EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"POR MI PATRIA Y POR MI BIEN"

TESIS

ALGORITMO MEMÉTICO PARA EL PROBLEMA DE VENTAS POR INTERNET CON COSTOS DE ENVÍO

Que para obtener el Grado de
Maestro en Ciencias de la Computación

Presenta
Ing. Miguel Ángel García Morales
G03500548

Director de Tesis
Dr. Héctor Joaquín Fraire Huacuja

Co-Director de Tesis
Dr. Mario César López Locés

Cd. Madero, Tamaulipas Mayo 2021

Cd. Madero, Tam. 18 de mayo de 2021

OFICIO No. : U.013/21
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

C. MIGUEL ÁNGEL GARCÍA MORALES
No. DE CONTROL G03500548
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su Examen de Grado de Maestría en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ALGORITMO MEMÉTICO PARA EL PROBLEMA DE VENTAS POR INTERNET CON COSTOS DE ENVÍO"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE:	DR.	NELSON RANGEL VALDEZ
SECRETARIO:	DRA.	MARÍA LUCILA MORALES RODRÍGUEZ
VOCAL:	DR.	HÉCTOR JOAQUÍN FRAIRE HUACUJA
SUPLENTE:	DRA.	LAURA CRUZ REYES
DIRECTOR DE TESIS:	DR.	HÉCTOR JOAQUÍN FRAIRE HUACUJA
CO-DIRECTOR DE TESIS:	DR.	MARIO CÉSAR LÓPEZ LOCÉS

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
Excelencia en Educación Tecnológica.
"Por mi patria y por mi bien".

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MARCO ANTONIO CORONEL GARCÍA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p.- Archivo
MACG 'mdcoa'

Declaración de originalidad

Mayo del 2021, Cd. Madero, Tamps., México.

Yo, Miguel Ángel García Morales, en mi calidad de autor manifiesto que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares. Por lo tanto la obra es de mi autoría y soy titular de los derechos que surgen de la misma.

También declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones. Esta tesis fue evaluada por la herramienta Turniting obteniendo como resultado el 3% de Similutud con otros trabajos.

Además, en caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, acepto toda la responsabilidad de tal infracción y relevo de ésta a mi director y codirectores de tesis, así como al Tecnológico Nacional de México, al Instituto Tecnológico de Ciudad Madero y a sus respectivas autoridades.

Miguel Ángel García Morales

Resumen

Este proyecto de investigación aborda el problema de Optimización de compras por Internet con costos de envío (IShOP). En el estado del arte, solo se reporta una solución metaheurística. Esta solución es un algoritmo de procesamiento celular (Pcell) que simula el procesamiento paralelo de dos o más procesos de búsqueda a través del espacio de soluciones y actualmente se considera la mejor solución de IShOP en el estado del arte. En este trabajo, se propone un nuevo algoritmo metaheurístico basado en la metodología del algoritmo memético. Se propone tambien una nueva representación vectorial de las soluciones candidatas que permite reducir la complejidad del temporal del cálculo de la función objetivo de $O(n^2)$ a $O(n)$. Para validar el enfoque propuesto se realizaron una serie de experimentos computacionales con instancias del estado del arte incluyendo un estudio comparativo del desempeño del algoritmo propuesto MAIShOP contra el del algoritmo Pcell. En los experimentos computacionales, se utiliza un amplio conjunto de instancias aleatorias y los resultados muestran una clara superioridad del algoritmo propuesto MAIShOP. Se aplicó la prueba no paramétrica de Wilcoxon, en la que se verificó la significancia de las diferencias observadas.

Agradecimientos

Agradezco a mi asesor en esta investigación, Dr. Hector Joaquín Fraire Huacuja, gracias por su apoyo, por todos sus consejos y por la paciencia que tuvo conmigo para aconsejarme durante todo este tiempo. Mi agradecimiento más sincero para usted por su hospitalidad y por todo el tiempo que compartimos juntos, lo que me ha llevado a apreciarlo con un cariño y respeto que siempre guardo en mi corazón.

Agradezco profundamente a mi asesor en la Universidad Autónoma de Nuevo León, Dr. Mario César López Locés, porque este trabajo no hubiera sido posible sin su dirección. Gracias por confiar en mi persona e invertir su conocimiento, su esfuerzo y su tiempo en mi formación. Su dirección, su experiencia, sus agudas observaciones y su apoyo fueron fundamental durante todo este proyecto.

Agradezco infinitamente a los integrantes de mi comité tutorial: Dra. María Lucila Morales Rodríguez, Dra. Laura Cruz Reyes, Dra. Claudia Guadalupe Gómez Santillán, y el Dr. Nelson Rangel Valdez por sus sabios consejos durante mis estudios de maestría. Mi más sincero reconocimiento, no sólo por ser unos excelentes docentes e investigadores, sino también por ser un ejemplo viviente de lucha constante, trabajo y superación. Nunca olvidaré las lecciones aprendidas de ustedes.

Reconozco agradecidamente el apoyo de las instituciones públicas que hicieron posible este trabajo: el Consejo Nacional de Ciencia y Tecnología (CONACYT), el Tecnológico Nacional de México y, sobre todo, al Instituto Tecnológico de Ciudad Madero.

Gracias a toda mi familia por apoyarme en esta etapa de mi vida, por la paciencia y comprensión que han tenido para mi desde siempre. Nunca podré expresar con palabras lo agradecido que estoy con ellos, son mi soporte de vida.

Tabla de Contenido

Capítulo 1. Introducción

El comercio electrónico se ha convertido en una parte esencial de la sociedad moderna, la implementación de tecnología novedosa y en constante crecimiento ha hecho que sea inevitable adaptarse a esta evolución [Musial, et al., 2014].

La principal ventaja del comercio electrónico es que las ofertas están disponibles para una audiencia más amplia, sin la mayoría de los costos asociados, como alquiler, impuestos, mantenimiento y publicidad. Los clientes pueden realizar compras desde cualquier lugar, en cualquier momento, con mejores precios y teniendo una gama de productos más amplia, siempre que tengan acceso a Internet [López Locés, 2016].

En el Problema de optimización de compras en Internet (IShOP) como se menciona en [Musial, 2012], se asume que un cliente con una lista de compras necesita comprar los productos en un conjunto de tiendas en línea al menor costo posible. Se propone la definición del problema [Mitsuo Gen and Runwei Cheng, 2000].

El problema es modelado formalmente como un problema de optimización [Blazewicz, et al., 2010], en donde se presenta una prueba de que pertenece al conjunto NP-Duro y se proponen dos algoritmos polinomiales deterministas para resolver casos especiales del problema de IShOP: greedy [Wojciechowski and Musial, 2010] y forecasting, [Blazewicz, et al., 2014b].

1.1. Antecedentes del proyecto

El proyecto forma parte de una investigación iniciada en el año 2012 cuyo propósito es el desarrollo de nuevos métodos de optimización para el problema de IShOP.

Esa investigación se encuentra relacionada con los siguientes trabajos:

- Trajectory metaheuristics for the internet shopping optimization problem. In *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization* (pp. 527-536). López Locés, M. C., Rege, K., Pecero, J. E., Bouvry, P., & Huacuja, H. J. F. (2015). Springer, Cham.
- Exact and Heuristic Approaches to Solve the Internet Shopping Optimization Problem with Delivery Costs. Mario C. López Locés, Jedrzej Musial, Johnatan E. Pecero, Hector J. Fraire-Huacuja, Jacek Blazewicz, Pascal Bouvry. Int. J. Appl. Math. Comput. Sci. (2016).
- Métodos de Optimización de Problemas NP-Duros Basados en Algoritmos de Procesamiento Celular. Mario C. López Locés, Director: Dr. Hector J. Fraire-Huacuja, Co-Director: Dr. Pascal Bouvry. ITTijuana, Trabajo de Tesis (2016).

El proyecto de tesis se propone el desarrollo de un algoritmo memético cuyo desempeño supere al mejor algoritmo del estado del arte, mismo que es un algoritmo celular [López Locés, 2016].

1.2. Problema de investigación

El problema de optimización de compras en Internet se define de la siguiente manera:

Un cliente desea comprar por internet un conjunto de *n* productos N, los cuales se pueden adquirir en un conjunto de *m* tiendas disponibles M. Ahora, el conjunto N_i contiene los productos disponibles en la tienda i, cada producto $j \in N_i$ tiene un costo de c_{ij} y un costo de envío d_i. El costo de envío se cobra solo si uno o más productos se compran en la tienda i. *El problema de compras por Internet con costos de envío (IShOP)* consiste en minimizar el costo total de la compra de todos los productos en *N* considerando el costo de los productos más los gastos de envío.

En la definición del problema se usa la notación de la Tabla 1.2:

Tabla 1.2: Tabla de notación.

Símbolo	Descripción
M	Conjunto de tiendas
N	Conjunto de productos
m	Número de tiendas, $\|M\|$
n	Número de productos, $\|N\|$
i	Indicador de tienda
j	Indicador de producto
N_i	Contenedor de productos disponibles de una tienda i
d_i	Costo de envío de todos los productos de la tienda i
c_{ij}	Costo del producto j en la tienda i
x_{ij}	Variable binaria que indica si el producto j en la tienda i o no
y_i	Variable binaria que indica si se agrega el costo de envío de la tienda i
T	Valor acumulado de todos los productos comprados en todas las tiendas
T_i	Valor acumulado de todos los productos comprados en la tienda i
$X = (X_i, \dots, X_m)$	Secuencia de selecciones de productos de tiendas $1, \dots, m$
$F(X)$	Suma de producto y costos de envío
$\sigma(x)$	Indicador de función para $x = 0$ y $x > 0$
X^*	Secuencia óptima de selecciones de productos
F^*	Costo total óptimo (mínimo)

Formalmente el problema consiste en determinar una partición de los productos a comprar en las diferentes tiendas $X = (X_1, \dots, X_m)$, de modo que $X_i \subseteq N_i$ y que se compren todos los productos $\cup_{i=1}^{m} X_i = N$ y que minimice el costo total. La función objetivo del problema se muestra en la Ec. 1.1:

$$F(X) = \sum_{i=1}^{m} \left(\sigma(|X_i|) d_i + \sum_{j \in X_i} c_{ij} \right) \qquad (1.1)$$

Donde:

$|X_i|$ es la cardinalidad del contenedor X_i, y $\sigma(x) = 0 \; si \; x = 0 \; y \; \sigma(x) = 1 \; si \; x > 0$.

1.3. Objetivos de la tesis

1.3.1. Objetivo general

Desarrollar un algoritmo memético para el problema de compras por internet considerando costos de envío, con un rendimiento estadísticamente similar o superior a los del estado del arte.

1.3.2. Objetivos específicos

- Definir tipo y tamaño de instancias a utilizar.
- Implementar el mejor algoritmo del estado del arte de IShOP.
- Desarrollar nuevas heurísticas y búsquedas locales.
- Desarrollar el algoritmo memético.
- Realizar una evaluación experimental del desempeño del algoritmo memético con respecto al mejor del estado del arte.

1.4. Alcances y delimitaciones

1.4.1. Alcances

El proyecto cubre el desarrollo de nuevas heurísticas y búsquedas locales, que serán usadas en el desarrollo de un algoritmo memético, que proporcione soluciones óptimas para instancias pequeñas del problema de IShOP.

Mientras tanto, para instancias de tamaño mediano y grande del IShOP, se desarrollaran algoritmos aproximados que podrán resolverse en un tiempo razonable.

1.4.2. Delimitaciones

El proyecto de investigación no considera la implementación de algoritmos paralelos para resolver el IShOP.

El software de optimización que se utiliza para implementar el ILP es el CPLEX Optimization Studio de IBM.

1.5. Justificación y beneficios

En la actualidad se genera una gran cantidad de información en internet, cuando más usuarios pueden generar y cargar nueva información, la búsqueda de información relevante puede llegar a ser abrumadora. Un caso particular del escenario anterior son las compras por internet. Debido a que los costos de las tiendas virtuales son sensiblemente más bajos en comparación con las tiendas físicas y aunado a esto la gran cantidad de proveedores que

ofrecen el mismo producto considerando las mismas características en la mayoría de los casos.

Esto provoca que un usuario particular en busca de la mejor oferta, mirando a través de toda la gama de tiendas y teniendo en cuenta toda la gama de productos y costos de envío asociados, pueda tener problemas en decidir cuál producto comprar y en que tienda.

Debido a la dureza del problema se hace necesario desarrollar métodos más eficientes para resolver el IShOP.

Por este motivo, el proyecto de investigación propone el desarrollo de nuevas heurísticas y búsquedas locales que permitan construir un algoritmo memético para obtener soluciones exactas en el caso de instancias pequeñas y soluciones aproximadas para los casos de instancias medianas y grandes para el problema de optimización.

Capítulo 2. Marco teórico

En el capítulo se describen los conceptos básicos relacionados con la optimización y su estrecha relación con las ciencias de la computación. Es imposible hablar de optimización sin considerar la teoría de la complejidad computacional. La complejidad computacional estudia la dificultad de los diferentes problemas [Garey, M. R. and Johnson, 1990]. Esta teoría nos indica que hay problemas que son intratables por una computadora de forma determinista en un tiempo computacional razonable (polinomial). El primer problema en probarse que no existe un algoritmo que lo resuelva en tiempo polinomial es el problema de satisfactibilidad sin restricciones (SAT) [Cook, S. A, 1971]. Dicho problema tiene la particularidad de ser el único al cual todos los problemas resolubles en tiempo polinomial por un algoritmo no determinista (aleatorio) se pueden reducir (transformar) a él, y no se conoce un algoritmo determinista en tiempo en tiempo polinomial que lo resuelva.

2.1. Notación *O* grande

Las definiciones de tiempo de ejecución polinomial y exponencial tienen su origen en la notación *O* grande, de las ciencias de la computación [Danziger, P. (2010)]. La notación *O* grande acota de manera asintótica el crecimiento del tiempo de ejecución. En otras palabras, el tiempo de ejecución crece en relación con el tamaño *n* de la entrada de datos. Algunos ejemplos de algoritmos con su orden se muestran en la siguiente tabla:

Tabla 2.1: Órdenes de complejidad comunes de menor a mayor

Notación	Orden	Algoritmo de ejemplo
O(1)	Constante	Determinar si un número es par
O(log n)	Logarítmico	Búsqueda binaria
O(< n)	Sublineal	Búsqueda paralela
O(n log n)	*n log* (*n*)	Quicksort
O(n2)	Cuadrático	Ordenamiento Burbuja
O(nc)	Polinomial	Enumerar permutaciones con repetición
O(cn)	Exponencial	Triángulo de Pascal
O(n!)	Factorial	Enumerar permutaciones sin repetición

- Dentro de la teoría de la complejidad computacional generalmente los órdenes se engloban en dos conjuntos, complejidad polinomial y complejidad exponencial. Donde complejidad polinomial se refiere al conjunto de tiempos de ejecución con crecimiento igual o menor al orden polinomial. Mientras que complejidad exponencial se refiere al conjunto de tiempos de ejecución con crecimiento igual o superior al orden exponencial. Ambos términos son utilizados en el presente capítulo.

2.2. Complejidad computacional

Aunque la teoría de la complejidad computacional tiene un gran impacto sobre las aplicaciones en el mundo real, ya que abre las puertas a buscar nuevas soluciones a problemas intratables, de los cuales no se conoce un algoritmo determinista de tiempo polinomial que lo resuelva. Si lo anterior no es posible, se justifica el uso de métodos no exhaustivos, los cuales evalúan un subconjunto de las posibles soluciones. Es por eso que es importante la clasificación de los problemas acorde a la teoría de la complejidad, clasificación que se describe a continuación.

2.2.1. Problema de decisión

La teoría de la complejidad computacional es originalmente desarrollada para problemas de decisión, los cuales tienen la característica de que su respuesta es sí o no [Sipser, M.

(2006)]. Por ejemplo la pregunta ¿Existe un camino con distancia menor a 560km entre Madrid y Barcelona? Plantea un problema de decisión donde la respuesta es sí o no.

2.2.2. *Clase P*

La *clase P* es un conjunto de problemas de decisión que pueden ser resueltos en tiempo polinomial con un algoritmo determinista [Sipser, M. (2006)]. Estos problemas se considera que son fáciles de resolver y se les denomina tratables. Para probar que un problema de decisión es tratable, basta construir un algoritmo determinista que encuentre su solución óptima. Probar que un problema es intratable es una tarea mucho más compleja, se tiene que probar que no existe un algoritmo determinista de tiempo polinomial que lo resuelva.

2.2.3. *Clase NP*

La *clase NP* es un conjunto de problemas de decisión que se pueden verificar en tiempo polinomial con un algoritmo no determinista. Por verificar se entiende que se pueda generar en tiempo polinomial una solución candidata con un algoritmo no determinista y que se pueda comprobar si es solución o no en tiempo polinomial con un algoritmo determinista [Sipser, M. (2006)].

2.2.4. Problema *NP-Completo*

La clase *NP-Completo* es el conjunto de todos los problemas denominados como intratables, un problema que pertenece a la clase *NP-Completo* es un problema para el cual se ha probado que no existe algoritmo determinista polinomial que lo resuelva. Un problema de decisión *B* pertenece a la clase *NP-Completo* si y solo si satisface las siguientes condiciones [Sipser, M. (2006)]:

- *B* pertenece a la clase *NP*
- Todo problema $A \in NP$ es reducible en tiempo polinomial a *B.*

Para la reducción es necesario definir una transformación, donde las instancias del problema *B* cuya respuesta es si se transforman a una instancia del problema $A \in NP - Completo$, donde la respuesta es también sí, y viceversa. Los problemas de esta clase se

consideran los más difíciles de la clase *NP*, ya que para probar que *P=NP* sería suficiente probar que es tratable uno solo de los problemas de *NP-Completo*.

2.2.5. Problema de optimización

En todos los problemas de optimización se trata de encontrar la mejor solución posible de un conjunto de posibles soluciones. Por ejemplo, nosotros podemos buscar localizar el clique más grande de un grafo, el recubrimiento de vértices más pequeño, o el camino más corto entre dos nodos [Sipser, M. (2006)]. En otras palabras, dada una función objetivo $f(\vec{x})$, el objetivo es encontrar la solución $\vec{x} *$ con el mejor valor objetivo, en el caso de problemas de minimizar la que produce el menor valor, en el caso de problemas de maximizar la que produce e mayor valor. Todo problema de optimización tiene asociado una versión de decisión.

2.2.6. Problema *NP-Duro*

Como se mencionó anteriormente la teoría de la complejidad computacional está desarrollada para problemas de decisión, los cuales son utilizados para caracterizar la dificultad de los problemas de optimización. Un problema *NP-Duro* es un problema de optimización cuya versión de decisión es *NP-Completo* [Garey, M. R., et al., (1979)]. De tal manera, desarrollando la versión de decisión de cualquier problema de optimización es posible probar si pertenece o no a esta clase, los más difíciles de resolver.

2.2.7. Decibilidad sobre espacios continuos

La teoría de la complejidad computacional fue desarrollada sobre problemas combinatorios discretos. No obstante, existen muchos problemas de optimización cuyo espacio de soluciones es infinito continuo. Diversos esfuerzos interesantes se han realizado para llenar el hueco en la teoría [Blum et al., (1989), Blum et al., (1990), Blum et al., (1998), Blum, (2004)]. En estos esfuerzos dirigidos por la profesora Lenore Blum de la Universidad de Carnegie Mellon, el concepto de máquina de Turing es reemplazado por el concepto de máquina sobre un anillo o campo, utilizando operaciones algebraicas y comparaciones de elementos en el anillo.

2.2.8. Decibilidad en funciones multivaluadas

Otra particularidad de la teoría de la complejidad computacional es que está desarrollada utilizando funciones univaluadas, en las cuales para un problema de decisión existe una sola salida SÍ o NO. El problema radica cuando nos encontramos con funciones multivaluadas, como naturalmente se encuentran en los problemas multiobjetivo.

Para los cuales un objetivo podrá cumplir la condición deseada mientras que otro no. Por ejemplo, en el caso de un problema con 2 objetivos pueden ocurrir las siguientes combinaciones: (sí, sí), (no, no), (si, no), (no, sí). Dadas estas combinaciones no es claro cuando un problema se satisface (es una solución al problema) o no. Por ejemplo ¿Es solución si solo cumple en un objetivo mientras que en otros no? ¿Es solución si y solo si cumple todos los objetivos? peor aún el caso cuando se sabe que los objetivos están encontrados significa que encontrar el óptimo en un objetivo supone encontrar la peor solución en otro objetivo. Imposibilitando encontrar SÍ salidas en todos los objetivos en una sola solución. Pocos esfuerzos se han hecho en tratar de llenar el hueco en la teoría de la complejidad, algunos ejemplos descritos [Glaßer et al., 2010, Fleszar et al., 2011, Bökler, 2017].

2.3. Métodos de optimización

Los diferentes métodos de optimización se pueden clasificar en dos tipos, métodos exactos y métodos heurísticos. Los métodos exactos garantizan encontrar de manera sistemática la solución óptima a un problema de optimización. Mientras que los métodos heurísticos encuentran una solución aproximada al problema, tratando de garantizar una calidad considerable sin garantizar encontrar la solución óptima al problema. La teoría de la complejidad computacional es una guía sobre que método ocupar para resolver determinado problema. Si el problema pertenece a la clase *P* debe ser resuelto con un método exacto lo que garantiza encontrar la solución óptima. Si el problema es *NP-Duro* se justifica el uso de métodos heurísticos.

2.3.1. Métodos exactos

Los métodos de optimización exactos a veces pueden resolver pequeños casos de un problema complejo de manera efectiva. Además, se puede ejecutar ejecutar un método exacta

durante mucho tiempo con el objetivo de obtener soluciones óptimas [Sörensen y Glover, 2013]. Actualmente existen diversos métodos disponibles tales como los métodos enumerativos, de ramificación y poda, back tracking, etc. En general estos métodos resuelven las instancias con una entrada *n* pequeña de un problema de optimización en un tiempo razonable. Sin embargo, cuando se aplican a instancias difíciles o grandes, el tiempo de solución crece exponencialmente, haciéndolos prácticamente inaplicables en estos casos.

Se presentan algunos ejemplos particulares a continuación:

- *Branch and Bound*: El método branch and bound es un enfoque casi enumerativo para resolver una variedad de problemas combinatorios. Esto es bastante eficiente para problemas de tamaño modesto, y la metodología general forma una parte importante del conjunto de métodos de solución para la clase general de problemas de programación lineal entera (ILP). La idea básica es dividir un problema dado en un número de subproblemas (branching). Establecer subproblemas que son más fáciles de resolver que el problema original, por su tamaño menor o estructura susceptible [Laguna y Delgado, 2007].
- *Enumeración Implícita*: Esta técnica es usualmente aplicada a problemas de programación entera cero-uno. Es similar al branch and bound; sin embargo, las reglas de ramificación, restricción y acotamiento son simplificadas y refinadas porque cada variable entera puede tomar solo valores cero o uno. Los problemas de programación entera cero-uno tienen un número finito de puntos factibles, 2^n puntos enteros factibles, donde n es el número de variables cero-uno [Cavalier M. y James P., 1994].
- *Planos de corte*: El objetivo es construir iterativamente el caso convexo en la vecindad de la solución óptima entera. La solución óptima de un problema de programación entera puede ser determinado resolviendo un problema de programación lineal única en la que el casco convexo es usado como la región factible. Esto es porque los puntos de los extremos del casco convexo corresponden a las soluciones enteras. Se hace de una manera sistemática introduciendo restricciones adicionales que cortan porciones de la región factible excluyendo algunos puntos enteros factibles [Gomory, 1958].

2.3.2. Métodos heurísticos

Los heurísticos utilizan reglas, estrategias o aproximaciones basadas generalmente en conocimientos empíricos para guiar la búsqueda. La palabra heurisken procede del griego y significa “descubrir, encontrar, inventar” (etimología que comparte eureka) [Polya, G., 1971]. Nicholson y Reeves ofrecen las siguientes definiciones de heurística:

- Es un procedimiento que permite resolver problemas a través de un método intuitivo en el cual la estructura del problema puede ser interpretado y explotado inteligentemente para obtener una solución razonable [Nicholson, T. A. J., 2007].
- Una técnica que busca buenas soluciones con un tiempo de computación razonable sin garantizar la optimalidad [Reeves, C. R., 1993].
- Un método heurístico siempre busca un camino corto a la solución del problema, pero sin garantizar obtener la mejor solución.

Los métodos constructivos incluyen las siguientes estrategias [López Locés, 2017]:

Algoritmo codicioso.- Comenzando desde un punto inicial, el algoritmo genera paso a paso una solución factible al agregar en cada iteración el componente que produce la mayor mejora en su estado actual. La principal desventaja del método es que no tiene conocimiento del impacto que las decisiones actuales tendrán en el futuro.

Método de descomposición**.-** Como su nombre lo indica, el algoritmo descompone el problema principal en instancias más pequeñas hasta que la solución de cada uno de los subproblemas generados sea simple para luego combinarlos y obtener la solución al problema general.

Método de reducción.- El enfoque intenta reducir el espacio de búsqueda de soluciones mediante la identificación de los atributos que se consideran parte de las soluciones de alta calidad y explorar solo en las regiones que las incluyen.

Modificación del Modelo.- El método simplifica el modelo original para resolver el problema de una manera más eficiente. Una vez que el problema se resuelve con la nueva formulación, la solución al problema original puede extrapolarse dada la solución obtenida. Algunas de las estrategias que entran en esta categoría incluyen la linealización, la agrupación de variables y la introducción de nuevas restricciones.

2.3.3. Óptimo local

En optimización una solución $\vec{x}'$ vecina de $\vec{x}$ es aquella que se encuentra dentro de su vecindario $N(\vec{x})$ es decir "próxima". Entiéndase por vecindario todas aquellas soluciones que pueden ser producidas a partir de $\vec{x}'$ mediante una sola perturbación. Un operador de vecindario transforma una solución inicial $\vec{x}$ y permite alcanzar todas sus soluciones en $N(\vec{x})$ mediante movimientos (aplicaciones del operador). En problemas de ordenamiento lineal usualmente se utilizan intercambios o inserciones, lo que se conoce también como formas de perturbación a la solución original. Un óptimo local es una solución $\forall y \in N(x) \quad f(x) \leq f(y)$ para el caso en que se minimiza el objetivo [Duarte Muñoz A. et al. 2010].

2.3.4. Métodos metaheurísticos

El término metaheurística o meta-heurística fue acuñado por F. Glover en el año 1986 [Muñoz et al., 2007]. Glover pretendía definir un procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimalidad local. Una de las definiciones más descriptivas del término metaheurística es la presentada por Osman y Kelly [Osman y Kelly, 1996], que se puede enunciar del siguiente modo:

"Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos 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"

Actualmente no existe una clara definición de la palabra metaheurístico y podría señalarse que el término está aún en proceso de definición. En algunos casos se define como un marco de referencia para la implementación de algoritmos y en otros se define a través de un conjunto de características algorítmicas. En el trabajo se utiliza la definición de Sörensen y Glover [Sörensen y Glover, 2013].

"Una metaheurística es un marco de referencia algorítmico de alto nivel que es independiente del problema y que provee un conjunto de lineamientos o estrategias para desarrollar algoritmos de optimización heurísticos. El termino también se usa para referirse

a una implementación orientada a un problema específico de un algoritmo de optimización heurístico de acuerdo con los lineamientos expresados en el marco de referencia"

Los métodos metaheurísticos esencialmente son métodos heurísticos, adquieren el prefijo meta ya que a diferencia de los heurísticos simples van más allá utilizando mecanismos para escapar de óptimos locales, las siguientes definiciones aportadas por diferentes investigadores describen sus principales propiedades:

- Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que los heurísticos clásicos no son eficientes [Duarte et al., 2006].
- Un metaheurístico es un proceso iterativo que combina conceptos derivados de las ciencias naturales, la biología, las matemáticas, la física y la inteligencia artificial, para guiar de forma inteligente una heurística subordinada durante la exploración y explotación del espacio de búsqueda con el fin de encontrar soluciones cercanas al óptimo de forma eficiente [Osman y Kelly, 1996]

Los métodos metaheurísticos se clasifican en dos grandes grupos: trayectoriales y poblacionales.

2.3.5. Metaheurísticos trayectoriales

Los metaheurísticos trayectoriales inician su proceso de optimización a partir de un punto de búsqueda (denominado solución inicial). La búsqueda sigue una trayectoria específica a partir de esta solución inicial mediante la aplicación de operadores de vecindario y cambios estocásticos, estos cambios se aceptan o no de acuerdo a un criterio de navegación (por ejemplo, si la nueva solución en la trayectoria mejora a la mejor solución conocida) propio del metaheurístico para generar el resultado final [Santiago Pineda, A.A. (2018)]. De las metaheurísticas más utilizadas ampliamente se encuentran:

- Búsqueda Tabú (TS).- Se le considera un procedimiento de búsqueda local que incorpora una memoria para escapar de óptimos locales.
- Recocido simulado (SA).- "*Es un algoritmo de búsqueda local capaz de escapar de óptimos locales permitiendo que bajo circunstancias se admitan movimientos*

que empeoren la mejor solución encontrada hasta el momento en el proceso de búsqueda".

- Búsqueda de vecindad variable (VNS).- "*Se basa en un cambio sistemático de las estructuras de vecindades dentro de un procedimiento de búsqueda local"* [Duarte Muñoz A. et al. 2010].

2.3.6. Metaheurísticos poblacionales

A diferencia de los trayectoriales, estos metaheurísticos comienzan el proceso de optimización a partir de diferentes puntos de búsqueda. Lo llevan a cabo manteniendo un conjunto de soluciones diverso llamado población. Al conjunto de individuos de la población se le asigna un valor de aptitud, conforme la búsqueda avanza los individuos con mejor aptitud conocida (mejores soluciones encontradas) son conservados, y los de baja aptitud descartados [Santiago Pineda, A.A. (2018)]. Dentro de los metaheurísticos poblacionales más utilizados se encuentran:

- Algoritmos genéticos (GA).- "*Se basa en una población de soluciones candidatas, que evoluciona por medio de los mecanismos genéticos neo-darwinistas de selección, cruce y mutación"*.
- Algoritmos meméticos (MA).- "*De la misma forma que en una población se transmiten los genes de los padres a los hijos, los memes se transmiten de cerebro a cerebro de la población, estos algoritmos basados en un algoritmo genético se extendien al agregar una búsqueda local"*.
- Búsqueda dispersa (SS).- "*La información sobre la calidad o el atractivo de un conjunto de reglas, restricciones o soluciones se puede utilizar mediante la combinación de éstas en lugar de aisladamente. De modo que, dadas dos soluciones de un problema, se podría obtener mediante su combinación, una nueva solución que mejore a las que la originaron"* [Duarte Muñoz A. et al. 2010].

2.3.7. Algoritmos evolutivos

Estos algoritmos son metaheurísticos guiados por una búsqueda poblacional, en la que los individuos de la población con mejores valores de aptitud tienen mayor posibilidad de supervivencia, así como de generar descendencia (soluciones derivadas) [Duarte Muñoz A.

et al. 2010]. Existe una variedad de algoritmos evolutivos, así como estrategias evolutivas empleadas, sin embargo, los algoritmos evolutivos más utilizados son: el algoritmo genético y el algoritmo memético (algoritmo genético extendido).

2.3.7.1 Algoritmo Genético (GA)

Estos algoritmos son una clase particular de algoritmo evolutivo, introducidos por Holland [Holland, 1992]. En general se compone de tres operadores diferentes reproducción, cruza y mutación [P. Moscato, 1989].

- *Reproducción*: este mecanismo favorece la copia de los individuos más aptos para tener descendencia.
- *Cruza*: intercambio de información entre dos individuos para crear uno nuevo tomando partes de los considerados padres.
- *Mutación*: alteración de una parte de un individuo.

Después de la aplicación de los operadores genéticos el proceso de selección ambiental permite depurar a los individuos menos aptos, sobreviviendo los de mayor valor de aptitud. En el caso de la optimización mono-objetivo generalmente son los que tienen mejor valor de la función objetivo. En el Algoritmo 2.3.7.1 se presenta un esquema general para los Algoritmos Genéticos.

Algoritmo 2.3.7.1 Algoritmo Genético

$t = 0$
inicializar [$P(t)$];
evaluar [$P(t)$];
mientras no terminar ***hacer***
$P'(t) \leftarrow$ selección_pareja [$P(t)$];
$P'(t) \leftarrow$ recombinación [$P'(t)$];
$P'(t) \leftarrow$ mutación [$P'(t)$];
evaluar [$P'(t)$];
$P(t+1) \leftarrow$ seleccionar_entorno [$P'(t) \cup P(t)$];
$t \leftarrow t + 1$;
fin mientras

2.3.7.2 Algoritmo Memético (MA)

MA debe su nombre al concepto de "meme" introducido por Dawkins [R. Dawkins, 1976]. Los algoritmos meméticos están estrechamente realacionados entre la búsqueda en una población global y la búsqueda local heurística que realiza cada uno de los individuos [P. Moscato, 1989]. Son algoritmos evolutivos que amplían los algoritmos genéticos tradicionales con métodos de optimización local [D'Aniello, G. 2014] y que combinan conceptos y estrategias de diferentes metaheurísticas para anuar las ventajas de las mismas. En el Algoritmo 2.3.7.2 se presenta un esquema general para los Algoritmos Meméticos.

Algoritmo 2.3.7.2 Algoritmo Memético

```
{x_best : TipoSolucion} ← MA(N:entero; Operador: vector [1. . .N] de TipoOperador;
f : TipoFuncionObjetivo)
variable
  x_g: vector [1. . .N] de TipoSolucion  // Soluciones con peso
  g, i: enteros
  inicio
    g ← 1  // Inicializar el número de generación
    {x} ← Inicializar(N)  // Inicializar aleatoriamente la población
    ⋁ i ∈ N hacer
      {x[i]} ← OptimizadorLocal(x[i],Operador[i], f )
    termina para
    repetir
      /*Paso generacional:*/
      {x} ← Seleccionar(x, f )  // Seleccionar los mejores individuos
      {x} ← Alterar(x); // Reproducir nuevos individuos
      ⋁ i ∈ N_s hacer
        {x[i]} ← OptimizadorLocal(x[i],Operador[i], f )
      termina para
      si convergencia(x) entonces
        {x} ← Inicializar(N_s); // Inicializar aleatoriamente la población
        ⋁ i ∈ N_s hacer
          {x[i]} ← OptimizadorLocal(x[i],Operador[i], f )
        termina para
      termina si
      g ← g + 1
    hasta terminacion
  {x_best} ← seleccionar(x, f ); // Seleccionar la mejor solución encontrada
```

---

fin

2.4. Búsqueda local (LS)

Los enfoques computacionales para resolver problemas de optimización combinatoria consisten principalmente en métodos LS. Estos métodos funcionan generando y evaluando soluciones candidatas iterativamente. Esos procesos, incluso para problemas combinatorios NP-Duros se pueden realizar en tiempo polinómico. Sin embargo, es el proceso de generación de soluciones candidatas el que tiene un mayor impacto sobre las propiedades teóricas y la eficiencia de los algoritmos LS [López Locés, 2017].

Los algoritmos LS se pueden clasificar en sistemática y local. Los primeros viajan sistemáticamente por el espacio de búsqueda, asegurando que encuentre la solución óptima [López Locés, 2017].

Por otro lado, los algoritmos LS, también llamados de mejora iterativa, comienzan desde una solución inicial en cualquier punto del espacio de búsqueda y realizan la exploración moviéndose a soluciones en su vecindario. Cada solución tiene un número relativamente pequeño de vecinos y cada movimiento está determinado por decisiones basadas en la información local disponible. Estos algoritmos no garantizan la solución óptima y también puede visitar la misma solución repetidamente durante el proceso de exploración [López Locés, 2017].

Entre los algoritmos de búsqueda local están la caminata aleatoria no informada (Uninformed Random Walk), remontando la colina (Hill Climbing), 2-Opt, entre otros [Schiavinoto and Stützle, 2004].

Dado un problema Π, un algoritmo de búsqueda local que resuelve una instancia $\pi \in \Pi$, esta definido por los siguientes componentes [Schiavinoto and Stützle, 2004]:

- El espacio de búsqueda $S(\pi)$ de la instancia π, el cual es un conjunto finito de soluciones candidatas $s \in S$ conocidas también como permutaciones, configuraciones o estados.
- Un conjunto de soluciones factibles $S'(\pi) \subseteq S(\pi)$
- Una relación de vecindad en $S(\pi), N(\pi) \subseteq S(\pi) \times S(\pi)$

- Un conjunto finito de estados memoria $M(\pi)$, o dado el caso de que el algoritmo de búsqueda local no implemente una estructura de memoria, su tamaño será de uno.
- Una función de inicialización $init(\pi): \emptyset \mapsto D(S(\pi) \times M(\pi))$, que especifica una distribución de probabilidad sobre las posiciones iniciales de búsqueda y estados de memoria.
- Una función de movimiento $step(\pi): S(\pi) \times M(\pi) \mapsto D(S(\pi) \times M(\pi))$ que asigna a cada posición y estado de memoria una distribución de probabilidad sobre las posiciones y estados de memoria de su vecindario.

Una condición de parada $terminate(\pi): S(\pi) \times M(\pi) \mapsto D(\{\top, \bot\})$ que asigna a cada posición y estado de memoria una distribución de probabilidad sobre los valores de verdad $\top = True, \bot = False$, los cuales indican la probabilidad de que la búsqueda local terminará al alcanzar un punto específico en el espacio de búsqueda o los estados de memoria.

El Algoritmo 2.4 muestra la estructura básica de un algoritmo LS.

Algoritmo 2.4 Búsqueda Local

Entrada: V Vector lineal.

E Vector con elementos a reasignar

1: E ← seleccionarElementos()

2: **mientras** E← ∅ **hacer**

3: $i \in E$

4: $j \leftarrow argumentomaximo_{k \in \{1,2,\dots,n\}}\{beneficio\{Insertarpredecesor(V,k,i)\}\}$

5: V ← $insertar(V,j,i)$

6: E ← E − $\{i\}$

7: **fin mientras**

2.4.1. Permutación

La búsqueda local opera sobre una solución candidata, misma que, es codificada en una estructura de datos en una representación dada, para ser manipulada por esta. A esta representación de una solución dada, se le denomina permutación y es definida de la siguiente manera.

Sea n un entero positivo, $\pi = \{a_1, a_2, \dots, a_n\}$ y sea a un conjunto de n objetos o elementos. Una permutación de n objetos $a_1, \dots, a_n$ es un arreglo ordenado de los objetos en π [Hromkovic and Oliva, 2001].

La exploración que realiza la función de búsqueda local es realizada a partir del estado actual de la permutación de la solución candidata que se desea mejorar, por medio de movimientos que pueden realizarse dentro del espacio de búsqueda, los cuales se denominan vecindarios.

Capítulo 3. Estado del arte

En el capítulo se presentan los trabajos relacionados con el problema de optimización de compras por internet (IShOP). El propósito de esta sección es hacer un diagnóstico de la situación actual y conocimiento disponible en cuanto a la solución del problema. Los trabajos que se describen se encuentran relacionados con el IShOP con costos de envío. Se finaliza con un análisis de los problemas o áreas abiertas.

3.1. Revisión de trabajos relacionados con el problema de IShOP con costos de envío

En 2010, Błażewicz realiza una definición formal del problema de IshOP y prueba que es NP-Duro en el sentido fuerte aun cuando los precios de los productos fueran iguales a cero y los costos de envío igual a 1 [Błażewicz, J., 2010]. Se construye una transformación pseudopolinomial del problema P1 al problema NP-Completo X3C. Se discute el caso especial antes mencionado; donde los precios de los productos son iguales a cero y el costo de envío es igual a 1, asociando un tipo de caso especial al problema MINIMUM SET COVER. Se proponen dos algoritmos polinomiales para resolver casos especiales: SHOP-

ENUM con una complejidad de tiempo $O(n2^m)$ y PRODUCT-ENUM con una complejidad de $O(nm^n)$ [Błażewicz, J., 2010].

Wojciechowski, realiza una evaluación de los mejores sitios web de comparación de precios, además se crea un modelo lo más parecido a las condiciones reales de las compras por internet; estudia la relación entre competitividad, publicidad, precios y dispersión de precios en las tiendas por internet [Wojciechowski, A., 2010]. Dicho modelo se enfoca en la compra de libros como producto de mayor demanda en ese entonces y con una amplia variedad en las tiendas virtuales. Dentro de ese modelo se utilizan ciertas fórmulas para calcular los precios de los productos de manera aleatoria y dividen los rangos de precios en 9 intervalos. Proponen una heurística llamada 2WBO, que ordena de manera descendente la lista productos y buscan la tienda con el menor costo para cada producto [Wojciechowski, A., 2010].

En 2011, Błażewicz describe los principios fundamentales del IShOP, y define de manera formal el problema [Błażewicz, J., 2011]. Además asocia el problema de IShOP sin descuentos con el problema de ubicación de instalaciones (FLP) y propone un algoritmo heurístico simple, en el que para obtener mejores resultados que los sitios de comparación de precios de libros se debe volver a ejecutar con diferentes pedidos [Błażewicz, J., 2011].

En 2015, Marszaªkowski define el problema B-ISOP en el que un cliente desea comprar una lista de productos considerando no rebasar un presupuesto establecido. Además las restricciones adicionales limitan a seleccionar un producto una sola vez y de ser así agregar el costo de envío de la tienda [Marszaªkowski, J, 2015]. Propone una formulación matemática del problema B-ISOP, y se considera maximizar la cantidad de productos que se pueden comprar con el presupuesto establecido. Se utiliza el problema BKP para probar la *NP-Completes* del problema, sin embargo para el caso especial en donde el cliente desea maximizar la cantidad de productos que recibe, se realiza una nueva transformación de esta problemática al problema MC. Incluyen una relación del B-ISOP con otros problemas como: Binary Knapsack Problem (BKP), Multiple Choice Knapsack Problem (MCKP), Weighted Maximum Coverage (WMC), Budgeted Maximum Coverage (BMC) y Generalized Maximum Coverage (GMC) [Marszaªkowski, J, 2015].

En 2015, López Locés aborda el problema IShOP con costos de envío en donde su objetivo principal es reducir el costo total de la compra (costo de los productos más el costo

de envío) bajo ciertas restricciones [López Locés, 2016]. Propone el uso de dos heurísticas basadas en la trayectoria como son: Búsqueda Tabú (TS) y Recocido simulado (SA) el objetivo de estas metaheurísticas es obtener una solución óptima para el problema de IShOP. Para ello se crean instancias a partir de condiciones reales de tiendas de libros en línea como Amazon, Barnes and Noble, entre otras. Se utiliza una representación matricial para representar las soluciones candidatas y se crean tres conjuntos de instancias (pequeñas, medianas y grandes) que a su vez contienen 3 subconjuntos cada uno. En las instancias pequeñas se consideraron 2, 4, y 5 productos y 20 tiendas. En las instancias medianas 5 y 50 productos y 240 y 400 tiendas. En las instancias grandes 50 y 100 productos y 240 y 400 tiendas. Las instancias incluyen los costos de los productos y los costos de envío para cada tienda. Se propone un modelo de programación lineal entera (ILP), mismo que permite obtener soluciones exactas en un tiempo razonable para casos reales de instancias pequeñas. Se propone el algoritmo heurístico MinMin que va recorriendo cada una de las tiendas, asignando y determinando si un producto tiene el menor costo dentro de esa tienda (costo del producto más el costo de envío asociado con la tienda). La contribución más relevante del trabajo de López Locés es un algoritmo metaheurístico de procesamiento celular que simula la ejecución paralela de varias células que exploran diferentes regiones del espacio de soluciónes para tratar de encontara la solución óptima. Este algoritmo se considera actualmente como el mejor del estado del arte de la solución del problema IshOP [López Locés, 2016].

3.2. Análisis del estado del arte

El presente trabajo de investigación se enfocará en el diseño e implementación de un algoritmo memético para el problema de ventas por internet con costos de envío, y en ese sentido, como se puede observar en el estado del arte, las metodologías reportadas para la solución de instancias de IShOP son: Tabú Search, Recocido Simulado [López Locés, 2015] y Procesamiento Celular [López Locés, 2016]. Hasta el momento el algoritmo de procesamiento celular, por los resultados que reporta, es considerado el mejor algoritmo para el problema de optimización de compras por internet con costos de envío (IShOP).

El propósito del trabajo de investigación es diseñar e implementar un algoritmo memético que iguale o mejore los resultados del mejor algoritmo propuesto en el estado del

arte, así como nuevas heurísticas y búsquedas locales que mejoren el desempeño de las soluciones reportadas por López Locés [López Locés, 2016].

En la Tabla 3.2 se puede apreciar las características de los trabajos mencionados en esta sección y una comparación con las características del proyecto propuesto en esta investigación. La primera columna de la tabla muestra el Autor y el año de cada trabajo, la segunda muestra las características de los trabajos que incluyen Modelos de programación Lineal Entera (ILP), la tercera muestra las características de los trabajos que usan Métodos Exactos, la cuarta muestra las características de los trabajos que usan Heurísticas Constructivas Deterministas, la quinta muestra las características de los trabajos que usan Heurísticas Constructivas Aleatorias, la sexta muestra las característica de los trabajos que usan Búsquedas Locales, la séptima muestra las características de los trabajos que usan Algoritmos Metaheurísticos y la octava columna muestra las características de los trabajos que definen Instancias para el problema del IShOP.

Tabla 3.2: Comparación de trabajos de IShOP.

Autor/Año	ILP	EXA	HED	HER	LS	META	INST
Błażewicz, J., 2010			✓				
Wojciechowski, A., 2010			✓				
Błażewicz, J., 2011			✓				✓
Marszaªkowski, J., 2015	✓						✓
López Locés, 2015						✓	✓
López Locés, 2016	✓		✓		✓		✓

Con base en la revisión de los trabajos reportados tanto para métodos exactos como métodos aproximados, se identifica lo siguiente:

- Hasta el momento no se han reportado algoritmos de solución que utilicen la metodología de algoritmos meméticos.
- No se ha reportado el uso de heurísticas constructivas aleatorias.

Capítulo 4. Descripción del mejor algoritmo del estado del arte.

4.1. Introducción

En este capítulo se realiza una breve descripción de los conceptos básicos y de la estructura del algoritmo de procesamiento celular, el cual se considera actualmente el mejor algoritmo en el estado del arte de la solución de IShOP.

Dicho algoritmo es un marco de trabajo flexible sobre el cual es posible construir soluciones escalables y configurables, concebida desde un principio para ejecutarsse en paralelo, pero que incluso en su versión secuencial, resulta altamente eficiente.

4.2. Algoritmo de procesamiento celular

Se describe el enfoque de la computación celular como una filosofía de diseño de algoritmos que se basa en tres principios interrelacionados. El principio de simplicidad establece que una célula de procesamiento idealmente realiza tareas muy simples que consumen poco tiempo. A continuación, el principio de paralelismo despliega muchas células individuales para resolver una sola tarea.

El tercero es el principio de localidad, que establece que, dada la gran cantidad de células, la comunicación entre todas ellas no es práctica, por lo tanto, se prefiere la comunicación local entre células vecinas [Sipper M., 1999].

El paradigma es especialmente adecuado para su uso en computación en la nube o centros de datos, ya que aprovecha la gran cantidad de procesadores disponibles.

En aplicaciones del mundo real, IShOP se tiene que resolver en contextos donde se requieren servidores centrales con varias unidades de cómputo para atender eficientemente las peticiones de múltiples usuarios de todo el mundo. Razón por la que se consideró utilizar un algoritmo que aplica la filosofía de la computación celular [López Locés, 2016].

Los algoritmos de procesamiento celular fueron propuestos inicialmente para resolver, con resultados prometedores, el problema del ordenamiento lineal con costos acumulativo [Terán-Villanueva et al., 2015].

A menudo es comparado con el enfoque hiperheurístico [Burke et al., 2003], a diferencia de dicha metodología en el procesamiento celular cada célula de procesamiento tiene un conocimiento completo del problema que se está resolviendo.

En contraste, el enfoque hiperheurístico tiene una barrera de dominio entre el controlador y las heurísticas de bajo nivel.

La segunda diferencia principal es el hecho de que las células de procesamiento están adaptadas al problema que se está resolviendo, mientras que el enfoque hiperheurístico depende de heurísticas genéricas de bajo nivel que pueden aplicarse a una mayor variedad de problemas de optimización [Burke et al., 2003].

4.3. Descripción del algoritmo de procesamiento celular

El algoritmo de procesamiento celular se basa en células de procesamiento independientes que mantienen su propia población y las procesa con su propio mecanismo. Los componentes de cada célula se describen a continuación:

La colección contiene una solución candidata o un conjunto de ellas que serán modificadas por el componente controlador celular. Es un método heurístico que resuelve y mejora las soluciones candidatas en la colección. El método que se incluye en el núcleo de procesamiento puede ser el mismo con pequeñas variaciones o completamente diferente en el algoritmo de procesamiento celular. El componente controlador celular controla el

estancamiento de la colección, pudiendo tomar medidas correctivas como generar un nuevo conjunto de soluciones candidatas en la colección, mover las existentes a otra región del espacio de búsqueda o detener su propia ejecución para evitar el mal uso de los recursos computacionales, como el tiempo de procesamiento computacional y el espacio de memoria.

Cada célula es bastante independiente entre sí, ya que comparte información sobre el estado de las soluciones candidatas en la colección con otras células del sistema. La comunicación se puede utilizar para reconfigurar los parámetros del núcleo de procesamiento para mover todas o algunas de las soluciones candidatas en la colección a una región diferente en el espacio de búsqueda para escapar del estado de estancamiento.

El mecanismo de comunicación permite realizar esta acción mientras el procesamiento se lleva a cabo en la célula después de que se haya procesado todo el conjunto de soluciones candidatas.

Para mejorar la ejecución de cada célula, estas se comunican con otras células durante su ejecución o después de que las células de procesamiento hayan evaluado a todos sus individuos en sus grupos respectivos.

El proceso completo generalmente termina cuando la mayoría de las células convergen a un óptimo local que podría o no ser la solución óptima para la instancia que se está resolviendo.

4.4. Implementación del algoritmo de procesamiento celular

El algoritmo celular implementado comienza construyendo una solución inicial con el método grasp para cada célula, que luego se mejora mediante la aplicación de un algoritmo de búsqueda local. La implementación de todas las células en el algoritmo de procesamiento celular como una unidad de procesamiento de búsqueda local iterativa se presenta en el Algoritmo 4.4. Se establecieron los siguientes parámetros del algoritmo de búsqueda local iterativa: número máximo de iteraciones sin mejora se estableció en 10 iteraciones, número máximo de iteraciones por cada célula se estableció en 1000. El algoritmo de búsqueda local encuentra la mejor posición para cada elemento de la solución, y, si ninguna otra posición mejora el valor objetivo, el elemento se deja en su lugar original. La comunicación se realiza compartiendo todas las soluciones con todas las células después de completar cada iteración.

Algoritmo 4.4 Algoritmo de Procesamiento Celular

Entrada: $Sol_i \leftarrow grasp()$: Creación de células utilizando grasp.

$MejorLocal_i \leftarrow LS(Sol_i, F(Sol_i))$: Mejor local por cada célula.

$MejorGlobal \leftarrow$ menor$(MejorLocal_i)$: El menor de los mejores locales.

$St_i \leftarrow 0$: Vector binario que indica si la célula *i* esta estancada o no.

Salida: $BG \leftarrow (X_1, \dots, X_m)$: Mejor global de selección de productos.

```
1: Repetir
2:     ∨ i ←1 a 2
3:       Solución(i) ← MejorLocal(i)
4:           ∨ i ←1 a 2
5:                si st(i) = 0
6:              mientras (Iteracion(i) % 50 ≠ 0 y Iteración(i) < IteraciónLocalMaxima)
7:            Solución(i) ← BúsquedaLocal(Solución(i))
8:            si Solución(i) < MejorLocal(i) entonces
9:               MejorLocal(i) ← Solución(i)
10:                    IteraciónSinMejora(i) ← 0
11:                de lo contrario
12:                    IteraciónSinMejora(i) ← IteraciónSinMejora(i) + 1
13:                Iteración(i) ← Iteración(i) + 1
14:                si (IteraciónSinMejora(i) > MaximaIteraciónSinMejora) entonces
15:                   st(i) ← 1,  IteraciónSinMejora(i) ← 0
16:           MejorLocal ← Menor(MejorLocal(1), MejorLocal(2))
17:           MejorGlobalAnterior ← MejorGlobal
18:               si (MejorLocal < MejorGlobalAnterior) entonces
19:                  MejorGlobal ← MejorLocal
20:                  IteraciónGlobalSinMejora  ← IteraciónGlobalSinMejora + 1
21:               de lo contrario
22:                 IteraciónGlobalSinMejora ← 0
      //Regenerar Células
23:  Solución1 ← grasp()
24:  Solución1_vector ← FunciónObjetivo(Solución1)
25:  Solución2 ← grasp()
26:  Solución2_vector ← FunciónObjetivo(Solución2)
     //Obtener MejorLocal
27:  MejorLocal(1) ← BúsquedaLocal(Solución1)
28:  MejorLocal(2) ← BúsquedaLocal(Solución2)
```

29: **hasta** *(IteraciónGlobalSinMejora < MaximaIteraciónGlobalSinMejora)*

4.4.1. Implementación de GRASP

El algoritmo inicia calculando el costo total de cada producto en cada una de las tiendas, posteriormente estos costos son ordenados de forma ascendente. Una vez asignada la lista de candidatos, de forma aleatoria se selecciona uno de ellos y posteriormente se determina a que tienda corresponde ese candidato seleccionado y se asigna la tienda en la posición actual del vector.

Esto continúa hasta que todos los productos tengan asignada una tienda donde deben comprarse los productos con el menor costo total. Lo anterior se puede observar en el Algoritmo 4.4.1.

Algoritmo 4.4.1 GRASP

Entrada:
solución: **vector** [1. . .*N*] Asignación de tienda por producto
costoxproducto: **vector** [1. . .*M*] Costo de producto en tienda seleccionada
tienda: entero
selección: real
Salida: *solución*

```
1:  inicio
2:     ⋁ i ∈ N hacer
3:          ⋁ j ∈ M hacer
4:          costoxproducto[j] ← FunciónObjetivo(j, i)
5:          termina para
6:          costoxproducto ← ordenar_ascendente(costoxproducto)
7:          selección ← seleccionaproducto(costoxproducto)
8:          tienda ← tiendaseleccionada(selección, costoxproducto)
9:          solución[i] ← tienda
10:     termina para
11:  retorna solución
12:  fin
```

4.4.2 Implementación de búsqueda local

El objetivo del algoritmo de búsqueda local es minimizar el costo total generado por el resultado obtenido en la solución inicial creada por el Algoritmo 4.4.1, para ello la búsqueda local inicia tomando la secuencia seleccionada de productos y el costo total de esa secuencia seleccionada de productos que se desean mejorar.

Se asigna a *i'* la tienda que corresponde a la posición dentro del vector, después en la posición se asigna la tienda j para calcular el costo total de la nueva selección de productos, si el costo es menor entonces es actualizado y j es asignada a *i'*.

Una vez que han sido exploradas todas las tiendas y localizado el menor costo del producto, la posición del vector es actualizada por el valor de *i'*.

Lo anterior se puede verificar en el Algoritmo 4.4.2.

Algoritmo 4.4.2 Búsqueda local (LS)

Entrada:

$X_{old} \leftarrow (X_{old1}, \dots, X_{oldm})$: Secuencia seleccionada de productos a mejorar

N_i: Multiconjunto de productos disponibles por tienda.

Salida:

$X \leftarrow (X_1, \dots, X_m)$**:** Secuencia seleccionada de productos mejorada

1: $X \leftarrow X_{old}$
2: $Costo \leftarrow FunciónObjetivo(X)$
1: **inicio**
2: $\bigvee i \in N$ **hacer**
3: *i'* ← *X[i]*
4: $\bigvee j \in M$ **hacer**
5: *X[i]* ← *j*
6: **si** *FunciónObjetivo(X)* < *Costo* **entonces**
7: *Costo* ← *FunciónObjetivo(X)*
8: *i'* ← *j*
9: **termina si**
10: **termina para**
11: *X[i]* ← *i'*
12: **termina para**
13: **fin**

Capítulo 5. Algoritmo memético propuesto

5.1. Introducción

En el proyecto se aborda el problema de optimización de compras por internet (IShOP) con costos de envío.

Las principales contribuciones en el trabajo son las siguientes:

- Un algoritmo memético novedoso para el problema de optimización de compras por Internet con costos de envío,
- una nueva representación vectorial de las soluciones candidatas que reduce la complejidad temporal de $O(n^2)$ a $O(n)$ para calcular los valores objetivos,
- dos nuevas heurísticas para crear individuos mejorados,
- dos nuevos operadores de mutación para el problema,
- un nuevo mecanismo que acelera el cálculo de los valores objetivos,
- y después de cada generación, en la población solo queda la mejor solución y el resto de los individuos se regeneran aleatoriamente.

5.2. Definición del problema

Un cliente desea comprar por internet un conjunto de *n* productos *N*, los cuales se pueden adquirir en un conjunto de *m* tiendas disponibles *M*. El conjunto N_i contiene los productos disponibles en la tienda *i*, cada producto $j \in N_i$ tiene un costo de c_{ij} y un costo de envío d_i.

El costo de envío se cobra solo si se compran uno o más productos en la tienda *i*. *El problema de compras por Internet con costos de envío (IShOP)* consiste en minimizar el costo total de la compra de todos los productos en *N* considerando el costo de los productos más los gastos de envío.

Formalmente el problema consiste en determinar una partición de los productos a comprar en las diferentes tiendas $X = (X_1, \dots, X_m)$, de modo que $X_i \subseteq N_i$ y que se compren todos los productos $U_{i=1}^{m} X_i = N$ y que se minimice el costo total, tal como se muestra en la Ec. 5.1:

$$F(X) = \sum_{i=1}^{m} \left(\sigma(|X_i|) d_i + \sum_{j \in X_i} c_{ij}\right) \qquad (5.1)$$

Donde: $|X_i|$ es la cardinalidad del contenedor X_i, y $\sigma(x) = 0 \; si \; x = 0 \; y \; \sigma(X) = 1 \; si \; x > 0$.

Si la lista de productos que se va a comprar está formada por *N* productos y hay *M* tiendas disponibles, entonces una solución se representa en un vector de longitud *N*, el cual contiene para cada producto la tienda donde se va a comprar.

5.3. Estructura general del algoritmo memético

5.3.1 Representación de la solución y reducción de la complejidad del cálculo de la función objetivo

El mecanismo de cálculo dado en la definición del problema se basa en una representación matricial de las soluciones por lo que la complejidad temporal de la evaluación de una solución es de $O(n^2)$. En esta propuesta las soluciones candidatas del problema se representan con un vector *I(j)* en el cual se especifican para cada producto *j* la tienda *i* donde se va a comprar, como se muestra en la Fig. 5.3.1.

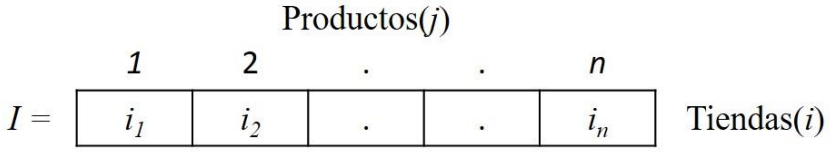

Figura 5.3.1**:** Representación de una solución.

Con esta representación se propone calcular el valor objetivo de una solución *I* usando la Ec. 5.2.

$$F(I) = \sum_{i=1}^{m} d_i + \sum_{j=1_{\forall j \,|tal\ que\ I(j)=i}}^{n} (c_{ij}) \qquad (5.2)$$

El índice *j* de la sumatoria interna toma solamente los valores de los productos entre 1 y *m* que se van a comprar en la tienda *i*. Cuando la sumatoria externa termina el número total de acumulaciones que se realizan es *n.* Por lo tanto la complejidad temporal del cálculo del valor objetivo de una solución es de *O(n).* Como este proceso se realiza en forma masiva en el algoritmo que se propone, este mecanismo puede tener un gran impacto en su eficiencia.

5.4. Heurística para generar soluciones

En esta sección se describen dos heurísticas de construcción de soluciones para generar soluciones. La idea principal detrás de estas heurísticas de construcción es asignar a cada producto, la tienda en la que el producto tenga menor costo, considerando el costo del producto y el costo de envío.

5.4.1 Heurística FirstPlus

Esta heurística inicia creando de manera aleatoria una solución candidata con su costo de envío, que incluye c_{ij} y d_i del multiconjunto N_i. Luego, esta solución se asigna a X y su valor objetivo se almacena en la variable $Costo$. Después se verifica para cada tienda i en M, desde que tienda podríamos comprar el producto para reducir el costo total de la selección de tiendas en X.

La búsqueda de esa tienda finaliza cuando el costo total de X es menor que $Costo$, y se pasa al siguiente producto que debe ser diferente al producto actual. El proceso continúa hasta que todas las tiendas i en M hayan sido revisadas. En el Algoritmo 5.4.1 se representa la estructura de dicho proceso.

El algoritmo es polinomial con una complejidad temporal de $O(n^2)$.

Algoritmo 5.4.1. *Heurística FirstPlus*

Entrada: $X_{old} \leftarrow (X_{old_1}, \dots, X_{old_m})$: Vector de elementos (producto, tienda) a mejorar

N_i: Multiconjunto de productos disponibles por tienda

N: Lista de compras

M: Lista de tiendas

Salida: $X \leftarrow (X_1, \dots, X_m)$: Vector de elementos (producto, tienda) mejorados

1: $X \leftarrow X_{old}$
2: *Costo* $\leftarrow Función Objetivo(X)$
3: $\bigvee j \in N$ **hacer**
4: $\bigvee i \in M$ **hacer**
5: *X[j]* ← *i*
6: **si** *FunciónObjetivo(X)* ≤ Costo **entonces**
7: *Costo* ← *FunciónObjetivo(X)*
8: *i'* ← *i*
9: salir
10: ***continuar***
11: *X[j]* ← *i'*
12: **retorna** *X*

5.4.2 Heurística FirstBest

La heurística comienza calculando el costo total de una solución seleccionada contenida en un vector de valores que contiene las tiendas i asignadas en las secuencia de selección de productos con su costo, que incluye c_{ij} y d_i del multiconjunto N_i. Luego, esta secuencia se asigna a X y su valor objetivo actual se almacena en la variable $Costo$. Después se verifica para cada tienda i en M, desde que tienda podríamos comprar el producto para reducir el

costo total de la selección de tiendas en X. Esto se realiza primero calculando el costo del producto en la nueva tienda que se desea asignar, posteriormente se calcula el costo del producto en la tienda seleccionada actualmente dentro de la solución. Si el costo de asignar a la nueva tienda es menor que el costo en la tienda ya seleccionada, entonces, se actualiza la nueva tienda.

La búsqueda de esta tienda finaliza cuando se han revisado todas las tiendas y el costo V es menor que $Costo$, se mueve a un nuevo producto que debe ser diferente al producto actual. El proceso continua hasta que todas las tiendas i en M hayan sido reasignadas. La diferencia con respecto a la heurística anterior es que $FirstBest$ no se detiene con la primera mejora en la solución, esta heurística continua hasta haber hecho todas las posibles combinaciones y obtener el menor costo en la tienda de compras. El Algoritmo 5.4.2 ilustra la estructura del proceso.

El algoritmo es polinomial con una complejidad temporal de $O(n^2)$.

Algoritmo 5.4.2 *Heurística FirstBest*

Entrada: $X_{old} \leftarrow (X_{old_1}, \dots, X_{old_m})$: Vector de elementos (producto, tienda) a mejorar

N_i: Multiconjunto de productos disponibles por tienda

N: Lista de compras

M: Lista de tiendas

Salida: $X \leftarrow (X_1, \dots, X_m)$: Vector de elementos (producto, tienda) mejorados

1: $X \leftarrow X_{old}$

2: $Costo \leftarrow FunciónObjetivo(X)$

3: $\bigvee j \in N$ **hacer**

4: $\bigvee i \in M$ **hacer**

5: *X[j]* ← *i*

6: **si** *FunciónObjetivo(X)* ≤ *Costo* **entonces**

7: *Costo* ← *FuciónObjetivo(X)*

8: $i' \leftarrow i$

9: $X[j] \leftarrow i'$

10: **retorna** X

5.5. Selección por torneo binario

La idea principal del método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo: Determinística y Probabilística.

En la versión determinística se selecciona al azar un número p de individuos (generalmente se escoge $p = 2$). De entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

Se seleccionan aleatoriamente *p* individuos de la población *pop* y el más apto es el que pasa a la siguiente generación.

En el algoritmo memético se realiza un torneo binario (con *p=2*) sobre la población pop [Blazewicz, et al., 2011]. Cada solución participa en dos torneos y la solución ganadora se selecciona y se inserta en *NewPop*. De esta manera, cualquier solución en *Pop* puede tener como máximo dos copias en *NewPop*. La Fig. 5.5 ilustra el proceso.

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo [0..1], si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo más apto y en caso contrario el menos apto. Generalmente p toma valores en el rango $0{,}5 < p \leq 1$ [6].

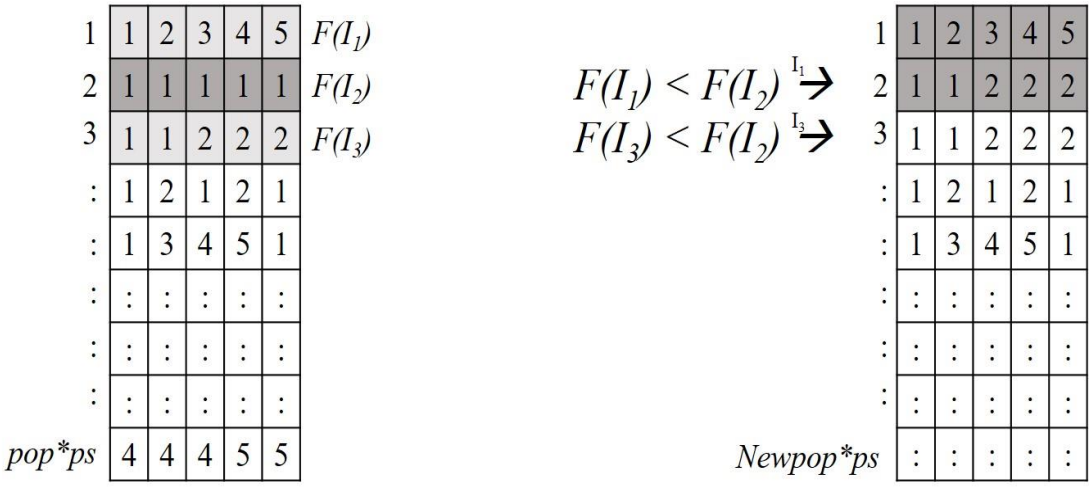

Figura 5.5**:** Ilustración del torneo binario.

5.6. Operador de cruza

El operador se aplica a un par de soluciones *padre₁* y *padre₂*, seleccionados de manera secuencial hasta que se hayan seleccionado un porcentaje de los individuos de la población [Holland, 1975]. Para generar la solución *hijo₁*, primero se toma la mitad inicial del *padre₁* y se une con la segunda mitad del *padre₂*. Posteriormente para formar la solución *hijo₂*, se toma la mitad inicial del *padre₂* y se une con la segunda mitad del *padre₁* [Umbakar and Sheth, 2015].

En el operador de cruza, el punto $\lfloor N/2 \rfloor$ ó $\lceil N/2 \rceil$ se selecciona como punto de cruce. La Fig. 5.6 ejemplifica el proceso de cruza en un punto para dos soluciones de diferentes tamaños.

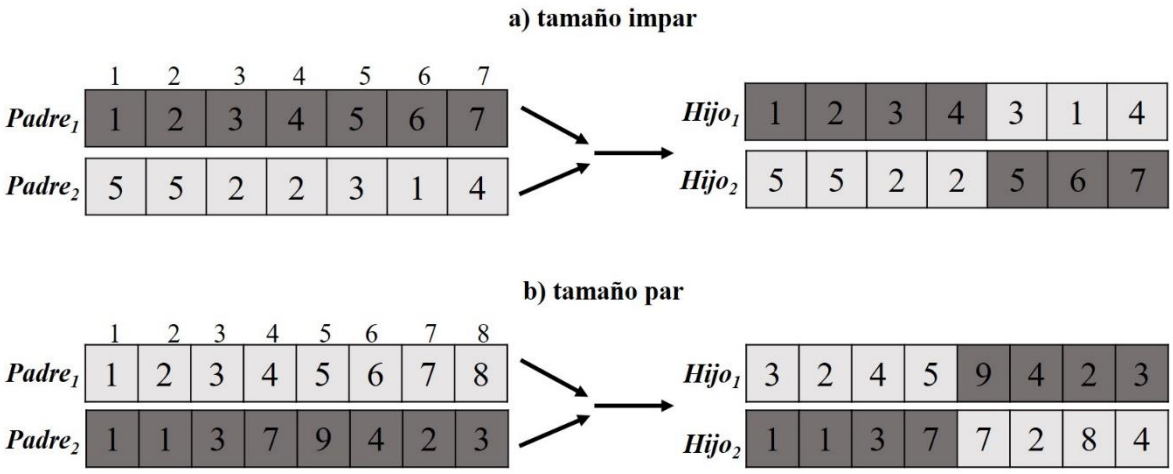

Figura 5.6**:** Punto de cruce en una solución.

5.7. Operadores de mutación

En esta sección se describen los dos operadores de mutación propuestos: el operador de mutación uniforme FirstBest y el operador de mutación heurística FirstBest.

5.7.1 Mutación uniforme FirstBest

Se diseñó un método de mutación uniforme, que se describe a continuación:

Seleccionando $ps - ps * er$ soluciones y para cada solución se genera un número aleatorio. Si el valor generado aleatoriamente es menor que *ps*pm*, se selecciona un elemento del vector solución. Posteriormente se genera un nuevo número aleatorio, si el valor es menor que *ps*pmu*, se generan todas las posibles combinaciones de asignación (*producto, tienda*), se evalúan todas estas combinaciones y se selecciona la mejor. Continuamos con el mismo procedimiento anterior hasta haber visitado todos los elementos del vector solución. Ilustramos la representación gráfica del proceso en la Fig. 5.7.1.

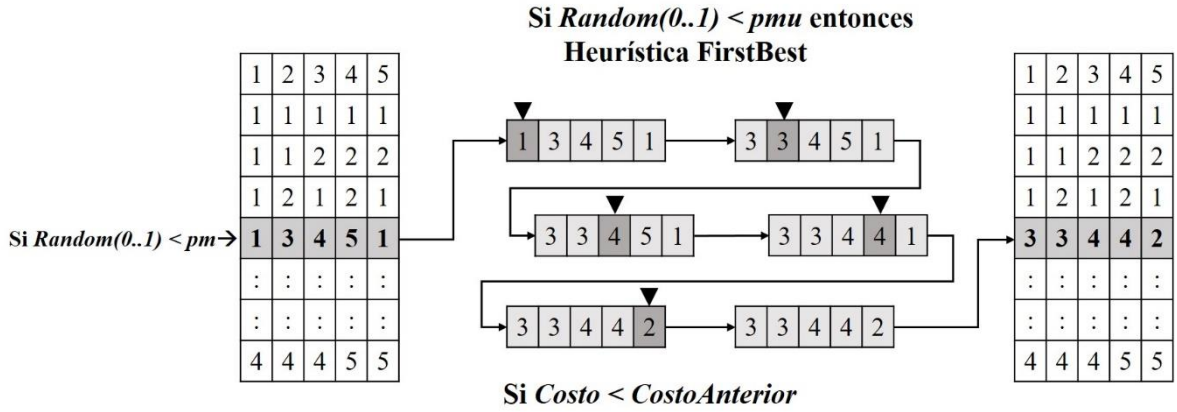

Figura 5.7.1: Proceso de la mutación uniforme FirstBest.

5.7.2 Mutación heurística FirstBest

Se diseñó un método de mutación heurística, considerando el siguiente procedimiento: Seleccionando $ps - ps * er$ soluciones y por cada una de estas se genera un número al azar, si el valor generado al azar es menor que *ps*pm* se procede a generar todas las posibles combinaciones de asignación $(producto, tienda)$, se evalúan todas estas combinaciones y se selecciona la mejor. La Fig. 5.7.2 ilustra gráficamente el proceso.

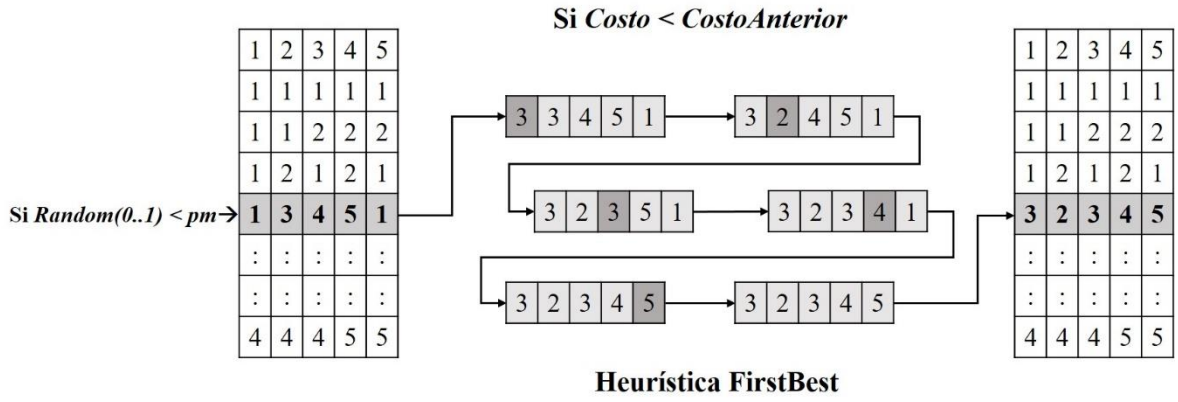

Figura 5.7.2**:** Proceso de la mutación heurística FirstBest.

5.8. Búsqueda local

En la Fig. 5.8 se muestra como el procedimiento comienza seleccionando una solución de *IntermediatePop*, siendo está un vector de valores asociados que contiene las tiendas *i* asignadas en la secuencia de selección de productos *X* con su costo, que incluye c_{ij} y d_i del multiconjunto N_i.

Posteriormente se verifica para cada producto, desde que tienda podríamos comprarlo para reducir el costo total de la selección de productos *X*.

Cuando finaliza la búsqueda de esa tienda, se mueve a un nuevo producto. El proceso continúa hasta que todas las tiendas *i* en *X* hayan sido revisadas. Concluye cuando todas las *X* hayan sido asignadas en *ChildPop*.

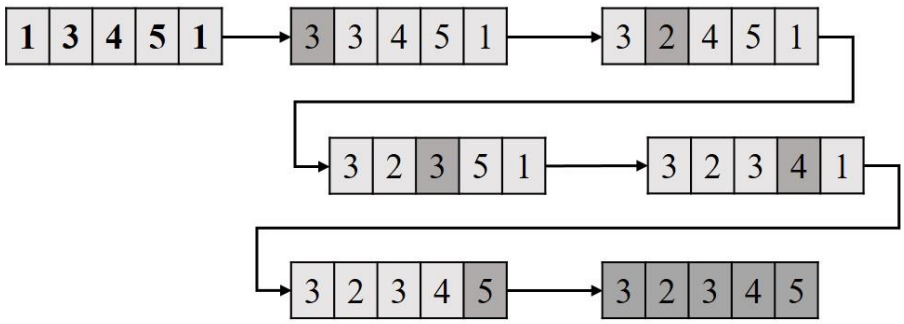

Figura 5.8: Ilustración de la búsqueda local basada en la heurística FirstBest.

5.9. Algoritmo memético (MAIShOP)

El Algoritmo 5.9 describe la estructura general del algoritmo MAIShOP. El algoritmo, comienza definiendo los parámetros tales como tamaño de la población inicial *ps*, porcentaje de cruza *pc*, probabilidad de mutación *pm*, porcentaje de elitismo *er*, criterio de terminación *timelimit*, enseguida se genera una población inicial (Paso 2) utilizando una heurística de construcción. En la fase generacional (Pasos 5-15), se aplica el operador de torneo binario para la selección en el Paso 6. Las mejores soluciones *ps*er* (elitismo) de *pop* se copian en *IntermediatePop* en el Paso 7. En *ps*pc* de las soluciones restantes en *NewPop*, se aplica el operador de cruza para generar más soluciones para *IntermediatePop* y las soluciones restantes se copian tal como están en *IntermediatePop* en el Paso 8. Se supone que *pc* se selecciona de una manera que *ps*pc* es menor o igual que el número de soluciones restantes en *NewPop*. En el Paso 9, los individuos en *IntermediatePop* sufren mutación excepto

aquellos individuos que fueron seleccionados en el Paso 6 usando elitismo. En el Paso 10, se aplica la búsqueda local a *IntermediatePop* para mejorar las soluciones. En el Paso 11, se obtiene la mejor solución local (*BestLocal*) de *ChildPop* y se compara con la solución global (*BestGlobal*) en el Paso 12 además se calcula el tiempo de ejecución (*timesolution*) una vez evaluado el paso, si *BestLocal* tiene un menor costo entonces reemplaza a *BestGlobal*, en el Paso 13 la mejor solución local (*BestLocal*) es copiada a la población inicial y todas las demás soluciones son recalculadas, se calcula el tiempo de ejecución total del algoritmo para cada generación en el paso 14 y finalmente se retorna la mejor solución global (*BestGlobal*) con el cálculo de tiempo de ejecución en la que se encontró esta solución en el paso 16.

Algoritmo 5.9. *Algoritmo memético (MAIShOP)*

Entrada: *tiempolimite:* Tiempo límite en la ejecución del algoritmo
Salida: *MejorGlobal:* Mejor solución global
tiemposolución: Tiempo de la solución

1: Se definen los siguientes elementos: tamaño de la población inicial *ps*, porcentaje de cruza *pc*, probabilidad de mutación *pm*, porcentaje de elitismo *er*, criterio de terminación *tiempolimite*
2: Generar población inicial *pop* de tamaño *ps*
3: Determinar *MejorLocal* en *pop*.
4: *MejorGlobal* ← *MejorLocal*.
5: **mientras** no se cumpla el criterio de terminación *tiempolimite* **hacer**
6: Aplicar operador de torneo binario en *pop* para obtener *NewPop*
7: *ps.er* soluciones élite de NewPop se pasan a *IntermediatePop*
8: Entre los individuos restantes en *NewPop*, se aplica el operador de cruza a *ps*pc* soluciones seleccionadas al azar y las soluciones restantes se copian tal como están a *IntermediatePop*
9: Aplique el operador de mutación en *IntermediatePop* con probabilidad *pm* excepto a las soluciones élite.
10: Se aplica la búsqueda local (*FirstBest*) a *IntermediatePop* para generar *ChildPop.*
11: Se obtiene la mejor solución (*MejorLocal)* de *ChildPop*
12: Se determina si el valor objetivo de la mejor solución local (BestLocal) es menor que la solución global (*MejorGlobal*)**,** si es así se actualiza la solución global. Se calcula el tiempo de ejecución del algoritmo *(timesolution).*
13: La mejor solución local (*MejorLocal) se copia a pop* y todos los demás individuos son generados nuevamente.
14: Se calcula el tiempo de ejecución del algoritmo para cada generación *(tiempototal).*
15: **termina mientras**
16: **retorna (***MejorGlobal***,** *tiemposolución*)

Capítulo 6. Experimentación computacional.

6.1. Introducción

En esta sección se describe una muestra representativa de los experimentos realizados en el proyecto. Los experimentos se realizaron en una plataforma de hardware y software que incluye un procesador Intel Core 2 Duo a 2,53 Ghz. con 4 GB de RAM, versión de Java 1.8.0_231. Básicamente se hicieron dos tipos de experimentos, una serie de experimentos para determinar la mejor configuración de los parámetros y un experimento para comparar el desempeño del algoritmo propuesto con la mejor configuración contra el mejor algoritmo del estado del arte.

6.2. Instancias utilizadas

Para generar las instancias utilizadas en los experimentos, se utilizó un modelo realista. En este se considera la relación entre la estructura competitiva, la publicidad, el precio y la dispersión de precios en las tiendas de Internet. Para segurar la representatividad de los productos, se eligen los libros, por una amplia oferta en tiendas virtuales (Internet) y la mayor frecuencia de compra.

Las instancias utilizadas en esta experimentación se generaron utilizando el generador disponible en la página del Proyecto de optimización de compras en Internet [1a].

El nombre de las instancias especifica su tamaño, *m* representa el número de tiendas y *n* el número de productos. Se crearon tres conjuntos de instancias de diferentes tamaños. Un conjunto de instancias pequeñas, con tres subconjuntos de 3, 4 y 5 productos y 20 tiendas, cada uno con 30 instancias. Un conjunto de instancias medianas, con tres subconjuntos de 5 y 50 productos con 240 y 400 tiendas, cada uno con 30 instancias. Un conjunto instancias grandes con tres subconjuntos: de 50 y 100 productos con 240 y 400 tiendas, cada uno también con 30 instancias del problema. Los nombres de los subconjuntos generados se define con el método descrito por Blazewicz [Blazewicz, et al., 2010], donde el nombre de cada subconjunto especifica el número de productos *n* y el de tiendas *m* de las instancias incluidas en el subconjunto. El conjunto de instancias pequeñas esta formado por los subconjuntos 3*n*20*m*, 4*n*20*m* y 5*n*20*m*, cada uno con 90 instancias. El conjunto de instancias medianas esta formado por los subconjuntos 5*n*20*m*, 5*n*400*m* y 50*n*240*m*), cada uno con treinta instancias. El conjunto de instancias grandes contiene los subconjuntos 50*n*400*m*, 100*n*240*m* y 100*n*400*m*, también cada uno con 30 instancias.

6.2. Configuración de los parámetros del algoritmo

Se realizó un experimento preliminar con grupos de instancias medianas. Se formaron cada grupo de 30 instancias, y para cada caso, se ejecutaron 30 corridas independientes, con cada uno de los valores de los parámetros que se configuraron. En cada serie, se consideró un límite de tiempo en segundos de CPU. Para las instancias de los grupos 5*n*240*m*, 5*n*400*m* y 50*n*240*m*, se usó 0.565, 1.49 y 31 como límites de tiempo, respectivamente. Para cada caso, se determinó la solución con el valor objetivo más bajo y el promedio de las 30 soluciones encontradas. Los parámetros que se configuraron son el tamaño de la población *(pop)*, el porcentaje de la cruz *(pc)*, la probabilidad de mutación *(pm)* y la tasa de elitismo *(er)*. La prueba de hipótesis no paramétrica de Friedman se aplicó con un 95% de confianza para determinar si las diferencias observadas son significativas o no.

[1a] http://www.cs.put.poznan.pl/ishop/.

La Tabla 6.2 muestra los resultados que se obtienen cuando el tamaño de la población toma los valores de 50, 100 y 150. La primera columna de la tabla contiene los nombres de los grupos de instancias utilizados en los experimentos, la segunda, tercera y cuarta columna contienen el promedio de las treinta mejores soluciones encontradas y para cada caso se muestra el valor del ranking de Friedman como subíndice. La quinta contiene el *p-value* reportado por la prueba de Friedman. Para cada grupo de instancias las diferencias observadas son significativas cuando el valor sea menor que 0.05 y la celda sombreada señala al algoritmo configurado que obtuvo el mejor ranking. El algoritmo con mejor desempeño reportado en la prueba es el que tiene el menor valor en el ranking. Como se observa en la tabla para los primeros dos grupos de instancias no existe diferencia significativa entre las tres configuraciones del tamaño de la población. Para el tercer grupo de instancias la configuración del algoritmo con un tamaño de población de 100 supera en promedio a las otras dos configuraciones y si hay diferencias significativas.

Tabla 6.2: Configuración del tamaño de la población (*pop*).

Instancia	*pop*=50	*pop*=100	*pop*=150	*p-value*
5*n*240*m*	75.58_{56}	77.32_{65}	76.57_{59}	0.49659
5*n*400*m*	$69.80_{56.5}$	70.63_{63}	$69.86_{60.5}$	0.69884
50*n*240*m*	550.16_{72}	477.41_{30}	552.12_{78}	0.00001

La Tabla 6.2.1, muestra los resultados que se obtienen cuando el porcentaje de cruza toma los valores de 0.4, 0.6 y 0.7. La primera columna de la tabla contiene los nombres de los grupos de instancias utilizados en los experimentos, la segunda, tercera y cuarta columna contienen el promedio de las treinta mejores soluciones encontradas y para cada caso se muestra el valor del ranking de Friedman como subíndice. La quinta contiene el *p-value* reportado por la prueba de Friedman. Para cada grupo de instancias las diferencias observadas son significativas cuando el valor sea menor que 0.05 y la celda sombreada señala al algoritmo configurado que obtuvo el mejor ranking. El algoritmo con mejor desempeño reportado en la prueba es el que tiene el menor valor en el ranking. Como se observa en la tabla para los primeros dos grupos de instancias no existe diferencia significativa entre las tres configuraciones del porcentaje de cruza. Para el tercer grupo de instancias la configuración del algoritmo con un porcentaje de cruza de 0.6 supera en promedio a las otras dos configuraciones y si hay diferencias significativas.

Tabla 6.2.1: Configuración del porcentaje de cruza (*pc*).

Instancia	*pc*=0.4	*pc*=0.6	*pc*=0.7	*p-value*
5*n*240*m*	75.25_{55}	77.32_{67}	75.85_{58}	0.27253
5*n*400*m*	70.05_{61}	$70.63_{60.5}$	$69.91_{58.5}$	0.94334
50*n*240*m*	$548.78_{79.5}$	477.41_{30}	$546.16_{70.5}$	0.00001

La Tabla 6.2.2, muestra los resultados que se obtienen cuando la probabilidad de mutación toma los valores de 0.01, 0.03 y 0.05. La primera columna de la tabla contiene los nombres de los grupos de instancias utilizados en los experimentos, la segunda, tercera y cuarta columna contienen el promedio de las treinta mejores soluciones encontradas y para cada caso se muestra el valor del ranking de Friedman como subíndice. La quinta contiene el *p-value* reportado por la prueba de Friedman. Para cada grupo de instancias las diferencias observadas son significativas cuando el valor sea menor que 0.05 y la celda sombreada señala al algoritmo configurado que obtuvo el mejor ranking. El algoritmo con mejor desempeño reportado en la prueba es el que tiene el menor valor en el ranking. Como se observa en la tabla para los primeros dos grupos de instancias no existe diferencia significativa entre las tres configuraciones de la probabilidad de mutación. Para el tercer grupo de instancias la configuración del algoritmo con una probabilidad de mutación de 0.01 supera en promedio a las otras dos configuraciones y si hay diferencias significativas.

Tabla 6.2.2: Configuración de la probabilidad de mutación (*pm*).

Instancia	*pm*=0.01	*pm*=0.03	*pm*=0.05	*p-value*
5*n*240*m*	77.32_{65}	76.18_{60}	75.42_{55}	0.4346
5*n*400*m*	$70.63_{61.5}$	70.02_{60}	$69.52_{58.5}$	0.92774
50*n*240*m*	477.41_{30}	545.42_{73}	549.20_{77}	0.00001

La Tabla 6.2.3, muestra los resultados que se obtienen cuando el porcentaje élite toma los valores de 0.03, 0.05 y 0.08. La primera columna de la tabla contiene los nombres de los grupos de instancias utilizados en los experimentos, la segunda, tercera y cuarta columna contienen el promedio de las treinta mejores soluciones encontradas y para cada caso se muestra el valor del ranking de Friedman como subíndice. La quinta contiene el *p-value* reportado por la prueba de Friedman. Para cada grupo de instancias las diferencias observadas son significativas cuando el valor sea menor que 0.05 y la celda sombreada señala al algoritmo configurado que obtuvo el mejor ranking. El algoritmo con mejor desempeño

reportado en la prueba es el que tiene el menor valor en el ranking. Como se observa en la tabla para los primeros dos grupos de instancias no existe diferencia significativa entre las tres configuraciones del porcentaje élite. Para el tercer grupo de instancias la configuración del algoritmo con un porcentaje élite de 0.05 supera en promedio a las otras dos configuraciones y si hay diferencias significativas.

Tabla 6.2.3: Configuración del porcentaje de elitismo (*er*).

Instancia	*er*=0.03	*er*=0.05	*er*=0.08	*p-value*
5*n*240*m*	$76.07_{56.5}$	$77.32_{64.5}$	75.97_{59}	0.57216
5*n*400*m*	$69.95_{59.5}$	70.63_{60}	$70.14_{60.5}$	0.9917
50*n*240*m*	$546.24_{69.5}$	477.41_{30}	$553.58_{80.5}$	0.00001

La Tabla 6.2.4 muestra los resultados obtenidos cuando la mutación uniforme toma los valores 0.01, 0.03 y 0.05. La primera columna de la tabla contiene los nombres de los grupos de instancias que se usaron en los experimentos; la segunda y tercera columna muestran el promedio de las 30 mejores soluciones que se encontraron. Para cada caso, se muestra el valor del ranking asignado por la prueba de hipótesis de Friedman (subíndice). La quinta columna muestra el valor *p-value* asignado por la prueba de Friedman. Para cada grupo de instancias, las diferencias observadas son significativas cuando es valor es menor que 0.05, y la celda sombreada indica el algoritmo configurado que obtuvo la mejor clasificación. El algoritmo con el mejor rendimiento observado en la prueba es el que tiene el valor de clasificación más bajo. Como se muestra en la tabla para los tres grupos de instancias, no hay diferencia significativa entre las tres configuraciones del porcentaje de probabilidad de mutación uniforme. Por lo tanto, se decide establecer el 0.01% como parámetro para la mutación uniforme. Debido a que para instancias de tamaño 50*n*240*m*, esta configuración obtiene el costo más bajo en la función objetivo.

Tabla 6.2.4: Configuración de la probabilidad de mutación uniforme (*pmu*).

Instancia	*pmu*= 0.01	*pmu*= 0.03	*pmu*= 0.05	*p-value*
5*n*240*m*	76.027_{62}	$75.347_{59.5}$	$75.612_{58.5}$	0.89733
5*n*400*m*	$69.697_{62.5}$	69.579_{60}	$69.506_{57.5}$	0.81194
50*n*240*m*	$536.898_{49.5}$	548.403_{67}	$546.097_{63.5}$	0.05736

La siguiente tabla muestra los resultados obtenidos con las dos mutaciones seleccionadas (mutación uniforme y mutación heurística). La primera columna de la tabla contiene los nombres de los grupos de instancias que se usan en los experimentos; la segunda y tercera columnas enumeran el promedio de las 30 mejores soluciones que se encontraron. La cuarta columna contiene el valor *p-value* obtenido por la prueba de hipótesis no paramétrica de Wilcoxon. Para cada grupo de instancias, las diferencias observadas son significativas cuando el valor es menor que 0.05, y la celda sombreada indica el algoritmo configurado que obtuvo la mejor clasificación.

La Tabla 6.2.5 muestra los resultados obtenidos con cada grupo de 30 instancias. Para cada grupo se muestra la desviación estándar (subíndice) del valor objetivo y el tiempo de solución. En las celdas de Mutación uniforme se indica si hay diferencias significativas a favor del tipo de Mutación uniforme (↓) o a favor de la Mutación heurística (↑) y si no hay diferencias significativas entre los dos algoritmos (--). La significancia se determinó con una confiabilidad de 95%. Las celdas sombreadas corresponden al algoritmo que obtuvo el mejor resultado en calidad o eficiencia.

Tabla 6.2.5**:** Configuración del tipo de mutación.

Instancia	*Mutación uniforme*	*Mutación heurística*	*p-value*
5n240*m*	$76.027_{0.094}{}^{0.139}\downarrow$	$77.32_{0.090}{}^{0.217}$	0.02202
5*n*400*m*	$69.697_{0.253}{}^{0.377}$--	$70.63_{0.193}{}^{0.609}$	0.13888
50*n*240*m*	$536.898_{3.862}{}^{7.808}\uparrow$	$477.41_{6.328}{}^{30.143}$	0.00001

6.3. Configuración de la población inicial

La Tabla 6.3 muestra los resultados obtenidos con cada grupo de 30 instancias. Para cada algoritmo se muestra el promedio del valor objetivo de la mejor solución encontrada y del tiempo de solución. En las celdas de FirstPlus(FO) y FirstPlus(Time) se indica si hay diferencias significativas a favor de la heurística FirstPlus (↓) o a favor de la heurística FirstBest (↑) y si no hay diferencias significativas entre los dos algoritmos (--). La significancia se determinó con una confiabilidad de 95%. Las celdas sombreadas corresponden al algoritmo que obtuvo el mejor resultado en calidad o eficiencia.

Tabla 6.3: Desempeño comparativo de las heurísticas.

Instancia	FirstPlus(FO)	FirstBest(FO)	FirstPlus(Time)	FirstBest(Time)
3*n*20*m*	116.917↑	71.786	4.67E-05 --	0.00036
4*n*20*m*	147.452↑	92.638	0.00004 --	0.00012
5*n*20*m*	185.764↑	119.297	5.34E-05↓	0.00013
5*n*240*m*	191.480↑	93.189	0.00014↓	0.00407
5*n*400*m*	178.956↑	89.598	0.00024↓	0.01136
50*n*240*m*	3231.827↑	1750.662	0.00524↓	0.27034
50*n*400*m*	3303.555↑	1686.851	0.00907↓	0.69741
100*n*240*m*	6180.628↑	3426.171	0.02053↓	1.08407
100*n*400*m*	6246.163↑	3412.699	0.03386↓	2.68293

La Tabla 6.3.1 muestra los resultados obtenidos con cada grupo de 30 instancias. Para cada algoritmo se muestra el promedio del valor objetivo de la mejor solución encontrada y del tiempo de solución. En las celdas de MAFirstBest(FO) y MAFirstBest(Time) se indica si hay diferencias significativas a favor del algoritmo MAFirstBest (↓) o a favor de MAFirstPlus (↑) y si no hay diferencias significativas entre los dos algoritmos (--). La significancia se determinó con una confiabilidad de 95%. Las celdas sombreadas corresponden al algoritmo que obtuvo el mejor resultado en calidad o eficiencia.

Tabla 6.3.1**:** Desempeño comparativo del algoritmo memético generando pop con las heurísticas.

Instancia	MAFirstBest (FO)	MAFirstPlus (FO)	MAFirstBest (Time)	MAFirstPlus (Time)
3*n*20*m*	62.834--	63.257	1.11E-06↓	0.0002
4*n*20*m*	78.734↓	80.977	1.11E-06↓	0.0004
5*n*20*m*	102.09↓	104.665	1.67E-05↓	0.0008
5*n*240*m*	74.995↓	81.824	0.1090↓	0.3830
5*n*400*m*	68.850↓	76.632	0.4408↓	1.0789
50*n*240*m*	824.985↑	499.065	31.8809↓	42.6214
50*n*400*m*	704.896↑	422.664	62.7846↓	74.2709
100*n*240*m*	2029.199↑	955.468	93.5592↓	145.1306
100*n*400*m*	1862.175↑	769.011	260.2112↓	282.0457

La Tabla 6.3.2 muestra los resultados obtenidos con cada grupo de 30 instancias. Para cada algoritmo se muestra el promedio del valor objetivo de la mejor solución encontrada y del tiempo de solución. En las celdas de MA(FO) y MA(Time) se indica si hay diferencias significativas a favor del algoritmo MA (↓) o a favor de MA-Reset (↑) y si no hay diferencias significativas entre los dos algoritmos (--). La significancia se determinó con una confiabilidad de 95%. Las celdas sombreadas corresponden al algoritmo que obtuvo el mejor resultado en calidad o eficiencia.

Tabla 6.3.2: Desempeño comparativo del algoritmo memético regenerando pop.

Instancia	MA (FO)	MA-Reset (FO)	MA (Time)	MA-Reset (Time)
3*n*20*m*	63.257--	62.760	0.0002--	0.0006
4*n*20*m*	80.977--	79.485	0.0004↓	0.0013
5*n*20*m*	104.665--	103.984	0.008↑	0.0018
5*n*240*m*	81.824--	80.69	0.383↑	0.2170
5*n*400*m*	76.632--	75.024	1.079↑	0.609
50*n*240*m*	499.065--	496.556	42.621↑	30.143
50*n*400*m*	422.664↑	417.541	74.271↑	67.110
100*n*240*m*	955.468↑	947.591	145.131--	108.138
100*n*400*m*	769.011--	774.222	282.046--	274.345

6.4. Resultados experimentales

La Tabla 6.4 muestra los resultados obtenidos con cada grupo de 30 instancias después de aplicar la prueba no paramétrica de Wilcoxon. Para cada algoritmo se muestra el promedio y la desviación estándar (subíndice) del valor objetivo y del tiempo de solución. En las celdas de Pcell se indica si hay diferencias significativas a favor de Pcell (↓) o a favor de MAIShOP (↑) y si no hay diferencias significativas entre los dos algoritmos (--). La significancia se determinó con una confiabilidad de 95%. Las celdas sombreadas corresponden al algoritmo que obtuvo el mejor resultado en calidad o eficiencia.

Tabla 6.4: Desempeño comparativo de los algoritmos Pcell vs MAIShOP aplicando Wilcoxon.

Instancia	Pcell (FO)	MAIShOP (FO)	Pcell (Time)	MAIShOP (Time)
3*n*20*m*	$63.03_{0.483}$ ↑	$62.76_{2.57E-14}$	$0.00049_{0.0006}$ ↑	$1.2222E-05_{6.21E-05}$
4*n*20*m*	$78.83_{0.40}$ ↑	$78.73_{2.45E-14}$	$0.00185_{0.0012}$ ↑	$6.6667E-06_{4.26E-05}$

5*n*20*m*	$102.56_{0.80}$ --	$102.09_{0.0043}$	$0.00273_{0.0014}$ ↑	$0.00016_{0.0002}$
5*n*240*m*	$75.43_{0.77}$ ↓	$76.32_{1.67}$	$0.27360_{0.0957}$ ↑	$0.0669_{0.0627}$
5*n*400*m*	$69.98_{1.72}$ --	$70.55_{1.68}$	$0.79002_{0.2882}$ ↑	$0.2425_{0.2139}$
50*n*240*m*	$1058.56_{103.06}$ ↑	$505.47_{17.60}$	$16.4441_{8.0266}$ ↓	$22.5010_{7.5014}$
50*n*400*m*	$951.08_{119.35}$ ↑	$438.74_{24.09}$	$48.6607_{25.3042}$ ↑	$33.1063_{3.8351}$
100*n*240*m*	$2349.94_{159.48}$ ↑	$993.70_{38.25}$	$73.8573_{39.7564}$ ↑	$50.4910_{5.6348}$
100*n*400*m*	$2213.90_{173.70}$ ↑	$801.51_{35.03}$	$181.9521_{95.3004}$ --	$176.1110_{41.5457}$
Promedio global	773.701	347.763	35.776	31.391

Como se puede observar en la Tabla 6.4 el estudio comparativo del rendimiento del algoritmo propuesto con la mejor configuración frente al mejor algoritmo del estado del arte (los parámetros de configuración del algoritmo Pcell, pueden consultarse en el capítulo 4 de este trabajo de investigación). Los resultados de los experimentos realizados muestran que el algoritmo propuesto supera al algoritmo del estado del arte en calidad y eficiencia. En calidad mejora significativamente en 8/9 conjuntos y en eficiencia mejora significativamente en 8/9 conjuntos.

La Tabla 6.5 muestra los resultados obtenidos con cada grupo de 30 instancias después de aplicar la prueba no paramétrica de Friedman. Para cada algoritmo se muestra el promedio y el ranking de Friedman (subíndice) del valor objetivo y del tiempo de solución. En las celdas de Pcell se indica si hay diferencias significativas a favor de Pcell (↓) o a favor de MAIShOP (↑) y si no hay diferencias significativas entre los dos algoritmos (--). La significancia se determinó con una confiabilidad de 95%. Las celdas sombreadas corresponden al algoritmo que obtuvo el mejor resultado en calidad o eficiencia.

Tabla 6.5: Desempeño comparativo de los algoritmos Pcell vs MAIShOP aplicando Friedman.

Instancia	Pcell (FO)	MAIShOP (FO)	Pcell (Time)	MAIShOP (Time)
3*n*20*m*	$63.03_{49.5}$ --	$62.76_{40.5}$	0.00049_{51} ↑	$1.2222E\text{-}05_{39}$
4*n*20*m*	78.83_{48} --	78.73_{42}	0.00185_{58} ↑	$6.6667E\text{-}06_{32}$
5*n*20*m*	102.56_{49} --	102.09_{41}	$0.00273_{58.5}$ ↑	$0.00016_{31.5}$
5*n*240*m*	$75.43_{53.5}$ ↓	$76.32_{36.5}$	0.27360_{57} ↑	0.0669_{33}
5*n*400*m*	$69.98_{56.5}$ ↓	$70.55_{33.5}$	0.79002_{58} ↑	0.2425_{32}
50*n*240*m*	1058.56_{60} ↑	505.47_{30}	16.4441_{33} ↓	22.5010_{57}
50*n*400*m*	951.08_{60} ↑	438.74_{30}	48.6607_{58} ↑	33.1063_{32}
100*n*240*m*	2349.94_{60} ↑	993.70_{30}	73.8573_{59} ↑	50.4910_{31}

100*n*400*m*	$2213.90_{60}\uparrow$	801.51_{30}	181.9521_{46} --	176.1110_{44}
Promedio global	773.701	347.763	35.776	31.391

Como se puede observar en la Tabla 6.5 el estudio comparativo del rendimiento del algoritmo propuesto con la mejor configuración frente al mejor algoritmo del estado del arte (los parámetros de configuración del algoritmo Pcell, pueden consultarse en el capítulo 4 de este trabajo de investigación). Los resultados de los experimentos realizados muestran que el algoritmo propuesto supera al algoritmo del estado del arte en calidad y eficiencia. En calidad mejora significativamente en 7/9 conjuntos y en eficiencia mejora significativamente en 8/9 conjuntos.

Capítulo 7. Conclusiones y trabajos futuros

7.1. Cumplimiento de objetivos

En el proyecto de tesis se cumplieron satisfactoriamente todos los objetivos planteados. En esta sección se describen las contribuciones principales, los productos científicos generados y los trabajos futuros que se han identificado.

7.1.1 Objetivo general

Desarrollar un algoritmo memético para el problema de ventas por internet considerando costos de envío, con un rendimiento estadísticamente similar o superior a los del estado del arte. (Ver capítulos 5 y 6).

7.1.2 Objetivos específicos

- Definición de tipo y tamaño de instancias. (Ver capítulo 6).

- Implementar el mejor algoritmo del estado del arte de IShOP. (Ver capítulo 4).
- Desarrollar nuevas heurísticas y búsquedas locales. (Ver capítulo 5).
- Desarrollar el algoritmo memético. (Ver capítulo 5).
- Realizar una evaluación experimental del desempeño del algoritmo memético con respecto al mejor del estado del arte. (Ver capítulo 6).

7.2. Contribuciones principales

Las contribuciones principales de esta investigación son las siguientes:

1. Un algoritmo memético novedoso para el problema de Optimización de Compras por Internet con Costos de Envío (Ver sección 5.9 Algoritmo memético (MAIShOP) del capítulo 5).
2. Una nueva representación vectorial de las soluciones candidatas que permite reducir la complejidad del temporal del cálculo de la función objetivo de $O(n^2)$ a $O(n)$ (Ver sección 5.3.1 Representación de la solución y reducción de la complejidad del cálculo de la función objetivo del capítulo 5).
3. Dos nuevas heurísticas para crear individuos mejorados (Ver secciones 5.4.1 Heurística FirstPlus y 5.7.2.1 Heurística FirstBest del capítulo 5).
4. Dos nuevos operadores de mutación (Ver secciones 5.7.1 Mutación uniforme y 5.7.2 Mutación heurística del capítulo 5).
5. Y después de cada generación, en la población solo queda la mejor solución y el resto de los individuos se regeneran aleatoriamente (Ver sección 5.9 Algoritmo memético (MAIShOP) del capítulo 5).

7.3. Productos científicos

Los productos científicos desarrollador a partir de la investigación son los siguientes:

- Artículo: *Nuevas Heurísticas para el problema de ventas por internet con costos de envío (IShOP)*. Presentado en el 12° Congreso Internacional "La Investigación Científica y Tecnológica impulsando la creatividad para innovar" de la Academia Mexicana Multidisciplinaria A.C. (Agosto 2020).

- Capítulo de libro: *Algoritmo Memético para el Problema de Ventas por Internet con Costos de Envío (MAIShOP)*. Presentado en el Encuentro Nacional de Computación ENC 2020. (Agosto 2020).
- Artículo: *Algoritmo Memético para el Problema de Ventas por Internet con Costos de Envío (MAIShOP)*. Publicado en la revista Komputer Sapiens, edición del año 13, Volumen 1. (Marzo 2021).
- Artículo: *Optimization of the Internet Shopping Problem with Shipping Costs*.
- Artículo: *Memetic Algorithm for Internet Shopping Optimization Problem with Shipping Costs (MAIShOP)*. Pendiente de ser enviado a la revista Information Sciences Elsevier.

7.4. Trabajos futuros

En el desarrollo de esta investigación se identificaron las siguientes líneas de investigación:

- Desarrollar un estudio de la solución del problema con diferentes metaheurísticas.
- Desarrollar nuevos operadores de cruza y mutación para el problema de IShOP con costos de envío.
- Introducir mecanismos de ajuste automático de los parámetros de control del algoritmo.

Bibliografía

Blazewicz, J., Bouvry, P., Kovalyov, M. Y., & Musial, J. (2014a). Internet shopping with price sensitive discounts. *4OR-A Quarterly Journal of Operations Research*, *12*(1): 35-48.

Blazewicz, J., Bouvry, P., Kovalyov, M. Y., & Musial, J. (2014b). Erratum to: Internet shopping with price sensitive discounts. *4OR-A Quarterly Journal of Operations Research*, *12*(4), 403-406.

Błażewicz, J., & Musiał, J. (2011). E-commerce evaluation–multi-item Internet shopping. Optimization and heuristic algorithms, in B. Hu et al. (Eds), *Operations Research Proceedings 2010*, Springer-Verlag, Berlin, pp. 149-154.

Blazewicz, J., Cheriere, N., Dutot, P. F., Musial, J., & Trystram, D. (2016). Novel dual discounting functions for the Internet shopping optimization problem: New algorithms. *Journal of Scheduling*, *19*(3), 245-255.

Błażewicz, J., Kovalyov, M. Y., Musiał, J., Urbański, A. P., & Wojciechowski, A. (2010). Internet shopping optimization problem. *International Journal of Applied Mathematics & Computer Science*, *20*(2): 385-390, DOI: 10.2478/v10006-010-0028-0.

Blum, L. (2004). Computing over the reals: Where turing meets newton. Notices of the AMS, 51(9):1024-1034.

Blum, L., Cucker, F., Shub, M., and Smale, S. (1998). Complexity and real computation. Springer Science & Business Media.

Blum, L., Shub, M., and Smale, S. (1990). A Theory of Computation and Complexity over the real numbers. International Computer Science Institute.

Blum, L., Shub, M., Smale, S., et al. (1989). On a theory of computation and complexity over the real numbers: np-completeness, recursive functions and universal machines. Bulletin (New Series) of the American Mathematical Society, 21(1):1-46.

Bökler, F. (2017). The Multiobjective Shortest Path Problem Is NP-Hard, or Is It?, pages 77-87. Springer International Publishing, Cham.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC'71, pages 151-158, New York, NY, USA. ACM.

Burke, E.,Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology, in F. Glover nd G.A. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, Vol. 57, Springer-Verlag, Berlín, pp. 457–474.

Cavalier M., T., & James P., I. (1994). *Linear Programming*. Prentice Hall.

D'Aniello, G., Gaeta, M., Loia, V., & Orciuoli, F. (2015). An ami-based software architecture enabling evolutionary computation in blended commerce: the shopping plan application. *Mobile Information Systems, 2015*.

D'Aniello, G., Orciuoli, F., Parente, M., & Vitiello, A. (2014, September). Enhancing an AmI-based framework for u-commerce by applying memetic algorithms to plan shopping. In *2014 International Conference on Intelligent Networking and Collaborative Systems* (pp. 169-175). IEEE.

Danziger, P. (2010). Big o notation. Source internet: http://www.scs. ryerson. ca/~ mth110/Handouts/PD/bigO. pdf, Retrieve: April.

Duarte Muñoz, A., Pantrigo Fernández, J. J., M. G. C. (2010). Metaheurísticas. Universidad Rey Juan Carlos.

Duarte, A., Laguna, M., and Martí, R. (2006). Tabu search for the linear ordering problem with cumulative costs. Computational Optimization and Applications, pages 1-19.

Fleszar, K., Glaßer, C., Lipp, F., Reitwießner, C., & Witek, M. (2011, April). The Complexity of Solving Multiobjective Optimization Problems and its Relation to Multivalued Functions. In *Electronic Colloquium on Computational Complexity (ECCC)* (Vol. 18, No. 53, p. 165).

Garey, M. R., Johnson, D. S., and Others (1979). Computers and Intractability: A Guide to the Theory of NP-completeness. WH freeman San Francisco.

Glaßer, C., Reitwießner, C., Schmitz, H., & Witek, M. (2010, June). Approximability and hardness in multi-objective optimization. In *Conference on Computability in Europe* (pp. 180-189). Springer, Berlin, Heidelberg.

Garey, M. R. and Johnson, D. S. (1990). Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, NY.

Gaeta, M., Loia, V., Orciuoli, F., & Parmentola, M. (2013, May). A genetic approach to plan shopping in the ami-based blended commerce. In *2013 IEEE International Symposium on Industrial Electronics* (pp. 1-6). IEEE.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. In *Bulletin of the American Mathematical Society* (pp. 275–278). Springer.

Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66-73.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press, And Arbor, Michigan*.

Józefczyk, J., & Ławrynowicz, M. (2018). Heuristic algorithms for the Internet shopping optimization problem with price sensitivity discounts. *Kybernetes*, *47*(4), 831-852.

Kelly, J. P. and Osman, I. H. (1996). Meta-Heuristics: Theory and Applications. Kluwer Academic Publishers Norwell, MA, USA.

Laguna, M., and Delgado, C. (2007). *Introducción a los metaheurísticos*. Monográfico 3.

López Locés, M. C., Musial, J., Pecero, J. E., Fraire-Huacuja, H. J., Blazewicz, J., & Bouvry, P. (2016). Exact and heuristic approaches to solve the Internet shopping optimization problem with delivery costs. *International Journal of Applied Mathematics and Computer Science*, *26*(2), 391-406.

López Locés, M.C. (2017). Métodos de Optimización de Problemas NP-Duros Basados en Algoritmos de Procesamiento Celular (tesis doctoral). TeCNM-Instituto Tecnológico de Tijuana, Baja California, México.

López Locés, M. C., Rege, K., Pecero, J. E., Bouvry, P., & Huacuja, H. J. F. (2015). Trajectory metaheuristics for the internet shopping optimization problem. In *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization* (pp. 527-536). Springer, Cham.

Marsza[a]kowski, J. Budgeted Internet Shopping Optimization Problem.

Mitsuo Gen and Runwei Cheng. (2000). Genetic Algorithms and Engineering Optimization. *Wiley Series in Engineering Design and Automation*. John Wiley and Sons, New York.

Muñoz, D., A., P. F. J., & Gallego Carrillo, M. (2007). *Metaheurısticas*. S.L. - DYKINSON.

Musial, J., & López Locés, M. C. (2017). Trustworthy online shopping with price impact. *Foundations of Computing and Decision Sciences*, *42*(2), 121-136.

Musial, J., Pecero, J. E., López Locés, M. C., Fraire-Huacuja, H. J., Bouvry, P., & Blazewicz, J. (2016). Algorithms solving the Internet shopping optimization problem with price discounts. *Bulletin of the Polish Academy of Sciences Technical Sciences*, *64*(3), 505-516.

Musial, J., Pecero, J., Dorronsoro, B., & Blazewicz, J. (2014). Internet Shopping Optimization Project (IShOP). European IST Projects, 16.

Musial, J., Pecero, J. E., López Locés, M. C., Fraire, H. J., Bouvry, P., & Blazewicz, J. (2014, August). How to efficiently solve Internet Shopping Optimization Problem with price sensitive discounts?. In *2014 11th International Conference on e-Business (ICE-B)* (pp. 209-215). IEEE.

Musial, J. (2012) Applications of Combinatorial Optimization for Online Shopping. NAKOM, Poznán.

Musial, J., Pecero, J., Dorronsoro, B., & Blazewicz, J. Internet Shopping Optimization Project (IShOP). *European IST Projects*, 16.

Nicholson, T. A. J. (2007). Optimization in industry: Optimization techniques. Aldine.

Orciuoli, F., Parente, M., & Vitiello, A. (2016). Solving the shopping plan problem through bio-inspired approaches. *Soft Computing*, *20*(5), 2077-2089.

P. Galinier, Z. Boujbel, M.C. Fernandes, An efficient memetic algorithm for graph partitioning problem, Ann. Oper. Res. 191 (2011) 1–22.

P. Moscato, C. Cotta, Memetic algorithm, in: T.F. Gonzalez (Ed.), Handbook of Approximation Algorithms and Metaheuristics, Chapman & Hall/CRC, 2007, pp. 27.1–27.12.

P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Material Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program, C3P Report 826, 1989.

Polya, G. (1971). How to solve it: A new aspect of mathematical method. Princeton University Press Princeton, NJ.

R. Bansal, K. Srivastava, A memetic algorithm for the cyclic antibandwidth maximization problem, soft computing –a fusion of foundations, Method. Appl. 5 (2011) 397–512.

R. Dawkins. The Selfish Gene. Clarendon Press, Oxford, UK, 1976.

R. Shang, J. Wang, L. Jiao, Y. Wang, An improved decomposition-based memetic algorithm for multi-objective capacitated arc routing problem, Appl. Soft Comput. 19 (2014) 343–361.

Reeves, C. R. (1993). Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc. New York, NY, USA.

S.J. Sadjadi, R. Soltani, A. Eskandarpour, Location based treatment activities for end of life products network design under uncertainty by a robust multi-objective memetic-based heuristic approach, Appl. Soft Comput. 23 (2014) 215–226.

S. Yang, K. Cheng, M. Wang D. Xie , L. Jiao , High resolution range-reflectivity estimation of radar targets via compressive sampling and memetic algorithm, Inf. Sci. 252 (2013) 144–156.

Santiago Pineda, A.A. (2018). Métodos de Optimización Multiobjetivo (tesis doctoral). TeCNM-Instituto Tecnológico de Tijuana, Baja California, México.

Sadollah, A., Gao, K., Barzegar, A., & Su, R. (2016, November). Improved model of combinatorial Internet shopping optimization problem using evolutionary algorithms. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)* (pp. 1-5). IEEE.

Sayyaadi, H., Sadollah, A., Yadav, A., & Yadav, N. (2018). Stability and iterative convergence of water cycle algorithm for computationally expensive and combinatorial Internet shopping optimisation problems. *Journal of Experimental & Theoretical Artificial Intelligence*, 1-21.

Schiavinotto, T. and Stützle, T. (2004). The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4): 367-402.

Sipser, M. (2006). Introduction to the Theory of Computation, 2nd Edition. Course Technology PTR.

Sipper, M. (1999). The emergence of cellular computing, Computer 32(7): 18–26.

Sörensen, K., & Glover, F. (2013). Metaheuristics. *Encyclopedia of Operations Research and Management Science*, *62*, 960–970.

Terán-Villanueva, J.D., Huacuja, H.J.F., Valadez, J.M.C., Rangel, R.P., Soberanes, H.J.P. and Flores, J.A.M. (2015). A heterogeneous celular processing algorithm for minimizing the power consumption in wireless comunications systems, Computational Optimization and Applications 62(3): 787-814.

Umbakar, A. J., and Sheth, P. D. (2015). Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1).

Verma, S., Sinha, A., Kumar, P., & Maitin, A. (2020, March). *Optimizing Online Shopping using Genetic Algorithm*. In *2020 3rd International Conference on Information and Computer Technologies (ICICT)* (pp. 271-275). IEEE.

Wojciechowski, A., and Musial, J. (2009). A customer assistance system: Optimizing basket cost, *Foundations of Computing and Decision Sciences* 34(1), 59-69.

Wojciechowski, A., and Musial, J. (2010). Towards optimal multi-item shopping basket management: Heuristic approach, in R. Meersman et al. (Eds), On the Move to Meaningful Internet Systems: *OTM 2010 Workshops, Lecture Notes in Computer Science*, Vol. 6428, Springer-Verlag, Berlin, pp. 349-357.

Z. Zhu , J. Xiao, S. He , Z. Ji , Y. Sun , A multi-objective memetic algorithm based on locality-sensitive hashing for one-to-many-to-one dynamic pick- up-and-delivery problem, Inf. Sci. 329 (2016) 73–89.