UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN INGENIERÍA
Y CIENCIAS APLICADAS

HEURÍSTICA COMPUTACIONAL PARA LA OPTIMIZACIÓN DE PAPEL EN IMPRENTAS

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE:

MAESTRÍA EN INGENIERÍA Y CIENCIAS APLICADAS

P R E S E N T A

I.C.I. YAINIER LABRADA NUEVA

ASESOR: DR. MARCO ANTONIO CRUZ CHÁVEZ
COASESOR: DR. JUAN MANUEL RENDÓN MANCHA

CUERNAVACA, MOR. ENERO 2016

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

CIICAp

Av. Universidad 1001 Col. Chamilpa, Cuernavaca, Morelos, México, CP 62209
Conmutador 01 (777) 329 70 00 Ext. 6242, Tel. 01 (777) 329 70 84, Fax. 01 (777) 329 79 84

OFICIO N°029/2015
ASUNTO: APROBACIÓN DE TESIS

C. LABRADA NUEVA YAINIER
P R E S E N T E

Por este conducto le notifico que su tesis de Maestría titulada,

"Heurística Computacional para la Optimización de papel en Imprentas"

Fue aprobada en su totalidad por el jurado revisor y examinador integrado por los ciudadanos

NOMBRE	FIRMA
DRA. MARGARITA TECPOYOTL TORRES	
DR. MARTÍN HERIBERTO CRUZ ROSALES	
DR. MARTÍN GERARDO MARTÍNEZ RANGEL	
DR. MARCO ANTONIO CRUZ CHÁVEZ (Asesor)	
DR. JUAN MANUEL RENDÓN MANCHA (Co-Asesor)	

Por consiguiente, se autoriza a editar la presentación definitiva de su trabajo de investigación para culminar en la defensa oral del mismo.

Sin otro particular aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E
"POR UNA HUMANIDAD CULTA"

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS
JEFATURA DE POSGRADO EN INGENIERÍA Y CIENCIAS APLICADAS
CIICAp

DR. JOSÉ ALFREDO HERNÁNDEZ PÉREZ
JEFATURA DE POSGRADO DE INGENIERÍA
Y CIENCIAS APLICADAS

DISTINTIVO Empresa Incluyente Gilberto Rincón Gallardo

Resumen

En el presente trabajo de investigación se aplica un algoritmo de Búsqueda Local para resolver el problema de empaquetamiento en contenedores en dos dimensiones, aplicado a la optimización del papel en imprentas. Para dar solución a este problema, se realizó un estudio sobre geometría computacional para evitar el traslape entre los objetos que se empaquetan, y así poder optimizar la función objetivo cumpliendo con las restricciones del problema, además se estudiaron diferentes estructuras de vecindad, con el objetivo de mejorar el desempeño del algoritmo.

Se aplicó una metodología de sintonización, la cual permitió realizar el análisis de sensibilidad de los parámetros de control del algoritmo de Búsqueda Local Iterada, lo que le permitió al algoritmo trabajar 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 en dos dimensiones al problema del desperdicio de papel en una imprenta y tratado con una heurística computacional.

Las pruebas experimentales realizadas a la implementación, mostraron que es posible lograr optimizar el papel en las imprentas.

Abstract

In the present research, local search algorithm is applied to solve the problem of packing in containers in two dimensions, applied to the optimization in printing paper. To solve this problem, a study on computational geometry was made to avoid overlapping between objects that are packaged, so we can optimize the objective function meeting the constraints of the problem, also different structures of neighborhood studied, aiming to improve the performance of the algorithm.

Tuning methodology was applied, which allowed for the sensitivity analysis of the control parameters iterated local search algorithm, which allowed the algorithm to work with the best performance in efficiency and effectiveness.

The contribution of this research is the implementation of a mapping of the problem of packing in two dimensions, the problem of waste paper in a printing and treated with a computational heuristic.

The experimental tests to implementation, showed that it is possible to achieve optimization in printing paper.

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. Juan Manuel Rendón Mancha, Dr. Martín Gerardo Martínez Rangel, Dr. Martín Heriberto Cruz Rosales, Margarita Tecpoyotl Torres, por sus comentarios, orientación y consejos para la realización de este trabajo de investigación.

Dedicatorias

A TODOS.

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

A los Doctores del Hospital Clínico Quirúrgico “Dr. Juan Bruno Zayas Alfonso” de Santiago de Cuba, Cuba, especialmente a la Dra. Haydee Leyva Díaz y al Dr. Adolfo Rafael Jaen Oropeza, que fueron los que me operaron y gracias a ellos pude seguir adelante con este proyecto tan importante para mi vida.

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

A los Doctores Rendón, Martín Gerardo, Martín Heriberto, Pecina, David Juárez, Héctor Castro, Jesús Castrellón, Grimalsky por mi formación.

A mi novia Candy la tucky-tucky por estar conmigo todo este tiempo.

Al Dr. Diego Seuret Jiménez por ser como un padre para mi en estos tiempos y a su esposa Rossy.

A todos mis amigos de la Universidad, Luis, Michel, Ismar, Yamy, Reynier, Alina, Dayana, Irving, Yadian, Ilmaris, León, Yisel, Dairon y El Titi entre otros más.

A Fernando por ser como un hermano para mi y a su esposa Paula.

A Alexeis Companioni por ser un gran amigo y ejemplo como profesional.

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, Rogelio,Carmen, Pedro, Luis Dévora, Marco, Juanita, Alfonso, Jazmin, Betty, Ariadna, Mara, Alán, Mireya, Erika, Roberto, Roy Susy, Armando, Arelis y a otros que no aparecen porque la lista sería interminable. A todos los aprecio y estimo.

A Don Victor y familia por su calida acogida, a Alfredo por ser mi amigo, al primo Javier y todos los que de una forma u otra han compartido esta etapa de mi vida.

Nomenclatura General

GA Siglas en inglés para los Algoritmos Genéticos (Genetic Algorithm).

SA Siglas en inglés para el algoritmo de Recocido Simulado (Simulated Annealing).

BPP Siglas en inglés para el Problema de Empaquetamiento (Bin Packing Problem).

GRASP Siglas en inglés para Procedimientos Voraces de Búsqueda Adaptables Aleatorizados (Greedy Randomized Adaptive Search Procedures).

ILS Siglas en inglés para el algoritmo de Búsqueda Local Iterada. (Iterated Local Search).

TS Siglas en inglés para el algoritmo de Búsqueda Tabú (Tabu Search).

VNS Siglas en inglés para el algoritmo de búsqueda con vecindario variable (Variable Neighborhood Search).

Parámetros de Control de la Búsqueda Local Iterada

S Solución Inicial

S' Solución Inicial perturbada.

Sm-ILS Solución mejorada con la Búsqueda Local Iterada

Contenido

Índice de Figuras

Índice de Tablas

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 empaquetamiento, son, en general, difíciles de resolver en la práctica. Estos problemas están incluidos en la clase de problemas NP-Completo [Garey y Johnson, 1979], 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 las ciencias computacionales, dedicada a la investigación de operaciones así como al estudio y tratamiento de problemas considerados difíciles de resolver [Papadimitriou y Steiglitz, 1998]; el objetivo a seguir en esta área es básicamente el desarrollo o mejora de métodos novedosos que permitan abordar problemas de optimización con el menor esfuerzo computacional posible. La característica principal de un problema de optimización

es la búsqueda de la mejor solución mediante la implementación de métodos que permitan reducir la dificultad 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 [Hillier, Lieberman, 2008].

Todos los problemas considerados en la literatura por la Optimización Combinatoria pueden ser clasificados de acuerdo al grado de dificultad para resolverlos, de aquí surge la teoría de la complejidad, la cual se encarga de dar una clasificación a estos problemas, dividiéndolos en P, NP y NP-Duros.

Los problemas P, son aquellos que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial, es decir, que la relación entre el tamaño del problema y su tiempo de ejecución es polinómica. Los problemas NP sólo pueden ser tratados por una máquina de Turing no determinista acotada en tiempo polinomial. [Garey y Johnson, 1979]. Una de las características de estos problemas, es que el tamaño de su espacio de soluciones es de comportamiento exponencial conforme se incrementa el tamaño de la instancia [Pinedo, 2008]. Los problemas NP-Duros presentan las mismas características que los NP, pero se diferencian en que estos son aún más difíciles de tratar y sus instancias resueltas hoy en día son más pequeñas en comparación con las resueltas para problemas NP. El problema que se está analizando surge en una imprenta y el objetivo fundamental de esta investigación es disminuir el desperdicio del papel. Este problema se enmarca dentro de la Optimización Combinatoria y tiene como nombre problema de empaquetamiento en contenedores y está clasificado desde el punto de vista computacional como NP-Completo [Garey y Johnson, 1979].

Para darle cumplimiento al objetivo de este trabajo de investigación se propone un algoritmo de Búsqueda Local Iterada, para el cuál se desarrolló una estructura de

vecindad que realiza pequeños movimientos a las figuras con el fin de lograr insertar más, además se valida que no existan traslapes entre ellas, permitiendo mejorar la explotación del espacio de soluciones.

Dentro de las pruebas experimentales fue necesario llevar a cabo un análisis de sensibilidad de los parámetros de control del algoritmo, para mejorar la eficacia del algoritmo propuesto.

La siguiente sección presenta una clasificación de los métodos de optimización más conocidos, así como una breve explicación de cada uno de ellos.

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 [Papadimitriou y Steiglitz, 1998], 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 [Wetzel, 1983]. 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 exactos, se puede mencionar que el Simplex Clásico [Hillier, Lieberman, 2008] 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 [Klee y Minty, 1972] que su complejidad es de orden exponencial, 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.

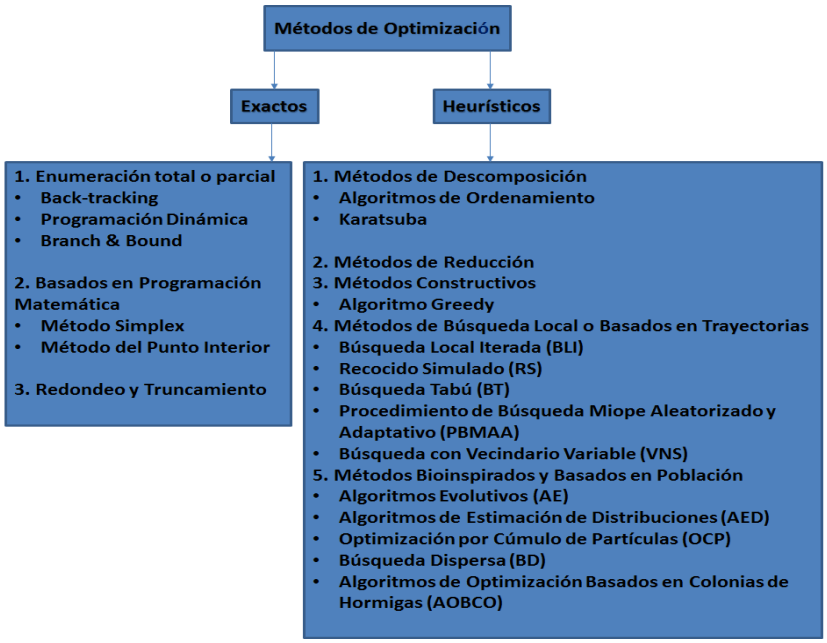


Figura 1.1. *Clasificación de los métodos de optimización. [Martínez, 2010] y [Salto, 2010].*

Debido a esto, muchos investigadores se han visto atraídos al estudio y 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 un problema de optimización bien definido, 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 [Riojas, 2005] para instancias medianas y grandes de problemas clasificados como difíciles. Es por esto, que surgen las denominadas metaheurísticas, las cuales son una clase de métodos aproximados que están diseñados para resolver problemas complejos de optimización combinatoria, en las que los heurísticas clásicas no son efectivos.

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 [Osman y Nelly, 1996]. 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 [Riojas, 2005] y [Alba, 2005].

Métodos de Descomposición (Divide y Vencerás). Son métodos donde el problema original es descompuesto en subproblemas más pequeños y por ende más sencillos de resolver, teniendo en cuenta que todos pertenecen al mismo problema. Algunos ejemplos de métodos de este tipo son los algoritmos de Ordenamiento [Aho, 1974] y [Michalewicz y Fogel, 2002].

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 paso a paso 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) [Michalewicz y Fogel, 2002].

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 analíticos es que estos no necesariamente encontraran una solución óptima. Ejemplos de éste tipo de métodos son Búsqueda Tabú, Recocido Simulado, entre otros [Michalewicz y Fogel, 2002].

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 [Téllez, 2007], 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 los Algoritmos

La Teoría de la Complejidad se enfoca en dos aspectos principales: el tiempo y los recursos necesarios para resolver un problema computacional (espacio). Estos parámetros son de vital importancia para definir si un algoritmo es eficiente o no; ya que si este requiere mucho tiempo para resolver un problema, no será de utilidad, lo mismo sucede si requiere gran cantidad de memoria. En el caso de que un algoritmo requiera más tiempo que otro para converger en una solución, se puede decir que este algoritmo tiene mayor complejidad temporal; para el caso de mayores requerimientos de memoria, se maneja el término de complejidad espacial.

Un algoritmo, independientemente de su funcionamiento o la forma en que realiza el proceso de solución de un problema, necesita datos de entrada, los cuales, dependiendo del problema tratado, influyen en el tiempo de convergencia del algoritmo, ya que mientras mayor sea el tamaño de la entrada, mayor será el tiempo de ejecución requerido para su procesamiento.

La complejidad temporal de un algoritmo se encuentra representada por medio de una función temporal, la cuál es dependiente del tamaño de la entrada y del número de instrucciones que requieren ser evaluadas para resolver el problema.

Un claro ejemplo de lo explicado anteriormente, es que si un algoritmo presenta un tiempo de ejecución constante T(n) = k, se puede decir que este algoritmo es eficiente, debido a que si el tamaño de la entrada aumenta, el tiempo necesario para su ejecución permanecerá constante. En el caso de un algoritmo con complejidad logarítmica T(n) = log(n), también es considerado eficiente, debido a que, por ejemplo, si el tamaño de la entrada se hace 100 veces más grande, el tiempo requerido para su procesamiento únicamente se duplica; por el contrario, si un algoritmo presenta una complejidad de tipo exponencial T(n) = 2^n, éste es considerado ineficiente [Bisbal, 2009] y [Aho, 1974]. En la Tabla 1-1 se muestran los tipos de complejidad existentes.

Velocidad de Crecimiento	Nombre
K	Constante
log(n)	Logarítmica
n	Lineal
nlog(n)	Cuasi-lineal
n^2	Cuadrática
n^k	Polinómica
2^n	Exponencial

Tabla 1-1. *Tipos de complejidad para el estudio de eficiencia de los algoritmos [Aho, 1974] y [Bisbal, 2009].*

El estudio de eficiencia se lleva a cabo analizando el caso extremo de los algoritmos, es decir, el peor caso, el cuál se encuentra definido como el valor en el que el algoritmo tendrá que realizar la mayor cantidad de operaciones. En la Figura 1-2 se

muestra el comportamiento de algunas de las funciones de complejidad con respecto al tiempo.

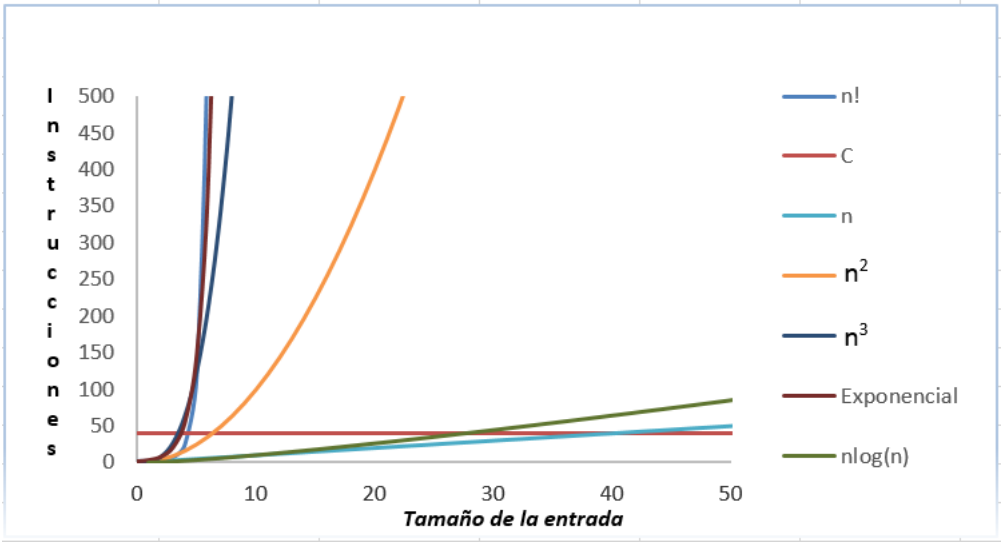


Figura 1-2. *Representación del crecimiento de funciones utilizadas comúnmente en las estimaciones con notación O.*

La importancia de los algoritmos en cuanto a la complejidad, se basa en diseñar algoritmos que cuenten con una complejidad logarítmica, debido a que independientemente del equipo en el que sea ejecutado, el tiempo de cómputo será razonable.

1.3. Estado del Arte

La utilización de metaheurísticas para resolver problemas de optimización se caracteriza principalmente por aplicar búsquedas en vecindades (conjunto de soluciones alcanzables a partir de una solución); razón principal para la construcción de algoritmos de búsqueda eficientes y eficaces que permitan la mejora considerable de soluciones obtenidas por las metaheurísticas.

Muchos han sido los investigadores de la comunidad científica que han estudiado el problema de empaquetamiento en dos dimensiones, por ejemplo [David Pisinger y Mikkel Sigurd, 2005] resolvieron este problema utilizando figuras rectangulares en donde los contenedores tenían diferentes tamaños y diferentes costos y el objetivo

fundamental era reducir al mínimo el costo total de los contenedores utilizados para el empaquetamiento de figuras rectangulares, estos autores aplicaron un algoritmo exacto con instancias de pruebas de hasta 100 elementos y los resultados fueron satisfactorios.

Los autores, [XIE, WANG y LIU, 2007], desarrollaron un algoritmo para anidar figuras irregulares en dos dimensiones con el fin de optimizar una lámina, el desarrollo fue basado en un algoritmo genético, y la aplicación de este algoritmo mostro que es posible acomodar de una forma eficiente las figuras para que se pueda optimizar el espacio en la lámina.

[Andreas M. Chwatal y Sandro Pirkwieser, 2012] solucionan el problema de empaquetamiento en dos dimensiones en donde un conjunto de artículos rectangulares conocidos como demanda, así como también un conjunto de elementos que representan las cajas, se tienen que posicionar de forma eficiente en los artículos rectangulares, cada elemento se puede rotar de manera opcional para su mejor posicionamiento y así disminuir los espacios desocupados, las figuras se posicionan con el fin que se produzca el corte de la guillotina, además se consideran objetos libres, para solucionar este problema estos autores utilizaron varios paquetes metaheurísticos como por ejemplo los algoritmos GRASP y VNS, se utilizaron instancias-benchmark y mostraron que en particular el algoritmo VNS obtuvo mejor resultado que el algoritmo GRASP.

Los autores [López, Ochoa, Terashima y Burke, 2013] resolvieron el problema de empaquetamiento en dos dimensiones con figuras irregulares en donde tratan de posicionar estas figuras en una pieza rectangular grande para maximizar la cantidad de figuras irregulares dadas, las figuras irregulares en la mayoría de los casos se van formando como un rompecabezas y se inscriben en polígonos convexos y estos polígonos son los que se posicionan en la figura rectangular grande, en este caso según se observa la utilización de los polígonos hace que haya mucho desperdicio de espacio porque en varios casos las figuras formadas no ocupan todo el espacio

del polígono convexo en donde se inscriben, se aplica un algoritmo exacto para darle solución al problema, pero según se observa los resultados que se obtienen no son los esperados.

Los autores [Blazewicz, Drozdowski, Soniewicki y Walkowiak, 1989], resolvieron el problema de empaquetamiento en contenedores en dos dimensiones para figuras irregulares con el fin de minimizar el desperdicio de una lámina de material y lo resolvieron con un algoritmo heurístico, los resultados obtenidos fueron básicos pero buenos, además crearon las bases para nuevas implementaciones de este tipo de algoritmos.

[Julia A. Bennell y Xiaozhou Zhao, 2012] construyen un algoritmo adaptado de búsqueda en un conjunto de soluciones para darle solución al problema de empaquetamiento en dos dimensiones utilizando figuras irregulares (polígonos triángulos escalenos entre otras más) en donde en dependencia de la geometría que tiene cada figura se va anidando una con la otra hasta formar un polígono y a la vez las figuras formadas se inscriben en un polígono convexo para así posicionarse en el objeto rectangular grande, y así facilitar el corte de la guillotina, vale destacar que estas figuras que se forman en forma de polígonos a la hora de posicionarse rotan para buscar una mejor posición con otra que se construya. Los resultados experimentales muestran que la estrategia utilizada por estos autores es eficiente y además obtienen buenos resultados.

El autor [Pasha, 2003] soluciona el problema de empaquetamiento en contenedores en dos dimensiones con figuras irregulares con una hibridación del algoritmo de algoritmo genético con un algoritmo de recocido simulado. El algoritmo propuesto en esta tesis es capaz de hallar las configuraciones óptimas de un conjunto de figuras de diferentes formas, es flexible y es capaz de adaptarse a cambios, además es integrable a sistemas de empaquetamientos autónomos.

[Ramin Halavati, Saeed B. Shouraki, Mahdieh Noroozian, y Saman H. Zadeh, 2007] solucionan el problema de empaquetamiento utilizando figuras irregulares y aplican un algoritmo genético, en donde presentan una población representada por figuras irregulares, en donde estas figuras están bien acopladas y la aptitud se computa en dependencia del espacio que exista entre las figuras, también vale recalcar que las figuras no están traslapadas, es decir, si hay algún traslape las figuras toman automáticamente otras posiciones, teniendo en cuenta los cortes de la guillotina. El algoritmo propuesto por estos autores no se degrada mientras el número de figuras aumenta y todo el proceso se realiza en menos de 15 minutos, obteniendo resultados eficaces y eficientes.

[Aline M. Del Valle, Thiago Alves de Queiroz, Flávio K. Miyazawa, Eduardo C. Xavier, 2011] proponen un Algoritmo Heurístico GRASP para resolver el Problema de la Mochila 0-1 y lo solucionan posicionando las figuras irregulares en una figura rectangular grande, validando que no exista traslape entre las figuras, teniendo en cuenta el corte de la guillotina, luego se introduce en una caja de forma rectangular más grande que simula el comportamiento de la Mochila. Con el algoritmo propuesto se logró ocupar con las figuras pequeñas, la figura rectangular grande en un 90 por ciento, lo que resulta que este algoritmo es en un 90 por ciento eficiente.

[Eva Hopper] dice que en que la mayoría de las bibliografías existentes utilizan Algoritmos Genéticos para resolver el Problema de empaquetamiento en contenedores en dos dimensiones con figuras tanto regulares como irregulares, en este preciso caso la autora proporciona un enfoque para solucionar ambos problemas, y enfatiza que con Algoritmos Genéticos obtuvo buenos resultados y que llegó a soluciones eficaces y eficientes.

[R. Andrade, E. G. Birgin, R. Morabito, 2013] formularon el Problema de Empaquetamiento en Contenedores en dos dimensiones para figuras rectangulares con la restricción del corte de la guillotina utilizando un Modelo de Programación Entera y aplicaron un algoritmo exacto en donde posicionan de mayor a menor las

figuras en un espacio rectangular grande, hasta que ya no se pueda posicionar la figura más pequeña, hay que aclarar que en este caso se tiene en cuenta el corte de la guillotina y este siempre es paralelo al ancho y paralelo al largo del contenedor que es la figura rectangular grande. Los modelos representados pueden ser modificados para tratar casos particulares. Se utilizaron diferentes instancias y los resultados obtenidos fueron satisfactorios.

El autor [Mahmud, 2006] desarrolló una investigación para resolver el problema de tira de empaquetamiento en dos dimensiones específicamente para su uso en las industrias de prendas de vestir, este problema está clasificado como un problema NP-duro. Para darle solución a este problema el autor utilizó un Algoritmo Genético y mantuvo la orientación de las piezas fijas. Este requisito de mantener la orientación fija de las piezas, se plantea por la naturaleza de las formas en que tienen que ser cortadas las mismas para hacer la ropa. El autor desarrolló un algoritmo que especifica como poder hacer los cortes de las piezas con el fin de optimizar los recursos y le presenta al usuario una salida en forma visual. El algoritmo genético utilizado está embebido en la herramienta que se encarga de mostrarle al usuario el resultado visual.

Los autores [Blum y Schmid, 2013] solucionan el problema de empaquetamiento en contenedores en dos dimensiones con un algoritmo evolutivo que hace un uso intensivo de una heurística de un paso al azar para la construcción de soluciones. Los resultados del algoritmo propuesto son en comparación con algunos de los mejores enfoques de la literatura. Esta comparación muestra que el algoritmo implementado es muy competitivo a los enfoques del estado de la técnica.

Los autores [Pintea, Pascan y Hajdu-Macelaru, 2012] hacen una comparación entre diferentes heurísticas para resolver el problema de empaquetamiento en contenedores. Los autores solucionan este problema utilizando figuras rectangulares y le asignan una posición fija a las figuras validando que estas no se traslapen entre sí, así como también que ninguna de las figuras pueda rotar, ambas

son restricciones del problema que se está resolviendo. En este trabajo se compara un algoritmo greedy con un algoritmo genético híbrido con el fin de ver cuál es la mejor técnica para el problema dado, y se concluye que la mejor técnicas de las dos utilizadas fue el algoritmo genético híbrido, cada uno de estos algoritmos son probados en diferentes tamaños de datos.

De lo anteriormente expuesto se escogen las principales ideas para resolver el Problema de empaquetamiento en contenedores dos dimensiones aplicado a las imprentas, el mismo se resuelve con figuras irregulares. Para iniciar el proceso se carga una figura de geometría irregular y esta se posiciona n-veces en la zona de trabajo o hoja de papel, se evalúan las restricciones del problema evitando que existan traslapes entre las figuras, cada figura se mueve de forma automática y se mueven en distintas posiciones, por ejemplo: para arriba, para abajo, para la izquierda, para la derecha y en forma diagonal, todas estas operaciones que se les aplican a las figuras, son a partir del análisis de sus pixeles. El segundo caso es similar al primer caso con la diferencia que en este caso se cargan varias figuras de geometrías irregulares. En ambos casos se calcula el área de las figuras a partir de los pixeles que las conforman, luego se suman todas las áreas, después se calcula el área de la figura rectangular grande (área de trabajo o hoja de papel) y a esa área rectangular grande se le resta la suma de las áreas de todas las figuras, obteniéndose como resultado un área residual que es la función objetivo del problema que se está analizando en esta investigación, por último el proceso se repite n-iteraciones, y cuando finaliza cada iteración, se verifica si el área residual disminuye, se guarda como una solución mejorada, el proceso se repite hasta llegar a la condición de parada y por último se obtiene la mejor solución con la configuración de las figuras en la pantalla, es decir, se obtiene como quedaron pintadas las figuras en la pantalla. Para llevar a cabo este trabajo se estudió una librería que permitió poder manipular las imágenes en forma de pixeles, esta librería lleva el nombre de Allegro.

Allegro es una biblioteca libre y de código abierto para la programación de videojuegos desarrollada en lenguaje C. Allegro es un acrónimo recursivo de rutinas de bajo nivel para videojuegos. Esta biblioteca cuenta con funciones para gráficos, manipulación de imágenes, texto, sonidos, dispositivos de entrada (teclado, mouse y mandos de juegos) y temporizadores, así como rutinas para aritmética de punto fijo y acceso al sistema de archivos. Existen dos versiones de Allegro que cuentan con soporte oficial por parte de los desarrolladores, la versión clásica Allegro 4 y la nueva versión Allegro 5. La versión más reciente de Allegro 4 incluye soporte para el manejo de archivos de datos y una implementación por software de funciones para gráficos en tres dimensiones. La versión 5 de Allegro cuenta con una nueva API y cambia la implementación por software de las rutinas gráficas por una implementación basada en OpenGL o Direct3D. Allegro ofrece una API en lenguaje C, actualmente existen envolventes y bibliotecas adicionales que permiten utilizarlo en otros lenguajes como C++, Java, C#, Visual Basic.NET, entre otros muchos más lenguajes de programación [Allegro, 2011].

1.4. Objetivos de la investigación

- Implementar una heurística computacional para resolver el problema de empaquetamiento en contenedores en dos dimensiones aplicado a imprentas.
- Implementar un algoritmo para la detección de traslapes de figuras irregulares para contenedores en dos dimensiones aplicado a imprentas.
- Implementar un algoritmo para el cálculo de áreas de figuras irregulares para contenedores en dos dimensiones aplicado a imprentas.
- Evaluar la eficiencia y la eficacia del algoritmo propuesto mediante pruebas experimentales.
- Implementar una herramienta con interfaz visual para la optimización de papel en imprentas.

1.5. Alcance de la investigación

- Se implementará un algoritmo de Búsqueda Local Iterada y será evaluado con una estructura de vecindad que permita una mejor explotación del espacio de soluciones.
- Se realizará un estudio de sensibilidad a los parámetros de control del algoritmo de Búsqueda Local Iterada, con el fin de encontrar el mejor desempeño en eficiencia y eficacia.
- Se implementará un algoritmo para el cálculo de áreas en las figuras irregulares, a partir de la implementación de una estructura de mallado de pixeles.
- Implementación de un algoritmo para la determinación de traslapes entre figuras irregulares.
- Implementación de una herramienta con interfaz visual para optimizar papel en Imprentas.

1.6. Contribución de la tesis

La aportación de este trabajo de investigación para resolver el problema de empaquetamiento en contenedores de dos dimensiones tratado, es el mapeo que se hace al problema real que aparece en una imprenta para disminuir el desperdicio del papel y el cual se resuelve con una heurística computacional.

En este trabajo de investigación se desarrolló un algoritmo de Búsqueda Local Iterada con una estructura de vecindad simple.

La estructura de vecindad consiste en hacer pequeños movimientos de forma aleatoria a las figuras con el objetivo de encontrar espacios para lograr insertar más figuras.

Se implementó un algoritmo para la validación de traslapes de figuras tanto regulares como irregulares.
Se implementó un algoritmo para el cálculo a áreas de figuras tanto regulares como irregulares.
Las pruebas experimentales realizadas muestran que es posible realizar una disminución del desperdicio de papel en una imprenta.

Una parte fundamental de este trabajo es que se aplica un problema de las ciencias de las computacionales (problema de empaquetamiento en contenedores en dos dimensiones) a un problema que surge en una imprenta y los resultados que se obtienen muestran que es posible lograr minimizar el desperdicio de papel. También hay que mencionar que la búsqueda local se aplica sobre las transformaciones de cada una de las figuras que se desean posicionar en el área de trabajo u hoja de papel, estas transformaciones son las traslaciones a la derecha, izquierda, arriba abajo y de forma diagonal en cualquier dirección, para así lograr sus posiciones óptimas en el espacio del área de trabajo que no es más que la hoja de papel.
De acuerdo a lo anterior, se propone una metodología para la realización del análisis de sensibilidad, la cual lleva a cabo la sintonización de las variables de entrada del algoritmo; esto permite conocer que tan sensible es el algoritmo al cambio de valor de ciertos parámetros y de esta forma determinar el rango de valores dentro del cual, la solución sigue siendo buena.

1.7. Organización de la Tesis

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 en Contenedores, así como su descripción conceptual.
En el capítulo 3 se da una explicación introductoria del algoritmo de Búsqueda Local Iterada, se explica ampliamente la estructura de vecindad propuesta aplicada a búsqueda local, se muestra un esquema sobre el algoritmo de colisión de figuras,

así como el algoritmo de cálculo de las áreas amorfas; se explica la metodología propuesta para la realización del análisis de sensibilidad utilizado para obtener la sintonización de los parámetros de control, además de presentar la función temporal correspondiente a la complejidad del algoritmo propuesto.

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al algoritmo de Búsqueda Local Iterada, donde se integran el algoritmo de detección de traslapes, el algoritmo de cálculo de áreas de figuras irregulares y la estructura de vecindad a la búsqueda local, dar solución al problema de empaquetamiento en contenedores en dos dimensiones aplicado a imprentas. En el capítulo 5 se plantean las conclusiones y los trabajos futuros.

Capítulo 2

2.0. Problema de empaquetamiento en contenedores en 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.

Para resolver este tipo de problemas, se trata de empaquetar un conjunto de piezas tanto regulares como irregulares en una pieza regular grande, en forma rectangular, además se debe validar que cada una de las piezas que se posicionan en la pieza de forma rectangular grande no se traslapen entre ellas, y que la suma de cada una de sus áreas sea menor o igual que el área de la pieza regular rectangular grande. Estos problemas, específicamente el problema de empaquetado es un problema que surge en las industrias. El mismo puede ser aplicado a Problemáticas que constan en Imprentas. En la actualidad existen imprentas que anualmente procesan 350 toneladas de papel al año y tienen un desperdicio del 10 % que representa 35 toneladas de papel.

En este trabajo de investigación se aplica un problema de las ciencias de la computación (problema de empaquetamiento en contenedores en dos dimensiones)

a un problema que surge en la industria de papel, específicamente en las imprentas, y el mismo se resuelve aplicando técnicas heurísticas.

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 en 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 también se aplica en el diseño de circuitos integrados, 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.

Los problemas de empaquetamiento 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) y del 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 en 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 anteriormente expuesto, se puede mencionar que los problemas de empaquetamiento aparecen bajo diferentes nombres en la literatura. [Hinxman, 1980], estos problemas pueden aparecer con el nombre de Dickhoff [Dickhoff, 1990], en donde se presenta una caracterización integrando todas las clases de problemas de corte y empaquetado, la cual generaliza la clasificación presentada por Hinxman [Hinxman, 1980] 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 euclideo (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 [Dickhoff, 1990 y Finke, 1992]:

- 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 refieren a la forma y cantidad de los elementos. Los problemas pueden consistir de 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. Recientemente, Wäscher et al. [Wäscher, Haußner y Schumann, 2007] han presentado 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.

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.

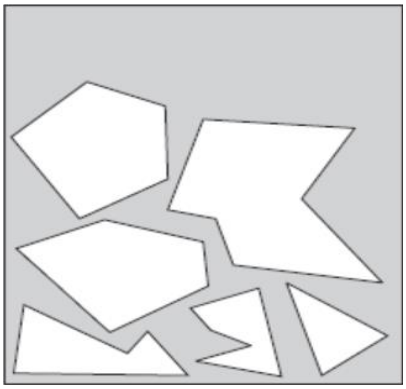


Figura 2.1: *Ejemplo de patrones con figuras irregulares. [Salto, 2010]*

En función de la forma que tengan los elementos o piezas, o a la geometría de los mismos, 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 la misma.

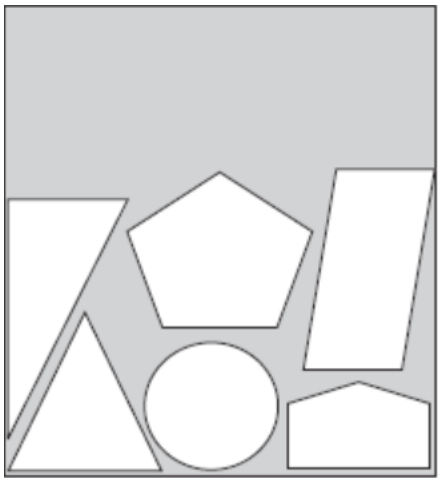
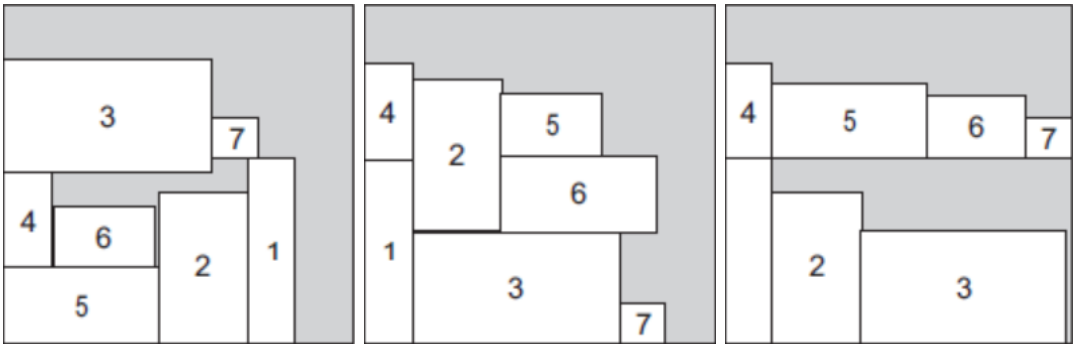


Figura 2.2: *Ejemplos de patrones con cortes no ortogonales. [Salto, 2010]*



a) b) c)

Figura 2.3: *Ejemplos de patrones con cortes ortogonales: (a) corte no guillotina; (b) corte guillotina; (c) corte guillotina en niveles. [Salto, 2010]*

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 rectangular

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

Dados n elementos (rectángulos pequeños) $I = \{1, 2, \ldots, n\}$, cada uno caracterizado por su ancho w_i y su altura h_i, 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 asignación, 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 w_i y su altura h_i, 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 w_i y su altura h_i, y un gran objeto cuyo ancho W y altura H son variables. El objetivo es minimizar el área $W \times H$ del objeto de manera que todos los artículos pueden ser empaquetados en la tira.

Otros dos problemas de empaquetamiento son el *problema de empaquetamieto en dos dimensiones y problema de la mochila,* 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 w_i y una altura h_i, 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 $i \in I$ 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 $I' \subseteq I$ con valor máximo total $\sum_{i \in I'} ci$ tal que todos los elementos $i \in I'$ puedan ser empaquetados en la mochila.

Para el problema de empaquetamiento en dos dimensiones, Lodi et al. [A. Lodi, S. Martello, y D. Vigo, 1999] proponen métodos heurísticos y algoritmos metaheurísticos y realizan experimentos sobre instancias de varios casos de prueba. Para el problema de la mochila en dos dimensiones, Wu et al. [Y. Wu, W. Huang, S. Lau, C.K. Wong, y G.H. Young, 2002] propone algoritmos heurísticos que son efectivos para muchas instancias de prueba. 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 w_i, una altura h_i 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 d_i copias en los distintos objetos).

Carga de paletas. Dada una cantidad suficientemente grande de elementos de idéntico tamaño ***(w, h)*** 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.

2.3.1. Modelo matemático

El problema que se está analizando en este trabajo se puede representar matemáticamente como un problema de minimización de áreas (***problema de empaquetamiento en dos dimensiones***) en donde dado n elementos, cada uno definido por su ancho w_i y su altura h_i, 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:

$$Min\ FO = ((W*H) - \sum A) \qquad (1)$$

Sujeto a:

$$l_{ij}+l_{ji}\ b_{ij}+b_{ji}+p_{ij}+p_{ji} \geq 1 \qquad \forall i, j \in I, i<j \qquad (2)$$

$$x_i - x_j + \mathbf{W} l_{ij} \leq W - w_i \qquad \forall i, j \in I \qquad (3)$$

$$y_i - y_j + \mathbf{H} b_{ij} \leq H - h_i \qquad \forall i, j \in I \qquad (4)$$

$$m_i - m_j + np_{ij} \leq H - h_i \qquad \forall i, j \in I \qquad (5)$$

$$x_i \leq W - w_i \qquad \forall i, j \in I \qquad (6)$$

$$y_i \leq H - h_i \qquad \forall i, j \in I \qquad (7)$$

$$m_i \leq v \qquad \forall i, j \in I \qquad (8)$$

$$1 \leq m_i \qquad \forall i, j \in I \qquad (9)$$

$$m_i \leq i \qquad \forall i, j \in I \qquad (10)$$

$$l_{ij}, b_{ij}, p_{ij} \in \{0,1\} \qquad \forall i, j \in I, i \neq j \qquad (11)$$

$$x_i, y_i, m_i, \mathbf{A} \in N \qquad (12)$$

Figura 2.4. *Modelo Matemático del problema de empaquetamiento en contenedores en dos dimensiones. [Christian Blum y Verena Schmid, 2013]*

Donde W, es el largo de la hoja, H es el ancho, w_i es el largo de la figura i, h_i, es el largo de la figura i, (x_i, y_i) son las coordenadas del elemento i, L_{ij}, b_{ij}, p_{ij} : $\{0, 1\}$ son variables binarias, A es la suma de las área de cada una de las figuras que se están posicionando

en la hoja de papel, m_i, es el contenedor en donde se van a colocar los elementos i. Para una mejor compresión del modelo matemático ver ***(Figura 2.5).***

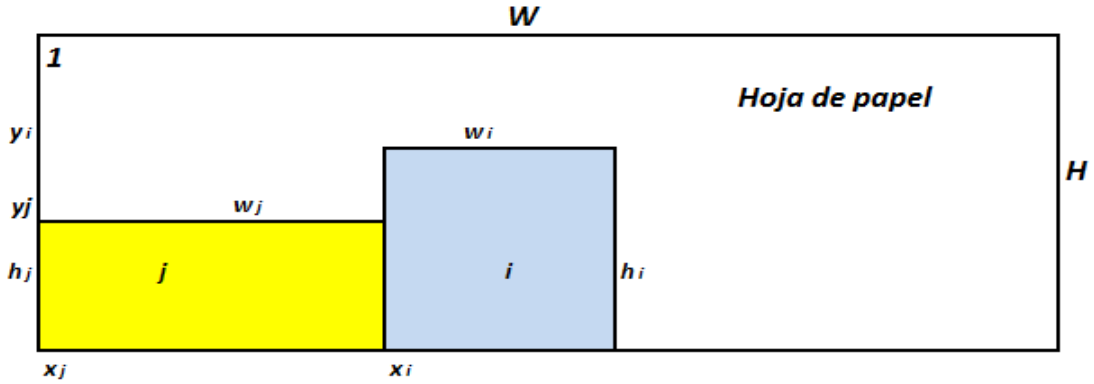


Figura 2.5. *Ejemplo de aplicación al modelo matemático. Representación de las variables.*

Las restricciones 2, 3 y 4 son las que validan que no existan traslapes entre las figuras que se están posicionando, es decir que una figura se posicione una a continuación de la otra, para una mejor comprensión, se sugiere que se observe la ***Figura 2.6.***

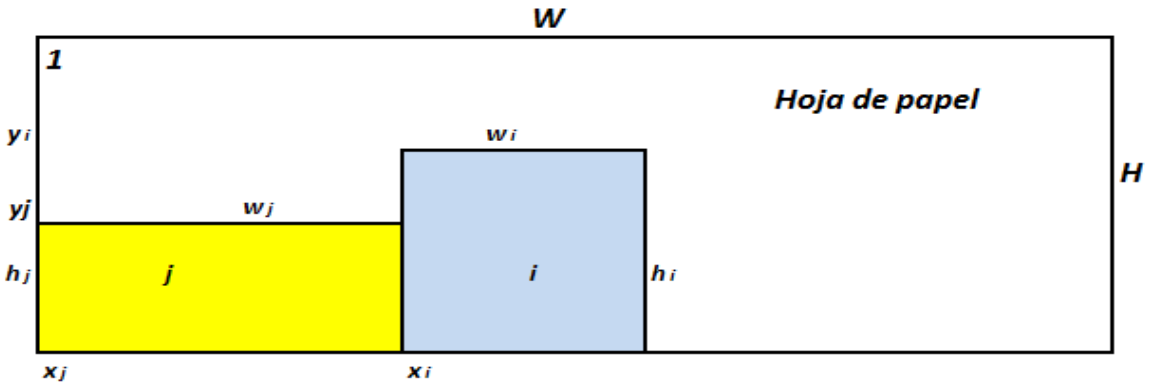


Figura 2.6. *Ejemplo de aplicación al modelo matemático. Posicionamiento de las figuras una a continuación de la otra.*

Las restricciones 6 y 7 garantizan que las dimensiones de las figuras siempre tienen que ser menores que las dimensiones de la hoja de papel, para una mejor comprensión, se sugiere que se observe la ***Figura 2.7.***

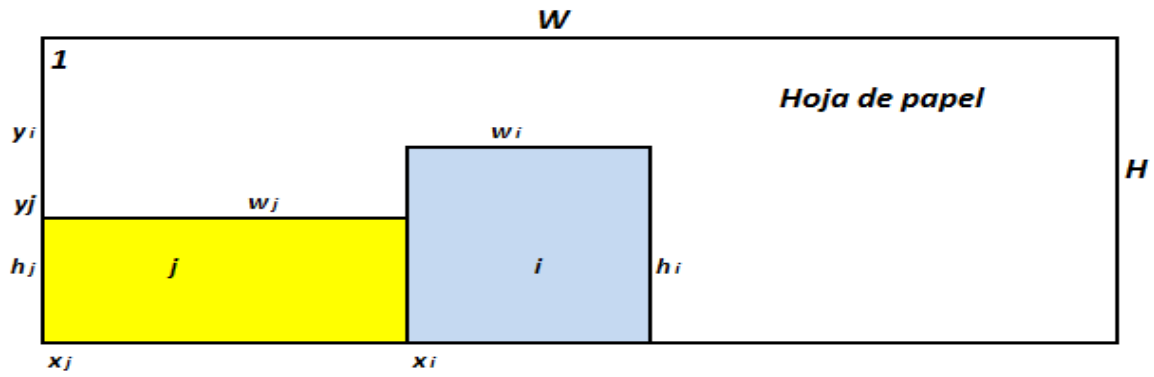


Figura 2.7. *Ejemplo de aplicación al modelo matemático. Las dimensiones de las figuras a posicionar, tienen que ser menor que la hoja de papel.*

La restricción 5 está relacionada con el número de contenedores u hoja de papel y el número de figuras, cada contenedor u hoja de papel tiene asociado un número de figuras y cada figura está posicionada en un contenedor en específico o en una hoja de papel, las variables m_i y m_j representan los contenedores o la hoja de papel, n es el número de figuras y p_{ij} es una variable binaria que toma valores de cero o uno en dependencia del contexto. La restricción 6 quiere decir que si se restan las figuras del contenedor u hoja de papel número 2 menos las figuras del contenedor u hoja de papel número 2, más la cantidad de figuras posicionadas, esta suma algebraica será igual a la cantidad de figuras menos 1, para comprender mejor lo anteriormente dicho se sugiere ver la ***Figura 2.8 (a) y (b).***

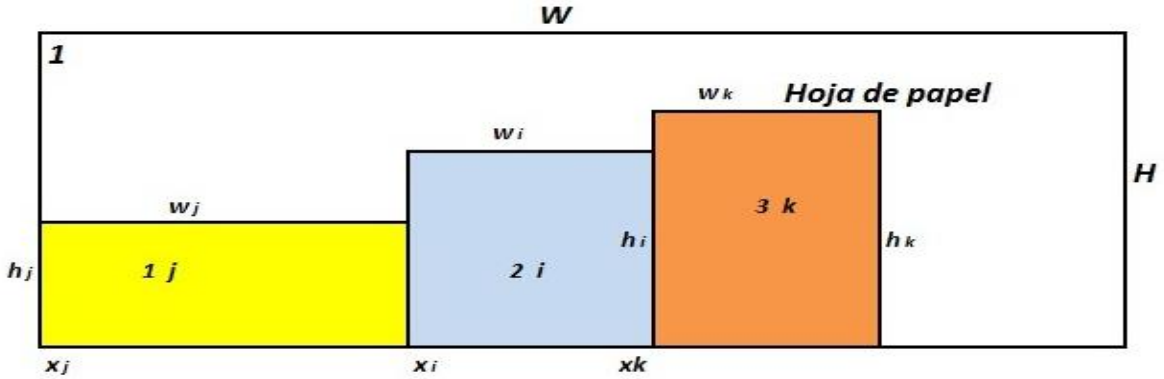


Figura 2.8. (a). *Ejemplo de aplicación al modelo matemático. Restricciones asociadas al número de contenedores u hoja de papel con el número de figuras.*

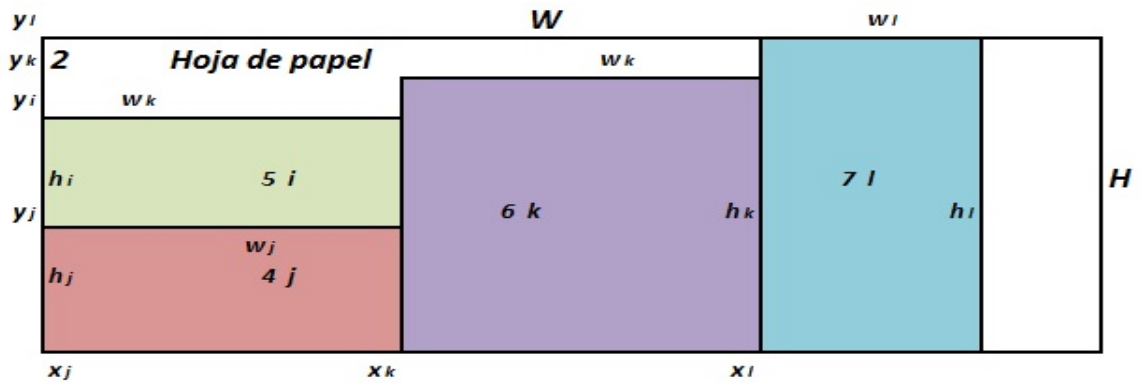


Figura 2.8. (b). *Ejemplo de aplicación al modelo matemático. Restricciones asociadas al número de contenedores u hoja de papel con el número de figuras.*

El problema de empaquetamiento en contenedores en dos dimensiones para figuras regulares (figuras rectangulares), se expresa a partir de un modelo matemático como el que anteriormente se explicó. En este trabajo de investigación se aplica un problema de las ciencias de la computacionales (problema de empaquetamiento en contenedores en dos dimensiones) a un problema que surge en las imprentas, y se resuelve con una heurística computacional, pero las figuras que se están trabajando no son figuras rectangulares sino, son figuras irregulares, entonces lo que se hace es; hacerle un mapeo al problema que surge en las imprentas y tratarlo como un problema rectangular, haciendo referencia al modelo matemático anteriormente explicado.

2.4. Problema de empaquetamiento en contenedores aplicado a imprentas.

El problema de empaquetamiento en contenedores es un problema que surge en las industrias, tanto de carga de mercancías, como en industrias del papel, este problema es un problema de optimización combinatoria que desde el punto de vista computacional se clasifica como un problema NP-Completo [M. Garey y D. Johnson, 1979], esto significa que no tiene una solución exacta en un tiempo polinomial, y para resolverlo se hace uso de métodos heurísticos, que no son más, que métodos con reglas empíricas para encontrar soluciones a problemas, están basados 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. El

problema del empaquetamiento en contenedores cuando se analiza en dos dimensiones se puede llegar a la conclusión, que independientemente de ser un problema de optimización combinatoria y de estar clasificado como un problema NP-Completo [M. Garey y D. Johnson, 1979], 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 validar que las piezas a empaquetar no colisionen entre ellas y que la suma de cada una de sus áreas sea menor o igual que el área de la pieza rectangular grande que representa la hoja de papel.

El problema de empaquetamiento en contenedores en dos dimensiones, al ser un problema de las ciencias computacionales, el mismo puede ser aplicado a problemáticas que surgen en imprentas. En la actualidad existen imprentas que anualmente procesan 350 toneladas de papel al año y tienen un desperdicio del 10 % que representa 35 toneladas de papel.

En este trabajo de investigación se aplica un problema de las ciencias computacionales (problema de empaquetamiento en contenedores en dos dimensiones) a un problema que surge en una imprenta, y se resolverá aplicando técnicas heurísticas.

Capítulo 3

Heurística Computacional Aplicada

3.1. Descripción de la Búsqueda Local Iterada

Existen diferentes métodos con los que se puede generar una solución mucho mejor en comparación con los que actualmente se utilizan. La Búsqueda local iterada es una técnica potente, simple de implementar, robusta y altamente eficiente [Ramalhino, Martin, Stützle y Glover, 2002]. Propone un esquema en el que se incluye una heurística a la medida base para mejorar los resultados de la repetición de dicha heurística *[Salto, 2010]*. 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. El método analizado es un método estocástico de búsqueda local que iterativamente aplica búsquedas locales a perturbaciones del punto actual (solución inicial) de una manera aleatoria en un espacio de soluciones óptimas.

Este método trabaja en conjunto con la búsqueda local, y es mediante éste que se obtiene la primera solución con la cual se iniciará el recorrido por el espacio de soluciones hasta mejorar progresivamente.

Para comprender mejor el funcionamiento de la búsqueda local iterada, es necesario entender, en primer lugar, cómo funciona la búsqueda local:

1. Del espacio de soluciones, se obtiene una primera distribución aleatoria de figuras generadas estocásticamente, la cual se evalúa mediante la función objetivo y se genera la solución inicial (S) [Michalewicz y Fogel, 2004].
2. A esta solución inicial se le aplica un cambio con alguna estructura de vecindad, se evalúa con la función objetivo y se obtiene una nueva solución (S').
3. Si S' es mejor que la actual, S será sustituida, de lo contrario permanecerá igual; es decir, si S'□ S entonces S ¬ S' .
4. Dentro de la búsqueda local se repiten los pasos 2 y 3 hasta que se cumpla con el criterio de paro.

El cambio que se aplique a S mediante alguna estructura de vecindad será el que determine el tipo de solución obtenida; si ésta fue mejor, será remplazada; de lo contrario, permanecerá hasta encontrar una que la sustituya.

Para salir del espacio de soluciones óptimas locales, es necesario hacer uso del algoritmo de búsqueda local iterada [Khebbache, Prins y Yalaoui, 2008], el cual, una vez obtenido el valor óptimo local, es tomado como el mejor e iniciará la siguiente iteración. Con este nuevo valor, si en las siguientes iteraciones se encuentra uno mucho mejor que el actual, lo reemplaza; si no es así, se genera un nuevo cambio hasta encontrar uno mejor que el actual.

En forma de pseudocódigo, el algoritmo de búsqueda local iterada se muestra en la Figura 3.1.

```
Hacer
    Generar solución inicial s
    Hacer
        s'=Estructura de Vecindad(s)
        Si (f(s')>=f(s)) entonces
            s=s'
            Si (f(s')>f(Sm-LS)) entonces
                Sm-LS=s'
            Fin-si
        Fin-si
    Hasta CP-LS
    Si (f(Sm-LS)>f(Sm-ILS)) entonces
        Sm-ILS=Sm-LS
    Fin-si
Mientras CP-ILS
```

Figura 3.1. Búsqueda local iterada [Cruz, Martínez y Serna, 2010]

3.2. Estructura de Vecindad Propuesta

Una vecindad es el conjunto de soluciones las cuales se pueden alcanzar a partir de una solución *s* por medio de un movimiento σ/s [Papadimitriou y Steiglitz, 1998], que puede ser un intercambio, inserción o eliminación entre los elementos de una solución s. Tomando en cuenta la definición anterior, una vecindad es definida como el conjunto de soluciones cercanas de una solución inicial, $s \epsilon S$ en una instancia del problema, tal que, el conjunto *N(s)* es factible en un punto cercano a *s*. El conjunto *N(s)* (ver figura 3-2) llamado vecindad de *s*, indica que cada solución $s' \epsilon N(s)$ puede ser alcanzada directamente desde el valor actual de la función objetivo en un sólo paso. De acuerdo con esto la vecindad es definida por la función $N: S \rightarrow 2^s$ [Martínez, 2010]. Para mejorar la solución *s*, es necesario moverse paso por paso desde la solución inicial factible hacia una solución que proporciona el mínimo valor de la función objetivo, la cual usualmente implica el costo. Para el problema que se está analizando en este trabajo de investigación, la función objetivo es minimizar el desperdicio de espacio de trabajo o de la hoja de papel que representan la función objetivo del problema.

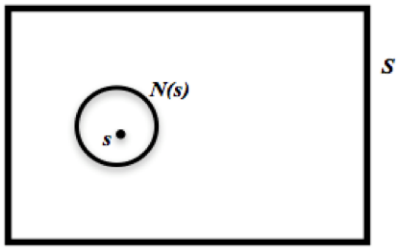


Figura 3.2. Representación del espacio de soluciones de una estructura de vecindad.

El procedimiento comienza de un punto *s* y el conjunto de soluciones *N(s)*, es elegido a través de una perturbación ***s'*** dada por un procedimiento estocástico, esto es, si *f(s') >= f(s), s'* es remplazada por la solución *s* y pasa a ser la solución actual del problema, la Figura 3.2 se muestra el algoritmo general de búsqueda de vecindad.

```
Generar solución inicial S
Hacer
    S' = solución inicial perturbada
    Si f(s') >= f(s) entonces
        S' = mejor solución
        S<-S'
    Fin-si
Mientras la solución continúe mejorando
```

Figura 3.3. Algoritmo general de búsqueda por vecindad.

La Figura 3.3 muestra un algoritmo para hacer búsquedas por vecindad. Las técnicas de búsqueda por vecindad se implementan para mejorar una solución minimizando el costo de la función objetivo, se recuerda que para este trabajo de investigación se requiere minimizar el espacio en la hoja de papel o maximizar el número de figuras a posicionar en la hoja de papel.

Para elegir la estructura que encuentra mejores soluciones al problema se define un movimiento que permita alcanzar una solución s' por medio de una perturbación sobre la solución s, siempre y cuando se cumpla el criterio de selección. Este proceso será de forma iterativa hasta alcanzar el número de iteraciones

determinadas en el algoritmo, si la solución encontrada no mejora, significa que se ha encontrado el óptimo local.

En el caso del problema que se está analizando en este trabajo de investigación, la estructura de vecindad la conforman los posibles movimientos que se realizan de forma aleatoria a las figuras para que estas obtengan una mejor posición en la hoja de papel con el fin de minimizar los espacios en ambos casos.

El proceso de perturbación aplicado a una estructura de vecindad simple, es realizado sobre una solución *s* inicializada de forma aleatoria en donde se especifican los valores de x e y, que representan las coordenadas de las posiciones en que están ubicadas las figuras, una vez obtenida una solución s, se prosigue a guardarla y cuando se aplica una Búsqueda Local Iterada, se carga la solución anteriormente guardada y se van actualizando de forma aleatoria las posiciones de las coordenadas, se destaca que el proceso de selección de un par de coordenadas es de forma aleatoria y a cada coordenada se le va sumando o restando un valor que se va generando de forma aleatoria por cada par, con el fin de conseguir espacios para poder insertar más figuras a la solución s y se obtiene una solución s' mejorada.

Figura 3.4. *Estructura de vecindad sencilla, solución inicial S.*

Luego de tener una solución inicial, esta solución se perturba mediante los pequeños movimientos de forma aleatoria que se le realizan a las figuras con el fin de obtener espacios para seguir insertando figuras, a este proceso se le denomina explotación de la solución con el fin de obtener más espacio disponible para lograr una mejor solución que en este caso sería la solución mejorada S' como lo muestra la ***Figura 3.4***, en esta figura se coloca un circulo de color blanco para indicar que existe espacio disponible para ubicar una figura, así como también se puede observar que dos de las figuras están acompañadas por flechas de color blanco y son una muestra de que ambas figuras están cerca pero no se están traslapando.

En la ***Figura 3.5,*** se puede observar que se insertó una nueva figura y esta se especifica con flechas de color blanco, además se le aplica pequeños movimientos aleatorios a las figuras con el fin de obtener espacios para seguir insertando más

figuras, en ambos casos tanto el caso de la ***Figura 3.4***, como el caso de la ***Figura 3.5,*** se hacen pequeños movimientos para lograr mejores soluciones. En la ***figura 3.5***, se logra mejorar la solución.

Figura 3.5. *Estructura de vecindad sencilla, solución mejorada S'.*

En el problema que se está analizando la estructura de vecindad está enmarcada sobre los movimientos de traslación de forma aleatoria ya sean a la izquierda o a la derecha, arriba, abajo y de forma diagonal en cualquiera de los sentidos, aquí las figuras se pueden mover en cualquier forma con el fin de posicionarse en el espacio de trabajo u hoja de papel y lograr que se minimice el desperdicio que conforma la función objetivo a minimizar.

Mientras se están posicionando las figuras en el área de trabajo, se están obteniendo soluciones parciales que se van perfeccionando con los movimientos

traslación de las figuras, una vez ubicadas todas las figuras que quepan en el espacio de trabajo se obtiene una solución total del problema.

3.3. Creación de una solución factible

En la Figura 3.6 se representa en diagrama de flujo el algoritmo que permite encontrar una solución factible. Para generar una solución factible se implementó un algoritmo que consiste en cargar un conjunto de imágenes de diferentes geometrías irregulares y posicionarlas aleatoriamente en la pantalla. La pantalla representa el área de trabajo, así como la hoja de papel en donde se van a imprimir las figuras. Una vez cargadas y posicionadas las figuras de diferentes geometrías, se verifican las restricciones del problema, estas consisten en que no exista traslapes entre las figuras y que haya una separación entre ellas, capaz de permitir el corte de la guillotina. Las figuras tienen la libertad de transformarse, cada una de ellas puede moverse a la izquierda, a la derecha, hacia arriba, hacia abajo y de forma diagonal, en dependencia del valor que tomen las x e y que representan las coordenadas de las posiciones de cada una de las figuras, todo el proceso es similar a armar un rompecabezas.

Cuando gran parte de las figuras están posicionadas en la pantalla y existe espacio para seguir ubicando figuras, pero este espacio no es suficiente porque las figuras están muy dispersas, lo que se aplica es un reagrupamiento, este paso consiste en que cada figura se mueve de forma automática porque a cada coordenada se le hace una actualización aleatoria, a este proceso se le llama reinicio aleatorio porque se realiza de forma aleatoria y no es más que el reinicio que se le aplica a una Búsqueda Local para conformar una Búsqueda Local Iterada.

Una vez aplicado el reinicio aleatorio, por cada figura se cuenta el número de pixeles que la conforman, con el fin de calcular el área de cada una de ellas y así poder sumar esas áreas para calcular la suma de las áreas de todas las figuras, con el propósito de hacer la resta entre el área de la hoja de papel menos el área de la suma de todas las figuras, obteniéndose un área residual que conforma la función

objetivo del problema que se está analizando. Una vez calculadas el área total de las figuras y el área que ocupa la hoja de papel, se llega a la conclusión, que si la suma total de las áreas de todas las figuras es mayor que el área que ocupa la hoja de papel, no es una solución porque se estarían violando las restricciones del problema, así como las restricciones de traslape y de separación entre las figuras, de lo contrario, si existe espacio, ese especio debe ser mayor que el área de la figura más pequeña, o el número de iteraciones al que se manda a ejecutar el algoritmo tiene que ser mayor que el total de iteraciones por el cual se comenzó a ejecutar dicho algoritmo, si se cumplen ambas condiciones entonces el algoritmo llega a su fin, sino comienza nuevamente todo el proceso que anteriormente se describió.

El diagrama de flujo de la ***Figura 3.6*** representa el algoritmo que se implementó para lograr obtener la primera solución factible en este trabajo de investigación.

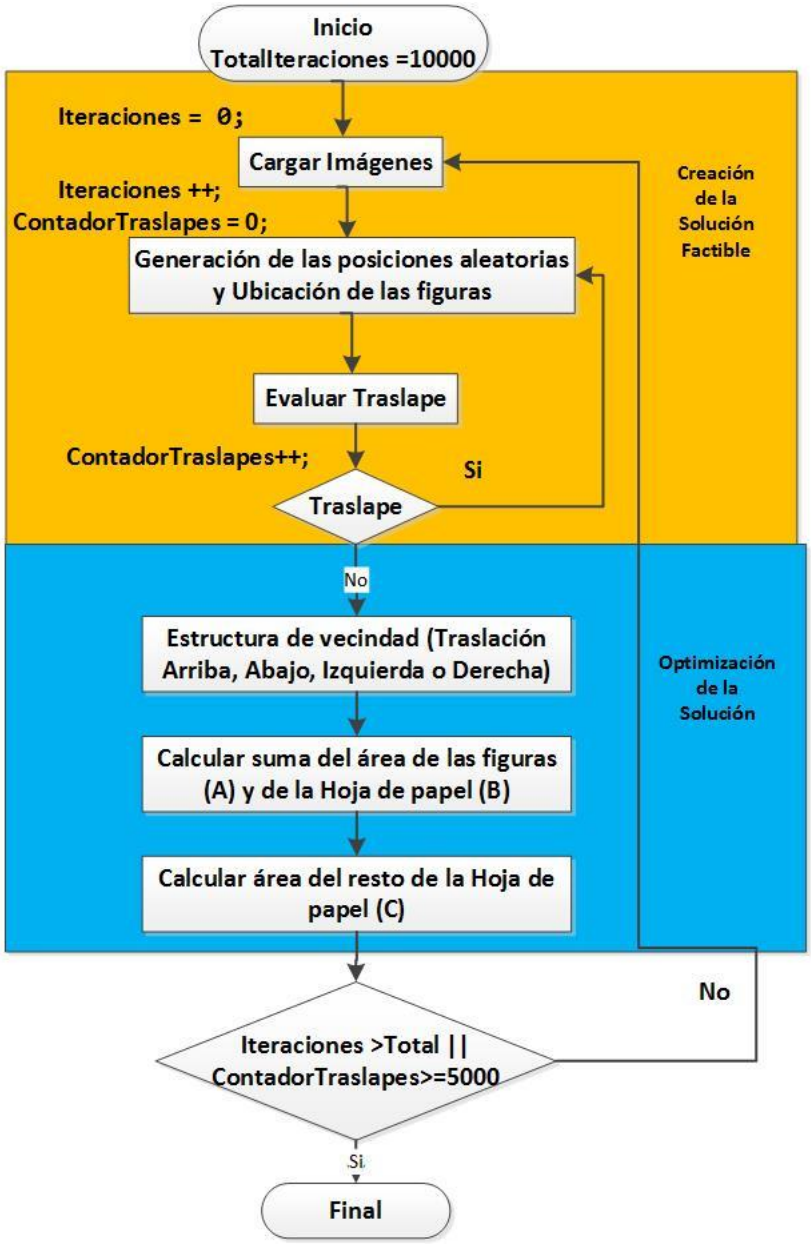


Figura 3.6. Representación en diagrama de flujo del Algoritmo que permite encontrar una solución factible.

Nótese que la estructura de vecindad, que anteriormente se describió en la sección 3.2 sección comienza en el cuadro del proceso donde dice estructura de vecindad, en donde se visualizan las transformaciones de las figuras, dígase las traslaciones

tanto a la izquierda como a la derecha en la búsqueda del espacio en el área de trabajo para la optimización del desperdicio del papel.

En la Figura 3.7 se muestra el algoritmo de detección de traslapes se describe de la siguiente forma. Cada figura está representada por una máscara de pixeles, que a su vez esta máscara está representada por una estructura que tiene un campo que es un arreglo de pixeles y dos valores enteros que significan el largo y el ancho de cada imagen que contienen figuras. Cada imagen, contiene una figura y se representa por una malla de pixeles en donde los pixeles toman dos valores binarios significativos, estos dos valores son 0 y 1, a los pixeles que tienen valor 1, se les llaman activos, y son los que comienzan a partir de los bordes de las figuras, además son los que conforman las figuras, los pixeles que poseen valor de 0, se les llaman desactivados son los que no están dentro de las figuras, pero que forman parte de la imagen. Para validar la existencia de traslapes entre las figuras se calcula la distancia entre ellas, si esa distancia, es muy grande entonces en ese caso no hay superposición, pero si esa distancia es pequeña se verifica, la existencia de traslapes entre pixeles activados y si existen pixeles activados entonces hay traslapes entre pixeles que a su vez significa, que ambas figuras se están traslapando una con la otra.

```
Mask_Collide ( * a, * b, x, y)
Inicio
      X = ( a->w + b->w ) / 2;
      Y = ( a->h + b->h ) / 2;
      Si (X<=0 || Y<=0)
          Return 0;
      Fin-si
      x1 = (a->w – X)*((x<0)?1:0);
      y1 = (a->h – Y)*((y<0)?1:0);
      x2 = (b->w – X)*((x<0)?0:1);
      y2 = (b->h – Y)*((y<0)?0:1);
      Para i = 0 hasta X
          Para j = 0 hasta Y
              Si (a->bits[(x1+i)*a->w+(y1+j)] == 1 && b->bits[(x2+i)*b->w+(y2 + j)] == 1)
                  Return 1;
              Fin Si
          Fin Para
      Fin Para
Return 0;
Fin del procedimiento
```

Figura 3.7. Algoritmo de detección de traslapes de figuras.

En la Figura 3.8 se muestra el algoritmo en seudocódigo que se desarrolló para calcular el área a cualquier figura irregular, este algoritmo consiste en hacerle un barrido a cada una de las imágenes, cuando se habla de hacer un barrido, esto significa que cada imagen se trató como una matriz, eso quiere decir que por cada una de las imágenes, se hizo un recorrido como mismo se recorre una matriz, pero en este caso cada imagen representaba una matriz de pixeles, y se contaron todos los pixeles activados y una vez teniendo el número de pixeles activados, ya se puede decir que se tiene el área que representa la figura que está dentro de una imagen, pero esta área esta expresada en pixeles.

```
Mask_GetPixels_On ( * a)
Inicio
      Pixel = 0;
      Para i = 0 hasta a->w
          Para j = 0 hasta a->h
              Si (a->bits[i*a->w+j] == 1)
                  Pixel++;
              Fin Si
          Fin Para
      Fin Para
Return Pixel;
Fin del procedimiento
```

Figura 3.8. Algoritmo para el cálculo de las de figuras irregulares.

3.4. Metodología de Sintonización

Para llevar a cabo una adecuada sintonización de los parámetros de control del algoritmo implementado, fue necesario llevar a cabo un análisis de sensibilidad.
El análisis de sensibilidad es una evaluación del comportamiento de las variables críticas de un problema (parámetros de control), con la finalidad de establecer un rango numérico, dentro del cual la solución obtenida por el algoritmo sigue siendo buena, además que permite conocer que tan sensible es el algoritmo a ciertos cambios en los valores de ciertas variables propias del problema.

El análisis de sensibilidad tiene su origen en la programación lineal (PL) debido a que facilita la toma de decisiones. Los cambios llevados a cabo durante dicha evaluación pueden ser en el entorno general del problema, en la empresa o bien en los datos característicos del problema.

El objetivo primordial del análisis de sensibilidad, es encontrar la adecuada sintonización de los parámetros de control del algoritmo, de modo que este tenga una mejora en cuanto a la eficiencia y eficacia. Los valores más estudiados en este análisis de forma general, son los coeficientes de la función objetivo y los términos independientes de las restricciones.

Por lo tanto, un objetivo fundamental en el análisis de sensibilidad es identificar los parámetros sensibles (es decir, los parámetros cuyos valores no pueden cambiar sin que cambie la solución óptima) [Hillier y Lieberman, 2010].

El parámetro de control inicial se toma de forma arbitraria, se comienza con 5000 y será dividido entre la cantidad mínima de pruebas que es 30, para obtener el valor de decremento o incremento [Cruz, 2005].

Se ha tomado la explicación de la metodología de sintonización de las tesis de [Cruz, 2005] y de [Martínez, 2010] para encontrar una adecuada proporción en los

valores correspondiente a los parámetros de control, tomando en cuenta tanto el problema como el método de solución implementado.

a) Selección de los parámetros de control.

Para determinar los parámetros de control del algoritmo, es necesario llevar a cabo una revisión de publicaciones que se relacionen con el algoritmo implementado, de esta forma es posible analizar los parámetros tomados en cuenta en investigaciones anteriores [Martínez, 2010].

Otra forma de identificar los parámetros de control, es identificar los parámetros críticos que influyen de cierta manera en la calidad de la solución.

b) Establecer Rangos de Evaluación.

Una vez que hemos identificado los parámetros de control, es necesario establecer los rangos que van a ser utilizados para el análisis de sensibilidad a cada uno de los parámetros. En caso de existir parámetros probados en la literatura para el mismo problema, especificar un rango de acción para cada parámetro de control será más sencillo. El tener una adecuada sintonización nos permite identificar la proporción adecuada entre los parámetros de control dentro del cual el algoritmo obtiene buenas soluciones.

c) Pruebas a Rangos de Evaluación.

Una vez establecidos los rangos se calcula una serie de muestras, de acuerdo al tamaño del rango, que permita evaluar el comportamiento del algoritmo cuando los parámetros de control toman valores distintos.

Una vez obtenido el conjunto de muestras, se procede a realizar las pruebas experimentales, para lo cual se recomienda realizar conjuntos mínimo de 30 pruebas para cada una de las muestras. Para llevar a cabo una adecuada sintonización de los parámetros de control, es necesario realizar un barrido de los valores correspondientes a una de las variables, manteniendo fijos los demás, hasta identificar el valor que mejore la calidad de las soluciones. Una vez obtenido el mejor valor para esta variable, se fija dicho valor y se comienza con la variación de otro

parámetro, llevando a cabo el mismo proceso hasta obtener el conjunto de valores que permitan al algoritmo mejorar la eficiencia y eficacia. Una vez obtenido los mejores valores para cada parámetro, realizamos muestras para un rango más pequeño, tomando como punto medio, el valor fijado, de modo que se vuelva a realizar el proceso de sintonización hasta fijar nuevamente los valores.

d) Sintonización de Parámetros.

Los valores obtenidos al final de cada una de las muestras al término de la evaluación, serán considerados como los valores de sintonización de los parámetros de control del algoritmo. Estos serán aquellos valores que influyan de manera positiva en la calidad de la solución, lo que permite tener un mejor desempeño del algoritmo.

3.5. Análisis de Complejidad del Algoritmo Búsqueda Local Iterada

Una vez definidos los parámetros de control que hacen que el algoritmo funcione correctamente, es necesario realizar un estudio que permita conocer su comportamiento y así poder medir su rendimiento, centrándose principalmente en su simplicidad y el uso eficiente de los recursos.

La complejidad computacional de un algoritmo se puede clasificar de acuerdo a la dificultad de resolverlo en función de los siguientes parámetros:

Espacio. Cantidad de memoria requerida para almacenamiento de los datos durante la ejecución del algoritmo.

Tiempo. Duración de la ejecución del algoritmo.

Ambos parámetros representan el costo requerido por el algoritmo según el tipo de problema que se está tratando, para encontrar una solución.

El tiempo de ejecución de un algoritmo o complejidad temporal T(n), donde n es el tamaño de la entrada, está en función de los parámetros: Datos de entrada, velocidad del procesador y complejidad del algoritmo. La complejidad temporal representa el número de instrucciones simples (asignaciones, comparaciones, operaciones aritméticas, entre otras más) que serán ejecutadas por el algoritmo. Por lo regular se considera la complejidad del algoritmo en el peor de los casos, aunque también es importante conocer la complejidad en el mejor y el caso promedio.

Para clasificar si un algoritmo es considerado como bueno o malo se tiene que basar en las siguientes convenciones:

1. Todos los algoritmos, desde constantes hasta polinomiales, son polinomiales y son clasificados cómo buenos algoritmos.

2. Todos los algoritmos exponenciales y factoriales, son exponenciales y son clasificados cómo malos algoritmos.

Para calcular la complejidad del algoritmo Búsqueda Local Iterada, es necesario realizar un análisis del número de instrucciones requeridas por el algoritmo para encontrar una solución al problema tratado. De acuerdo a esto, se muestra la ecuación correspondiente a la función temporal del algoritmo (Ecuación 1).

$$T(n) = 56 + 17n^3 + n^2 (128 + 69m^2) + n^4 (148 + 24m^2)$$

Donde ***n*** representa el donde n es el tamaño de la entrada para el algoritmo, y ***m*** las dimensiones de las figuras que se posicionan en la pantalla, estas dimensiones por convenio tienen el mismo valor en pixeles, área de trabajo u hoja de papel.

De acuerdo a esto, se puede concluir que la complejidad del algoritmo secuencial de Búsqueda local Iterada con la estructura de vecindad simple que se le aplicó a la búsqueda local para el problema de empaquetamiento en contenedores aplicado a imprentas, tiene una complejidad en el peor de los casos es de $O(n^4 * m^2)$.

Capítulo 4

Resultados Experimentales

En este capítulo se presentan los resultados obtenidos en las pruebas experimentales realizadas al Algoritmo de Búsqueda Local Iterada, donde se integra la estructura de vecindad a la búsqueda local, para el Problema de empaquetamiento en contenedores en dos dimensiones aplicado a imprentas. Estos resultados son comparados con los obtenidos por el mismo algoritmo cambiando la estructura de vecindad a una sencilla, la que va actualizando cada una de las posiciones en las que se encuentran ubicadas las figuras.

Finalmente, se llevó a cabo un análisis para mostrar la eficiencia y eficacia del algoritmo, comparando los resultados obtenidos por la Búsqueda Local Iterada en sus dos versiones (con una estructura de vecindad sencilla y con una estructura de vecindad en la cual se actualizan todas las posiciones de las figuras), con los obtenidos por un algoritmo a partir del cual se obtiene la solución óptima.

4.1. Descripción del equipo utilizado

Para llevar a cabo la implementación del Algoritmo de Búsqueda Local Iterada aplicada al problema de empaquetamiento en contenedores en dos dimensiones aplicado a imprentas, se utilizó una computadora portátil con las siguientes características:

- Procesador: Intel(R) Core(TM) i5-4200 CPU @ 2.3 Ghz
- Memoria: 8.0 GB
- Sistema Operativo: Windows 8.1 64 bit
- Compilador: Microsoft Visual C++ 2012

En el caso de las pruebas realizadas al Algoritmo de Búsqueda Local Iterada aplicada al problema de empaquetamiento en contenedores en dos dimensiones en

imprentas, se utilizó equipo perteneciente al laboratorio de Optimización. A continuación se enlistan las características del equipo utilizado:

- Procesador: Intel(R) Core(TM) i7 CPU 870 @ 2.93 GHz
- Memoria: 5.0 GB
- Sistema Operativo: Windows 7 Ultimate 64 bits
- Compilador: Microsoft Visual C++ 2012

4.2. Análisis de Sensibilidad

Para llevar a cabo el análisis de sensibilidad de los parámetros de control del algoritmo Búsqueda Local Iterada aplicado al problema de empaquetamiento de contenedores en dos dimensiones en imprentas, se propuso una metodología, la cual fue explicada anteriormente (Capítulo 3). A continuación se aplica la metodología al algoritmo Búsqueda Local Iterada.

a) Selección de los parámetros de control.

De acuerdo al análisis realizado tanto al método aplicado como al problema tratado, las variables utilizadas para llevar a cabo el análisis de sensibilidad del algoritmo Búsqueda Local Iterada aplicado al problema de empaquetamiento en contenedores en dos dimensiones en imprentas son las siguientes:
Cantidad de traslapes por soluciones (***Traslapes***)
Cantidad de iteraciones por unidad de tiempo (***Iteraciones***)
Iteraciones en función del tamaño de la estructura de vecindad (***CK***), donde C es una constante, y ***K*** es tamaño de la estructura de vecindad.

b) Establecer Rangos de Evaluación.

Una vez que los parámetros de control se han identificado, es necesario establecer los rangos que van a ser utilizados para realizar el análisis de sensibilidad a cada una de las variables antes mencionadas.

Haciendo un análisis de los valores de cada una de las variables de control, se tiene que si ***Traslapes*** es muy pequeña puede que el algoritmo rápidamente arroje una solución en un tiempo determinado, siendo este tiempo muy pequeño, entonces lo que se estableció fue fijar un valor de la cantidad de veces que se traslaparon las figuras, este valor fue grande para darle la posibilidad al algoritmo de seguir calculando posiciones e ir buscando combinaciones para que se insertaran más figuras, se recuerda que este algoritmo es una Heurística de Inserción, que cada vez se quiere que se inserten más figuras a la hoja de papel.

En cuanto a la cantidad de iteraciones (***Iteraciones***), ¿Si este parámetro es pequeño? Entonces puede que el algoritmo rápidamente arroje una solución en un tiempo muy pequeño, entonces lo que se estableció fue fijar un valor de ***Iteraciones*** grande para darle la posibilidad al algoritmo de seguir calculando posiciones e ir buscando combinaciones para que se insertaran más figuras a la hoja de papel.

En cuanto al parámetro ***K***, este parámetro representa el tamaño de la estructura de vecindad, por ejemplo la mejor solución que arrojó el algoritmo implementado fue para 18 figuras, es decir se pudieron introducir 18 figuras en una zona de trabajo que en pixeles mide 800x600 pixeles, y las imágenes son de 150x150 pixeles, vale decir que cada imagen tiene asociada una figura, cada una de esas figuras tienen las libertades de moverse a la izquierda, a la derecha, arriba, abajo, y de forma diagonal en ambos sentidos, las ***Figuras 4.1 y la Figura 4.1.0,*** muestran las posibles formas en que se puede mover una figura dentro del área de trabajo u hoja de papel.

Figura 4.1. *Posiciones en las cuales se pueden mover cada una de las figuras.*

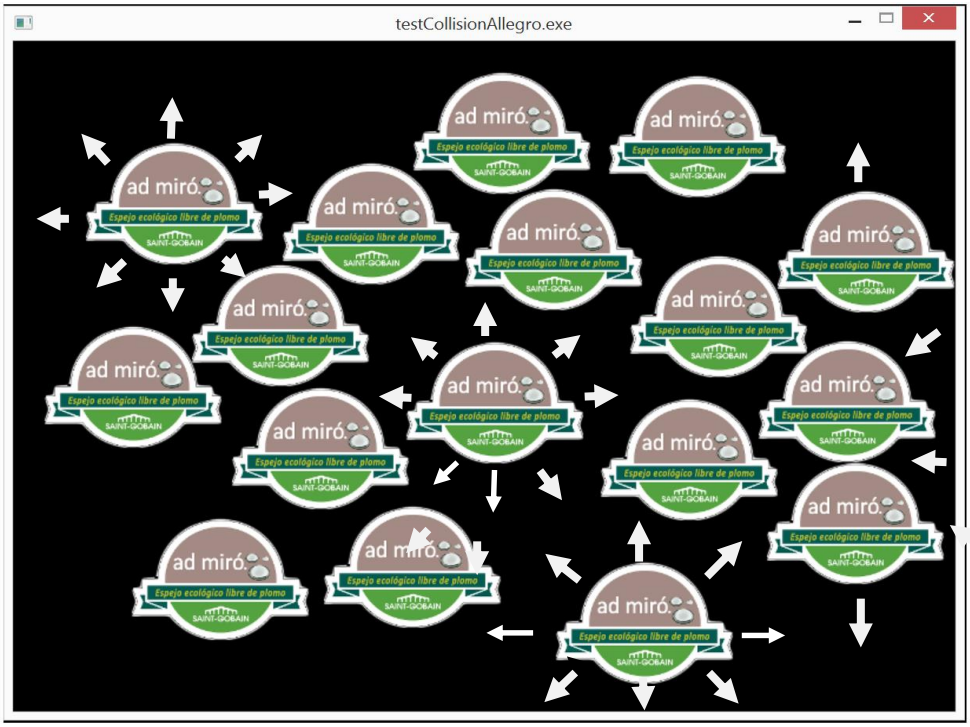


Figura 4.1.0. *Posiciones en las cuales se pueden mover cada una de las figuras.*

Según lo que se muestra en la ***Figura 4.1,*** cada figura tiene la posibilidad de moverse en la dirección que indican las flechas de color negro, entonces, las figuras tienes 8 posibles movimientos que pueden hacer, y la mejor solución arrojada por el algoritmo es de 18 figuras, entonces multiplicando el número máximo de figuras insertadas en la hoja de papel, conjuntamente con el número de posibles movimientos, da como resultado: ***K*** = 18x8 = 144, este valor de ***K*** = 144, representa el tamaño de la estructura de vecindad.

El número de iteraciones para el que se sintonizo el algoritmo y a partir del cual se obtuvo la mejor solución fue el valor del ***#Iteraciones*** = 10000, por lo que este valor se divide entre el tamaño de la estructura de vecindad y da como resultado el valor de 70, por lo que el parámetro Iteraciones en función del tamaño de la estructura de vecindad, queda sintonizado con el valor 70k, donde ***k*** = 144, y C = 70, cuando se multiplica el valor de $CK \approx 10000$, y en el caso del tiempo sería aproximadamente dos horas porque hay que recordar que este algoritmo es para aplicarlo a una imprenta y tiene que ser eficiente en cuanto al tiempo, por lo que lo máximo en que se debe tardar para arrojar una solución debe ser aproximadamente dos horas, porque así se le da la posibilidad al algoritmo de ir buscando combinaciones para que se insertaran más figuras a la hoja de papel.

Por lo que se concluye que es de vital importancia encontrar la proporción adecuada entre los parámetros de control, dentro de la cuál el algoritmo obtiene buenas soluciones. De acuerdo a lo mencionado anteriormente y a los valores utilizados, se proponen los siguientes rangos a evaluar para cada una de las variables de control (Tabla 4-1).

Tabla 4-1. *Rangos utilizados para realizar el análisis de sensibilidad al algoritmo Búsqueda Local Iterada.*

Parámetro de Control	Límite Inferior	Límite Superior
#Traslapes	100	10000
#Iteraciones	1000	10000
CK	70k	70k

c) Pruebas a Rangos de Evaluación.

Para realizar el análisis de sensibilidad a los parámetros de control mencionados anteriormente, se lleva a cabo lo siguiente:

Inicialmente, para cada uno de los rangos mostrados en la tabla anterior se calcularon 10 muestras con los incrementos mostrados en la tabla 4-2.

Tabla 4-2*. Valores de incrementos utilizados para los rangos de los parámetros de control utilizados para el análisis de sensibilidad.*

Parámetro de Control	Incremento
#Traslapes	5000
#Iteraciones	10000
CK	70k

El análisis de sensibilidad se llevó a cabo realizando un barrido de las 10 muestras de #***Traslapes***, de acuerdo al incremento presentado en la Tabla 4-2, manteniendo constante el valor de los parámetros #***Iteraciones***, ***CK***, es decir el tiempo sería de una a dos hora por cada serie de pruebas. Cabe mencionar que, por cada muestra evaluada se realizaron un total de 30 ejecuciones.

Una vez que se ha realizado la evaluación de los resultados obtenidos para cada una de las muestras, se fija el valor de #***Iteraciones*** que haya obtenido los mejores resultados de acuerdo a la función objetivo del problema tratado.

Se vuelven a realizar las pruebas, esta vez realizando un barrido del valor de #***Traslapes*** y manteniendo constante los valores de los parámetros #***Iteraciones*** y ***el*** parámetro ***CK*** respectivamente.

Al igual que en el caso de las #***Iteraciones*** se realiza una serie de 30 ejecuciones por cada incremento evaluado, fijando los valores de #***Traslapes*** y parámetro ***CK,*** que obtenga las mejores soluciones.

El mismo proceso se lleva a cabo para los valores de #***Iteraciones*** y el parámetro ***CK***, de modo que prevalezcan sólo aquellos valores que permitan obtener los mejores resultados. De acuerdo a esto, se obtuvieron los siguientes valores (Tabla 4-3).

Tabla 4-3. *Valores de los parámetros de control, fijados de acuerdo al análisis de sensibilidad.*

Parámetro de Control	Valor Sintonizado
#Traslapes	5000
#Iteraciones	10000
CK	70k

d) Sintonización de Parámetros.

Los valores obtenidos para cada uno de los parámetros de control evaluados, al término de la evaluación de todas y cada uno de los incrementos (Tabla 4-2), dan como resultado la sintonización de los parámetros de control.

La sintonización de parámetros de control se lleva a cabo con la finalidad de identificar aquellos valores correspondientes a las variables de control, que influyen de manera positiva en la calidad de la solución, lo que permite obtener una mejora en el desempeño del algoritmo.

Con base en el análisis de sensibilidad realizado, se obtuvieron los siguientes valores sintonizados (Tabla 4-4).

***Tabla 4-4**. Valores sintonizados correspondientes a los parámetros de control evaluados durante el análisis de sensibilidad.*

Parámetro de Control	Valor Sintonizado
Traslapes	5000
Iteraciones	10000
CK	70k

4.3. Aplicación de la Estructura de Vecindad

Para mejorar la calidad de las soluciones obtenidas por el algoritmo de Búsqueda Local Iterada propuesto para el problema de empaquetamiento en contenedores en dos dimensiones aplicado a Imprentas, se decidió llevar a cabo la implementación de una estructura de vecindad sencilla, que permitió tener una mejor explotación del espacio de soluciones.

Para aplicar la estructura de vecindad fue necesario realizar una evaluación del algoritmo Búsqueda Local Iterada que permitió identificar el proceso que requería ser mejorado por dicha estructura, de esta forma, se llegó a la conclusión de que cada vez que se termina una iteración, es decir, cuando se hayan ubicado las figuras de forma aleatoria en la zona de trabajo o hoja de papel, se lleva a cabo la comparación entre las soluciones obtenidas para identificar la mejor solución hasta el momento, una vez que se tiene identificado este mejor valor de la función objetivo hasta el momento, se le aplica la búsqueda local con la estructura de vecindad

simple, de modo que en cada iteración se vaya mejorando el mejor valor de la función objetivo hasta el momento.

La mejor solución que arrojo el algoritmo implementado fue para 18 figuras, es decir se pudieron introducir 18 figuras en la zona de trabajo que en pixeles mide 800x600 pixeles cuadrados, y las imágenes son de 150x150 pixeles cuadrados, entonces multiplicando el número máximo de figuras insertadas en la zona de trabajo u hoja de papel, conjuntamente con el número de posibles movimientos, da como resultado un valor igual a 144 porque se multiplica el número de figuras por la cantidad de movimientos que estas figuras pueden hacer y esa cantidad de movimientos es 8, estos movimientos son, arriba, abajo a la izquierda a la derecha y de forma diagonal en ambos sentidos, el valor de 144, representa el tamaño de la estructura de vecindad, para entender este proceso se especifica la ***Figura 4.1.1 que*** muestra las posibles formas en que se pueden mover las misma dentro de la zona de trabajo u hoja de papel, y es la mejor solución obtenida por el algoritmo implementado.

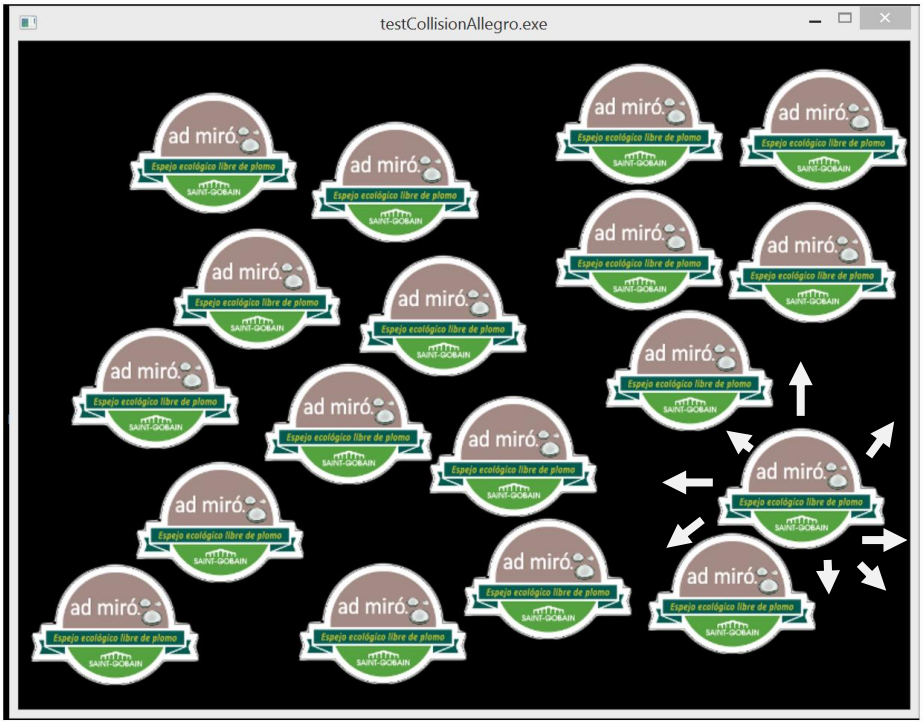


Figura 4.1.1. *Posiciones en las cuales se pueden mover cada una de las figuras cuando se aplica la estructura de vecindad.*

4.4. Resultados Experimentales

Para conocer el comportamiento de un algoritmo, es necesario recurrir al cálculo de parámetros estadísticos como son la media y desviación estándar. La media permite obtener el promedio de las soluciones encontradas para cada instancia. La desviación estándar permite conocer que tan dispersas están las soluciones con respecto a la media aritmética. De acuerdo a los datos obtenidos se puede determinar si un algoritmo es bueno o no para cierto tipo de problema. El error relativo permite conocer el porcentaje de error de la mejor solución encontrada con respecto a una cota establecida, que para este caso es la solución óptima correspondiente a cada instancia. De acuerdo a los resultados obtenidos por dichos parámetros, se puede determinar si un algoritmo es bueno o no para cierto problema.

Para probar el funcionamiento del algoritmo, se generaron soluciones aleatorias, específicamente con una figura. En la ***Figura 4.2,*** se muestra una solución, y esta figura, representa una instancia real con la que trabajan las imprentas.

Figura 4.2*. Solución inicial, generada aleatoriamente a partir de una figura que representa una instancia real con la que trabajan las imprentas.*

Una vez obtenida esta solución inicial como se muestra en la ***Figura 4.2***, se puede observar que existen espacios vacíos en donde pueden posicionarse figuras, se ilustra un ejemplo y se coloca un circulo de color de fondo blanco, una vez obtenida esta solución inicial se prosigue a mejorarla con una estructura de vecindad simple que ya se ha explicado en varias ocasiones. Luego de aplicarse esa estructura de vecindad se puede observar que la solución mejora, ya que aumenta el número de figuras, la ***Figura 4.3,*** ilustra que se insertó una nueva figura y esta es la que está rodeada por las flechas de color blanco.

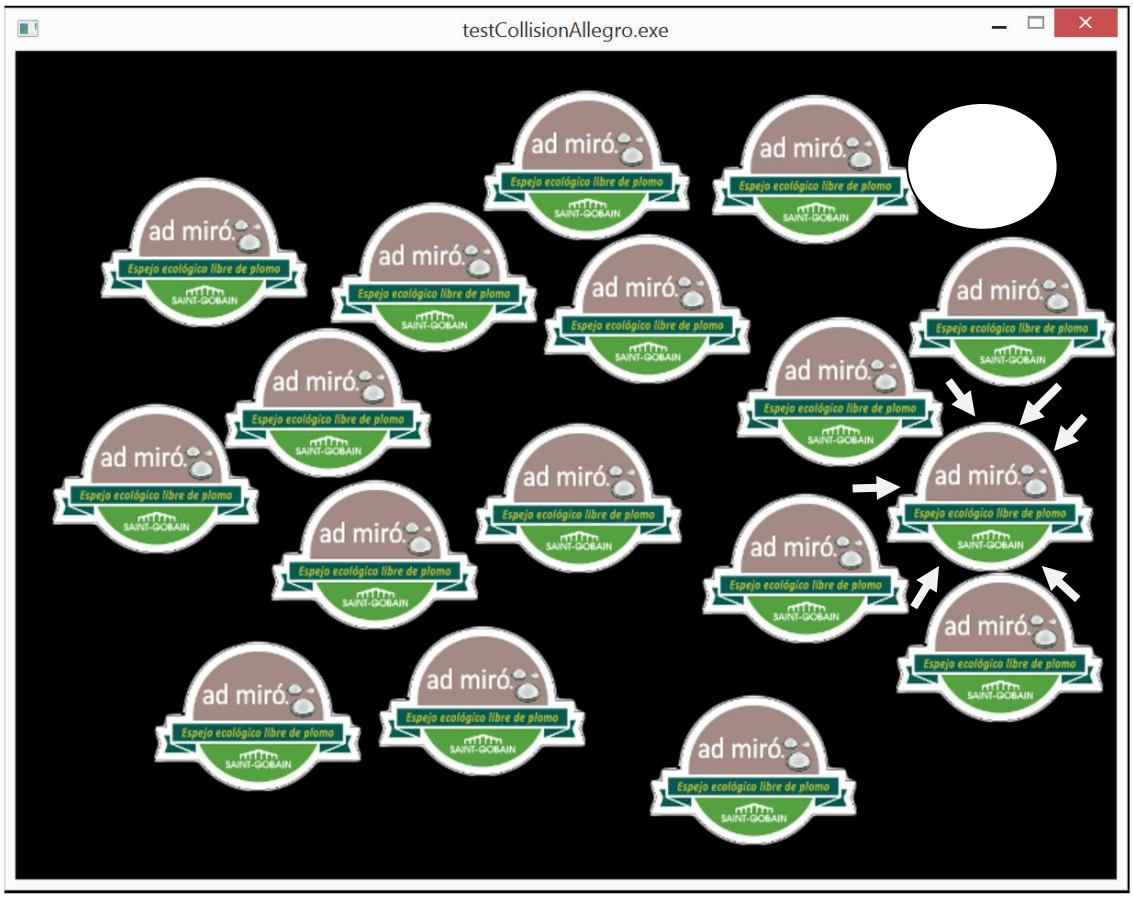


Figura 4.3. *Solución mejorada, a partir de la aplicación de una estructura de vecindad.*

Como se puede observar existen espacios disponibles y un ejemplo del mismo es donde está situado el círculo de color de fondo blanco, entonces al aplicar la búsqueda local iterada se puede decir que se aumentó en número de figuras la solución, por lo que se puede concluir que una vez aplicada la búsqueda local iterada se llegó a una mejor solución.

Figura 4.4. *Solución mejorada, a partir de la aplicación la búsqueda local iterada.*

En la ***Figura 4.4,*** muestra que se aumentó en el número de figuras la solución, por lo tanto se puede concluir que la solución mejoró, por ejemplo la figura que se insertó una vez que se aplicó la búsqueda local iterada es la que está rodeada de flechas de color. Esta es la mejor solución que se obtuvo al aplicar la estructura de vecindad que anteriormente se explicó.

Cabe especificar que al algoritmo implementado se le realizaron 30 pruebas experimentales, y una vez realizadas estas pruebas se obtuvo el promedio de las mismas, y una vez obtenido ese promedio, se puede decir que el algoritmo en el peor de los casos arrojó una solución de ***325496*** medida en unidades de pixeles cuadrados, en el mejor de los casos arrojó una solución de ***281352*** medida en pixeles cuadrados, obteniéndose una solución promedio de ***303424*** medida en pixeles cuadrados, este valor representa la media, también se hizo el cálculo de la

desviación estándar, hay que especificar que este cálculo se hizo con Excel, la desviación estándar arrojó un valor de ***1.50***, la Tabla 4-5, muestra estos resultados, así como también la cantidad de figuras insertadas en la zona de trabajo u hoja de papel por cada una de las soluciones, también se especifica que cada figura con la que se realizaron las pruebas, tienen un ancho y un largo, que se mide en dimensiones de pixeles, y estos valores, son los mismos, y miden 150 pixeles, es decir las dimensiones de las imágenes en pixeles es de 150x150, para un área total de ***22500*** pixeles cuadrados, y un área de pixeles activos de ***11036*** pixeles cuadrados, los pixeles activos son aquellos que conforman la figura en la imagen, y tienen un valor de 1, también existen los pixeles no activos que son aquellos que no aparecen dentro de las figuras en la imagen y que tienen un valor de 0. Las imágenes se trataron en el programa de diseño Photoshop, el cual proporciona herramientas para eliminar los fondos blancos de las imágenes, que en este caso el Photoshop fue el software que se utilizó para poner en 0 los pixeles que conforman la imagen, pero que no forman parte de los bordes de la figura y como tal de la figura. Otras cuestiones que hay que mencionar es que las dimensiones de la zona de trabajo u hoja de papel se expresaron en pixeles, en donde el ancho mide 800 pixeles y el alto mide 600 pixeles, es decir que las dimensiones en pixeles del área de trabajo u hoja de papel de 800x600 pixeles, para una área total de ***480000*** pixeles. La ***Tabla 4-5*** muestra los valores obtenidos del promedio de las 30 pruebas experimentales hechas al algoritmo implementado. La ***Tabla 4-6*** muestra la mejor solución, la peor solución y los ***Parámetros Estadísticos Desviación Estándar, Media, Mediana y Moda.***

En la ***Figura 4.4.1,*** se muestran los resultados obtenidos a partir del promedio de las pruebas realizadas al algoritmo implementado.

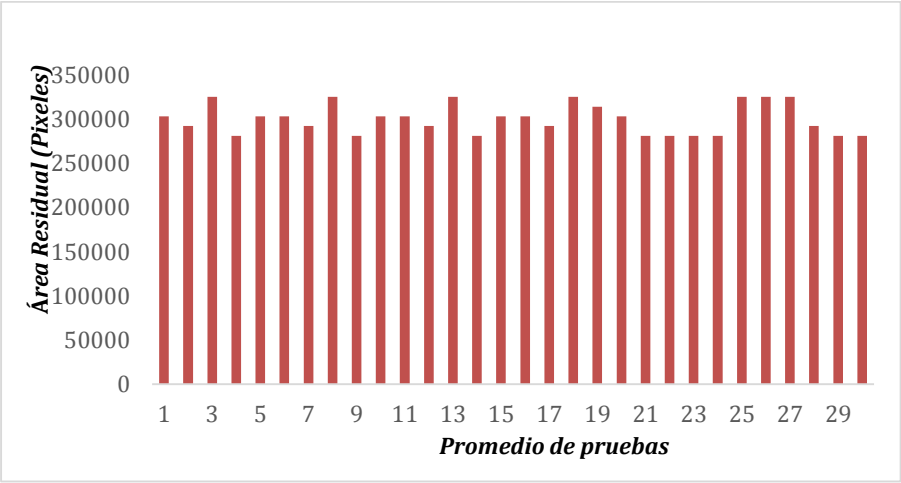


Figura 4.4.1. *Gráfica de la función objetivo o Área residual en unidades de pixeles en función del promedio del número de pruebas.*

Cabe resaltar que la gráfica de la ***Figura 4.4.1,*** representa las 30 pruebas que se le hicieron al algoritmo implementado, en donde se lograron insertar hasta 18 elementos o figura dentro del área de trabajo u hoja de papel y dentro del vector que representa la solución.

En la ***Tabla 4-5***, se muestran los resultados obtenidos a partir de las 30 pruebas experimentales hecha al algoritmo implementado, hay que especificar que la tabla está conformada por tres campos, estos son: Número de Pruebas, Función Objetivo o Área residual medida en pixeles cuadrados, así como la Cantidad de Figuras respecto a la Función Objetivo y los 30 resultados por cada una de las pruebas, los colores que se muestran en esta tabla tienes el siguiente significado, el color rojo significa que es la peor solución, el color rojo vino significa que la solución es muy cercana a la peor pero no es la peor, el color amarillo oscuro es una solución intermedia que está entre la peor solución y la mejor solución, el color verde claro es una solución que está próxima a la mejor solución y el color verde significa que se está en presencia de la mejor solución. A continuación se muestra la ***Tabla 4-5.***

***Tabla 4-5**. Valores de las soluciones obtenidas.*

Número de Pruebas	Función Objetivo o Área residual en pixeles cuadrados	Cantidad de Figuras
1	303424	16
2	292388	17
3	325496	14
4	281352	18
5	303424	16
6	303424	16
7	292388	17
8	325496	14
9	281352	18
10	303424	16
11	303424	16
12	292388	17
13	325496	14
14	281352	18
15	303424	16
16	303424	16
17	292388	17
18	325496	14
19	314460	15
20	303424	16
21	281352	18
22	281352	18
23	281352	18
24	281352	18
25	325496	14
26	325496	14
27	325496	14
28	292388	17
29	281352	18
30	281352	18

***Tabla 4-6**. Valores de las soluciones obtenidas, y valores estadísticos.*

Calidad de la Solución	Función Objetivo o Área residual en pixeles cuadrados	Cantidad de Figuras
Peor	325496	14
Mejor	281352	18
Desviación estándar	16882.66303	1.50
Media	300481.07	16.233
Mediana	302434	16
Moda	281352	18

En la ***Tabla 4-6 se*** muestran la mejor solución, la peor solución y los ***Parámetros Estadísticos Desviación Estándar, Media, Mediana y Moda,*** el color rojo pertenece a la peor solución obtenida y el color verde a la mejor solución obtenida, el color azul claro pertenece a los parámetros estadísticos que se calcularon a partir de los resultados experimentales.

También se implementaron casos de pruebas con distintos tipos de figuras, y estas figuras representan instancias reales de figuras que trabajan las imprentas, las ***Figura 4.5 (a) y (b),*** muestran que es posible la realización de un algoritmo que permita hacer la combinación de figuras en una misma solución, es decir que se puede en una misma solución tener figuras de distintos tipos, y precisamente esta parte es la parte novedosa de este trabajo de investigación.

Figura 4.5. (a) *.Caso de prueba con varias figuras.*

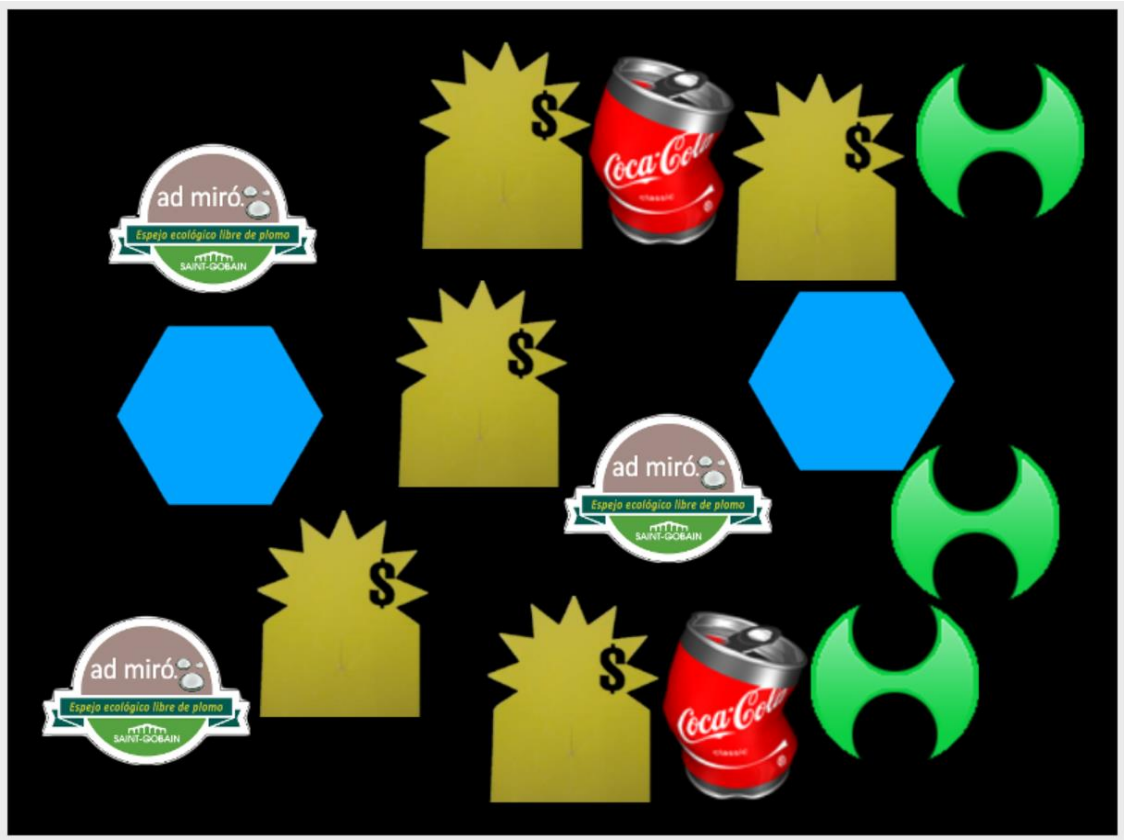


Figura 4.5. (b) *Caso de prueba con varias figuras.*

4.5. Análisis de los Resultados

Para la implementación de este algoritmo se utilizó el lenguaje de programación C++ en la plataforma Visual Studio .Net 2012, y como hardware se utilizó una laptop con procesador Core i5 con 8 Gigas de Memoria RAM. Luego de hacer 30 pruebas se presentan los resultados a partir del promedio de las pruebas hechas al algoritmo implementado.

En la ***Figura 4.6,*** muestra la gráfica de la función objetivo o área residual en unidades de pixeles, en función de la cantidad de iteraciones.

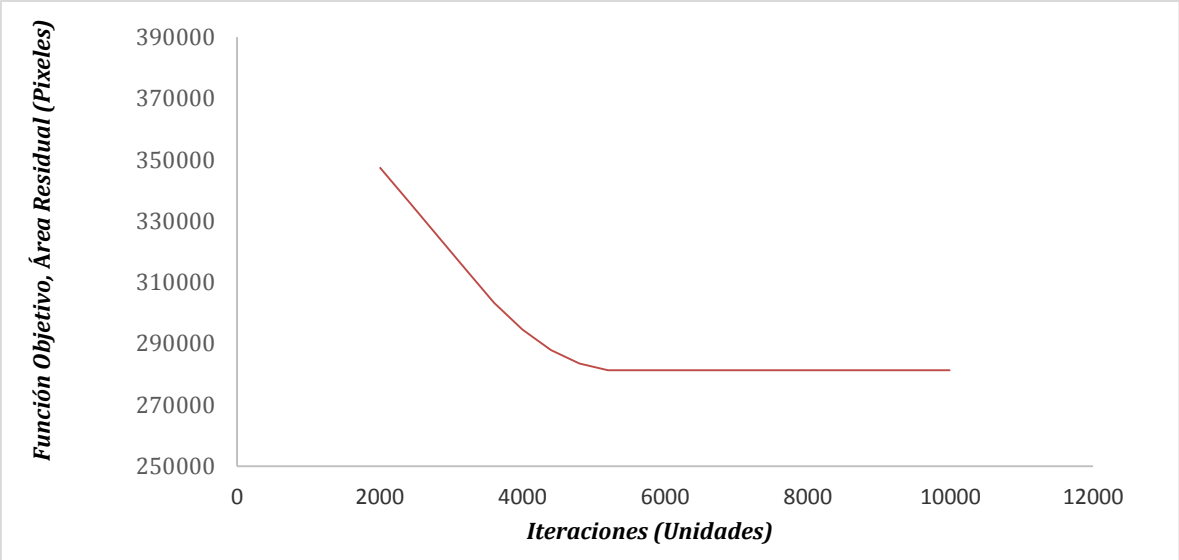


Figura 4.6*. Gráfica de la función objetivo o Área residual en unidades de pixeles en función del número de iteraciones.*

Aquí se puede apreciar que el área residual o la función objetivo decrece en forma lineal hasta llegar a un valor determinado que es el menor valor de la función objetivo que se obtiene; luego de llegar a este valor, el área residual o función objetivo se mantiene constante a medida que aumentan las iteraciones del algoritmo. Lo anterior significa que a partir de este valor el algoritmo comienza a converger aproximadamente en las 5000 iteraciones y cuando llega a esta cantidad de iteraciones alcanza su mínimo local, la función. Después de 5000 iteraciones aproximadamente, el área residual o función objetivo comienza a ser constante, es decir el mismo; es entonces cuando se dice que se llega a un mínimo local.

En la ***Figura 4.7*** se representa la gráfica del área residual o función objetivo en función del tiempo. Aquí se puede apreciar que, a medida que transcurre el tiempo, el área residual o función objetivo disminuye. Como se observa en la ***Figura 4.7***, desde que comenzó la ejecución del algoritmo hasta transcurridos aproximadamente 200 minutos, el valor del área residual o función objetivo comienza a decrecer y llega un momento en que comienza a ser constante. Este valor donde el área residual o función objetivo comienza a ser constante, representa el mínimo local, luego se observa que, aproximadamente a partir de los 200 minutos y hasta los 600 minutos, el área residual o función objetivo, comienza a ser constante y entonces se manifiesta la convergencia del algoritmo implementado.

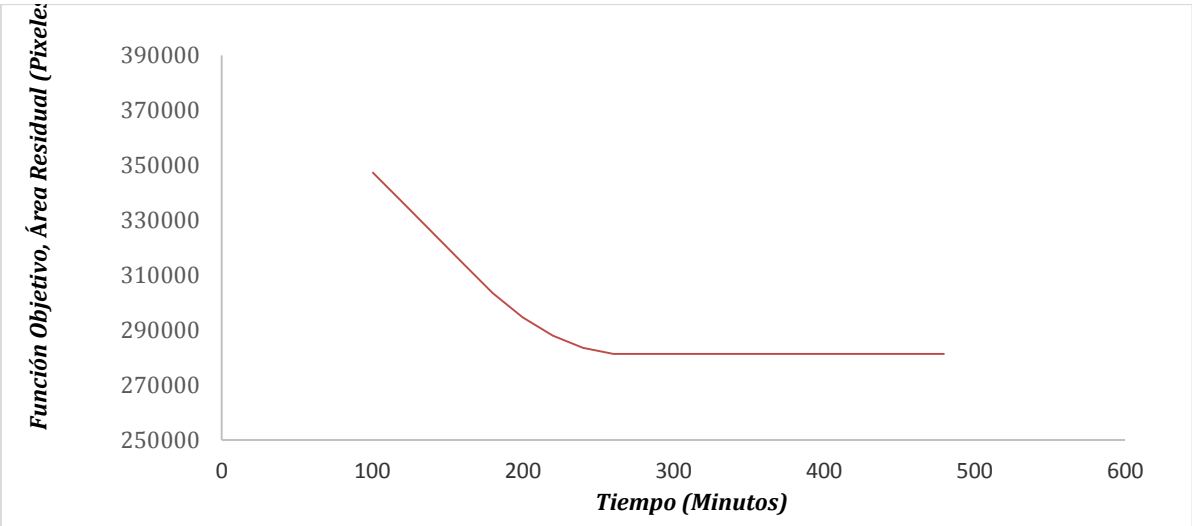


Figura 4.7. *Gráfica Área residual en unidades de pixeles en función del tiempo.*

Tanto la ***Figura 4.6*** como la ***Figura 4.7*** muestran las pruebas experimentales de convergencia de algoritmo implementado. La primera gráfica, ***Figura 4.6*** representa el promedio de las pruebas realizadas para encontrar la convergencia del algoritmo y la misma representa la disminución del área residual o función objetivo a partir de un número de iteraciones dados.

La segunda gráfica, ***Figura 4.7***, representa las pruebas realizadas para encontrar la convergencia del algoritmo y la misma representa la disminución del área residual o función objetivo a partir de un tiempo determinado. En ambas pruebas se lograron

insertar 18 elementos o figuras dentro del área de trabajo u hoja de papel, así como también se insertaron en el vector que representa las soluciones.

4.6 Análisis de Eficacia y Eficiencia del Algoritmo

El análisis de eficacia y eficiencia del algoritmo propuesto se lleva a cabo para evaluar la calidad de las soluciones obtenidas, como los recursos y tiempo necesarios para su ejecución.

Una vez implementado el Algoritmo de Búsqueda Local Iterada para la darle solución al problema de empaquetamiento en contenedores en dos dimensiones aplicado a imprentas se, puede decir, que según el valor de
$T(n) = 56 + 17n^3 + n^2 (128 + 69m^2) + n^4 (148 + 24m^2)$, el algoritmo implementado tiene una complejidad temporal para el peor de los casos de $O(n^4 * m^2)$, y se puede decir que el algoritmo no es eficiente, porque como se puede observar, n está elevado a la cuarta y el comportamiento que tiene es un comportamiento parabólico, lo cual hace que no sea eficiente debido a este comportamiento n^4, pero si se logra algo de eficacia porque con el algoritmo se logra posicionar figuras tanto regulares como irregulares en la pantalla sin que se produzcan traslapes entre ellas. Esta falta de eficacia es debido a que la estructura de vecindad utilizada no es del todo buena, por lo que se deberá mejorar. El algoritmo implementado es completamente eficaz en cuanto a la detección de traslapes entre las figuras, así como también en cuanto al cálculo del área de cualquier tipo de figura ya sea regular o irregular.

Como se expresó en el párrafo anterior, este algoritmo tiene deficiencias en cuanto a la estructura de vecindad, ya que esta estructura de vecindad no es cien por ciento eficaz y teniendo en cuenta esto, en los trabajos futuros se recomienda, diseñar y desarrollar una estructura de vecindad que sea lo suficientemente inteligente para ganar en espacios y así poder insertar más figuras a la hoja de papel. La ***Figura 4.8,*** muestra que el algoritmo implementado es eficiente en cuando a la detección se

traslapes entre las figuras, pero también muestra que se tiene que desarrollar una estructura de vecindad que permita ocupar los espacios vacíos en la hoja de papel.

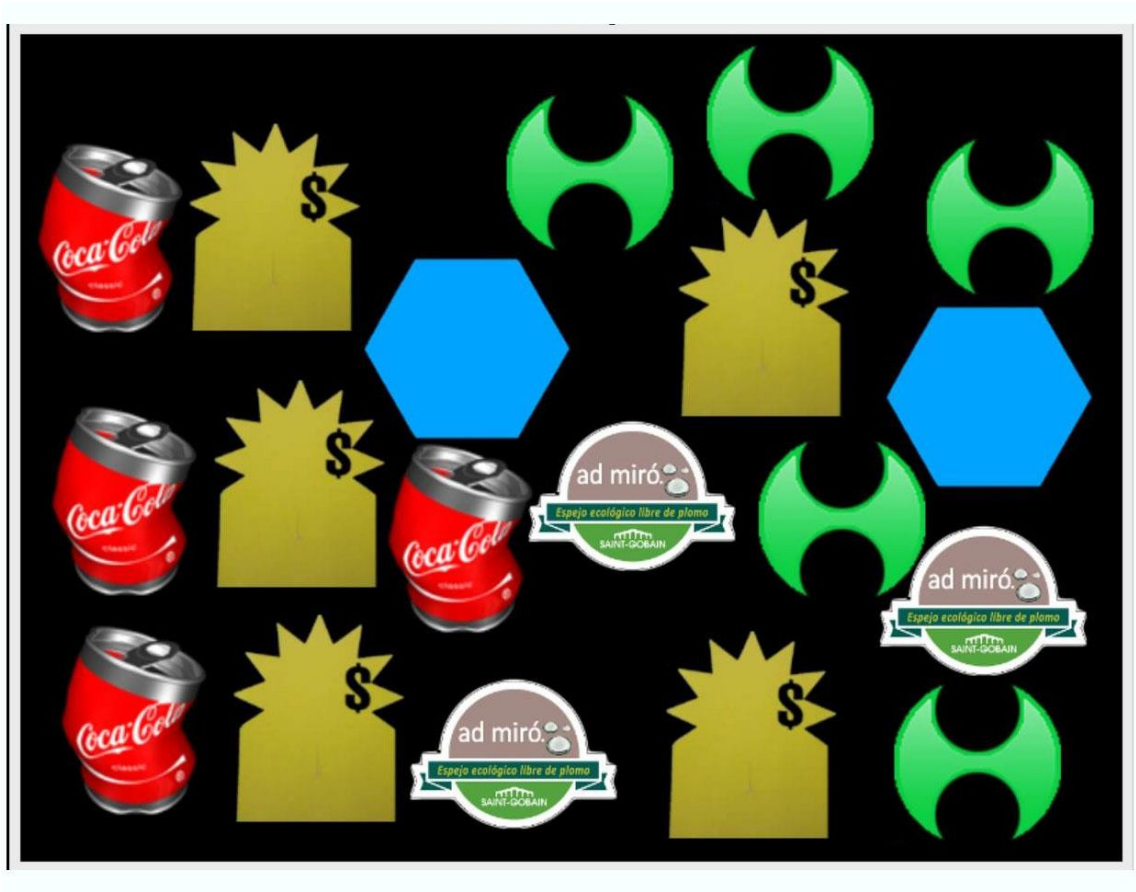


Figura 4.8. *Eficacia en el algoritmo de detección de traslapes.*

Capítulo 5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

La aportación de este trabajo de investigación es la implementación de un mapeo del problema de empaquetamiento en dos dimensiones, al problema del desperdicio de papel en una imprenta y tratado con una heurística computacional.

Una vez concluido este trabajo de investigación se realizó la implementación de una heurística computacional para resolver el problema de empaquetamiento en contenedores en dos dimensiones, así como una estructura de vecindad simple, con el fin de encontrar espacios disponibles para insertar las figuras en el contenedor u hoja de papel. Se percibió que existieron deficiencias en cuanto al diseño de la estructura de vecindad, ya que las estructuras de vecindades conocidas hasta el momento no funcionaron con la eficiencia que se esperaban para la aplicación de este trabajo de investigación, por lo que para este tipo de problemas se requiere el diseño de una nueva estructura de vecindad.

El algoritmo implementado que resuelve el problema de empaquetamiento en contenedores en dos dimensiones aplicado al problema de desperdicio de papel en imprentas, es eficaz en cuanto a la detección de traslapes entre las figuras, así como también en cuanto al cálculo del área de cualquier tipo de figura ya sea regular o irregular.

La herramienta generada de la interfaz visual sirve para validar de forma simple la solución encontrada al problema de empaquetamiento en contenedores debido a que ahí se puede observar si existen o no traslapes de figuras tanto regulares como irregulares.

Las pruebas experimentales realizadas a la implementación, mostraron que es posible optimizar el papel en las imprentas.

5.2. 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 herramienta para la disminución del desperdicio de papel en una imprenta, por ejemplo una aplicación visual que contenga el algoritmo implementado embebido, y que se pueda acceder desde internet.

Realizar una mejora en la estructura de vecindad, es decir desarrollar una estructura de vecindad específicamente para este problema, porque las que se conocen hasta el momento, a la hora de aplicarlas acá no funcionan, porque trabajan con un número de elementos que es fijo, y en el caso de este problema, el número de elemento o figuras en todo momento estará cambiando porque el algoritmo trata de insertar más elementos a la hoja de papel, a este tipo de procedimiento, se le llamo en este trabajo de investigación, heurística de inserción. La estructura de vecindad tiene que ser capaz de incluir todos los tipos de transformaciones que tengan las figuras, por ejemplo, tienen que incluir los movimientos a la izquierda, derecha, arriba, hacia abajo, de forma diagonal en todos los sentidos y además poder rotar, con el fin que se puedan obtener mejores soluciones.

Implementar un algoritmo genético para este problema con el fin de explorar el espacio de soluciones de forma más profunda y luego teniendo ya el desarrollo de una estructura de vecindad eficiente y eficaz poder obtener mejores resultados.

La implementación en paralelo mediante Ambiente Grid, del algoritmo genético que anteriormente se mencionó, con el propósito de disminuir los tiempos de respuestas del algoritmo implementado, 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, y por eso se recomienda el uso del ambiente Grid para poder disminuir los tiempos de respuestas.

Referencias

[Andrade, 2013]. Two-stage two-dimensional guillotine cutting stock problems with usable leftover. Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. Department of Production Engineering, Federal University of Carlos, Via Washington Luiz km. 235, 13565-905, São Carlos, SP - Brazil.

[Agarwal y Grossman, 2004]. Agarwal A., Grossman I. E. Modeling for Optimization of Hybrid Dynamic Networks. Submitted to Computers Chemical Engineering, 2004.

[Aho, 1974]. Aho Alfred V., Hopcroft E. John, Ullman Jeffrey D. The Design and Analysis of Computer Algorithms. Ed. Addison-Wesley Publishing Company. ISBN. 0-201 00029-6. pp. 2, 60. 1974.

[Alba, 2005]. AlbaEnrique. Parallel Metaheuristics. A New Class of Algorithms. ISBN-13 978-0-471-67806-9, ISBN-10 0-471-67806-6. Canada. 2005.

[Allegro, 2011]. A game programming library. Retrieved Enero 7, 2016, from http://liballeg.org/readme.html

[Alonso, Cordón y Fernández-de Viana, 2004]. Alonso Sergio, Cordón Oscar, Fernández-de Viana Iñaki, Herrera Francisco. La Metaheurística de Optimización Basada en Colonia de Hormigas: Modelos y Nuevos Enfoques. Optimización Inteligente: Técnicas de Inteligencia Computacional para Optimización. ISBN. 84-9747-034-6. pp. 261-314- 2004.

[Anagnostopoulos y Rabadi, 2002]. Anagnostopoulos Georgios C., Rabadi Ghaith. A Simulated Annealing Algorithm for the Unrelated Parallel Machine Scheduling

Problem. School of Electrical Engineering & Computer Science, Orlando Florida, 2002.

[Anastova y Dror, 1998]. Anastasova K., Dror M. Intelligent Scheduler for Processing Help Requests on Unrelated Parallel Machines in a Computer Support Administration System, Conference on Systems, Man, and Cybernetics, IEEE, pp. 372-377, ISBN: 0-7803-4778-1, 1998.

[Anil y Marathe, 2005]. Anil Kumar V.S., Marathe M. V., Approximation Algorithms for Scheduling on Multiple Machines, Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), pp. 254-263, ISBN: 0-7695-2468-0/05, 2005.

[Arnaout, Musa y Ghaith, 2008]. Arnaout Jean-Paul, Musa Rami, Rabadi Ghaith. Ant Colony Optimization Algorithm to Parallel Machine Scheduling Problem with Setups. 4th IEEE Conference on Automation Science and Engineering. ISBN: 978-1-4244-2023-0/08. 2008.

[Bennell, 2012]. Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints. European Journal of Operational Research. 2013 Elsevier B.V. All rights reserved.

[Bisbal, 2009]. Bisbal Riera Jesús. "Manual de Algorítmica". Recursividad, Complejidad y Diseño de Algoritmos. Ed. UOC. Primera Edición en Castellano. ISBN: 978-84-9788-027-5. pp. 54-59. Barcelona. 2009.

[Blazewicz, Drozdowski, Soniewicki y Walkowiak, 1989] Blazewicz, Drozdowski, Soniewicki, Walkowiak., "TWO-DIMENSIONAL CUTTING PROBLEM BASIC COMPLEXITY RESULTS AND ALGORITHMS FOR IRREGULAR SHAPES", 1989.

[Bullnheimer, Hartl y Strauss, 1997]. Bullnheimer Bernd, Hartl Richard F., Strauss Christine. Applying the Ant System to the Vehicle Routing Problem. 2nd International Conference on Metaheuristics. MIC 97. Francia. 1997.

[Blum y Schmid, 2013], Blum Ch., Schmid V, Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm. International Conference on Computational Science, ICCS 2013.

[Brassard, Gilles, Bratley y Paul, 1997], Algoritmos voraces. Fundamentos de Algoritmia. Madrid: PRENTICE HALL. ISBN 84-89660-00-X.

[Chun y Chuen, 2006]. Chun-Lung Chen, Chuen-Lung Chen. A Heuristic Method for a Flexible Flow Line with Unrelated Parallel Machines Problem. IEEE Conference on Robotics, Automation and Mechatronics. Digital Object Identifier: 10.1109/RAMECH.2006.252673. pp. 1 – 4. 2006.

[Chun, 2008]. Chun-Lung Chen, An Iterated Local Search for Unrelated Parallel Machines Problem with Unequal Ready Times, Proceedings of the IEEE International Conference on Automation and Logistics, pp. 2044-2047, Qingdao, China September, ISBN: 978-1-4244-2502-0. 2008.

[Cruz, 2005] Cruz-Chávez, M.A. (2005). Cooperación de Procesos para el Problema de Jop Shop Scheduling Aplicando Recocido Simulado, Tesis de Doctorado, Marzo 2005.

[Cruz, Juárez, Ávila y Martínez, 2009a]. Cruz-Chávez Marco Antonio, Juárez-Pérez Fredy, Ávila-Melgar Erika Yesenia, Martínez-Oropeza Alina. Simulated Annealing Algorithm for the Weighted Unrelated Parallel Machines Problem. Cerma 2009. ISBN-13:978-0-7695-3799-3. pp. 94. 2009

[Cruz, Martínez, Zavala-Díaz y Martínez-Rangel, 2009b]. Cruz-Chávez Marco Antonio, Martínez-Oropeza Alina, Zavala-Díaz José Crispín, Martínez-Rangel Martín G. Relajación del Problema de Calendarización de Trabajos en un Taller de Manufactura utilizando un Grafo Bipartita. 7mo. Congreso Internacional de Cómputo en Optimización y Software AGECOMP-CICos 2009. ISBN: 978-970.9750-26.3.2009

[Cruz, Martínez y Rivera, 2010a]. Cruz-Chávez Marco Antonio, Martínez Oropeza Alina, Rivera López Rafael. Relaxation of Job Shop Scheduling Problem using a Bipartite Graph. Accepted as paper in Cerma 2010. To publish in IEEE. 2010.

[Cruz, Martínez y Serna, 2010b]. Cruz-Chávez Marco Antonio, Martínez-Oropeza Alina, Serna-Barquera Sergio A. Neighborhood Hybrid Structure for Discrete Optimization Problems. Accepted as paper in Cerma 2010. To publish in IEEE.2010.

[Cruz y Juárez, 2010]. Cruz-Chávez Marco Antonio, Juárez Pérez Fredy. Algoritmo de Recocido Simulado con Paso de Mensajes en Ambiente Grid para el Problema de Máquinas en Paralelo no Relacionadas. A enviar a Journal of Grid Computing. ISSN (e): 1570-7873. Springer. 2010.

[Cruz, Martinez y Serna 2010]. Cruz-Chávez, M. A., Martínez-Oropeza, A., Serna-Barquera, S. A. Neighborhood hybrid structure for discrete optimization problems. En: Electronics, Robotics and Automotive Mechanics Conference, CERMA2010. México: IEEE-Computer Society, 2010, 108-113.

[Dickhoff, 1990, Dickhoff y Finke, 1992]. A typology of cutting and packing problems. European Journal of Operational Research, 44:145–159, 1990.

[Dickhoff y Finke, 1992]. Cutting and packing in production and distribution. Springer Verlag, Berlin, 1992.

[Dorigo, Maniezzo y Colorni, 1991a]. Dorigo M., Maniezzo V., Colorni A. Positive Feedback as a Search Strategy. Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1991a.

[Dorigo, Maniezzo y Colorni, 1991b]. Dorigo M., Maniezzo V., Colorni A. The Ant System: An Autocatalytic Optimizing Process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1991b.

[Dorigo, 1992]. Dorigo M. Optimization, Learning and Natural Algorithms [in Italian]. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1992.

[Dorigo , Maniezzo y Colorni, 1996]. Dorigo M., Maniezzo V., Colorni A. Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man. and Cybernetics–Part B, 26(1):29–41, 1996.

[Dorigo y Stützle, 2003]. Dorigo M., Stützle T. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Adavances. Handbook of Metaheuristics, Springer. 2003.

[Dowsland y Adenso, 2003]. Dowsland Kathryn A., Adenso-Díaz Belarmino. Diseño de Heurística y Fundamentos del Recocido Simulado. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. Asociación Española para la Inteligencia Artificial. Año/Vol. 7, No. 019. ISSN(i): 1137-3601, ISSN(e): 1988-3064. Valencia, España. 2003.

[Feo y Resende, 1989]. Feo T. A., Resende M. G. Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. Operation Research Letters. Vol. 8. Issue 2. pp. 67 – 71. 1989.

[Garey y Johnson, 1976]. Garey M. R., Johnson, D.S., Shethi, R. The Complexity of Flow Shop and Job Shop Scheduling, in Mathematics of Operation Research, vol. 1; No.2. pp. 117 129. 1976.

[Garey y Johnson, 1979]. Garey M. R., D. Johnson. “Computers and Intractability” a Guide to the Theory of NP-Completeness. Freeman. New York NY. ISBN 0-7167-1044-7. 1979.

[Graham, Lawler, Lenstra y Rinnooy, 1979]. Graham R.L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. Ann. Discrete Math., pp. 287–326, 1979.

[Grassé, 1959]. Grassé P. P. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. La théorie de la Stigmergie: essai d interpretation du Comportement des Termites Constructeurs, pages 41-81, 1959.

[Gilmore y Gomory, 1965]. Multistage cutting stock problems of two and more dimensions. Operational Research, 13:94–119, 1965.

[Glover, 1989]. Glover F. Tabu Search. Part I. ORSA Journal of Computing. Vol. 1. No. 3. pp. 190 – 206. 1989.

[Gómez, Mateus y Rocha 2007]. Gómez Ravetti, Geraldo R. Mateus, Pedro L. Rocha. A Scheduling Problem with Unrelated Parallel Machines and Sequence Dependent Setups. 380 Int. J. Operational Research, Vol. 2, No. 4. Copyright © 2007 Inderscience Enterprises Ltd. 2007.

[Huang, 1994]. Gu Jun, Huang Xiaofei. Efficient Local Search with Search Space Smoothing : A Case Study of the Traveling Salesman Problem (TSP). IEEE Transactions on Systems, Man, and Cybernetics. Vol.24. No. 5. 1994.

[Hopper, 2000] Salto, C., “Two-dimensional Packing utilising Evolutionary Algorithms and other Meta-Heuristic Methods”, Tesis de Doctorado, 2000.

[Guo, Lim, Rodriguez y Yang, 2007]. Guo Y., Lim. A., Rodrigues B., Yang L. Minimizing the Makespan for Unrelated Parallel Machines. International Journal on Artificial Intelligence Tools (IJAIT). Vol. 16. Issue 3. pp. 399 – 415 Junio, 2007.

[Halavati, B. Shouraki, Noroozian, y Zadeh, 2007]. Optimizing Allocation of Two Dimensional Irregular Shapes using an Agent Based Approach World Academy of Science, Engineering and Technology Vol:1 2007-11-27

[Halavati, 2007]. Optimizing Allocation of Two Dimensional Irregular Shapes using an Agent Based Approach. World Academy of Science, Engineering and Technology. Vol:1 2007-11-27.

[Hillier, Lieberman, 2008]. Hillier S., Lieberman G.J., Introduction to Operation Research, 8th ed.,Mc Graw Hill , ISBN: 0073017795, 2008.

[Hinxman, 1980] A. Hinxman. The trim loss and assortment problems. European Journal of Operational Research, 5:8–18, 1980.

[Hong, 2007]. Hong Zhou, Zhengdao Li, Xuejing Wu. Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization. Proceeding of the IEEE International Conference on Automation and Logistic. Jinan, China. 2007.

[Khebbache, Prins y Yalaoui, 2008]. Khebbache, S., Prins, C., Yalaoui, A. Iterated local search algorithm for the constrained two-dimensional non-guillotine cutting problem. Journal of Industrial and Systems Engineering. 2008, 2 (3), 164-179.

[Klee y Minty, 1972]. Klee V., Minty G.J.. "How Good is the Simplex Algorithm?" In O. Shisha, editor, Inequalities, III, pages 159–175. Academic Press, New York, NY, 1972.

[Koranne, 2002]. Koranne S. Formulating SoC Test Scheduling as a Network Transportation Problem, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, december 2002.

[Labrada, Enríquez y García, 2014]. Labrada-Nueva Y, Enríquez-Urbano J, García-Ojito Y. Un algoritmo de búsqueda local iterada como solución al problema de la mochila. Universidad Autónoma del Estado de Morelos, Centro de Investigación en Ingeniería y Ciencias Aplicadas. Programación Matemática y Software (2014) 6 (2): 57-64. ISSN: 2007-3283

[López, 2013]. An effective heuristic for the two-dimensional irregular bin packing problem. Springer Science+Business Media New York 2013.

[Lourenço, 2003]. Iterated Local Search, in Glover, F. and Kochenberger, G. (Eds.): Handbook of Metaheuristics, volume 57 of International Series in Research & Management, chapter Iterated Local Search, pp. 321-353. Kluwer Academic Publisers, Boston.

[Lodi, Martello, Vigo, 1999] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal on Computing, 11:345–357, 1999.

[Martínez, 2006]. Martínez Morales A. A. Algoritmo Basado en Tabu Search para el Cálculo del Índice de Transmisión de un Grafo. Departamento de Computación Facultad de Ciencias y Tecnología. FARAUTE de Ciencias y Tecnología, Vol. 1, No.

1, pp. 31-39. Universidad de Carabobo, Valencia, Estado Carabobo, Venezuela. 2006.

[Martínez, Martín, 2003]. Martínez Gil Francisco A., Martín Quetglás Gregorio. Introducción a la Programación Estructura en C. ISBN. 84-370-5666-7. Universidad de Valencia. pp. 209-212. Ed. Maite Simon.2003.

[Martínez, 2010] Martínez, O. A., "Solución al Problema de Máquinas en Paralelo no Relacionadas mediante un Algoritmo de Colonia de Hormigas", Tesis de Maestría, Agosto 2010.

[Mateo, 2009]. Mateo Manuel, Ribas Imma, Ramón Companys. A GRASP Procedure for Scheduling Orders on Parallel Machines with Setup Times. 3rd. International Conference on Industrial Engineering and Industrial Management. XIII Congreso de Ingeniería de Organización. Barcelona-Terrassa, 2009.

[Morabito y Morales, 1998]. A simple and effective recursive procedure for the manufacturer's pallet loading problem. Journal of the Operational Research Society, 49(8):819–828, 1998.

[Michalewicz y Fogel, 2002]. Michalewicz Zbigniew, Fogel David B. How to Solve It: Modern Heuristics. Springer-Verlag Berlin Heidelberg New York. ISBN. 3-540-66061-5. pp. 55 – 90. 2002.

[Michalewicz y Fogel, 2004]. Michaelewicz, Z., Fogel, D. B. How to Solve it: Modern Heuristics. Berlín: Springer-Verlag, 2004.

[Mönch, 2008]. Mönch L. Heuristics to Minimize Total Weighted Tardiness of Jobs on Unrelated Parallel Machines, 4th IEEE Conference on Automation Science and Engineering Key Bridge Marriott, Washington DC, USA, pp. 572-577, August 23-26, ISBN: 978-1-4244 2022-3, 2008.

[Muñoz y Moraga, 2006]. Muñoz Valdez, R. J. Moraga Cuarzo. Heurística Constructiva Visionaria para el Problema de Máquinas Paralelas no Relacionadas con tiempos de Setup Dependientes de la Secuencia (Look-Ahead Constructive Heuristic for the Unrelated Parallel Machine Problem with Sequence Dependent Setup Times). Revista Ingeniería Industrial – Año 5, Num. 1, Segundo Semestre 2006. ISSN 0717-9103. 2006.

[Mahmud, 2006] Mahmud, K., “Solving 2d-strip packing problem using genetic algorithm”, Tesis de Maestría, Agosto 2006.

[Osman y Kelly, 1996]. Osman, I.H., Kelly, J.P. Meta-Heuristics: Theory and Applications, 39 Kluwer Academic Publishers. ISBN: 0792397002. USA, 1996.

[Papadimitriou y Steiglitz, 1998]. Papadimitriou C. H., Steiglitz Kenneth. Combinatorial Optmization. Algorithms and Complexity. ISBN. 0-486-40258-4. USA. 1998.

[PASHA,2003]. Geometric bin packing algorithm for arbitrary shapes, Tesis de Doctorado, 2005.

[Pei y Shih, 2007]. Pei-Chann Chang, Shih-Hsin Chen. Integrating Dominance Properties with Genetic Algorithms for Parallel Machine Scheduling Problems with Setup Times. Applied Soft Computing. ScienceDirect, 2007.

[Pessan et al. 2007]. Pessan C., Bouquard J. L., Néron E., An Unrelated Parallel Machines Model for Production Resetting Optimization, International Conference on Service Systems and Service Management IEEE, Vol 2. ISBN: 1-4244-0450-9. pp. 1178-1182. 2007.

[Pinedo, 2001]. Pinedo M. Scheduling Theory, Algorithms, and Systems. ISBN: 0130281387, 2 ed. Prentice Hall. USA, Aug. 2001.

[Pinedo, 2008]. Pinedo Michael L. Scheduling Theory, Algorithms, and Systems. Third Edition. New York University. ISBN: 978-0-387-78934-7, e-ISBN: 978-0-387-78935-4. Ed. Prentice Hall. Springer. pp. 14. 2008.

[Pintea, Pascan, y Hajdu-Macelaru, 2012]. Comparing several heuristics for a packing problem. 2012.

[Ramalhino, Martin, Stützle, 2001]. Ramalhino, Martin, y Stützle. A beginner´s introduction to iterated local search. Proceedings of MIC 2001, pages 1–6, July 2001.

[Ramalhino, Martin, Stützle y Glover, 2002]. Ramalhino, Martin, Stützle, Glover y Kochenber-ger (eds) Handbook of Metaheuristics, chapter Iterated local search, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[Riojas, 2005]. Riojas Cañari, Alicia Cirila. Conceptos, Algoritmo y Aplicación al Problema de las N-Reinas. Capítulo 2. Heurística y Metaheurística. Monografía para optar el Título de Licenciada de Investigación Operativa. Lima – Perú, 2005.

[Rosen, 2007]. Rosen Kenneth H. Discrete Mathematics and its Applications. Fifth Edition. Ed. Mc. Graw Hill. International Edition 2003. pp. 549. 2003.

[Salto, 2010] Salto, C., “Metaheurísticas Híbridas paralelas para problemas industriales de corte, empaquetado y otros relacionados”, Tesis de Doctorado, Octubre 2009.

[Stattennberger et al., 2007]. Stattenberger Günther, Dankesreiter Markus, Baumgartner Florian, Scheider Johannes J. On the Neighborhood Structure of the

Traveling Salesman Problem Generated by Local Search Moves. J Stat Phys129:623-648. DOI 10.1007/s10955-007-9382-1. Springer Science + Business Media, LLC 2007.

[Téllez, 2007]. Téllez Enríquez Emanuel. Uso de una Colonia de Hormigas para Resolver Problemas de Programación de Horarios. Tesis para obtener el grado de Maestro en Ciencias de la Computación. Laboratorio Nacional de Informática Avanzada (LANIA). Veracruz, 2007.

[Valdés, Parajón, y Tamarit, 2002]. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. Computers and Operations Research, 29(7):925–947, 2002.

[Vallada y Ruiz, 2009]. Vallada Eva, Ruiz Rubén. Metaheurísticas para el problema de Secuenciación en Máquinas Paralelas no Relacionadas con Tiempos de Cambio. 3rd. Internacional Conference on Industrial Engineering and Industrial Management. XIII Congreso de Ingeniería de Organización. Barcelona-Terrassa, 2009.

[Vanderbeck, 2001]. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. Management Science, 47(6):864–879, 2001.

[Wäscher, Haußner, y Schumann, 2007] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. European Journal of Operational Research, 183:1109–1130, 2007.

[Wetzel, 1983]. Wetzel. A. Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization. University of Pittsburgh, Pittsburgh (unpublished). 1983.

[Wu, Huang, Lau, Wong, y Young, 2002] Y. Wu, W. Huang, S. Lau, C.K. Wong, and G.H. Young. An effective quasi-human based heuristic for solving the rectangle packing problem. European Journal of Operational Research, 141:341–358, 2002.

[XIE, WANG y LIU, 2007]. Nesting of two-dimensional irregular parts: an integrated approach. International Journal of Computer Integrated Manufacturing, Vol. 20, No. 8, December 2007, 741 – 756.

[Yamada y Nakano, 1997]. Yamada Takeshi, Nakano Ryohei. Genetic Algorithms for Job Shop Scheduling Problems. Proceedings of Modern Heuristic for Decisión Support, pp. 67 81, UNICOM Seminar, London 1997.

[Zalzala y Fleming, 1997]. Zalzala A. M. S., Flemming P. J: Genetic Algorithms in Engineering Systems. Published by: Institute of Electrical Engineers. London, United Kingdom. ISBN. 0 85296 902 3.1997.

[Zhou, 2007]. Zhou H., Li Z., Wu X. Scheduling Unrelated Parallel Machine to Minimize Total Weighted Tardiness Using Ant Colony Optimization. Proceedings of the IEEE International Conference on Automation and Logistics. pp. 132-136, August 18 - 21, Jinan, China, 2007.

Apéndice A

Estructuras de Datos Utilizadas en el Algoritmo de Búsqueda Local Iterada.

Para darle solución al Problema de Empaquetamiento en Contenedores aplicado a Imprentas por medio de un algoritmo de Búsqueda Local Iterada, 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 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;
    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 área de la zona de trabajo o hoja de papel, la suma de las área de todas las figuras y el área residual que es el resultado de la resta del área de la zona de trabajo u hoja de papel menos la suma

del área de todas las figuras que se introducen en la hoja de papel, también guarda una imagen, así como su máscara.

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 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	m^2 Ya que por convenio w = h
xover*yover	m^2 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 = m^2	

`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 + m^2	

`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 = m^2	

`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 +*** $5m^2$	

`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 m^2	

`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, al_map_rgba_f(.75, 0, .75, .75));`	2
`}`	
`}`	
`}`	
T(n) = 6 + 5w*h = 6 + m^2	

`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; j++)`	h
`{`	
`if(m->bits[i * m->w + j] == 1)`	3
`pixel = pixel + 1;`	2
`}`	
`}`	
`return pixel;`	1
`}`	
T(n) = 2 + 5w*h = 2 + 5 m^2	

`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 feasibleSolution()`	
`{`	
`int residualAreas = 0;`	1
`int dx = 0;`	1
`int dy = 0;`	1
`Shapes test;`	1
`char *img = "150.png";`	1
`test.image = al_load_bitmap(img);`	1
`test.w = al_get_bitmap_width(test.image);`	1
`test.h = al_get_bitmap_height(test.image);`	1
`test.x = getNewPositionX(test.w);`	4
`test.y = getNewPositionY(test.h);`	4
`test.mask = Mask_New(test.image);`	***16 + 5*** m^2
`for (int i=0;i<NUMSHAPES&&SHAPES<NUMSHAPES;i++,SHAPES++)`	n
`{`	
`bool overlap = false;`	1
`do`	
`{`	
`shapes[SHAPES] = clone(test);`	
`for(int j = 0; j < SHAPES; j ++)`	n
`{`	
`dx = shapes[SHAPES].x – shapes[j].x;`	2
`dy = shapes[SHAPES].y - shapes[j].y;`	2
`if(Mask_Collide(shapes[SHAPES].mask,`	
`shapes[j].mask, dx, dy))`	***33 + 12*** m^2
`{`	
`overlap = true;`	1
`countOverlap++`	2

break;	
}	
overlap = false;	1
}	
} while(overlap);	n
if(countOverlap >= 5000)	1
{	
residualAreas = gettotalArea() - SHAPES *	3
Mask_Area_Pixels(test.mask);	
break;	
}	
return residualAreas;	1
}	
$T(n) = 37 + 42n^2 + n^3 + 12n^2*m^2$	

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	

int Solution_OptimaS()	
{	
Shapes figure;	1
int dx = 0;	1
int dy = 0;	1
int signs = 0;	1
int flag = 0;	1
int positionRandom = 0;	1
int coordinateVariationX = 0;	1
int coordinateVariationY = 0;	1
int valueUpdateX = 0;	1
int valueUpdateX = 0;	1
figure.w = al_get_bitmap_width(figure.image);	1
figure.h = al_get_bitmap_height(figure.image);	1
figure.mask = Mask_New(figure.image);	***16 + 5*** m^2
for(int i=0;i<NUMSHAPES&&SHAPES<NUMSHAPES;i++,SHAPES++)	***n***
{	
bool overlap = false;	1
do	
{	
if(SHAPES < nshapes)	1
{	
signs = Sign();	8
positionRandom = rand()%nshapes;	2
valueUpdateX = ((WIDTH - figure.w)/50);	3
valueUpdateY = ((HEIGHT - figure.h)/50);	3
coordinateVariationX =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateX);	1
coordinateVariationY =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateY);	1

shapesSolutions[positionRandom].x =	
shapesSolutions[positionRandom].x +	
signs * coordinateVariationX;	3
shapesSolutions[positionRandom].y =	
shapesSolutions[positionRandom].y +	
signs * coordinateVariationY;	3
}	
else	
{	
else if (SHAPES >= nshapes)	1
{	
shapesSolutions[SHAPES].image = figure.image;	1
shapesSolutions[SHAPES].w = figure.w;	1
shapesSolutions[SHAPES].h = figure.h;	1
shapesSolutions[SHAPES].x =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateX);	1
shapesSolutions[SHAPES].y =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateY);	1
shapesSolutions[SHAPES].mask = figure.mask;	***16 + 5*** m^2
}	
for(int j = 0; j < SHAPES; j ++)	***n***
{	
dx = shapesSolutions[SHAPES].x - shapesSolutions[j].x;	***2***
dy = shapesSolutions[SHAPES].y - shapesSolutions[j].y;	***2***
if(Mask_Collide(shapesSolutions[SHAPES].mask,	
shapesSolutions[j].mask, dx, dy))	***33 + 12*** m^2
{	
overlap = true;	1
countOverlap++;	2
break;	
}	
overlap = false;	1
}	
} while(overlap);	n
if(countOverlap >= 5000)	1
{	
residualAreasOptima = gettotalArea() - SHAPES *	
Mask_Area_Pixels(figure.mask);	3
break;	
}	
return residualAreasOptima;	1

}	
T(n) = $28 + 5\ m^2 + 73\ n^2 + 12\ n^2 * m^2$	

void saveSolutionOptima()	
{	
FILE *file;	1
char * name = "SOp.txt";	1
file = fopen(name,"w+");	1
fprintf(file,"%d\n", nshapes);	1
for(int i = 0; i < nshapes; 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	

void readInstancesS()	
{	
Shapes figure;	1
figure.image = al_load_bitmap(img);	1
figure.w = al_get_bitmap_width(figure.image);	1
figure.h = al_get_bitmap_height(figure.image);	1
figure.mask = Mask_New(figure.image);	***16 + 5*** m^2
if(!dataFile)	1
{	
printf ("Error...");	1
exit(0);	1
}	
else	
{	
fscanf(dataFile," %d\t ",&nshapes);	1
for(int i = 0; i < nshapes; i++)	n
{	
fscanf(dataFile," %d\t ",&Coordinates.x[i]);	1
fscanf(dataFile," %d\t ",&Coordinates.y[i]);	1
}	

fscanf(dataFile," %d\t ",&residualAreaInstance);	1
fclose(dataFile);	1
}	
for(int i = 0; i < nshapes; i++)	n
{	
shapesSolutions[i].image = figure.image;	1
shapesSolutions[i].w = figure.w;	1
shapesSolutions[i].h = figure.h;	1
shapesSolutions[i].x = Coordinates.x[i];	1
shapesSolutions[i].y = Coordinates.y[i];	1
shapesSolutions[i].mask = figure.mask;	1
}	
}	
T(n) = 26 + 5m^2 + 8n	

int disturbedSolution()	
{	
Shapes figure;	1
int dx = 0;	1
int dy = 0;	1
int signs = 0;	1
int flag = 0;	1
int positionRandom = 0;	1
int coordinateVariationX = 0;	1
int coordinateVariationY = 0;	1
int valueUpdateX = 0;	1
int valueUpdateX = 0;	1
figure.w = al_get_bitmap_width(figure.image);	1
figure.h = al_get_bitmap_height(figure.image);	1
figure.mask = Mask_New(figure.image);	***16 + 5*** m^2
for(int i=0;i<NUMSHAPES&&SHAPES<NUMSHAPES;i++,SHAPES++)	***n***
{	
bool overlap = false;	1
do	
{	
if(SHAPES < nshapes)	1
{	
signs = Sign();	8
positionRandom = rand()%nshapes;	2
valueUpdateX = ((WIDTH - figure.w)/50);	3

valueUpdateY = ((HEIGHT - figure.h)/50);	3
coordinateVariationX =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateX);	1
coordinateVariationY =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateY);	1
shapesSolutions[positionRandom].x =	
shapesSolutions[positionRandom].x +	
signs * coordinateVariationX;	3
shapesSolutions[positionRandom].y =	
shapesSolutions[positionRandom].y +	
signs * coordinateVariationY;	3
}	
else	
{	
else if (SHAPES >= nshapes)	1
{	
shapesSolutions[SHAPES].image = figure.image;	1
shapesSolutions[SHAPES].w = figure.w;	1
shapesSolutions[SHAPES].h = figure.h;	1
shapesSolutions[SHAPES].x =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateX);	1
shapesSolutions[SHAPES].y =	
Rand_LimitLower_To_LimitUpper(0,valueUpdateY);	1
shapesSolutions[SHAPES].mask = figure.mask;	***16 + 5*** m^2
}	
for(int j = 0; j < SHAPES; j ++)	***n***
{	
dx = shapesSolutions[SHAPES].x -	***2***
shapesSolutions[j].x;	
dy = shapesSolutions[SHAPES].y -	***2***
shapesSolutions[j].y;	
if(Mask_Collide(shapesSolutions[SHAPES].mask,	
shapesSolutions[j].mask, dx, dy))	***33 + 12*** m^2
{	
overlap = true;	1
countOverlap++;	2
break;	
}	
overlap = false;	1
}	
} while(overlap);	n
if(countOverlap >= 5000)	1

{ residualAreasOptima = gettotalArea() - SHAPES *	
Mask_Area_Pixels(figure.mask);	3
savefileSolution();	***T(n)=6 + 3n***
break;	
}	
return residualAreasOptima;	1
}	
T(n) = 31 + ***5*** m^2 + 75 n^2 + 12 n^2*m^2	

void iteratedLocalSearch()	
{	
printf("Starting of the program...\n");	1
printf("\n");	1
time_t start_t, end_t;	1
double diff_t;	2
time(&start_t);	1
int s0 = 0;	1
int sp = gettotalArea();	3
int SO = 0;	1
printf("Residuala/#Shapes/#Collisions");	1
printf("\n");	1
s0 = feasibleSolution();	***T(n)=37 + 42n^2 + n^3 + 12n^2*m^2***
do	
{	
for(int l = 1; l <= ITERATIONS; l++)	***n***
{	
if(s0 < sp)	1
{	
sp = s0;	1
printf("%d\t", sp);	1
printf("%d\t", SHAPES);	1
printf("%d\t", countOverlap);	1
printf("\n");	1
}	
cleanShapes();	9n
SHAPES = 0;	1
countOverlap = 0;	1
s0 = disturbedSolution();	***T(n)*** = 31 + ***5*** m^2 + 75 n^2 + 12 n^2*m^2
sp = s0;	1
readInstancesS();	***T(n) = 26 + 5***m^2 + 8n
int SO = Solution_OptimaS();	28 + ***5*** m^2 + 73 n^2 + 12 n^2*m^2
printf("%d\t", SO);	1
printf("%d\t", SHAPES);	1
printf("%d\t", countOverlap);	1
}	

`} while(s0 < sp);`	n
`time(&end_t);`	1
`diff_t = difftime(end_t, start_t);`	1
`printf("\n");`	1
`printf("Execution time in seconds = %f\n", diff_t);`	1
`printf("\n");`	1
`printf("Exiting of the program...\n");`	1
`}`	
$T(n) = 56 + 69n^2m^2 + 128n^2 + 17n^3 + 148n^4 + 24n^4m^2$	
$T(n) = 56 + 17n^3 + n^2(128 + 69m^2) + n^4(148 + 24m^2)$	

Apéndice C

Código fuente del Algoritmo de Búsqueda Local Iterada.

```
void iteratedLocalSearch()
{
    printf("Starting of the program...\n");
    printf("\n");
    time_t start_t, end_t;
    double diff_t;
    time(&start_t);
    int s0 = 0;
    int sp = gettotalArea();
    int SO = 0;
    printf("Residuala/#Shapes/#Collisions");
    printf("\n");
    s0 = feasibleSolution();
    do
    {
         for(int l = 1; l <= ITERATIONS; l++)
         {
             if(s0 < sp)
               {
                   sp = s0;
                   printf("%d\t", sp);
                   printf("%d\t", SHAPES);
                   printf("%d\t", countOverlap);
                   printf("\n");
               }
               cleanShapes();
               SHAPES = 0;
               countOverlap = 0;
               s0 = disturbedSolution();
               sp = s0;
               readInstancesS();
               int SO = Solution_OptimaS();
               printf("%d\t", SO);
               printf("%d\t", SHAPES);
               printf("%d\t", countOverlap);
         }
    } while(s0 < sp);
    time(&end_t);
    diff_t = difftime(end_t, start_t);
    printf("\n");
    printf("Execution time in seconds = %f\n", diff_t);
    printf("\n");
    printf("Exiting of the program...\n");
}
```

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.

Análisis de Sensibilidad: Evaluación del comportamiento de las variables críticas de un problema con la finalidad de establecer un rango numérico, dentro del cuál, 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.

Costo: 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 cuál 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.