Laguna, M.; Glover, F. Bandwidth Packing: A Tabu Search Approach. *Manage. Sci.* **1993**, *39*, 492–500.
54. Taillard, E. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Comput.* **1991**, *17*, 443–455.
55. Battiti, R.; Protasi, M. Approximate Algorithms and Heuristics for MAX-SAT. *Handb. Comb. Optim.* **1998**, 77–148, doi:10.1007/978-1-4613-0303-9_2.
56. Burke, E.K.; Bykov, Y. A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems.; 2008.
57. Feo, T.A.; Resende, M.G.C. Greedy Randomized Adaptive Search Procedures. *J. Glob. Optim.* **1995**, *6*, 109–133.
58. Aiex, R.M.; Binato, S.; Resende, M.G.C. Parallel GRASP with Path-Relinking for Job Shop Scheduling. *Parallel Comput.* **2003**, *29*, 393–430.
59. Festa, P.; Resende, M.G.C. Grasp: An Annotated Bibliography. In *Operations Research/Computer Science Interfaces Series*; Springer, Boston, MA, 2002; pp. 325–367.
60. Resende, M.G.C.; Ribeiro, C.C. *Greedy Randomized Adaptive Search*

Procedures; Springer, Boston, MA, 2003.

61. Mladenović, N.; Hansen, P. Variable Neighborhood Search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100.
62. Fleszar, K.; Hindi, K.S. New Heuristics for One-Dimensional Bin-Packing. *Comput. Oper. Res.* **2002**, *29*, 821–839, doi:10.1016/S0305-0548(00)00082-4.
63. Hansen, P.; Mladenovic, N.; Andrés Moreno Pérez, J.; Sánchez, F.; Cruz de Tenerife, S. Variable Neighbourhood Search. *Rev. Iberoam. Intel. Artif. No* **2003**, *19*, 77–92.
64. Whittaker, J.K.; Garbarino, J. *Social Support Networks : Informal Helping in the Human Services*; Aldine Pub. Co, 1983; ISBN 0-202-36031-8.
65. Bräysy, O. A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows. *J. Comput.* **2003**, *15*, 347–368.
66. Burke, E.K.; Cowling, P.I.; Keuthen, R. Effective Local and Guided Variable Neighbourhood Search Methods for the Asymmetric Travelling Salesman Problem. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* **2001**, *2037*, 203–212, doi:10.1007/3-540-45365-2_21/COVER.
67. Hansen, P.; Mladenović, N. An Introduction to Variable Neighborhood Search. *Meta-Heuristics Adv. Trends Local Search Paradig. Optim.* **1999**, 433–458.
68. Lourenço, H.R.; Martin, O.C.; Stützle, T. Iterated Local Search: Framework and Applications. *Int. Ser. Oper. Res. Manag. Sci.* **2019**, *272*, 129–168.
69. Stützle, T.; Hoos, H. The Max-Min ANT System and Local Search for Combinatorial Optimization Problems. In *Meta-Heuristics: Advances and*

Trends in Local Search Paradigms for Optimization; Springer, Boston, MA, 1999; pp. 313–329.

70. Fogel, R.W. The New Economic History. I. Its Findings and Methods. *Econ. Hist. Rev.* **1966**, *19*, 642.
71. Rechenberg, H.; Billard, L.; Chamberod, A.; Natta, M. Champs Hyperfins et Modele Semi-Microscopique Non-Local de l'invar. *J. Phys. Chem. Solids* **1973**, *34*, 1251–1265.
72. Holland, J.H. Genetic Algorithms and Adaptation. *Adapt. Control Ill-Defined Syst.* **1984**, 317–333, doi:10.1007/978-1-4684-8941-5_21.
73. Reeves, C.R.; Rowe, J.E. *Genetic Algorithms—Principles and Perspectives*; Operations Research/Computer Science Interfaces Series; Springer US: Boston, MA, 2002; Vol. 20; ISBN 978-1-4020-7240-6.
74. Mühlenbein, H.; Mahnig, T.; Rodriguez, A.O. Schemata, Distributions and Graphical Models in Evolutionary Optimization. *J. Heuristics* **1999**, *5*, 215–247.
75. Larranaga, P.; Lozano, J.A. Estimation of Distribution Algorithms Applied To Combinatorial Optimization Problems Article in Inteligencia Artificial Revista Iberoamericana de Inteligencia Artificial · Thermal Camera View Project Development of a New Metaheuristic Algorithm for Combinatorial Optimization Based on Instance Reduction View Project. **2003**, doi:10.4114/ia.v7i19.722.
76. Kennedy, J.; Eberhart, R. Particle Swarm Optimization. *Proc. ICNN'95 - Int. Conf. Neural Networks* **1995**, *4*, 1942–1948, doi:10.1109/ICNN.1995.488968.
77. Gudise, V.G.; Venayagamoorthy, G.K. Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural

Networks. *2003 IEEE Swarm Intell. Symp.* **2003**, 110–117.

78. Glover, D.; Doidge, J.; Hardwick, B.; Wilkinson London, J. *Book Reviews*; SAGE PublicationsLondon, Thousand Oaks and New Delhi, 1999; Vol. 27; ISBN 0749424176.
79. Martí, R.; Lourenço, H.; Laguna, M. *Assigning Proctors to Exams with Scatter Search*; Springer, Boston, MA, 2000.
80. Campos, V.; Glover, F.; Laguna, M.; Martí, R. An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. *J. Glob. Optim.* **2001**, *21*, 397–414.
81. Dorigo, M.; Di Caro, G. Ant Colony Optimization: A New Meta-Heuristic. *Proc. 1999 Congr. Evol. Comput. CEC 1999* **1999**, *2*, 1470–1477.
82. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. Syst. Man, Cybern. Part B Cybern.* **1996**, *26*, 29–41.
83. Colorni, A.; Dorigo, M.; Maniezzo, V.; Trubian, M.; Leonardo Da Vinci, P. Ant System for Job-Shop Scheduling. *J. Oper. Res. Stat. Comput. Sci.* **1994**, *34*, 39–53.
84. Gambardella, L.M.; Dorigo, M. An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *J. Comput.* **2000**, *12*, 237–255.
85. Dorigo, M.; Gambardella, L.M. Ant Colonies for the Travelling Salesman Problem. *Biosystems* **1997**, *43*, 73–81, doi:10.1016/S0303-2647(97)01708-5.
86. Darwin, C. *On the Origin of Species, 1859*; Routledge, 2004; ISBN 9780203509104.

87. Mendel, G. EXPERIMENTS IN PLANT HYBRIDIZATION (1865). *Electron. Sch. Publ. Proj.* **1996**.

88. Eiben, A.E.; Raué, P.E.; Ruttkay, Z. Genetic Algorithms with Multi-Parent Recombination. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* **1994**, *866 LNCS*, 78–87, doi:10.1007/3-540-58484-6_252/COVER.

89. DeJong, T.M. A Comparison of Three Diversity Indices Based on Their Components of Richness and Evenness. *Oikos* **1975**, *26*, 222.

90. Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*; 2013.

91. Mahfoud, S.W. A Comparison of Parallel and Sequential Niching Methods. **1995**.

92. Puchinger, J.; Raidi, G.R.; Koller, G. Solving a Real-World Glass Cutting Problem. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* **2004**, *3004*, 165–176.

93. Dudek, A.Z.; Arodz, T.; Galvez, J. Computational Methods in Developing Quantitative Structure-Activity Relationships (QSAR): A Review. *Comb. Chem. High Throughput Screen.* **2006**, *9*, 213–228.

94. Herrera, F.; Lozano, M.; Verdegay, J.L. Tackling Real-Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. *Artif. Intell. Rev.* **1998**, *12*, 265–319, doi:10.1023/A:1006504901164/METRICS.

95. Rogers, A.; Prügel-Bennett, A. Genetic Drift in Genetic Algorithm Selection Schemes. *IEEE Trans. Evol. Comput.* **1999**, *3*, 298–303, doi:10.1109/4235.797972.

96. Goldberg, R.B.; Barker, S.J.; Perez-Grau, L. Regulation of Gene Expression during Plant Embryogenesis. *Cell* **1989**, *56*, 149–160.

97. Horner, A. *Genetic Algorithms and Computer-Assisted Music Composition Musical Timbre Perception View Project Reverberation and Music Emotion View Project*; 1991.

98. Reeves, C.R. Genetic Algorithms. **2010**, 109–139, doi:10.1007/978-1-4419-1665-5_5.

99. Berger, E.; Amor, H. Ben; Amor, B.; Vogt, D.; Jung, B. Towards a Simulator for Imitation Learning with Kinesthetic Bootstrapping Mining-RoX: Autonomous Robots in Underground Mining View Project ARIDuA-Autonomous Robots and Internet of Things in Underground Mining View Project Towards a Simulator for Imitation. In Proceedings of the Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS; 2008.

100. Kumar, M.; Husain, D.M.; Upreti, N.; Gupta, D. Genetic Algorithm: Review and Application. *SSRN Electron. J.* **2010**, doi:10.2139/SSRN.3529843.

101. Cruz-Chávez, M.A.; Martínez-Rangel, M.G.; Cruz-Rosales, M.H. *Accelerated Simulated Annealing Algorithm Applied to the Flexible Job Shop Scheduling Problem*; John Wiley & Sons, Ltd, 2017; Vol. 24. doi: 10.1111/itor.12195

102. Garófalo Gaibor, E.H. Estudio de Optimización Aplicando Algoritmos Genéticos En Estructuras Paramétricas Tipo Celosia de Acero., Quito : EPN, 2022., 2022.

103. Martínez-Bahena, B.; Cruz-Chávez, M.A.; Ávila-Melgar, E.Y.; Cruz-Rosales, M.H.; Rivera-Lopez, R. Using a Genetic Algorithm with a Mathematical

Programming Solver to Optimize a Real Water Distribution System. *Water 2018, Vol. 10, Page 1318* **2018**, *10*, 1318.

104. Shukla, A.; Pandey, H.M.; Mehrotra, D. Comparative Review of Selection Techniques in Genetic Algorithm. In Proceedings of the 1st International Conference on Futuristic Trends in Computational Analysis and Knowledge Management.; Institute of Electrical and Electronics Engineers Inc., July 10 2015; pp. 515–519.
105. Reca, J.; Martínez, J.; Gil, C.; Baños, R. Application of Several Meta-Heuristic Techniques to the Optimization of Real Looped Water Distribution Networks. *Water Resour. Manag.* **2008**, *22*, 1367–1379.
106. Ille R ', B.L.M.; Goldberg, D.E. Genetic Algorithms , Tournament Selection, and the Effects of Noise. *Complex Syst.* **1995**, *9*, 193–212.
107. Mendenhall, W.R.J.B.B.M.B. *Introducción a La Probabilidad Estadística*; Cengage Learning, 2010.
108. Gonzalez, S. *Programación Matemática*. Editorial Díaz Santos, 2001. ISBN 10: 8479785047

Apéndice A

Estructuras de Datos Utilizadas en el Algoritmo Genético.

Para darle solución al Problema de Empaquetamiento de Contenedores de dos Dimensiones Aplicado a la Optimización de Papel en Imprentas Digitales se hizo por medio de un Algoritmo Genético, fue necesaria la utilización de estructuras de datos que permitieran el uso ordenado y eficiente de la información, esto es, debido a que una estructura es una colección de uno o más tipos de elementos, donde cada uno puede ser de diferente tipo de dato.

A continuación, se explican cada una de las estructuras utilizadas en el Algoritmo Genético propuesto.

```
typedef struct mask
{
    int w, h;
    int *bits;
} mask_t;
```

La estructura `mask_t`, permite almacenar la información del largo y el ancho de una imagen, así como el número de pixeles que las representan, tanto los pixeles que conforman las figuras, como las que no lo conforman.

```
struct Shapes
{
    int x;
    int y;
    int w;
```

```
    int h;
    float angle;
    int totalArea;
    int totalAreaShape;
    int residualArea;
    ALLEGRO_BITMAP *image;
    mask_t *mask;
};
```

La estructura `Shapes,` permite almacenar la información del largo y el ancho de una imagen, las coordenadas de las posiciones x e y, el ángulo de rotación, el área de la hoja de papel, la suma de las áreas de todas las figuras y el área residual que es el resultado de la resta entre el área de la hoja de papel y la suma del área de todas las figuras que se introducen en la hoja de papel, también guarda una imagen, así como su máscara de la figura amorfa y de la figura amorfa rotada.

Apéndice B

Complejidad Temporal por pasos.

Una parte importante de un algoritmo es conocer su eficiencia, y para conocer su eficiencia se lleva a cabo el cálculo de su complejidad. Considerando que cualquiera de las instrucciones de un algoritmo puede ser considerada un paso, sin tomar en cuenta ciclos, condiciones y funciones, cuyos pasos están en función de las operaciones realizadas, [Martínez y Martín, 2003] dan una explicación correspondiente a la complejidad (coste) en pasos para algunas instrucciones comúnmente utilizadas.

- La complejidad de declaraciones de variables, constantes y comentarios es de 0 pasos.

- Para el caso de expresiones, el número de pasos es 1 solo sí no se incluyen llamadas a funciones, de lo contrario, el coste será el correspondiente a la función. En caso de asignaciones simples, es decir, correspondiente a una variable, el coste es de 1 paso.

- Para el caso de sentencias condicionales, por ejemplo:

```
if (<expresión1>) <sentencia 1> else <sentencia 2>
```

Donde, para calcular el número total de pasos, se suman los pasos correspondientes a la expresión 1 con sentencia 1, o bien, con sentencia 2, según sea el caso.

- La complejidad correspondiente a sentencias iterativas se encuentra en función del número de pasos requeridos para resolver las expresiones de control de las sentencias. Por ejemplo, para:

```
for (<asignación>;<expresión 1>; <expresión 2>)
```

La complejidad será el número total de pasos necesarios para resolver

```
<asignación>, <expresión 1>,  <expresión 2>
```

Lo mismo sucede para el caso de

```
while (<expresión 3) y do… while (<expresión 4>)
```

Estas sentencias tendrán el número de pasos correspondientes a expresión

3 y expresión 4 respectivamente, cuyos pasos serán contados cada vez que el flujo del programa incluya dichas sentencias.

- Un caso especial es para una sentencia condicional múltiple **switch**, donde el número de pasos será el número de comparaciones necesarias para buscar sentencia por sentencia aquella requerida. Por ejemplo:

```
switch (n)
{
   n=1: <sentencia>si n vale 1, el coste del switch es 1
   n=2: <sentencias>si n vale 2, el coste del switch es 2
   n=3: <sentencias>
```

```
   default: <sentencias> si n no es 1, ni 2, ni 3, el coste del switch es 3
}
```

- Llamadas a funciones: Constarán de un paso, salvo que dependan de algún parámetro que indique el tamaño del problema. En este caso el coste será el de la expresión correspondiente a la variable.

- En caso de una función recursiva, se debe tomar en cuenta el coste de las variables locales de la función.

A continuación, se muestra un ejemplo del cálculo de la complejidad.

```
long mult2 (int i)
{
        long aux = 0;
        int j, k;
        for (j = 0; j< i; j++)
            for (k = 0; k < i; k++)
                  aux = aux + 1;
    return (aux);
}
```

Para este caso, se tiene que:

Líneas 1, 2 y 3: 0 pasos

Línea 4: 3 pasos (1 – asignación, 1- comparación y 1 - incremento), realizados i veces

Línea 5: 3 pasos (1 – asignación, 1 – comparación y 1 - incremento), realizados $\mathbf{i^2}$ veces

Línea 6: 1 paso, ejecutado i2 veces

Línea 7: 1 paso

De acuerdo a esto, se tiene que la función temporal de ésta función es:

$$T(i): 3i + 3i^2 + i^2 + 1 = 4i^2 + 3i+1 \text{ pasos}$$

Teniendo como resultado un polinomio de segundo orden respecto a la entrada, por lo que la complejidad de la función es de $O(i^2)$.

Una vez entendido el proceso de cálculo de complejidad de los algoritmos se procede a calcular la complejidad del algoritmo de Búsqueda Local Iterada desarrollado en este trabajo de investigación.

w*h	m2 Ya que por convenio w = h
xover*yover	m2 Ya que por convenio w = h

void Mask_Clear(mask_t *m)	
{	
for(int i = 0; i < m->w * m->h; i++)	w*h
m->bits[i] = 0;	1
}	
T(n) = w*h = m2	

mask_t *Mask_Create(int w, int h)	
{	
mask_t *temp = new mask_t;	1
temp->w = w;	1
temp->h = h;	1
temp->bits = new int[w * h];	1
Mask_Clear(temp);	w*h
if (!temp)	1
return 0;	1
return temp;	1
}	
T(n) = 7 + w*h = 7 + m2	

void Mask_Fill(mask_t *m)	
{	
for(int i = 0; i < m->w * m->h; i++)	w*h
m->bits[i] = 1;	1
}	
T(n) = w*h = m2	

void Mask_Fill(mask_t *m)	
{	

for(int i = 0; i < m->w * m->h; i++)	w*h
m->bits[i] = 1;	1
}	
T(n) = w*h = m2	

mask_t *Mask_New(ALLEGRO_BITMAP *bmp)	
{	
mask_t *temp;	1
int h = al_get_bitmap_height(bmp);	1
int w = al_get_bitmap_width(bmp);	1
ALLEGRO_COLOR transColor = al_map_rgb(255, 0, 255);	1
ALLEGRO_COLOR pixel;	1
temp = Mask_Create(w,h);	7 + w*h
if (!temp)	1
return 0;	1
Mask_Clear(temp);	w*h
for(int i = 0; i < h; i++)	h
{	
for (int j = 0; j < w; j++)	w
{	
pixel = al_get_pixel(bmp, j, i);	1
if(!Color_Equiv(pixel, transColor) && !Transparent(pixel))	2
Mask_SetBit(temp, j, i);	1
}	
}	
return temp;	1
}	
T(n) = 16 + 2w*h + 3w*h = 16 + 5w*h = 16 + 5m2	

mask_t *Mask_New(ALLEGRO_BITMAP *bmp)	
{	
mask_t *temp;	1
int h = al_get_bitmap_height(bmp);	1
int w = al_get_bitmap_width(bmp);	1
ALLEGRO_COLOR transColor = al_map_rgb(255, 0, 255);	1
ALLEGRO_COLOR pixel;	1
temp = Mask_Create(w,h);	7 + w*h
if (!temp)	1
return 0;	1
Mask_Clear(temp);	w*h
for(int i = 0; i < h; i++)	h
{	
for (int j = 0; j < w; j++)	w
{	

pixel = al_get_pixel(bmp, j, i);	1
if(!Color_Equiv(pixel, transColor) && !Transparent(pixel))	2
Mask_SetBit(temp, j, i);	1
}	
}	
return temp;	1
}	
T(n) = 16 + 2w*h + 3w*h = 16 + 5w*h = 16 + 5m2	

void Mask_SetBit(mask_t *m, int x, int y)	
{	
m->bits[x * m->w + y] = 1;	1
}	
T(n) = 1	

void Mask_UnsetBit(mask_t *m, int x, int y)	
{	
m->bits[x * m->w + y] = 0;	1
}	
T(n) = 1	

void Mask_Delete(mask_t *m)	
{	
if(m->bits != NULL)	1
delete [] m->bits;	1
if(m != NULL)	1
delete m;	1
}	
T(n) = 4	

int Mask_Collide(const mask_t *a, const mask_t *b, int xoffset, int yoffset)	
{	
int xover = (a->w + b->w) / 2 - abs(xoffset);	4
int yover = (a->h + b->h) / 2 - abs(yoffset);	4
if (xover <= 0 || yover <= 0)	3
return 0;	1
int x1 = (a->w - xover) * ((xoffset < 0) ? 1 : 0);	5
int y1 = (a->h - yover) * ((yoffset < 0) ? 1 : 0);	5
int x2 = (b->w - xover) * ((xoffset < 0) ? 0 : 1);	5
int y2 = (b->h - yover) * ((yoffset < 0) ? 0 : 1);	5
for(int i = 0; i < xover; i++)	xover
{	
for(int j = 0; j < yover; j++)	yover
{	
if(a->bits[(x1 + i) * a->w + (y1 + j)] == 1 &&	6
b->bits[(x2 + i) * b->w + (y2 + j)] == 1)	5
return 1;	1
}	
}	
return 0;	1
}	

T(n) = 33 + 12*xover*yover = 33 + 12 m2	

mask_t *Mask_Rotate(ALLEGRO_BITMAP *bmp, float theta)	
{	
mask_t *temp;	1
int h = al_get_bitmap_height(bmp);	1
int w = al_get_bitmap_width(bmp);	1
ALLEGRO_COLOR transColor = al_map_rgb(255, 0, 255);	1
ALLEGRO_COLOR pixel;	1
temp = Mask_Create(w,h);	7 + w*h
if (!temp)	1
return 0;	1
Mask_Clear(temp);	w*h
for(int i = 0; i < h; i++)	h
{	
for (int j = 0; j < w; j++)	w
{	
int hwidth = w / 2;	1
int hheight = h / 2;	1
int xt = i - hwidth;	1
int yt = j - hheight;	1
float sinma = sin(-theta);	1
float cosma = cos(-theta);	1
int xs = (int)((cosma * xt - sinma * yt) + hwidth);	1
int ys = (int)((sinma * xt + cosma * yt) + hheight);	1
if(xs >= 0 && xs < w && ys >= 0 && ys < h)	1
{	
pixel = al_get_pixel(bmp, j, i);	
if(!Color_Equiv(pixel, transColor) && !Transparent(pixel))	2
{	
Mask_SetBit(temp, j, i);	1
}	
}	
}	
return temp;	1
}	
T(n) = 16 + 2w*h + 3w*h = 16 + 5w*h = 16 + 5m2	

int Color_Equiv(ALLEGRO_COLOR col1, ALLEGRO_COLOR col2)	
{	
return col1.r == col2.r && col1.g == col2.g && col1.b == col2.b && col1.a == col2.a;	1
}	
T (n) = 1	

int Transparent(ALLEGRO_COLOR col1)	
{	
return col1.a == 0;	1
}	
T (n) = 1	

```
void Mask_Draw(mask_t *m, int x, int y)                                  
{                                                                        
            x = x - m->w / 2;                                            3
            y = y - m->h / 2;                                            3
            for(int i = 0; i < m->w; i++)                                w
            {                                                            
                        for(int j = 0; j < m->h; j++)                    h
                        {                                                
                                    if(m->bits[i * m->w + j] == 1)       3
                                       al_put_pixel(x+i, y+j,            2
al_map_rgba_f(.75, 0, .75, .75));
                        }                                                
          }                                                              
}                                                                        
T(n) = 6 + 5w*h = 6 + m2
```

```
int Mask_Area_Pixels(mask_t *m)                                  
{                                                                
            int pixel = 0;                                       1
            for(int i = 0; i < m->w; i++)                        w
            {                                                    
                    for(int j = 0; j < m->h;                     h
j++)
                        {                                        
                                    if(m->bits[i                 3
* m->w + j] == 1)
                                                                 2
pixel = pixel + 1;
                        }                                        
             }                                                   
             return pixel;                                       1
}                                                                
T(n) = 2 + 5w*h = 2 + 5 m2
```

```
int getNewPositionX(int w)                                       
{                                                                
    int x = rand() % (WIDTH - w);                                3
    return x;                                                    1
}                                                                
T(n) = 4
```

```
int getNewPositionY(int h)                                       
{                                                                
    int y = rand() % (HEIGHT - h);                               3
    return y;                                                    1
}                                                                
T(n) = 4
```

```
int gettotalArea()                                               
{                                                                
    int totalArea = WIDTH*HEIGHT;                                2
    return totalArea;                                            1
}
```

T(n) = 3	

Shapes clone(Shapes a)	
{	
Shapes clone;	1
clone.h = a.h;	1
clone.y = getNewPositionY(clone.h);	4
clone.w = a.w;	1
clone.x = getNewPositionX(clone.w);	4
clone.image = a.image;	1
clone.mask = a.mask;	1
clone.totalArea = gettotalArea();	1
return clone;	1
}	
T(n) = 3	

int Sign()	
{	
float number = rand()%2 - 0.5;	3
if(number > 0)	1
return 1;	1
if(number < 0)	1
return -1;	1
Else	
return 0;	1
}	
T(n) = 8	

int Rand_LimitLower_To_LimitUpper(int LimitLower, int LimitUpper)	
{	
return LimitLower + (rand()/(RAND_MAX/(LimitUpper-LimitLower)));	1
}	
T(n) = 1	

int getNewPositionX2(int w)	
{	
return Rand_LimitLower_To_LimitUpper(0,WIDTH - w);	1
}	
T(n) = 1	

int getNewPositionY2(int h)	
{	
return Rand_LimitLower_To_LimitUpper(0,HEIGHT - h);	1
}	
T(n) = 1	

void cleanShapes()	
{	

for(int i = 0; i < SHAPES + 1; i++)	n
{	
shapes[i].x = 0;	1
shapes[i].y = 0;	1
shapes[i].w = 0;	1
shapes[i].h = 0;	1
shapes[i].totalArea = 0;	1
shapes[i].totalAreaShape = 0;	1
shapes[i].residualArea = 0;	1
shapes[i].image = NULL;	1
shapes[i].mask = NULL;	1
}	
}	
T(n) = 9n	

void savefileSolution()	
{	
FILE *file;	1
char * name = "S3.txt";	1
file = fopen(name,"w+");	1
fprintf(file,"%d\n", SHAPES);	1
for(int i = 0; i < SHAPES; i++)	n
{	
fprintf(file," %d ", shapes[i].x);	1
fprintf(file," %d ", shapes[i].y);	1
fprintf(file,"\n");	1
}	
fprintf(file," %d ", residualAreasTotal);	1
fclose(file);	1
}	
T(n) = 6 + 3n	

```
//Población inicial
void inicializarPoblacion()
{
	int psheet = 0;
	int mask_a = 0;
	int label = 0;

	ALLEGRO_BITMAP * batman = al_load_bitmap("batman.png");
	ALLEGRO_BITMAP * conejo = al_load_bitmap("conejo.png");
	ALLEGRO_BITMAP * nube = al_load_bitmap("nube.png");
	ALLEGRO_BITMAP * estrella = al_load_bitmap("estrella.png");

	arrayShapes[0].image = batman;
	arrayShapes[1].image = conejo;
	arrayShapes[2].image = nube;
	arrayShapes[3].image = estrella;

	for (int mk = 0; mk < 4; mk++)
```

```
	{
		arrayShapes[mk].mask = Mask_New(arrayShapes[mk].image);
	}

	printf("//.....Individuos.....//\n");

    for(int p = 0; p < POP_SIZE; p++)
    {
		for (int i = 0; i < GEN_NUMSHAPES && SHAPES < GEN_NUMSHAPES; i++, SHAPES++)
		{
			bool overlap = false;
			do
			{
				label = rand()%4;

				shapes[SHAPES] = arrayShapes[label];

				shapes[SHAPES].w = al_get_bitmap_width(arrayShapes[label].image);
				shapes[SHAPES].h = al_get_bitmap_height(arrayShapes[label].image);
				shapes[SHAPES].x = getNewPositionX(arrayShapes[label].w);
				shapes[SHAPES].y = getNewPositionY(arrayShapes[label].h);
				shapes[SHAPES].image = arrayShapes[label].image;
				shapes[SHAPES].mask = arrayShapes[label].mask;

				Poblacion[p].shapes_Individuos[SHAPES] = shapes[SHAPES];
				Poblacion[p].shapes_Individuos[SHAPES].w = shapes[SHAPES].w;
				Poblacion[p].shapes_Individuos[SHAPES].h = shapes[SHAPES].h;
				Poblacion[p].shapes_Individuos[SHAPES].x = shapes[SHAPES].x;
				Poblacion[p].shapes_Individuos[SHAPES].y = shapes[SHAPES].y;
				Poblacion[p].shapes_Individuos[SHAPES].image = shapes[SHAPES].image;
				Poblacion[p].shapes_Individuos[SHAPES].mask = shapes[SHAPES].mask;

				for (int j = 0; j < SHAPES; j ++)
				{

					if (Mask_Collide(shapes[SHAPES].mask, shapes[j].mask, shapes[SHAPES].x - shapes[j].x, shapes[SHAPES].y - shapes[j].y))
					{
						overlap = true;
						countOverlap++;
```

```
                                        break;
                                    }
                                    overlap = false;
                                }
                        }while(overlap);

                        psheet = gettotalArea();
                        mask_a = mask_a + Mask_Area(shapes[SHAPES].mask);
                        RA = psheet - mask_a;
                        porcientoRA = (RA*100)/(psheet);

                        Poblacion[p].RAREA = RA;

                        printf("%d %d %d %d %d %d %d %d\n", SHAPES, shapes[SHAPES].x,
shapes[SHAPES].y, psheet, mask_a, RA, label, porcientoRA);
                }

                cleanShapes();
                SHAPES = 0;
                psheet = 0;
                mask_a = 0;
                RA = 0;
                porcientoRA = 0;

                printf("//================================//\n");
        }
}

// cruzamiento de un punto
void cruzamiento(int indice_padre1, int indice_padre2, Shapes hijo[GEN_NUMSHAPES])
{
        int punto = rand() % GEN_NUMSHAPES;
        for(int i = 0; i <= punto; i++)
        {
                hijo[i] = Poblacion[indice_padre1].shapes_Individuos[i];
        }

        for(int i = punto + 1; i < GEN_NUMSHAPES; i++)
        {
                hijo[i] = Poblacion[indice_padre2].shapes_Individuos[i];
        }
}
```

```
// función de reparación para cada hijo después del cruzamiento
void repararHijo(Shapes hijo[GEN_NUMSHAPES])
{
```

```
	int label = 0;

	ALLEGRO_BITMAP * batman = al_load_bitmap("batman.png");
	ALLEGRO_BITMAP * conejo = al_load_bitmap("conejo.png");
	ALLEGRO_BITMAP * nube = al_load_bitmap("nube.png");
	ALLEGRO_BITMAP * estrella = al_load_bitmap("estrella.png");

	arrayShapesRepair[0].image = batman;
	arrayShapesRepair[1].image = conejo;
	arrayShapesRepair[2].image = nube;
	arrayShapesRepair[3].image = estrella;

	for (int mk = 0; mk < 4; mk++)
	{
		arrayShapesRepair[mk].mask = Mask_New(arrayShapesRepair[mk].image);
	}

	for (int i = 0; i < GEN_NUMSHAPES && SHAPES < GEN_NUMSHAPES; i++, SHAPES++)
	{
		bool overlap = false;
		do
		{
			hijo[SHAPES] = clone(test);

			label = rand()%4;

			hijo[SHAPES] = arrayShapesRepair[label];

			hijo[SHAPES].w =
al_get_bitmap_width(arrayShapesRepair[label].image);
			hijo[SHAPES].h =
al_get_bitmap_height(arrayShapesRepair[label].image);
			hijo[SHAPES].x = getNewPositionX(arrayShapesRepair[label].w);
			hijo[SHAPES].y = getNewPositionY(arrayShapesRepair[label].h);
			hijo[SHAPES].image = arrayShapesRepair[label].image;
			hijo[SHAPES].mask = arrayShapesRepair[label].mask;

			for (int j = 0; j < SHAPES; j ++)
			{
				if (Mask_Collide(hijo[SHAPES].mask, hijo[j].mask,
hijo[SHAPES].x - hijo[j].x, hijo[SHAPES].y - hijo[j].y))
				{
					overlap = true;
					break;
				}
				overlap = false;
			}
		} while(overlap);
```

```
	}
}
```

```
// mutacion
void mutacion()
{

		for (int i = 0; i < GEN_NUMSHAPES && SHAPES < GEN_NUMSHAPES; i++, SHAPES++)
	{
		bool overlap = false;
		do
		{
			ALLEGRO_TRANSFORM t;
			al_identity_transform(&t);
			shapes[SHAPES].image = test.image;
			shapes[SHAPES].w = al_get_bitmap_width(shapes[SHAPES].image);
			shapes[SHAPES].h = al_get_bitmap_height(shapes[SHAPES].image);
			shapes[SHAPES].x = getNewPositionX(shapes[SHAPES].w);
			shapes[SHAPES].y = getNewPositionY(shapes[SHAPES].h);
			shapes[SHAPES].angle = getAngle();
			al_translate_transform(&t, -shapes[SHAPES].x - shapes[SHAPES].w/2, -shapes[SHAPES].y - shapes[SHAPES].h/2);
			al_rotate_transform(&t, shapes[SHAPES].angle*ALLEGRO_PI/180);
			al_translate_transform(&t, shapes[SHAPES].x + shapes[SHAPES].w/2, shapes[SHAPES].y + shapes[SHAPES].h/2);

			al_use_transform(&t);

			shapes[SHAPES].mask = Mask_Rotate(shapes[SHAPES].image, shapes[SHAPES].angle*ALLEGRO_PI/180);

			for (int j = 0; j < SHAPES; j ++)
			{
				if (Mask_Collide(shapes[SHAPES].mask, shapes[j].mask, shapes[SHAPES].x - shapes[j].x, shapes[SHAPES].y - shapes[j].y))
				{
					overlap = true;
					countOverlap++;
					break;
				}
				overlap = false;
			}
		}while(overlap);

		int mask_a = Mask_Area(shapes[SHAPES].mask);
		int psheet = gettotalArea();
		RA = psheet - SHAPES*mask_a;
```

```
            porcientoRA = (RA*100)/(psheet);
            printf("%d %d %d %f %d %d %d %d %d\n", SHAPES, shapes[SHAPES].x,
shapes[SHAPES].y, shapes[SHAPES].angle, psheet, mask_a, RA, countOverlap
,porcientoRA);


      }
}
```

```
// retorna el área residual
int obtenerPuntuacion(Shapes individuo[GEN_NUMSHAPES])
{
      int ra = 0;
      int mask_a = 0;
      int area_shapes = 0;
      int psheet = gettotalArea();

      for(int i = 0; i < GEN_NUMSHAPES; i++)
      {
            area_shapes = Mask_Area(individuo[i].mask);
            mask_a = mask_a + area_shapes;
            ra = psheet - mask_a;
      }
      return ra;
}
// retorna el índice del mejor individuo de la población
int obtenerMejor()
{
      int indice_mejor = 0;
      int score_mejor = obtenerPuntuacion(Poblacion[0].shapes_Individuos);

      for(int i = 1; i < POP_SIZE; i++)
      {
            int score = obtenerPuntuacion(Poblacion[i].shapes_Individuos);
            if(score < score_mejor)
            {
                  indice_mejor = i;
                  score_mejor = score;
            }
      }

      return indice_mejor;
}
```

```
void AG()
{
	srand(time(NULL));

	inicializarPoblacion();

	for(int i = 0; i < generaciones; i++)
	{
		for(int j = 0; j < tam_torneo; j++)
		{
			printf("//Calcula la probabilidad de cruzamiento//\n");

			double prob = ((double) rand() / ((double)RAND_MAX + 1));

			if(prob < prob_cruz)
			{
				printf("//Escoge dos padres//\n");

				int indice_padre1 = rand() % POP_SIZE;

				int indice_padre2;

				printf("//Garantiza que los indices de los padres no son iguales//\n");

				do
				{
					indice_padre2 = rand() % POP_SIZE;
				}
				while(indice_padre1 == indice_padre2);

				Shapes hijo[GEN_NUMSHAPES];

				printf("//Aplica el cruzamiento de un punto//\n");

				cruzamiento(indice_padre1, indice_padre2, hijo);

				printf("//Repara hijo despues del cruzamiento//\n");

				repararHijo(hijo);

				printf("//Calcula la probabilidad de mutacion//\n");

				prob = ((double) rand() / ((double)RAND_MAX + 1));

				if(prob < prob_mut)
				{
```

```
                                    mutacion(hijo);
                              }

                              int score_padre =
obtenerPuntuacion(Poblacion[indice_padre1].shapes_Individuos);

                              int score_hijo = obtenerPuntuacion(hijo);

                              if(score_hijo >= score_padre)
                              {
                                    for(int k = 0; k < GEN_NUMSHAPES; k++)

      Poblacion[indice_padre1].shapes_Individuos[k] = hijo[k];
                              }
                        }
                  }
                  printf("Generacion: %d\t", i + 1 );

                  printf("Mejor: \n");

                  int indice_mejor = obtenerMejor();

                  int score_mejor =
obtenerPuntuacion(Poblacion[indice_mejor].shapes_Individuos);

                  for(int j = 0; j < GEN_NUMSHAPES; j++)
                  {
                        printf("%d
%d\n",Poblacion[indice_mejor].shapes_Individuos[j].x,Poblacion[indice_mejor].shapes_
Individuos[j].y);
                  }

                  printf("Puntuacion: %d\n",score_mejor);
      }
}
```

Glosario de Términos.

Algoritmo Determinístico: Es un algoritmo en el que una misma entrada obtiene siempre el mismo resultado.

Algoritmo no Determinístico: Algoritmo en el que a una misma entrada se obtienen diferentes salidas, de modo que no es posible saber de manera previa, el resultado que de la ejecución de un algoritmo de este tipo.

Algoritmo Greedy: Un algoritmo voraz, también conocido como ávido, devorador o goloso es aquel que, para resolver un determinado problema, sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. [Brassard, Gilles, Bratley y Paul, 1997].

Algoritmo Secuencial: Algoritmo programado de manera estructurada que al ser ejecutado utiliza un solo procesador.

Allegro 5: es una biblioteca libre y de código abierto para la programación de videojuegos desarrollada en lenguaje C. Allegro es un acrónimo de «Atari Low Level Game Routines» (rutinas de bajo nivel para videojuegos).

Análisis de Sensibilidad: Evaluación del comportamiento de las variables críticas de un problema con la finalidad de establecer un rango numérico, dentro del cual, la solución obtenida por el algoritmo sigue siendo buena, además de que permite conocer que tan sensible es el algoritmo a cambios en los valores de ciertas variables propias del problema.

API: Interfaz de programación de aplicaciones.

Benchmark: Palabra del inglés utilizada para nombrar a los problemas de prueba utilizados por diversos investigadores, para medir el rendimiento de un sistema o algoritmo para un problema dado.

Búsqueda Local: Técnica iterativa de que permite explorar el espacio de soluciones de un problema dado, a partir de una solución inicial, por medio de movimientos, de modo que vaya mejorando la solución obtenida de acuerdo a la función objetivo. Este tipo de técnicas son utilizadas para mejorar la calidad de las soluciones obtenidas por un algoritmo, así como para reducir el tiempo en que se obtienen dichas soluciones.

Búsqueda Tabú (TS): Técnica basada en la inteligencia artificial, empleando el concepto de memoria e implementándolo mediante estructuras simples, con el objetivo de dirigir la búsqueda de la solución final en función de los resultados alcanzados [Glover, 1989]. TS considera dos tipos de memoria que interaccionan entre sí, a corto y largo plazo respectivamente. Este tipo de datos darán lugar a estrategias de intensificación y/o diversificación dentro del ámbito local o global.

Complejidad Espacial: Es la cantidad de memoria requerida por el algoritmo durante la ejecución.

Complejidad Temporal: Es el número de pasos necesarios (tiempo) para obtener una solución a una instancia de un problema dado.

Valor de Aptitud: Para el Problema de Optimización de Papel en Imprentas, son las unidades de tiempo requeridas para poder insertar una o varias figuras.

Direct3D: Conjunto de bibliotecas para multimedia, propiedad de Microsoft, consiste en una API para la programación de gráficos en tres dimensiones.

Estructura de Vecindad: Tipo de movimiento (permutación, inserción o eliminación) utilizado para explorar el espacio de soluciones de un problema de

optimización.

GRASP: (Greedy Randomized Adaptative Search Procedures). Es una procedimiento iterativo que consiste en una fase de construcción y una de mejora [Feo y Resende, 1989]. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial, considerando una lista restringida de los mejores candidatos y seleccionando un elemento de dicha lista, de forma aleatoria. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local.

Heurística: Procedimiento intuitivo bien definido que proporciona buenas soluciones aproximadas a problemas difíciles de resolver sin garantizar la optimalidad, en un tiempo computacional razonable [Wetzel, 1983].

Lista Tabú: Lista utilizada en el algoritmo Colonia de Hormigas, la cual contiene los nodos (dependiendo del problema, en el caso de UPMP, son los trabajos) visitados por cada hormiga, misma que tiene la función de evitar que un nodo sea visitado más de una vez. Ya que la lista tabú se ha completado, ésta contendrá la solución construida por la hormiga a un problema dado.

Metaheurísticas: Métodos aproximados que mejoran procedimientos heurísticos, los cuales son diseñados para ser aplicados a problemas considerados difíciles de resolver, donde las heurísticas no son eficientes [Osman y Kelly, 1996].

Modelo de Programación entera binaria: El formato del modelo de programación entera binaria requiere tener una función objetivo lineal, ya sea a minimizar o maximizar. La característica de este modelo, es que su conjunto de variables solo puede tomar valores binarios, es decir, 0s y 1s.

Óptimo Global: Es la mejor solución de un espacio de soluciones f(x).

Óptimo Local: Representa la mejor solución de f(x) en un entorno x.

Recocido Simulado (SA): Es una metaheurística de búsqueda aleatoria utilizada en la solución de problemas de optimización combinatoria [Kirkpatrick, 1983]. Se basa en la analogía del proceso de recocido, es cuando un material se somete a un calentamiento a temperatura muy alta llegando al punto de fusión y después es enfriado gradualmente, sus moléculas se acomodan de tal forma que la energía potencial de la configuración de las moléculas es mínima.

OpenGL: Open Graphics Library es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos en dos y tres dimensiones.

Óptimo Global: Es la mejor solución de un espacio de soluciones f(x).

Óptimo Local: Representa la mejor solución de f(x) en un entorno x.

Punto fijo: El punto fijo de una función es un punto cuya imagen producida por la función es el mismo.

Sintonización: Es la proporción adecuada en cuanto a los valores obtenidos mediante el análisis de sensibilidad aplicado a los parámetros de control, tomando en cuenta el problema y el método de optimización utilizado, de modo que el algoritmo muestre una mejora tanto en eficiencia como en eficacia.

Tiempo polinomial: En las ciencias computacionales, el tiempo viene expresado generalmente en función del tamaño de la entrada. Se considera que un algoritmo puede ser resuelto en tiempo polinomial sí su función de complejidad es de orden O (p(n)), es decir, que puede ser representado por un polinomio.

Vecindad: Es el conjunto de soluciones que pueden ser alcanzadas desde una solución dada por medio de un movimiento (inserción, eliminación, permutación).

Problema clase P: Es un conjunto de todos los problemas de decisión que pueden ser resueltos por un algoritmo determinístico en tiempo polinomial.

Instancia*:* *E*s un problema de prueba que puede ser definido como el tamaño de entrada de los datos del problema.

Cuernavaca, Morelos, 11 de mayo de 2023.

DR. JUAN CARLOS GARCÍA CASTREJÓN
COORDINADOR DEL POSGRADO
EN INGENIERÍA Y CIENCIAS APLICADAS
P R E S E N T E

Atendiendo a la solicitud para emitir DICTAMEN sobre la revisión de la TESIS titulada: ALGORITMO GENÉTICO PARA EL PROBLEMA DE EMPAQUETAMIENTO DE CONTENEDORES DE DOS DIMENSIONES, APLICADO PARA LA OPTIMIZACIÓN DE PAPEL EN IMPRENTAS DIGITALES, que presenta el alumno **YAINIER LABRADA NUEVA**, para obtener el título de **DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS**.

Nos permitimos informarle que nuestro voto es **APROBATORIO.**

Atentamente
Por una humanidad culta
Una universidad de excelencia

DR. ÁLVARO ZAMUDIO LARA
DRA. MARGARITA TECPOYOTL TORRES
DR. MARTÍN HERIBERTO CRUZ ROSALES
DRA. ALINA MARTÍNEZ OROPEZA
DRA. MARTA LILIA ERAÑA DÍAZ
DR. JUAN MANUEL RENDON MANCHA
DR. MARCO ANTONIO CRUZ CHAVEZ

Se adiciona efirma UAEM

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento firmado electrónicamente de conformidad con el ACUERDO GENERAL PARA LA CONTINUIDAD DEL FUNCIONAMIENTO DE LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS DURANTE LA EMERGENCIA SANITARIA PROVOCADA POR EL VIRUS SARS-COV2 (COVID-19) emitido el 27 de abril del 2020.

El presente documento cuenta con la firma electrónica UAEM del funcionario universitario competente, amparada por un certificado vigente a la fecha de su elaboración y es válido de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DE ESTADO DE MORELOS emitidos el 13 de noviembre del 2019 mediante circular No. 32.

Sello electrónico

ALVARO ZAMUDIO LARA | Fecha:2023-05-11 12:43:49 | Firmante

EXDNojcZPdiuTQGTpg7iH8lwJirxYJ7By9EBOZJSeBfkFhYYyJy++ml1ShSmEyr8WHxyfbUucTeYDHOy4iQJfJ1ydFqKqXyzS4Qb6LC7pyvFljEO5cRpLuhrcm5VY0YO3ALuoOaPkyff3XEdr+BD1sc7JdX6fgSg6ZHkSVBszObnzGGITDU9wD4URJAKhBgfthPyU0/SQy1LkIj0SqzoB1Rs+qR9VSWtCWqQGe+5joFooumEF8fjKLs9TfxqusBEPFWFQZ/sDKdGUm+Uo9LbUdhluzfHD/ueqdFv6cdqHx2J41vKdV49CdhVc+zkw5DFUE5vadyu5EH7qTeVuYLYRA==

MARGARITA TECPOYOTL TORRES | Fecha:2023-05-11 14:53:57 | Firmante

fEqJE3O93OKyrLZijWfqm9BZ4VRvYFbUTsTnK+ajlZl4oj/k3phd0sBO6wlmU91NvRfyC2kizZU14MjnNomKWTJFqj0n0TmGDYYBgAURRKOxmwi7+1NSYaH5ek7Is/t3vVXnnZFl2YkpFiGeVhgqUYEm3KgE7CmPQ4dx/fHKLMzsvK5f4rzisyy2CIzBDX0gSYJwuUcwJyRblrqadRWD4FNTq8h3OHsLZvkgmnhrvG7DMd5aflcjf6KoeQ06gEhhPw+fb/X8cCYS6F46UJtTU3M30fPeH5C01vfzikSfd5e5+11Z1H76aukX2ftqfDInU75kyvqbYl23hUvW1Qkvfg==

MARCO ANTONIO CRUZ CHAVEZ | Fecha:2023-05-11 16:17:51 | Firmante

dtwmq03lLs6NRWo6inEjQ9k2Q34JgffpVbYWcRdzxBj0OLH9yokJnE7wgN88Ht9yByr9VUyAgxHH5UHi4boD/tcWb8M+gTwkOkm7jBnojRwtMknH2HZ0egi/4UeZ8BgsAVVT/OVn62BZsLL6ywlyYstonHVhJVe2F3m2MTi0DdmwryiEylTER14j5EO3vnsYLXV2rd/brmoqx9CA3mAc7s49qpCjL1fpw0OFtvKIlM7djXoP+LYy0l/VZnZ0NVmn38kJNvgtx7Da8LN3BPwI7AFD2MXQQ8YBzwXQfuc/WA47wcJblQp3OkYCe5wmxj5QevGqEsQ908SftV1kTHuLKA==

JUAN MANUEL RENDON MANCHA | Fecha:2023-05-11 21:00:41 | Firmante

L2CJ7/PeiI/RwKTgsqEtufMr0iIQnQ97Pf+BLNf1GsTjXMjnadbpQ25iuPO295mdrzqNUsbbIwJQzl9ENyvfC+bw3vNj/remkmvvsjVYtFOgJqLedkKC7VpY9oMF64X3SLgDg2rE/KBtDgDUS/m8BJsz+t4s6Pi4Go1Usz2mQm65VCEm3kighS1GsYZPO3+dfnzrk0mR+3fzbbvMlqFS8sBTvIJ4/V2LsGX7mJXUi/G8CiynauT+qRhyWCcFmaAuKf/b/2tDi+9rMqkxWU80Znfm7saxNnjdAvEy1c0M52FNa6Rmxk5PgIxpZtTTHqHCsWrThBAGpKGKFXg1rhn2ug==

MARTA LILIA ERAÑA DIAZ | Fecha:2023-05-11 21:15:41 | Firmante

BllqAD1VElOxjDWvtDeEWSypUk3JSt4gz0Pax7qOSkcmBDpMOahff8ja5xISZ0fBHQo90PnePN7ymqgNfbIlW0n4HwwfPS4iBSk8FUvGQ2yftEcvG8TWULPY+WJ4m5Y59pDlRPjGPaHV2KpjnpD+GMn0ufLW9oin9pnUO6TaSkQ7rvnmhrlJzEaSyXUli6r7fglqtZwE779Ud3jVINxoJoV8XVnIQ1LmjiG9We6/GxZyajswq5oPW/YNJp4oRrX7ttXnyIlcKL4R2HyWIHzU+VNCK/0mDyQKUrufa5IWe0BbIDZibx/iU//06Tgmo6guQrmIPSfSyfCZ/RLvCsrEvg==

ALINA MARTINEZ OROPEZA | Fecha:2023-05-12 12:19:04 | Firmante

aH2Q1AhHJNNOX9W3lRal0JJxx7YFSKzIolRKT2SEVugtjX2hQBSoOKbZex4yXI0dTk/vphl5mjwGjpr78/G88zdzJxoo41uCzc1P0ajr9Jny6tQenmX9VQPvY3zX7hg16KqDzp5dnRkvHGRrVhl0KHkYjmEWooXmsRg9EArWko47LD2u1MMxBZ+XUznyNseNrCbpKyWX8PF5r6FYRQvMefuzm6gca1Xms9nTSEDNrvx78lhkZhlGocO/bh7ug5KF1PYfguwrq4yzMRsKQvW4BszbvbTiBY0oPS1JVyY7f4qO2xMWKd5Dl7BlIgcZ85UpUAaKyPW10A0/FyMetxntAg==

MARTIN HERIBERTO CRUZ ROSALES | Fecha:2023-05-12 15:43:54 | Firmante

WjyiJgAWjAw5vJMUxmNf/e8rpY95vLn0ELdrpBEf0pqv7klvyLgmyjEusuK1NU9gOHq9mcMjjDkhWmMAaK4YoVA2o2YH4lyRytcCx901kRCTsk6Sw8AWS0ArnRKOj+Vz6GbRYp/bDxA/d4pjY7dx0VSgmhgZ+uEOiA4gi4eJ1GC3W1kg2yHVIkKCIO1sVaHitAlX6Fj5wOhceFvajjvZ605nMEBH32YfnsqLeDU96ZyybVNjszvvfAVMr+EA9vew7u4/zoTfizhAFK+xdWAkFhXp0wcRV0kCEavlXg3OS8KS/bQJq+ZdgfYTonWwV7gC73crAi+ygVE2Mwglqrp+Uw==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

MTdt1HzJa

https://efirma.uaem.mx/noRepudio/ZkgfjrtttuHjeNSXNerntpgCtLkMgOgR