INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

TESIS

ALGORITMO DE BÚSQUEDA ARMÓNICA PARA EL PROBLEMA DE COMPRAS POR INTERNET CON COSTOS DE ENVÍO

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

Presenta
Ing. Eduardo Villegas Huerta
G21073007
1154004

Director de Tesis
Dr. Héctor Joaquín Fraire Huacuja

Co-director de Tesis
Dr. Fausto Antonio Balderas Jaramillo

Cd. Madero, Tamaulipas **septiembre 2023**

Ciudad Madero, Tamaulipas, 07/junio/2023

Oficio No.: U.076/2023
Asunto: Autorización de impresión de tesis

C. EDUARDO VILLEGAS HUERTA
No. DE CONTROL G21073007
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 DE BÚSQUEDA ARMÓNICA PARA EL PROBLEMA DE COMPRAS POR INTERNET CON COSTOS DE ENVÍO"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTA:	DRA.	LAURA CRUZ REYES
SECRETARIO:	DR.	NELSON RANGEL VALDEZ
VOCAL:	DR.	FAUSTO ANTONIO BALDERAS JARAMILLO
SUPLENTE:	DRA.	CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
DIRECTOR DE TESIS:	DR.	HÉCTOR JOAQUÍN FRAIRE HUACUJA
CO-DIRECTOR:	DR.	FAUSTO ANTONIO BALDERAS JARAMILLO

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

ccp. Archivo
MACG 'MLMR'

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO INVESTIGACIÓN

Por medio de la presente se hace constar que el C. EDUARDO VILLEGAS HUERTA. con número de control G21073007 Estudiante de la MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN no tiene adeudo en el departamento a mi cargo:

CENTRO DE CÓMPUTO DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

FECHA: 07 de junio de 2023

FIRMA:

SELLO:

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

COORDINACIÓN DE LA MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

FECHA: 07 de junio de 2023

FIRMA: Ma Lucila Morales Rdz
DRA. MARÍA LUCILA MORALES RODRIGUEZ

SELLO:

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

JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

FECHA: 07 de junio de 2023

FIRMA:
DR. MARCO ANTONIO CORONEL GARCÍA

SELLO:

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

Resumen

Este proyecto de investigación aborda el problema de Compras por Internet con costos de envío. En el estado del arte, solo se reporta que la mejor solución del problema es un algoritmo memético (MAIShOP). En este proyecto se propone un nuevo algoritmo de búsqueda armónica que usa un mecanismo de ajuste de parámetros que le permite ajustar dinámicamente la búsqueda de nuevas soluciones candidatas para tratar de avanzar rápidamente hacia la solución óptima, evitando el estancamiento. Para validar los resultados se realizaron una serie de experimentos computacionales con instancias del estado del arte incluyendo un estudio comparativo del desempeño del algoritmo propuesto contra el mejor algoritmo del estado del arte MAIShOP. En los experimentos computacionales, se utiliza un amplio conjunto de instancias desde chicas, medianas y grandes y los resultados muestran una clara superioridad del algoritmo propuesto. Para validar los resultados, se aplicó la prueba no paramétrica de Wilcoxon, en la que se verificó la significancia de las diferencias observadas tanto en valor como en tiempo.

Agradecimientos

Quiero dar mi agradecimiento a mi asesor en esta investigación, el Dr. Héctor Joaquín Fraire Huacuja, gracias por todo su apoyo que me dio de verdad que tuvo una paciencia y atenciones conmigo, y apoyarme a superar muchas dificultades que se atravesaron en mi camino. Muchas gracias por su hospitalidad y por todo el tiempo que compartimos juntos, de verdad le agradezco muchísimo fue un gran apoyo para poder lograr mis metas a lo largo de mi estadía.

También quiero agradecer a mi comité tutorial: la Dra. Laura Cruz Reyes, Dra. Claudia Guadalupe Gómez Santillán, y el Dr. Nelson Rangel muchísimas gracias por todos los consejos que me dieron, de verdad les agradezco todo su apoyo para lograr completar mis metas y poder entregar un trabajo más completo.

Al igual que quiero agradecer a mis compañeros con los que estuve conviviendo y me aconsejaron para no tener dificultades en la organización y calidad de trabajo. De verdad siempre estaré muy agradecido con ellos por haberme apoyado y haberme hecho sentir muy bien con su hospitalidad.

Y por último quiero agradecer a mi familia que me apoyo bastante la verdad sin su apoyo no lo hubiera podido lograr siempre me apoyaron y animaron a continuar para poder seguir con mi maestría y no dejar que me rindiera con mis metas hasta lograr cumplirlas, siempre estaré muy orgulloso de ellos por ser mis guías y apoyarme siempre.

Contenido

Capítulo 1. Introducción

En 2012, se dio inicio a este proyecto de investigación en el ITCM con el propósito de desarrollar enfoques de optimización innovadores para resolver el desafío de las compras por internet con costos de envío. El propósito de esta tesis es desarrollar un algoritmo basado en armonías musicales que supere o sea comparable en términos estadísticos al algoritmo más destacado en el estado del arte, el cual fue presentado en la tesis de Maestría de Miguel Ángel García Morales titulada "Algoritmo memético para el problema de compras por internet con costos de envío" [García, 2021].

1.1. Problema que se estudia

La problemática de mejora para compras por internet se analiza de la siguiente forma:

Una persona quiere adquirir en línea un grupo de n productos N, en las que se logren obtener de un grupo de m tiendas accesibles M. Entonces, el grupo N_i almacena cada producto accesible de la tienda i, cada indicador $j \in N_i$ obtiene un valor de c_{ij} y a su vez también un valor de envío d_i. El valor de envío se obtiene únicamente cuando un conjunto de productos es adquirido de la tienda i. La problemática de las compras en línea con un valor de envío consta de reducir el valor total de cada compra del conjunto de productos N dado este valor de cada producto, agregando también el valor del envío de cada uno.

Tabla 1.2: Tabla de notación [García, 2021].

Símbolo	Descripción
m	Grupo disponible de tiendas
n	Grupo disponible de productos
M	Cantidad de tiendas, $\|M\|$
N	Cantidad de productos, $\|N\|$
I	Señalador de tienda
J	Señalador de producto
N_i	Inventario de artículos disponibles en una tienda específica i.
d_i	Valor de envío del conjunto de productos de la tienda i
c_{ij}	Valor del conjunto de productos j de la tienda i
x_{ij}	Una variable booleana que determina si el producto j está disponible en la tienda i o no.

Explícitamente la problemática es encontrar una distribución de los productos a adquirir en cada una de estas tiendas $X = (X_1, \dots, X_m)$, de esta forma se dice que $X_i \subseteq N_i$ y se compren cada uno de los productos $\cup_{i=1}^{m} X_i = N$ y reduzca el valor final. La función objetivo para esta problemática se define de la siguiente manera:

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

1.2. Objetivos de la tesis

1.2.1. Objetivo general

Desarrollar un enfoque novedoso utilizando búsqueda armónica para abordar el problema de compras por internet con costos de envío, con el objetivo de lograr una eficiencia comparable o superior a la del algoritmo más avanzado en el campo de estudio.

1.2.1. Objetivos específicos

- Análisis exhaustivo de la literatura existente
- Elaboración de un marco teórico sólido
- Diseño e implementación de la estructura de armonías para abordar el problema IShOP
- Diseño e implementación de una memoria de armonías eficiente
- Diseño e implementación de un método de improvisación de armonías efectivo
- Diseño e implementación de un método para ajustar el tono de una armonía
- Evaluación experimental del rendimiento del algoritmo en pruebas y análisis de resultados obtenidos

1.3. Alcances y delimitaciones

1.3.1. Alcances

1. Se empleará la versión 8 del lenguaje de programación Java.
2. Se llevará a cabo una comparación entre el rendimiento del algoritmo de búsqueda armónica y el mejor algoritmo actual para esta problemática, para evaluar su desempeño relativo.

1.4. Hipótesis de Investigación

Se puede desarrollar y aplicar un algoritmo de búsqueda armónica capaz de resolver el problema IShOP con un rendimiento comparable o superior al del algoritmo memético MAIShOP, según se evidencia estadísticamente.

1.5. Justificación y beneficio

Hoy en día, existe la creciente demanda de diversas plataformas digitales de comercio electrónico altamente útiles, los cuales han adquirido elementos fundamentales en la planificación comercial de varias empresas, en cual se muestra en la Figura 1.1. Este ámbito similar a considerar es el de las compras en línea o comercio electrónico. Dado que cada uno de los precios de cada producto en los diversos portales de las tiendas en línea suelen ser más bajos que en las tiendas físicas, donde considerando que varios proveedores dan a ofrecer el mismo producto, con lo cual existe una gran demanda de plataformas digitales de compras por internet altamente eficientes. Aquellas plataformas que no cumplen con este nivel de eficiencia pueden ocasionar que los clientes opten por cambiar de proveedor de servicios de comercio electrónico.

La problemática de las compras por internet que se aborda en esta tesis ya ha sido estudiada en otros trabajos previos citados en la literatura. En esta tesis, se propone la adaptación de un método metaheurístico basado en la búsqueda armónica, la cual implica que aumentara la exploración que hay a través de vecindarios. Durante este proceso, se tiene en consideración la incorporación de técnicas de búsqueda local y heurísticas disponibles en la literatura, con el objetivo de mejorar el espacio de soluciones y potenciar la capacidad de búsqueda del mejor algoritmo actualmente que ha sido reportado para esta problemática de la actualidad.

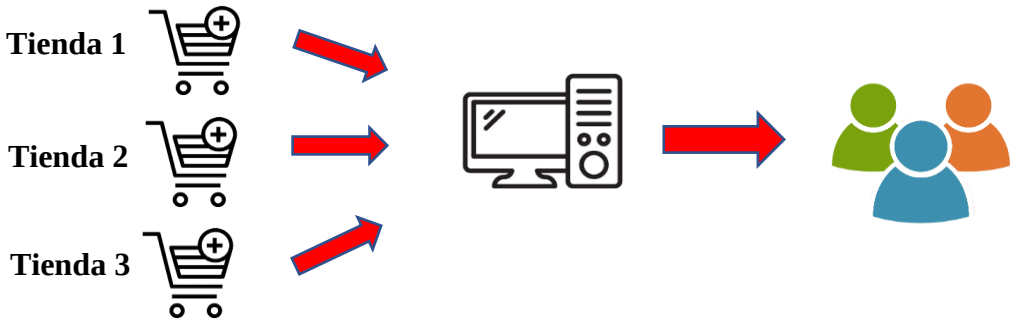

Figura 1.1. Diferentes compras en línea realizadas por una persona.

Capítulo 2. Marco teórico

En esta seccion se presentarán los principios fundamentales relacionados con la optimización. En el cual estos problemas de optimización se centran en localizar un mejor resultado dentro de un grupo de posibles resultados que cumplen ciertas restricciones. También se explorará la complejidad computacional, la cual analiza la dificultad de diversos problemas [Garey, M. R et al., 1990]. Además, se discutirán los conceptos básicos de los algoritmos de búsqueda armónica y el algoritmo memético, teniendo así el propósito de poder desarrollar una comprensión de su funcionamiento y aplicación en los métodos de optimización.

2.1. Notación O grande

Se puede llegar a entender que la notación O grande es una notación matemática que se utiliza para describir la complejidad temporal o espacial de un algoritmo. Esta una forma de medir la eficiencia de un algoritmo al proporcionar un límite superior en la tasa de crecimiento del algoritmo a medida que aumenta el tamaño de la entrada.

En términos más simples, la notación O grande nos dice cómo se escalará el rendimiento de un algoritmo con el tamaño de los datos de entrada. Nos ayuda a comparar diferentes algoritmos y elegir el que será más eficiente para un problema en particular.

La notación está representada por la letra "*O*" seguida de una función que describe la tasa de crecimiento del algoritmo. Por ejemplo, $O(1)$ significa que el algoritmo tardará una cantidad de tiempo constante en ejecutarse, independientemente del tamaño de entrada. $O(n)$ significa que la complejidad temporal del algoritmo crecerá linealmente con el tamaño de la entrada, mientras que $O(n^2)$ significa que la complejidad temporal del algoritmo crecerá cuadráticamente con el tamaño de la entrada. [Danziger, 2010].

Es importante tener en cuenta que la notación O grande proporciona un límite superior en la tasa de crecimiento de un algoritmo, por lo que es posible que un algoritmo pueda funcionar mejor de lo que sugiere su notación O grande. Sin embargo, es una herramienta

útil para analizar y comparar la eficiencia de diferentes algoritmos. Cabe mencionar que existen aún más notaciones O grande las cuales se representan en la Tabla 2.1.

Tabla 2.1. Órdenes de complejidad comunes de menor a mayor

Notación	Orden
$O(1)$	Orden constante
$O(log\ n)$	Orden logarítmico
$O(n)$	Orden lineal
$O(n^2)$	Orden cuadrático
$O(n^a)$	Orden polinomial (a>2)
$O(a^n)$	Orden exponencial(a>2)
$O(n!)$	Orden factorial

2.2. Complejidad computacional

La complejidad computacional es el estudio de los recursos, como el tiempo y la memoria, necesarios para resolver un problema computacional. Se ocupa de comprender cómo el tamaño de la entrada afecta la eficiencia de un algoritmo o programa de computadora que resuelve el problema.

El campo de la complejidad computacional es interdisciplinario y se basa en la informática, las matemáticas y la ingeniería. Incluye el estudio de varias clases de problemas, como problemas de decisión, problemas de optimización y problemas de búsqueda, y sus correspondientes algoritmos. Uno de los primeros y más influyentes trabajos en el campo de la complejidad computacional es el artículo “On Computable Numbers, with an Application to the Entscheidungsproblem” [Turing, 1937]. En este artículo, introdujo el concepto de una máquina de Turing, un modelo teórico de una máquina que puede realizar cualquier cálculo que pueda ser realizado por una computadora física.

Una de las medidas más conocidas de complejidad computacional es la notación O grande, que se describe anteriormente. La notación proporciona una forma de expresar el límite superior del tiempo de ejecución o el espacio requerido por un algoritmo a medida que crece el tamaño de la entrada.

Otras medidas de la complejidad computacional incluyen la complejidad del espacio, que es la cantidad de memoria requerida para resolver un problema, y la complejidad de la comunicación, que es la cantidad de bits de comunicación necesarios entre dos partes para resolver un problema.

2.2.1. Problema de decisión

Un problema de decisión es un problema computacional que involucra decidir si una entrada dada satisface una propiedad o criterio específico. Es un concepto fundamental en informática y se utiliza a menudo como base para el estudio de la complejidad computacional.

Un ejemplo bien conocido de un problema de decisión es el "problema del agente viajero (TSP)" [Menger,1954], que pregunta si existe una ruta que visita un conjunto de ciudades y regresa a la ciudad de origen, de modo que la distancia total recorrida sea menor que un valor dado.

Los problemas de decisión son importantes porque proporcionan un marco para analizar la complejidad de los algoritmos y para comprender los límites de la computación. Al estudiar los problemas de decisión y su complejidad, podemos obtener información sobre la eficiencia y escalabilidad de los algoritmos y sistemas.

2.2.2. Clase P

Esta categoría de problemas de decisión se refiere a aquellos que pueden ser resueltos por un algoritmo en un tiempo polinómico. La clase P es una propiedad común a todos los modelos de cálculo que son polinomial mente equivalentes a una máquina de Turing determinista. Representa aproximadamente la clase de problemas que pueden ser resueltos de manera realista en una computadora y se considera una clase matemática sólida [Sipser, 2006]. Estos problemas fueron inicialmente abordados utilizando una máquina de Turing

determinista de una cinta, que adopta un enfoque para resolverlos en tiempo polinomial., estos conceptos de problemas son mencionados en el trabajo “The complexity of theorem-proving procedures” [Cook, 1971]. También se a demostrado que este tipo de problemas tienen una equivalencia a los NP ya que pueden ser resueltos en tiempo polinomial gracias a la máquina de Turing.

2.2.3. Clase NP

La categoría de NP se refiere al grupo de problemas de decisión en las que se verifican en tiempo polinomial por un algoritmo no determinista. Esto implica que, en un tiempo polinomial, es posible generar un resultado optimo utilizando un algoritmo no determinista y luego checar si este resultado es válido utilizando un algoritmo determinista [Sipser, 2006]. Lo más relevante de esta clase de problemas es su capacidad para abarcar problemas de búsqueda y optimización, donde se busca determinar la existencia o encontrar soluciones óptimas para un problema específico.

Un ejemplo para este tipo de problemáticas es el problema del viajante (también conocido como "problema del agente viajero"), en el cual se busca determinar si se puede encontrar una ruta que sea óptima y logre pasar por todos los nodos dentro de un grafo dado. Otro ejemplo, es el problema de satisfactibilidad booleana, en el cual este busca determinar si una fórmula lógica proposicional puede ser satisfecha por un conjunto determinado de valores booleanos asignados a las variables correspondientes. [Cook, 1971].

2.2.4. Problema NP Completo

La clase completa NP (Tiempo polinomial no determinista) es una clase compleja aplicada en problemáticas de decisión en las cuales son resueltas mediante un algoritmo no determinista en tiempo polinomial. Es un subconjunto de la clase de mayor complejidad de problemas NP, que incluye problemáticas en la toma de decisión con las que se puedan analizar en un tiempo polinomial.

Se dice que un problema está en la clase NP completo si existe un algoritmo no determinista que puede resolverlo en tiempo polinomial. La clase NP completa es importante porque incluye muchos problemas que son difíciles de resolver usando algoritmos clásicos, como el problema del viajante de comercio y el problema de satisfactibilidad booleano. Estos problemas se conocen como problemas NP-completos, que son los problemas más difíciles de la clase NP [Garey, M. R. et al.,1990].

2.2.5. Problema de Optimización

En este tipo de problemas son en las que se necesita encontrar la mejor solución entre todas las soluciones posibles que satisfagan ciertas restricciones. El objetivo es maximizar o minimizar una cierta función objetivo que mide la calidad de la solución.

Los problemas de optimización se pueden encontrar en muchas áreas, como la ingeniería, la economía y las finanzas. Este es uno de los ejemplos de problemas de optimización:

1. Optimización de cartera: La optimización de cartera es un problema de optimización en el que un inversor intenta encontrar la asignación óptima de inversiones a diferentes activos que maximice el rendimiento esperado y minimice el riesgo. Tiene aplicaciones en finanzas y gestión de inversiones [Markowitz, 1952].

2.3. Métodos de optimización

Estos métodos son técnicas utilizadas para encontrar la mejor solución posible a un problema de optimización, y se utilizan para minimizar o maximizar una función objetivo ajustando las variables o parámetros de entrada y aquí es donde la complejidad computacional se aplica para poder seleccionar que método se puede aplicar para poder realizar el problema. Hay varios tipos de métodos de optimización, incluidos los métodos exactos, los métodos heurísticos y los métodos metaheurísticos y en la que si pertenece a la clase P y pueden ser realizados con algún método exacto para garantizar la solución óptima. [Sipser, 2006].

De igual manera también se pueden categorizar ampliamente en dos tipos: métodos deterministas y métodos estocásticos. Los métodos deterministas son algoritmos que utilizan

un enfoque sistemático para encontrar la solución óptima. Los ejemplos de métodos deterministas incluyen:

- Método de Newton: el método de Newton es un algoritmo de búsqueda de raíces que se puede utilizar para la optimización. Implica encontrar iterativamente las raíces de la primera y segunda derivada de la función objetivo.
- Método Simplex: El método Simplex es un algoritmo para problemas de programación lineal. Implica encontrar la solución óptima moviéndose a lo largo de los bordes de un politopo de forma sistemática [Dantzing, 1951].

Los métodos estocásticos son algoritmos que utilizan la aleatorización para encontrar la solución óptima. Ejemplos de métodos estocásticos incluyen:

- Recocido simulado: El recocido simulado es un algoritmo probabilístico que se utiliza para encontrar el mínimo global de una función de costo. Implica ajustar iterativamente los parámetros de una manera que permita que el algoritmo escape de los mínimos locales [Kirkpatrick et al., 1983].
- Algoritmos genéticos: Los algoritmos genéticos son algoritmos de optimización inspirados en el proceso de selección natural. Implican generar iterativamente una población de soluciones y aplicar operadores genéticos como mutación y cruce para generar nuevas soluciones [Holland, 1975].

2.3.1. Métodos exactos

Los métodos exactos son un conjunto de técnicas de optimización que garantizan la solución óptima a un problema de optimización en un tiempo finito. Estos métodos utilizan principios matemáticos y se basan en la enumeración completa de soluciones factibles para encontrar el óptimo global donde uno de los pioneros en utilizar los principios de los métodos exactos fue en el método simplex que fue desarrollado por George Dantzing en 1947 y es actualmente utilizado para resolver problemas de programación lineal [Nash, 2000].

En general, los métodos exactos son una herramienta importante para resolver problemas de optimización que requieren una solución óptima y pueden ser un complemento útil para los métodos heurísticos cuando el tamaño del problema es pequeño o cuando la optimización de la solución es crítica.

2.3.2. Métodos heurísticos

Los métodos heurísticos son técnicas de optimización que utilizan algoritmos aproximados para encontrar soluciones casi óptimas a problemas complejos. Estos son un procedimiento el cual permite resolver problemas a través de un método intuitivo donde la estructura del problema es comprendida para así poder ser explotado de forma ingeniosa para obtener soluciones factibles [Nicholson, T. A. J., 2007].

Algunos ejemplos de métodos heurísticos incluyen recocido simulado, algoritmos genéticos, optimización de colonias de hormigas, búsqueda tabú y optimización de enjambres de partículas. Estos métodos se inspiran en fenómenos naturales o heurísticas específicas de problemas y utilizan varias estrategias para explorar el espacio de búsqueda y converger hacia buenas soluciones.

La heurística se ha aplicado con éxito a una amplia gama de problemas de optimización en varios campos, incluidos la ingeniería, las finanzas y la informática. Por ejemplo, “Se han utilizado algoritmos genéticos para optimizar los sistemas de gestión de la cadena de suministro” [Jauhar et al., 2015], mientras que la optimización de colonias de hormigas “Se ha utilizado para problemas de enrutamiento en redes de comunicación” [Blum, 2005].

2.3.3. Métodos metaheurísticos

Los métodos metaheurísticos son estrategias de optimización de alto nivel que guían el proceso de búsqueda manipulando los parámetros o características de otros algoritmos de optimización. Estos métodos se utilizan a menudo para problemas de optimización complejos a gran escala en los que los métodos de optimización tradicionales pueden no ser efectivos.

Los métodos metaheurísticos están estructuradas para poder realizar problemáticas con un nivel alto de dificultad para optimización y para equilibrar la exploración y la explotación del espacio de búsqueda y, a menudo, pueden encontrar rápidamente soluciones de alta calidad mediante el uso de estrategias de búsqueda inteligentes [Kelly et al., 1996].

Los ejemplos de métodos metaheurísticos incluyen la programación genética, la evolución diferencial, así como también algunos de estos ejemplos:

1. Algoritmos evolutivos: como el algoritmo genético, la estrategia evolutiva, la programación genética, entre otros

2. Optimización por enjambre de partículas (PSO): como el algoritmo originalmente propuesto por Kennedy y Eberhart en el año 2002.

2.3.4. Metaheurísticos poblacionales

Las metaheurísticas de población son una clase de algoritmos metaheurísticos que utilizan una población de soluciones candidatas para buscar la solución óptima. Estos métodos están diseñados para equilibrar la exploración y la explotación mediante el uso de un conjunto diverso de soluciones candidatas para explorar el espacio de búsqueda y converger en la mejor solución.

Los ejemplos de metaheurísticas de población incluyen algoritmos genéticos, optimización de enjambres de partículas [Kennedy et al., 2002] y optimización de colonias de hormigas [Dorigo et al., 2007]. Los algoritmos genéticos (GA) utilizan técnicas inspiradas en la evolución para hacer evolucionar una población de soluciones candidatas hacia una solución óptima. Y también la optimización de enjambre de partículas (PSO) es una metaheurística poblacional ya que simula el comportamiento de un enjambre de partículas que se mueven a través de un espacio de búsqueda para encontrar la solución óptima dentro de una población establecida.

2.3.5. Algoritmos evolutivos

Los algoritmos evolutivos (EA) son una clase de algoritmos de optimización metaheurística que se inspiran en el proceso de evolución natural. Estos utilizan un enfoque basado en la población para evolucionar iterativamente un conjunto de soluciones candidatas hacia la solución óptima. Los algoritmos evolutivos imitan el proceso de evolución mediante la selección de los individuos más aptos de una población para utilizarlos en la recombinación y la mutación, creando nuevas soluciones que luego se evalúan en cuanto a su aptitud.

Los ejemplos de algoritmos evolutivos son algoritmos genéticos (GA), evolución diferencial (DE) y estrategias evolutivas (ES). Los algoritmos genéticos [Holland, 1992], utilizan operadores de selección, cruce y mutación para crear nuevas soluciones candidatas en cada generación [Goldberg, 1989]. La evolución diferencial, genera nuevas soluciones candidatas mediante la creación de combinaciones lineales de diferencias entre individuos elegidos al azar. Las estrategias evolutivas, utilizan operadores de mutación y selección para crear nuevas soluciones candidatas en cada generación.

2.3.6. Algoritmo meméticos

Los algoritmos meméticos (MA) son una clase de algoritmos de optimización metaheurística estos son la combinación de los algoritmos evolutivos con búsqueda local los cuales intentarían imitar la evolución cultural [Moscato, 1989]. Este método está inspirado en modelos de sistemas naturales que utilizan un enfoque basado en la población para evolucionar iterativamente un conjunto de soluciones candidatas hacia la solución óptima y luego aplican operadores de búsqueda locales para refinar las mejores soluciones.

El término "memético" en MA se deriva de la palabra "meme", se refiere a las estrategias (por ejemplo, refinamiento local, perturbación o métodos constructivos, etc.), y representan pequeñas unidades de conocimiento que pueden ser compartidas y adaptadas por la población de soluciones candidatas los cuales se emplean para mejorar a los individuos de una población [Krasnogor, 2005].

En general, los MA brindan un enfoque flexible y efectivo para la optimización que puede aprovechar el conocimiento específico del dominio para mejorar la calidad y la eficiencia del proceso de búsqueda.

La idea de un algoritmo memético fue propuesta por primera vez por L. Davis en 1985, pero el término "algoritmo memético" y fue descrito en el libro llamado "Ant Colony Optimization" [Dorigo et al.,1999].

2.3.7. Algoritmo de búsqueda armónica

La búsqueda armónica (BA) es un algoritmo metaheurístico basado en la población inspirado en el proceso musical de búsqueda de un estado perfecto de armonía. Fue propuesto por Zong Woo Geem en 2001 [Geem et al, 2001].

La metodología de búsqueda armónica está inspirada en el proceso de improvisación musical. En este, un número predefinido de músicos intenta afinar el tono de sus instrumentos hasta lograr una armonía agradable. En la naturaleza, una armonía se define como una relación entre varias ondas sonoras que tienen diferentes frecuencias. La calidad de la armonía improvisada está determinada por la estimación estética, con el fin de mejorar la valoración estética y encontrar la mejor armonía, los músicos realizan múltiples ensayos [Askarzadeh & Esmat, 2017].

En el proceso de improvisación musical, todos los músicos emiten tonos dentro del rango posible para crear una armonía

2.3.7.1. Estructura general del algoritmo de Búsqueda Armónica

En un algoritmo de búsqueda armónica, una armonía es una solución factible y a cada variable de decisión de una armonía se le llama nota [Geem et al, 2001]. Un número determinado de armonías (HMS) en estas son guardas en una memoria de armonías denominada como (HM). Suponiendo que la meta es reducir o aumentar una función de aptitud (*f*) con respecto a variables de decisión d, el problema es definido en la ecuación 2.1

$$min\ or\ max \quad f(x_1, x_2, \dots, x_d) \tag{2.1}$$

En la que *f* es una función de aptitud, x_i = (*i*=1, 2, …, *d*) y esta es la variable de decisión *i*, y que la variable *d* es para numerar las dimensiones en este planteamiento. Se muestra a continuación un diagrama de flujo de su funcionamiento en la Figura 2.1.

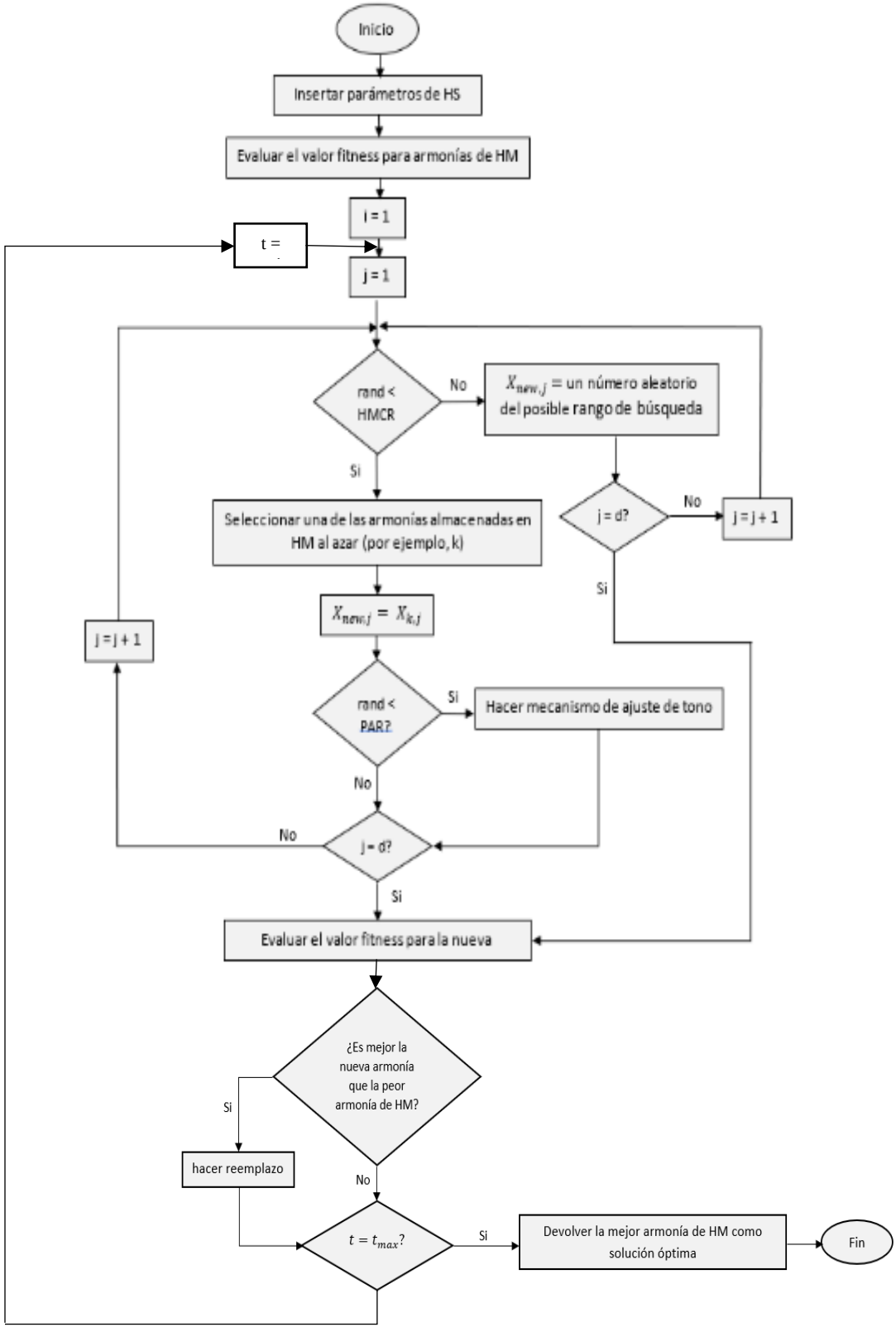

Figura 2.1. Diagrama de flujo del algoritmo de búsqueda armónica.

2.4. Búsqueda local

La búsqueda local es una familia de métodos de optimización que mejoran iterativamente una solución candidata explorando las soluciones vecinas en un espacio de búsqueda. La idea principal es comenzar con una solución inicial y aplicar un conjunto de operadores que modifiquen la solución de manera que mejore la función objetivo.

Una de las principales ventajas de la búsqueda local es su capacidad para manejar espacios de búsqueda grandes y complejos centrándose en las regiones más prometedoras del espacio de búsqueda. También es computacionalmente eficiente, ya que solo considera un pequeño subconjunto del espacio de búsqueda en cada iteración.

Existen varias variantes de los métodos de búsqueda local, como la escalada, el recocido simulado, la búsqueda tabú y la búsqueda local genética. Cada uno de estos métodos tiene su propio conjunto de operadores y estrategias para explorar el espacio de búsqueda.

Hay diversos ejemplos de aplicaciones de métodos de búsqueda local. Como, por ejemplo, se aplicó un algoritmo de búsqueda local al problema de enrutamiento de vehículos con ventanas de tiempo [Bräysy et al., 2005], mientras que otro estudio aplicaron un algoritmo de búsqueda tabú híbrido al problema de horarios de exámenes [Burke et al., 2010].

2.5. Sintonización de parámetros

La sintonización de parámetros en un algoritmo se refiere al proceso de encontrar los mejores valores para los parámetros que controlan el comportamiento del algoritmo. En la mayoría de los algoritmos, hay uno o más parámetros que deben ser configurados para adaptarse a la naturaleza del problema específico que se está resolviendo. La sintonización de parámetros es importante porque los valores óptimos de los parámetros pueden variar ampliamente según el problema que se esté abordando, y los valores subóptimos pueden llevar a un rendimiento pobre o incluso a la falta de convergencia.

Existen varias técnicas para el ajuste de parámetros, como la búsqueda en cuadrícula, la búsqueda aleatoria y la optimización bayesiana. Y la sintonización de parámetros se aplicado en diversos algoritmos, por ejemplo, el Particle Swarm Optimization (PSO) es un algoritmo de optimización global que utiliza la sintonización de parámetros para mejorar su rendimiento para converger y obtener nuevas soluciones optimas y también el algoritmo

Random Search que explora aleatoriamente el espacio de parámetros para encontrar valores óptimos [Bergstra et al., 2012].

2.6. Ajuste adaptativo de operadores

El ajuste adaptativo de operadores se refiere a una familia de técnicas de optimización que ajustan los operadores de búsqueda de un algoritmo durante el tiempo de ejecución para mejorar su rendimiento. Este tiene como objetivo ajustar los parámetros de los operadores de búsqueda o aprender una mejor combinación de operadores a lo largo del tiempo en función del rendimiento del algoritmo.

Existen diferentes enfoques para el ajuste adaptativo de operadores, entre ellos podemos encontrar:

1. Autoadaptación: este enfoque utiliza un mecanismo interno dentro del algoritmo para ajustar los parámetros de los operadores durante el tiempo de ejecución. Los parámetros se ajustan en función del rendimiento del algoritmo, como la aptitud de la población o la tasa de convergencia.
2. Ajuste adaptativo basado en bandidos: en este implica el uso de algoritmos de bandidos de multibrazo para seleccionar los mejores operadores genéticos. En este enfoque, las diferentes configuraciones de operadores se tratan como brazos de un problema de bandidos. El algoritmo de bandidos asigna una recompensa a cada brazo, que se actualiza a medida que avanza el algoritmo. Y después, se selecciona el brazo con el mejor desempeño y el algoritmo utiliza la configuración de parámetros correspondiente de ese mismo.
3. Métodos probabilísticos: aquí se implican el uso de distribuciones de probabilidad para ajustar los parámetros. En este método, los parámetros del algoritmo se modelan como variables aleatorias con distribuciones conocidas. Luego, el algoritmo toma muestras de estas distribuciones para determinar los valores de los parámetros. Las distribuciones se actualizan a medida que avanza el algoritmo, de modo que el algoritmo pueda converger a la configuración de parámetros más óptima. Aquí podemos encontrar las estrategias Probability matching y Adaptive pursuit

Un ejemplo de ajuste de operador adaptativo es el algoritmo de estrategia de evolución de adaptación de matriz de covarianza (CMA-ES), que ajusta el operador de mutación en función de la matriz de covarianza de la población [Hansen et al., 2001].

2.6.1. Fitness-Rate-Rank-based MultiArmed Bandit (FRRMAB)

La selección adaptativa de operadores (AOS) es una técnica utilizada en procesos de optimización para ajustar las tasas de aplicación de diferentes operadores en función de su rendimiento reciente. Ke Li ha propuesto un método de AOS basado en bandidos llamado Fitness-Rate-Rank-based MultiArmed Bandit (FRRMAB) [Li et al., 2014] que permite mejorar la exploración de soluciones en algoritmos de optimización. Este enfoque se utiliza para agilizar el proceso de búsqueda al asignar de manera adaptativa las tasas de aplicación de los operadores en función de su desempeño en el proceso de optimización.

FRRMAB se compone de dos componentes principales; la asignación de créditos, y la selección de operadores.

Asignación de créditos

Para esta tarea, previamente se ha creado un pool de operadores, en el cual, a cada índice le corresponde un operador de cruza y uno de muta, Ke Li utiliza cuatro operadores de mutación de evolución diferencial (DE).

1. DE/rand/1
2. DE/rand/2
3. DE/current-to-rand/2
4. DE/current-to-rand/1

Durante el proceso evolutivo, se calcula la recompensa para cada operador a la cual se le llama Fitness Improvement Rate (FIR), y se calcula con la siguiente formula:

$$FIR_{i,t} = \frac{Pf_{i,t} - cf_{i,t}}{pf_{i,t}} \qquad (2.2)$$

Donde $Pf_{i,t}$ es el valor de aptitud del padre y $cf_{i,t}$ es el valor de aptitud de la decendencia. El valor FIR para cada operador se almacena en una estructura de tipo FIFO llamada Sliding Window (ver figura 2.3), en dicha estructura los nuevos operadores evaluados se insertan al final, si el numero de elementos rebasa el tamaño *W* de la estructura, se removerán los elementos del inicio, esto permite tener las evaluaciones mas recientes de cada operador.

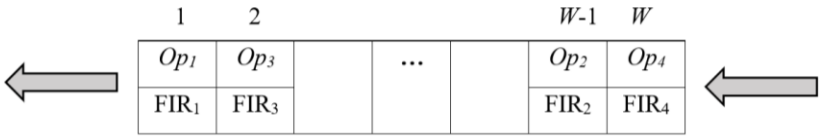

Figura 2.2. Sliding window [Ke Li et al., 2014]

Para calcular las recompensas (*Reward*$_i$), se suman los valores FIR para cada operador *i* almacenado en Sliding Window, después, todas las recompensas se ordenan de forma descendente y se ranquean, *Rank*$_i$ es el ranque para cada operador *i*. En este método, los autores incluyen un factor de decadencia, que permite que la calidad de cada operador se vaya degradando en el tiempo, esto de hace aplicando la siguiente formula:

$$Decay_i = D^{Rank_i} * Reward_i \tag{2.3}$$

Donde el factor de decadencia es un numero entre 0 y 1 definido por el usuario.

Para la asignación de créditos a cada operador se utiliza la siguiente formula:

$$FRR_{i,t} = \frac{Decay_i}{\sum_{j=1}^{k} Decay_j} \tag{2.4}$$

Selección de operadores

Basado en los créditos recibidos para cada operador, se aplica la selección de operadores basada en bandidos, la cual se puede observar en el algoritmo 2.3.

Algoritmo 2.3. Selección de operadores basado en bandidos

Inputs: op_t: número de operador utilizado, FIR*:* valor fitness del operador usado

Outputs: *op_t:* operador con mejor fitness.

1. ***if*** *Hay operadores que no han sido seleccionados* ***then***
2. $opt = sera\ seleccionado\ de\ manera\ aleatoria$
3. ***else***
4. $$op_t = argmax\left(FRR_{i,t} + C * \sqrt{\frac{2*In\sum_{j=1}^{k} n}{n_{i,t}}}\right)$$
5. ***endif***

2.7. Pruebas de hipótesis

La prueba de hipótesis es un método estadístico utilizado para determinar si una hipótesis sobre una población es cierta o no, en función de datos de muestra. El proceso implica hacer una declaración sobre un parámetro de la población (como una media o una proporción), recopilar datos de una muestra y usar análisis estadísticos para determinar si los datos respaldan o refutan la hipótesis.

Hay dos tipos principales de pruebas de hipótesis:

1. Hipótesis nula (*H0*): Esta hipótesis afirma que no hay una diferencia o relación significativa entre las variables que están siendo estudiadas.
2. Hipótesis Alternativa (*Ha*): Esta hipótesis plantea la existencia de una relación o diferencia significativa entre las variables investigadas, y es el objetivo del investigador probar o establecer dicha hipótesis.

En general, la prueba de hipótesis es una herramienta crucial en estadística e investigación, que nos permite sacar conclusiones y tomar decisiones basadas en la evidencia disponible.

2.7.1. Test de Wilcoxon

La prueba de Wilcoxon es una prueba estadística no paramétrica utilizada para determinar si dos muestras relacionadas provienen de poblaciones con la misma distribución. Es una alternativa a la prueba t pareada cuando los datos no cumplen con el supuesto de normalidad.

La prueba de Wilcoxon lleva el nombre de Frank Wilcoxon y se introdujo por primera vez en su artículo "Comparaciones individuales mediante métodos de clasificación" [Wilcoxon, 1945]. La prueba se basa en clasificar las diferencias entre las dos muestras y luego comparar las sumas de los rangos para las diferencias positivas y negativas.

La prueba de Wilcoxon se usa comúnmente en varios campos, como la biología, la medicina y las ciencias sociales. Es particularmente útil cuando el tamaño de la muestra es pequeño o cuando los datos no siguen una distribución normal.

Capítulo 3. Estado del arte

En este capítulo, se hace una revisión de forma breve aquellos trabajos que tengan alguna relación o hayan sido aplicados a resolver el problema de compras en internet con costos de envío. Se analizan sus puntos de vista y se revisa de forma superficial como han sido desarrollados.

A continuación, la Tabla 3.1 se observan las particularidades de algunos de los principales trabajos relacionados.

Tabla 3.1. Trabajos similares al problema planteado.

Autor/año	ILP	EXA	HED	HER	LS	META	INST
Geem, Kim y Loganthan, 2001				✓	✓	Busqueda armónica básico	
Blazewiez, 2010			✓			Reglas del IShOP	✓
Castillos, 2011		✓		✓		Lógica difusa armónica	
Nesmachnow, 2013					✓	MinMin	✓
Lopez-Loces, M. C., 2016	✓		✓		✓	GRASP P.Celular	✓
Garcia-Morales MA., 2021				✓	✓	Memético	✓
Este Trabajo				✓	✓	Armonía	✓

ILP: se refiere a modelos de programación lineal entera,

EXA: se refiere a métodos exactos

HED: se refiere a heurísticas constructivas deterministas

HER: se refiere a heurísticas constructivas aleatorias

LS: se refiere a búsquedas locales

META: se refiere a algoritmos metaheurísticos

INST: se utiliza para definir instancias.

3.1 Análisis de trabajos relacionados con el algoritmo de búsqueda armónica.

El algoritmo de búsqueda armónica es un algoritmo metaheurístico el cual fue desarrollado como en base a la improvisación de las notas musicales para la afinación de la música [Geem, 2001]. Desde entonces, el algoritmo se ha aplicado a una amplia gama de problemas de optimización y en la literatura se han sugerido varias variaciones y mejoras. Por ello se hablará de algunos trabajos notables relacionados con los algoritmos de búsqueda armónica y la forma en que fueron utilizadas para su desarrollo.

3.1.1 New heuristic optimization algorithm: Harmonious search

El trabajo original presenta la primera propuesta del algoritmo de búsqueda armónica, donde se habla de muchos problemas de optimización en varios campos que se han resuelto utilizando diversos algoritmos de optimización. Las técnicas tradicionales de optimización, como la programación lineal (LP), la programación no lineal (NLP) y la programación dinámica (DP), han tenido un papel importante en la solución de estos problemas, sin embargo, aún existen algunas posibilidades de idear nuevos algoritmos heurísticos basados en analogías con fenómenos naturales o artificiales. Es aquí donde se ha desarrollado un nuevo algoritmo heurístico, que imita la improvisación de los reproductores de música, y se ha denominado Harmony Search [Geem et al., 2001].

3.1.2 Improved harmony search algorithm for global optimization

En este artículo se habla del algoritmo de búsqueda de armonía (HS), y se explica que, este fácilmente queda atrapado en óptimos locales cuando busca óptimos globales. Para resolver este problema, crearon un algoritmo híbrido de búsqueda de armonía (HHS), que se basa en la concepción de la inteligencia de enjambre. HHS empleó un método de búsqueda local para reemplazar la operación de ajuste de tono y diseñó una estrategia de preservación elitista para modificar la operación de selección. [Wang et al., 2018].

3.2 Análisis de trabajos relacionados con el problema de compras por internet con costos de envió.

En esta sección se analizarán aquellos trabajos previos que hayan abordado el problema de las compras por Internet ya que es una de las actividades en línea más comunes y realizadas por millones de usuarios todos los días y como esta aumenta también la dificultad para conseguir el mejor precio entre todas las tiendas se vuelve algo cade vez más demandante y se hablara brevemente la forma en la que se resolvió.

3.2.1 Erratum to: Internet shopping with price-sensitive discounts

En este artículo habla de un problema de compra con descuentos sensibles al precio, en el que un cliente quiere comprar un conjunto determinado de productos en un conjunto determinado de tiendas de Internet, en el cual este problema es una extensión del problema original de optimización de compras por Internet (ISOP) [Blazewicz et al. 2010]. En la cual se quiere definir aún más la problemática y ampliar los parámetros sabidos para esta clase de problema. El problema es comprar todos los productos requeridos al precio mínimo total con descuento en lo cual por cuestiones de a proximidad en lo cual Blazewicz aplican 2 heurísticas para poder analizar los resultados de este problema denotados como G1 y G2 que sirven para considera los productos en un cierto orden por su costo total y optimizando así la mejor solución final para el cliente. [Blazewicz et al., 2014].

3.2.2 Estrategias de búsqueda local para el problema del ancho de corte

En esta tesis la investigación se enfocó en el análisis de funciones de búsqueda local, implementando diferentes configuraciones de los diversos componentes dentro de ellas, y resolviendo instancias del problema de ancho de corte [López, 2012.]. La evaluación del desempeño de las funciones desarrolladas se realizó con los métodos de búsqueda local solos e integrados en las metaheurísticas GRASP y algoritmo de procesamiento celular para determinar cuál de estas estrategias superar la resolución de los conjuntos de instancias considerados. Después de la finalización de los estudios experimentales, se concluyó que es la sinergia entre las funciones de búsqueda local y las metaheurísticas que es capaz de brindar soluciones de alta calidad, con resultados comparables a los del estado del arte. Y en los resultados, fueron evaluados por medio de Friedman estadístico de prueba dado un nivel de significancia alfa de 0.05

3.2.3 Exact and heuristic approaches to solve the Internet shopping optimization problem with delivery costs

Un artículo que habla del problema de las compras por Internet en donde esta ha sido una de las actividades en línea más comunes, realizadas por millones de usuarios todos los días y como esta aumenta también la dificultad para conseguir el mejor precio entre todas las tiendas. Por ello propone un modelo de programación lineal entera (ILP) y dos soluciones heurísticas, el algoritmo MinMin y el algoritmo de procesamiento celular (PCELL), obteniendo asi resultados mejores por las heurísticas, y para pequeños escenarios de casos reales, ofreciendo soluciones exactas en una cantidad de tiempo razonable gracias al uso de ILP [López et al., 2016].

3.2.4 Trustworthy online shopping with price impact

En este artículo se aborda el problema de optimización las compras por Internet (ISOP) [Blazewicz et al. 2010], Donde este propone analizar una muy interesante, y realmente sustancial, extensión de la ISOP. Es decir, en este se aplicaron factores de confianza donde se sometieron a un análisis cuidadoso desde el punto de vista del cliente. Así, crearon una definición de un nuevo modelo matemático del problema, y probar su afiliación a la clase de problemas NP-completos [López et al., 2017].

3.2.5 Algoritmo memético para el problema de ventas por internet con costos de envío

En esta tesis se investigó la problemática de la Optimización de compras por internet con costos de envío (IShOP). Se propuso un nuevo algoritmo metaheurístico basado en la metodología del algoritmo memético, el cual se considera actualmente como el mejor algoritmo para abordar esta problemática. Se introdujo una nueva representación vectorial de las soluciones candidatas que reduce la complejidad temporal del cálculo de la función objetivo de $O(n^2)$ a $O(n)$ [García, 2021]. Además, se realizaron experimentos computacionales comparando el rendimiento del algoritmo memético MAIShOP con el algoritmo Pcell [López et al., 2016]. Los resultados de estos experimentos, que incluyeron diversas instancias aleatorias, demostraron claramente la superioridad del algoritmo propuesto MAIShOP.

3.3 Conclusión del estado del arte

Tomando en cuenta los trabajos anteriores, se puede decir que el mejor algoritmo del estado del arte al momento de la elaboración de este documento es un algoritmo memético propuesto en la tesis de maestría de Miguel Ángel García Morales en al año 2021 llamado MAIShOP [García, 2021]. Este algoritmo integra dos nuevas heurísticas para generar soluciones llamadas First Plus y First Best, además de dos nuevos operadores de mutación llamados Mutación uniforme FirstBest y Mutación heurística FirstBest. Este algoritmo se tomará como algoritmo de referencia en el capítulo de evaluación experimental, donde se comparará contra el algoritmo propuesto en este trabajo.

Capítulo 4. Metodología de solución

En este capítulo se explica la adaptación del algoritmo de búsqueda armónica para abordar el desafío de optimizar el proceso de compras por internet con costos de envió. Además, se detallan los nuevos mecanismos implementados para mejorar la calidad de las soluciones obtenidas. El objetivo es optimizar el proceso de compra en línea considerando los costos de envio, y para ello se han realizado ajustes y mejoras en el algoritmo de búsqueda armónica.

4.1 Algoritmo búsqueda armónica adaptado al problema de compras por internet con costos de envío

Inicialmente, la búsqueda armónica fue concebida para abordar problemas de optimización en el dominio continuo. Sin embargo, en el caso del problema IShOP, nos encontramos frente a un problema discreto, lo cual implica la necesidad de adaptar esta metodología a este tipo de situaciones. En consecuencia, se requiere realizar ajustes y modificaciones en la búsqueda armónica para que sea aplicable y efectiva en el contexto de problemas de optimización discreta como IShOP, por lo cual se ha decido utilizar la representación de la solución propuesta para el algoritmo MAIShOP [García, 2021].

En el algoritmo de búsqueda armónica una solución es llamada armonía, por lo cual, una armonía *i* para este problema se representará de la siguiente manera:

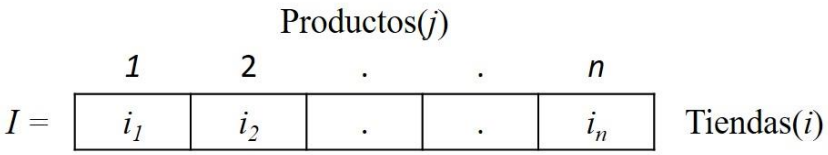

Figura 4.1. Representación de una armonía.

El algoritmo de búsqueda armónica (BA) emplea una estructura de memoria conocida como "memoria de armonías" (HM) que tiene la capacidad de almacenar un número específico de armonías, denominado "tamaño de memoria de armonías" (HMS).

El cálculo de la función objetivo que se utilizará para evaluar cada una de las armonías almacenadas en HM es el propuesto para el problema IShOP, el cual se realiza con la ecuación 4.1 consiste de la siguiente manera:

$$F(I) = \sum_{i=1}^{m} d_i + \sum_{j=1_{\forall j \,|tal\ que\ \ I(j)=i}}^{n} \left(c_{ij}\right) \qquad (4.1)$$

4.1.2. Mecanismo de ajuste adaptativo de parámetros

Se ha tomado el método de selección de operadores adaptativa Fitness-Rate-Rank-based MultiArmed Bandit (FRRMAB) propuesto por Ke Li [Li et al., 2014] y se ha hecho una modificación en su pool de operadores para convertirlo en un pool de acciones, dichas acciones consisten en el ajuste de los valores de dos parámetros del algoritmo de búsqueda armónica, dichos parámetros se muestran en la tabla 4.1.

Tabla 4.1. Parámetros que se ajustaran adaptativamente

Parámetro	Descripción	Valor inicial
HMCR	Tasa de consideración de memoria de armonías	0.95
PAR	Tasa de ajuste de tono	0.7

El pool de acciones ha quedado de la siguiente forma:

1) $HMCR{+}{=}\ 0.0001, PAR{+}{=}\ 0.0001$
2) $HMCR{-}{=}\ 0.0001, PAR{-}{=}\ 0.0001$
3) $HMCR{+}{=}\ 0.0001$
4) $PAR{+}{=}\ 0.0001$
5) $HMCR{-}{=}\ 0.0001$
6) $PAR{-}{=}\ 0.0001$

La nueva Sliding Window queda de la siguiente forma:

Figura 4.2. Sliding Window adaptada a acciones

Por último, se ha modificado el mecanismo de selección de operadores, por un mecanismo selector de acciones, este se muestra en el algoritmo 4.1.

Algoritmo 4.1. Selección de acción del método de bandidos para HS

Inputs: ac_t: número de acción utilizado, FIR*:* valor fitness de la acción usada

Outputs: ac_t*:* acción con mejor fitness.

1. ***if*** *Hay acciones que no han sido seleccionadas* ***then***
2. $ac_t \leftarrow$ *sera seleccionado de manera aleatoria*
3. ***else***
4. $$ac_t \leftarrow argmax\left(FRR_{i,t} + C * \sqrt{\frac{2*ln\sum_{j=1}^{k} n}{n_{i,t}}}\right)$$
5. ***endif***

4.1.3. Mecanismo de Re inicialización de memoria de armonías

Para evitar el estancamiento, se ha implementado un método de Re inicialización de armonías, el cual funciona de la siguiente manera: durante el proceso evolutivo, si la mejor solución no cambia durante un número determinado de iteraciones (por lo regular el 20% del total) se toma la memoria de armonías (HM) y se reemplaza un porcentaje (generalmente el 10%) de sus armonías por nuevas armonías generadas de forma aleatoria, lo que permite al algoritmo tener una mayor diversificación de la que originalmente tiene.

4.1.4. Algoritmo de Búsqueda armónica Adaptativo

El Algoritmo 4.2 muestra la versión del algoritmo de búsqueda armónica con los mecanismos de ajuste de parámetros adaptativo y el método de Re inicialización de la memoria de armonías.

Algoritmo 4.2. Algoritmo de Búsqueda armónica Adaptativo

Inputs: *file*: nombre del archivo de instancia, *MaxIt:* número máximo de iteraciones, *HMS:* tamaño de la memoria de armonías, *nNew:* tamaño de la memoria de nuevas armonías, *HMCR:* Tasa de consideración de la memoria de armonías, *PAR:* Tasa de ajuste de tono, *FW:* ancho de banda, *FW_damp:* actualización del ancho de banda, ς: porcentaje de estancamiento permitido.
Outputs: *bestSol:* mejor solución encontrada.

Functions:
LectorDeInstancia(file): Realiza la lectura del archivo de la instancia para almacenar y usar los datos.
GenerarMemoriaArmonias (): Aquí se van a generar de manera aleatoria según el método las primeras armonías con los datos de la instancia y se almacenan en una lista.
Evaluar (x_{new}): Se evalúan los valores de la suma de las armonías con el coste de envió.

1. *LectorDeInstancias (file)*
2. *Inicializar nueva memoria de armonías aleatorias de HM de tamaño HMS*
3. *Ordenar lista HM por Costo de menor a mayor*
4. *bestSol* ← *HM (0)*
5. ***for*** $it \leftarrow 1$ ***to*** $MaxIt$ ***do***
6. ***for*** $k \leftarrow 1$ ***to*** $nNEW$ ***do***
7. $Generar\ nueva\ armonía\ aleatoria\ x_{new}$
8. ***for*** $j \leftarrow 1$ ***to*** d ***do***
9. $r \leftarrow número\_aleatorio\ [0, 1]$
10. ***if*** $r <= HMCR$ ***then***
11. $Selecciona\ aleatoriamente\ la\ armonía\ x_i\ almacenada\ en\ HM$
12. $x_{new,j} \leftarrow x_{i,j}$
13. $EvaluarMejora(x_{new})$
14. ***endif***
15. $r \leftarrow número\_aleatorio[0, 1]$
16. ***if*** $r < PAR$ ***then***
17. $x_{new,j} \leftarrow x_{new,j} + fw \times rand$
18. $EvaluarMejora(x_{new})$
19. ***endif***
20. ***endfor***
21. $Añadir\ x_{new}\ a\ memoria\ de\ nuevas\ armonías\ NEW$
22. ***endfor***
23. $HM \cup NEW$
24. $Ordenar\ HM\ por\ costo$
25. $Truncar\ HM\ a\ HMS\ armonías$
26. ***if*** $it > 0$ ***then***
27. ***if*** $HM(0) == bestSol$ ***then***

28. *estancamiento* ← *estancamiento* + 1
29. **else**
30. *estancamiento* ← 0
31. **endif**
32. **if** *estancamiento* > *HMS* × ç **then**
33. ReinicializacionHarmonyMemory ()
34. *Ordenar lista HM por Costo de menor a mayor*
35. **endif**
36. **endif**
37. *bestSol* ← *HM (0)*
38. *fw* ← *fw x fw_damp*
39. **endfor**
40. **return** *bestSol*

Adicional al algoritmo anterior, también se presenta un diagrama de flujo, en el cual se muestra de forma generalizada el funcionamiento del nuevo algoritmo propuesto.

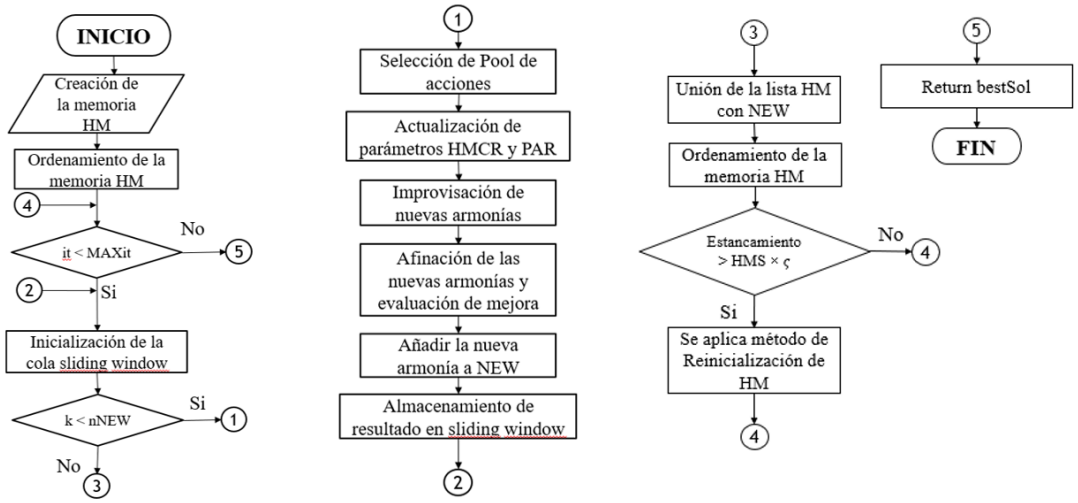

Figura 4.3. Diagrama de flujo del algoritmo de búsqueda armónico con parámetros adaptativos.

Capítulo 5. Experimentación computacional

En este capítulo se describe la experimentación computacional realizada para comparar el desempeño del algoritmo de búsqueda armónica adaptativo comparado contra el mejor algoritmo del estado del arte MAIShOP.

5.1. Instancias utilizadas

Se proporcionan los detalles de los grupos de instancias, que se caracterizan por las dimensiones "m" y "n", que representan la cantidad de tiendas y productos, respectivamente. Se crearon tres grupos de instancias con diferentes dimensiones. El grupo de instancias menores consiste en tres subconjuntos con 3, 4 y 5 productos, y 20 tiendas, cada uno de ellos con treinta instancias. El grupo de instancias intermedias comprende tres subconjuntos con 5 productos y 50 tiendas, con 240 y 400 instancias, respectivamente, todas ellas evaluadas en treinta instancias. El grupo de instancias mayores se compone de tres subconjuntos con 50 y 100 productos, y 240 y 400 tiendas, respectivamente, también con treinta instancias evaluadas. Los detalles de cada subconjunto se definen según el método propuesto por Blazewicz en 2010, que describe la cantidad de productos "n" y tiendas "m" de las instancias incluidas en cada subconjunto. Los grupos de instancias menores se crean con los subconjuntos 3n20m, 4n20m y 5n20m, cada uno con 90 instancias. Los grupos de instancias intermedias se componen de los subconjuntos 5n20m, 5n400m y 50n240m, todos ellos con 30 instancias. Los grupos de instancias mayores incluyen los subconjuntos 50n400m, 100n240m y 100n400m, también con treinta instancias evaluadas en las pruebas.

5.2. Experimentación computacional

Para la experimentación se han implementado los algoritmos en Java 8, y se han ejecutado en una PC con un procesador Ryzen 7 5700X con 32 GB de RAM sobre el sistema operativo Windows 10. Se han ejecutado treinta experimentos independientes para cada algoritmo y cada instancia. Para ejecutar los algoritmos en condiciones similares, se han utilizado los mismos valores de tamaño de población y numero máximos de iteraciones. Los valores de los parámetros y variables utilizados en los algoritmos se muestran en la tabla 5.1.

Tabla 5.1. Parámetros y variables utilizados.

Parámetro	Búsqueda armónica adaptativa	MAIShOP
MaxIt	100	100
HMS, nPop	100	100
HMCR, Cruza	0.95	0.6
PAR, Muta	0.7	0.01
cFW, er	0.05	0.05
FW_DAMP, pmu	0.995	0.03
nNew	100	NA
Porcentaje de estancamiento permitido(ς)	0.2	NA

5.3. Resultados experimentales

Se realizaron las comparativas de los resultados, primeramente, comparando el rendimiento del algoritmo de búsqueda armónica básico contra la versión adaptativa, posteriormente, se comparó el mejor algoritmo del estado del arte MAIShOP contra la versión adaptativa del algoritmo de búsqueda armónica.

5.3.1. Algoritmo de búsqueda armónica adaptativo vs Algoritmo de búsqueda armónica básico

En la Tabla 5.2 se presentan los resultados logrados para cada conjunto de 30 instancias luego de aplicar la prueba no paramétrica de Wilcoxon. En cada algoritmo se muestran el promedio y la desviación estándar (indicados con subíndices) así como también el valor objetivo y del tiempo de cada solución generada. En cada celda se señala si existe alguna diferencia significativa que este a favor del BA adaptativo (representado por una flecha hacia abajo) o del BA básico (representado por una flecha hacia arriba), o si no hay diferencia

significativa (indicado con un guion). La significancia es determinada con una confiabilidad del 95%. Y en las celdas sombreadas corresponden al algoritmo que obtuvo los mejores resultados en términos de calidad o eficiencia.

Tabla 5.2. Desempeño comparativo de los algoritmos BA adaptativo vs BA básico aplicando Wilcoxon.

Grupo Instancias (10)	Harmony Search Adaptativo (FO)	Harmony Search (FO)	Harmony Search Adaptativo (Time)	Harmony Search (Time)
3*n*20*m*	62.75_{96}	62.75_{96}--	0.08711798	0.00057341 ↑
4*n*20*m*	79.198_{03}	79.198_{03}--	0.09527112	0.08028623 ↑
5*n*20*m*	104.26_{03}	102.57_{36}↑	0.10677036	0.088676 ↑
5*n*240*m*	79.892_{07}	80.759_{06}↓	0.16423998	0.141833 ↑
5*n*400*m*	75.979_{32}	76.080_{36}↓	0.23597365	0.19311805 ↑
50*n*240*m*	460.74_{48}	606.184_{91}↓	6.95542011	6.88831774 --
50*n*400*m*	396.56_{57}	567.103_{95}↓	12.2669563	12.19577967 --
100*n*240*m*	912.59_{36}	2073.493_{81}↓	32.4812952	25.6634592 ↑
100*n*400*m*	751.71_{08}	1917.310_{03}↓	64.8651755	50.1274242 ↑
Promedio global	90.4967111	220.0077367↓	7.216921498	5.569777512 ↑

En la tabla anterior se evidencia que el algoritmo de búsqueda armónica que utiliza el método de ajuste de parámetros adaptativos FRRMAB muestra un rendimiento superior. En términos de calidad, se observa una mejora significativa en 6 de los 9 grupos, mientras que en los restantes se mantiene a la par sin mejoras notables. Esto demuestra que el algoritmo de búsqueda armónica logra una notable mejora en la calidad de la solución final, superando los resultados obtenidos previamente, como se muestra en la tabla.

5.3.2. Algoritmo MAIShOP vs Algoritmo de búsqueda armónica adaptativo

En la Tabla 5.3 se presentan los resultados logrados para cada conjunto de 30 instancias luego de aplicar la prueba no paramétrica de Wilcoxon. En cada algoritmo se muestran el promedio y la desviación estándar (indicados con subíndices) así como también el valor objetivo y del tiempo de cada solución generada. En cada celda se señala si existe alguna diferencia significativa que este a favor del algoritmo de MAIShOP (↓), del algoritmo de búsqueda armónica adaptativo (↑), o si no se observan diferencias significativas (--). La

significancia se estableció con un nivel de confianza del 95%. Las celdas resaltadas corresponden al algoritmo que obtuvo el mejor resultado en términos de calidad o eficiencia.

Tabla 5.3. Desempeño comparativo de los algoritmos MAIShop vs BA adaptativo aplicando Wilcoxon.

Grupo Instancias (10)	MAIShOP (FO)	Harmony Search (FO)	MAIShOP (Time)	Harmony Search (Time)
3*n*20*m*	63.05_{92}	62.75_{96}--	0.00057341	0.08711798↓
4*n*20*m*	80.031_{36}	79.198_{03}--	0.00128667	0.09527112↓
5*n*20*m*	104.26_{03}	102.57_{36}--	0.00179728	0.10677036↓
5*n*240*m*	80.917_{03}	79.892_{07}--	0.21702556	0.16423998↑
5*n*400*m*	72.73_{66}	75.979_{32}--	0.36987333	0.23597365↑
50*n*240*m*	497.92_{76}	460.74_{48}↑	30.1426144	6.95542011↑
50*n*400*m*	438.74_{30}	396.56_{57}↑	33.1063367	12.2669563↑
100*n*240*m*	993.70_{39}	912.59_{36}↑	50.4910367	32.4812952↑
100*n*400*m*	801.50_{81}	751.71_{08}↑	176.110986	64.8651755↑
Promedio global	348.0986	324.668635↑	32.2712811	13.0286911↑

En la tabla anterior se puede observar que el algoritmo de búsqueda armónica que utiliza el método de ajuste de parámetros adaptativos FRRMAB muestra un desempeño aceptable en comparación con MAIShOP. En términos de calidad, se observa una mejora significativa en 4 de los 9 grupos, mientras que en los restantes no se observan mejoras significativas. En cuanto a eficiencia, el algoritmo supera a MAIShOP en 6 de los 9 grupos. Es importante señalar que el algoritmo de búsqueda armónica muestra una mejora notable en cuestión de calidad de los resultados finales, especialmente en los grupos de instancias más grandes.

6.4. Conclusiones de la experimentación

Como se puede observar en los resultados, el algoritmo de búsqueda armónica adaptativo en las instancias más grandes logra mejorar el desempeño del mejor algoritmo del estado del arte. En el promedio global, se puede ver que el algoritmo de búsqueda armónica adaptativo es mejor que MAIShOP tanto en calidad como en eficiencia, por lo que se puede considerar que es un algoritmo competitivo.

Capítulo 6. Conclusiones y trabajo futuro.

6.1. Realización y Cumplimiento de objetivos

En el desarrollo de la tesis, se han lograron alcanzar exitosamente todos los objetivos propuestos. En esta sección se detallan las principales contribuciones realizadas utilizando el algoritmo de búsqueda armónica, así como los resultados científicos y pruebas obtenidos a lo largo de la investigación.

6.1.1 Objetivo general

Crear un algoritmo en la cual tenga como estructura la búsqueda armónica que resuelva el planteamiento de compras por internet con costos de envío, con un rendimiento estadísticamente mejor que los algoritmos existentes mencionados en el capítulo 4. Además, llevar a cabo pruebas estadísticas para comparar el rendimiento del algoritmo propuesto con el algoritmo del estado del arte, como se describe en el capítulo 5.

6.1.2 Objetivos específicos

- Análisis exhaustivo de la literatura existente
- Elaboración de un marco teórico sólido
- Diseño e implementación de la estructura de armonías para abordar el problema IShOP
- Diseño e implementación de una memoria de armonías eficiente
- Diseño e implementación de un método de improvisación de armonías efectivo
- Diseño e implementación de un método para ajustar el tono de una armonía
- Evaluación experimental del rendimiento del algoritmo en pruebas y análisis de resultados obtenidos

6.2. Contribuciones principales

Las contribuciones principales de esta investigación son las siguientes:

1. Un algoritmo de búsqueda armónica para el problema de Compras por Internet con Costos de Envío (ver sección 4.1.4. Algoritmo de Búsqueda armónica Adaptativo en el capítulo 4).
2. La implementación de un nuevo método de ajuste de parámetros adaptativo y su incorporación al algoritmo de búsqueda armónica (Ver sección 4.1.2. Mecanismo de ajuste adaptativo de parámetros del capítulo 4).
3. La incorporación de un mecanismo de Re inicialización de la memoria de armonías que previene el estancamiento del algoritmo. (Ver sección 4.1.3. Mecanismo de Re inicialización de memoria de armonías en el capítulo 4).

6.3. Productos científicos

Durante el desarrollo de este proyecto de tesis, se tuvo participación en el siguiente producto científico:

- Artículo: *Hybrid Harmony Search Optimization Algorithm for Continuous Functions* en la revista Mathematical and Computational Applications. DOI: 10.3390/mca28020029. ISSN: **1300-686X.** Emerging Sources Citation Index

6.4. Trabajo futuro

Para continuar esta investigación se tiene considerado aplicar el enfoque de búsqueda armónica a otros modelos del problema IShOP. Otra área de oportunidad es la aplicación de este enfoque a la optimización de problemas multiobjetivo estáticos y dinámicos.

Anexos

Representación de la solución de Heurística FirstBest

Para calcular el costo de los productos de las tiendas se aplica la fórmula de la función objetivo en donde tenemos que cada casilla del vector en *i* contiene la posición del producto de la tienda seleccionada c_{ij} con base en eso se hace la sumatoria de los costos y se toma en cuenta el costo de envió d_i también tomando de referencia la posición del producto seleccionado, en dado caso que haya dos o más productos escogidos en la misma tienda entonces la suma del costo de envió solo se aplicara una sola vez, una vez terminado el proceso se almacena el valor de *X* y este repite el ciclo *n* veces en este caso por el número de productos *j* y en dado caso de que disminuya el costo este valor reemplazara al valor contenido en *X* hasta que termine el ciclo [García, 2021].

Ahora, se presenta un ejemplo de cómo se realiza el cálculo de la heurística firstbest.:

Aquí tenemos 3 vectores de i productos que contienen la posición de las tiendas *m* las cuales y *n* es el número de ciclos del cual se repetirá el proceso sus valores están almacenados en el conjunto c_{ij} junto con su conjunto de costos de envió d_i.

n=3 m=5

	1	2	3	4	5
$I_1 =$	3	1	1	2	2
$I_2 =$	2	3	2	3	1
$I_3 =$	2	4	1	1	3

$$c_{ij} = \begin{bmatrix} 20 & 32 & 26 & 40 & 60 \\ 22 & 35 & 23 & 52 & 45 \\ 28 & 40 & 23 & 56 & 43 \\ 24 & 27 & 14 & 42 & 59 \\ 23 & 48 & 21 & 50 & 53 \end{bmatrix} \quad di= \begin{bmatrix} 12 \\ 15 \\ 14 \\ 14 \\ 11 \end{bmatrix}$$

Entonces para iniciar a calcular tenemos que obtener su costo total, eso se hace sumando los precios de la matriz c_{ij} donde el costo de envoi no se vuelve a sumar en caso de que se repita la posición de la tienda en donde se compró ejemplo [1,1,1] quiere decir que los

primeros 3 productos se compraron en la tienda 1 entonces solo se sumara su precio de envío di una sola vez.

$I_1 = [3,1,1,2,2]$ $F(I_1) = 224$

$X = I_1$ costo = 224

i = 1	j = 1	I = (1,1,1,2,2)	F(I) = 202	costo = 202	i = 1
	j = 2	I = (2,1,1,2,2)	F(I) =204		
	j = 3	I = (3,1,1,2,2)	F(I) =224	costo = 224	i' = 1
	j = 4	I = (4,1,1,2,2)	F(I) = 220		
	j = 5	I = (5,1,1,2,2)	F(I) = 216		

Una vez ya calculado su costo total se escoge la solución en el valor más optimo en este caso lo que se quiere hacer es minimizar el precio total entonces se toma el valor con el costo total más bajo y se almacena en la variable F(X) y el ciclo se vuelve repetir hasta que se complete el ciclo definido por el tamaño en *n*.

X = [1,1,1,2,2] F(X) = 202 costo = 202

i = 2	j = 1	I = (1,1,1,2,2)	F(I) = 202	costo = 202	i' = 1
	j = 2	I = (1,2,1,2,2)	F(I) =205		
	j = 3	I = (1,3,1,2,2)	F(I) =224		
	j = 4	I = (1,4,1,2,2)	F(I) = 211		
	j = 5	I = (1,5,1,2,2)	F(I) = 229		

X = [1,1,1,2,2] F(X) = 202 costo = 202

i = 3	j = 1	I = (1,1,1,2,2)	F(I) = 202	costo = 202	i' = 1
	j = 2	I = (1,1,2,2,2)	F(I) =199	costo = 199	i' = 2
	j = 3	I = (1,1,3,2,2)	F(I) =213		
	j = 4	I = (1,1,4,2,2)	F(I) = 204		
	j = 5	I = (1,1,5,2,2)	F(I) = 208		

Ejemplo del funcionamiento del algoritmo de búsqueda armónica

Para realizar la estructura de un algoritmo de búsqueda armónica HS se realizan diversos procedimientos principales los cuales son la creación de la memoria de armonías, la improvisación de nuevas armonías para mejorar la búsqueda en la población y la evaluación final de las armonías más optimas obtenidas [Geem et al., 2001].

Memoria de armonías

La primera parte para poder realizar un algoritmo de búsqueda armónica (HS), es necesario crear primero un numero de *X* armonías obtenidas mediante la información del problema que se quiere resolver y estas armonías son almacenadas en una memoria de armonías (HM) para poder trabajar en ellas ya que cada armonía son posibles soluciones optimas. En la Tabla 1 se muestra como es la composición de la memoria de armonías (HM), en donde la armonía i es representada como un vector, $i = [x_{i,1},\ x_{i,2}, \ldots\ x_{i,d}]$.

Para el llenado de la memoria de armonías primeramente se inicializan de manera aleatoria los vectores de las armonías, mediante el uso de la siguiente formula:

$$x_{i,j} = l_j + rand * (u_i - l_j) \qquad i = 1,2,\ldots,N; \qquad j = 1,2,\ldots,d \tag{2}$$

En la que l_j y u_i representan los límites inferior y superior de la variable de decisión j, respectivamente, la variable *rand* es un número aleatorio generado de manera uniforme en el intervalo [0 1].

Tabla 1. Estructura de una memoria de armonía (HM).

	x_1	x_2	…	x_d	f
Armonía 1	$x_{1,1}$	$x_{1,2}$	…	$x_{1,d}$	f_1
Armonía 2	$x_{2,1}$	$x_{2,2}$	…	$x_{2,d}$	f_2
…	…	…			
Armonía N	$x_{N,1}$	$x_{N,2}$	…	$x_{N,d}$	f_N

Ejemplo

Tenemos la siguiente función de aptitud en donde f (x_1, x_2,x3) = $x_1^2+x_2^2+x_3^2$, en base a ella se crea una memoria de armonías de un tamaño para almacenar 3 armonías.

Primera mente se aplica la fórmula 2 para poder generar aleatoriamente las armonías en el cual l_j = -1 y u_i = 1, así como también se especifica el tamaño N = 3 y d = 3 (número de dimensiones) la i = 1,2,3, y j = 1,2,3.

Armonía	x_1	x_2	x_3	F(x_1,x_2,x_3)
1	0 con rand = 0.5	-0.5 con rand = 0.25	0.5 Con rand = 0.75	0.5
2	-0.5 con rand = 0.25	0.8 con rand = 0.9	0 con rand 0.5	0.89*
3	0.2 Con rand = 0.6	-0.8 Con rand = 0.1	-0.2 Con rand = 0.4	0.72

*Mejor armonía.

Improvisación de una nueva armonía

Una vez completado el llenado de la memoria de armonías, se procede a crear o improvisar nuevas armonías, aplicando la siguiente formula $X_{new} = [X_{new,1}, X_{new,2} \dots, X_{new,d}]$ Estas nuevas armonías se generan en base a todas las armonías ya existentes que están almacenadas en la memoria de armonías HM para ello se usan 2 etapas en el proceso de creación de las nuevas armonías.

Etapa 1: uso del parámetro HMCR

Durante este proceso, se obtiene un valor aleatorio entre 0 y 1 con una distribución uniforme (rand) para hacer comparativa con HCMR. Si *rand* es mayor que *HMCR*, entonces la variable de decisión *j* generada de la nueva armonía ($X_{new,j}$) no cambia y se utiliza para generar aleatoriamente un nuevo valor aplicando la siguiente formula:

$$X_{new,j} = l_j + rand * (u_i - l_j) \qquad (3)$$

En el caso contrario donde, si *rand* es menor que *HMCR*, entonces este se elige de manera aleatoria una de las armonías que están guardadas en *HM*, donde el valor *k* de la armonía almacenada en la posición *j* es asignada a el valor *j* en la armonía *k* elegida de la nueva armonía creada, y es mostrada en la siguiente formula.

$$X_{new,j} = X_{k,j} \qquad (4)$$

Ahora en caso de que rand sea menor que *HMCR* entonces se aplicaría la fórmula para reemplazar la posición *j* de la armonía actual por otro valor de otra armonía en la misma posición.

Si *rand* es igual a 0.2 y menor que *HMCR*

J = 1

Y se selecciona de manera aleatoria la armonía *k* en la posición 2

Armonía	x_1	x_2	x_3	$F(x_1,x_2,x_3)$
2	-0.5 con rand = 0.25	0.8 con rand = 0.9	0 con rand 0.5	0.89

$X_{new,1} = X_{2,1} = -0.5$

Y ahora como podemos ver la posición 1 de j en la nueva armonía ya ha sido reemplazada por el valor de j en la armonía escogida aleatoriamente

Etapa 2: uso del parámetro PAR

Una vez completado la etapa 1, el HS procede a utilizar otro parámetro que es un mecanismo de ajuste de tono denominada como *PAR*, este varia entre 0 y 1, tiene el propósito de evitar óptimos locales. En esta etapa, se aplican los mismos métodos de comparación que

en la etapa anterior. Si un número aleatorio (*rand*) es menor que *PAR*, la nota improvisada se cambia a un valor vecino utilizando la Ecuación 5. Por otro lado, si rand es mayor que *PAR*, la nota improvisada permanece sin cambios.

$$X_{new,j} = X_{new,j} + bw * (rand - 0.5) * abs(u_j - l_j) \qquad (5)$$

En donde tenemos la variable *bw* que se denomina como ancho de banda de generación. La expresión (rand - 0.5) produce un número aleatorio dentro del rango [-0.5, 0.5] esto con el fin de poder seleccionar de manera aleatoria la dirección de movimiento.

Ejemplo

En primer lugar, se lleva a cabo el ajuste de cada valor aplicando la ecuación 5. En el cual, se muestra el ajuste de la primera variable de decisión y tenemos que *PAR* igual a 0.3 y *bw* igual a 0.5, y se estable el valor de *J* como 1

Y se selecciona de manera aleatoria la armonía *k* en la posición 2

Armonía	x_1	x_2	x_3	$F(x_1,x_2,x_3)$
2	-0.5 con rand = 0.25	0.8 con rand = 0.9	0 con rand 0.5	0.89

$X_{new,1} = X_{2,1} = -0.5$

Tenemos que si el valor de *rand* es igual a 0.2 entonces *rand* es menor que *PAR*, y el valor de $X_{new,1}$ entonces se aplica la fórmula de la ecuación 5. Entonces para este caso se tendría que:

$$X_{new,1} = -0.5 + 0.5 * (0.4 - 0.5) * 2 \; = \; -0.6$$

En el caso contrario de que *rand* mayor que *PAR* simplemente no se modifica el valor y no se aplica ninguna fórmula.

Reemplazo

Una vez que se han generado las nuevas armonías, se procede a incorporarlas y sustituir las armonías existentes en la memoria de armonías (HM). Estas armonías se ordenan de forma descendente según su valor objetivo, de menor a mayor en el caso de minimización de la función objetivo. Si la nueva armonía (f_{new}) es superior a la armonía existente (f_{ith}), se elimina esta última de la HM y se reemplaza por la nueva armonía generada. Por otro lado, si la nueva armonía es inferior a la armonía existente, se descarta y se mantiene la armonía previa en la HM.

Criterio de parada

Una vez finalizado el proceso de generación y reemplazo de nuevas armonías, se aplica un criterio de terminación basado en un número predefinido de iteraciones (W_{max}). Si se ha alcanzado este límite de iteraciones, el algoritmo finaliza su ejecución. En caso contrario, el proceso de generación y reemplazo de armonías continúa hasta que se cumpla el criterio de terminación establecido.

Fase final del Algoritmo de búsqueda armónica

En esta etapa, una vez que el algoritmo ha finalizado su ejecución, se obtiene como resultado la mejor armonía almacenada en la memoria de armonías (HM), que representa la solución óptima encontrada.

Referencias

Blazewicz, 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., Bouvry, P., Kovalyov, M. Y., & Musial, J. (2014). Erratum to: Internet shopping with price-sensitive discounts. *4OR*, *12*(4), 403–406.

López-Loces, 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.

Musial, J., & López-Locés, M. C. (2017). Trustworthy Online Shopping with Price Impact. *Foundations of Computing and Decision Sciences*, *42*(2), 121-136.

Geem, Z. W., Kim J. H., and Loganathan G. V., (2001). A new heuristic optimization algorithm: harmony search, simulation, vol. 76, pp. 60-68.

Miguel Ángel García Morales. (2021). Algoritmo memético para el problema de Ventas por Internet con Costos de Envío. Instituto Tecnológico de Ciudad Madero.

Li, K., Fialho, Á., Kwong, S., & Zhang, Q. (2014). Adaptive Operator Selection With Bandits for a Multiobjective Evolutionary Algorithm Based on Decomposition. IEEE Transactions on Evolutionary Computation, 18(1),

Mario Cesar López Locés. (2012). Estrategias de Búsqueda Local Para el Problema del Ancho de Corte. Instituto Tecnológico de Ciudad Madero.

Borgwardt, K. H. (1987). The Simplex Method. En *Algorithms and combinatorics*. Springer Nature.

Goldberg, D. E. (1988). Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, 412.

Reinelt, G. (1994). The traveling salesman: computational solutions for TSP applications. En Springer eBooks.

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.

Markowitz, H. M. (1952). Portfolio Selection. *Journal of Finance, 7*(1), 77.

Holland, J. H. (1975). Adaptation in natural and artificial systems. University of Michigan Press

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

Blum, C. (2005). Ant colony optimization: introduction and recent trends. Physics of life reviews, 2(4), 353-373.

Kennedy, J. F., & Eberhart, R. C. (2002). *Particle swarm optimization*. IEEE, Australia

Dorigo, M., Birattari, M., & Stützle, T. (2007). Ant Colony Optimization. En Chapman and Hall/CRC eBooks (pp. 417-430).

Bräysy, O., & Gendreau, M. (2005). Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. Transportation Science, 39(1), 104-118.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305.

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 9(2), 159-195.

Wilcoxon, F. (1945). "Individual comparisons by ranking methods". Biometrics Bulletin, 1(6), 80-83

Friedman, M. (1937). The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. Journal of the American Statistical Association, 32(200), 675-701.

Turing, A. M. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of The London Mathematical Society, s2-42(1), 230-265.

Jauhar, S. K., & Pant, M. (2015). Genetic Algorithms, a Nature-Inspired Tool: Review of Applications in Supply Chain Management. En *Advances in intelligent systems and computing* (pp. 71-86). Springer Nature.

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

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

Li, G., & Wang, H. (2018). Improved harmony search algorithm for global optimization.

Blazewicz, J., Kovalyov, M. Y., Musiał, J., Urbański, A., & Wojciechowski, A. (2010). Internet shopping optimization problem. International Journal of Applied Mathematics and Computer Science, 20(2), 385-390.

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.

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

Krasnogor, N., & Smith, J. G. (2005). A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues. *IEEE Transactions on Evolutionary Computation*, *9*(5), 474-488.

P. Moscato, (1989) “On evolution, search, optimization, GAs and martial arts: toward memetic algorithms,” California Inst. Technol., Pasadena, CA, Tech. Rep. Caltech Concurrent Comput. Prog. Rep. 826.

Nash, J. C. (2000). The (Dantzig) simplex method for linear programming. Computing in Science and Engineering, 2(1), 29-31.

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.