API RESTful evolutiva para optimización de rutas

Sergio Antonio de Jesús Saldaña Pacheco, Fidelio Castillo Romero, Rosa Gómez Domínguez, Miguel Pérez Vasconcelos

Tecnológico Nacional de México/Instituto Tecnológico de Villahermosa, División de Estudios de Posgrado e Investigación, Carretera Villahermosa - Frontera Km. 3.5 Ciudad Industrial Villahermosa, Tabasco, México. C.P. 86010.

Resumen

En este artículo se aplicó una de las técnicas de la inteligencia artificial para resolver el problema de optimización de rutas conocido como el problema del viajero. Este problema se caracteriza por ser difícil de resolver de manera exacta en un tiempo razonable, por esa razón, se tomó como alternativa de solución el uso de algoritmos evolutivos. Basado en la arquitectura REST, se presentó el diseño de una API que combinó la funcionalidad de Spring Boot con el poder de las metaheurísticas de MOEA Framework para resolver los casos de prueba, encontrando el valor óptimo en algunos casos y en otros, una aproximación muy cercana al óptimo. Finalmente, la API demostró la utilidad y eficiencia de los algoritmos evolutivos en la resolución de problemas de optimización TSP.

Abstract

In this article, one of the artificial intelligence techniques was applied to solve the route optimization problem known as the Travelling Salesman Problem. This problem is characterized by being difficult to solve exactly in a reasonable time, for this reason, the use of evolutionary algorithms was taken as an alternative solution. Based on the REST architecture, the design of an API was presented that combined the functionality of Spring Boot with the power of the MOEA Framework metaheuristics to solve the test cases, finding the optimal value in some cases and in others, a very close approximation. close to optimal. Finally, the API demonstrated the usefulness and efficiency of evolutionary algorithms in solving TSP optimization problems.

Palabras clave: API RESTful, Servicios Web, Metaheurística, Algoritmo Evolutivo, MOEA Framework
Keywords: API RESTful, Web Services, Metaheuristic, Evolutionary Algorithm, MOEA Framework

1. INTRODUCCIÓN

Los problemas de optimización están presentes en muchas áreas del conocimiento como la administración, negocios, ciencia y la ingeniería no es la excepción. Estos problemas tienen muchos objetivos, restricciones o espacios de búsqueda muy grandes y no se pueden resolver de una manera exacta en una cantidad de tiempo razonable. La principal alternativa para resolver esta clase de problemas es utilizar algoritmos aproximados [1]. Entre los problemas de optimización del mundo real se incluyen: asignación de tareas, planificación de sistemas de producción, planificación de rutas para el transporte, planificación de vuelos y diseño de circuitos VLSI.

Este artículo está basado en el famoso problema del viajero (TSP - Travelling Salesman Problem), el cual es un problema de optimización combinatoria de tipo NP-difícil (NP-Hard) [11], en el que dado un conjunto finito de ciudades y las distancias entre cada par de ellas hay que encontrar la ruta más corta que el viajero puede recorrer visitando cada ciudad una sola vez y regresar a la ciudad origen [2]. Además, se considera que el problema TSP es simétrico, si para cada par de ciudades (i,j), la distancia de la ciudad i a la ciudad j es la misma distancia que de la ciudad j a la i [5].

Existen técnicas de la Inteligencia Artificial que se utilizan para resolver problemas de optimización complejos como el TSP. Estas técnicas forman parte del Cómputo Evolutivo [2], se les conoce como algoritmos evolutivos

o metaheurísticas bio-inspiradas los cuales se basan en la evolución natural de las especies o en el comportamiento social de los animales o insectos [10].

Nuestro objetivo principal en este artículo es dar a conocer el diseño de una API que combina la tecnología REST con una técnica de la inteligencia artificial para obtener un servicio web con funcionalidad metaheurística, interoperable y disponible en cualquier plataforma para resolver el problema de optimización del viajero. En general, el diseño de la API se basa en una arquitectura de tres capas -capa del controlador que maneja las peticiones del cliente, capa intermedia o capa de servicio y capa del mecanismo de optimización que implementa dos algoritmos evolutivos de MOEA Framework-.

2. METAHEURÍSTICAS BIO-INSPIRADAS

La palabra metaheurística se deriva del prefijo griego *meta* que significa "metodología de nivel superior" y la heurística se deriva de la palabra griega *heuriskein* que significa el arte de descubrir nuevas estrategias (reglas) para resolver problemas. Los métodos de búsqueda metaheurística se consideran metodologías generales de nivel superior que se puede utilizar como una guía en el diseño de heurísticas para resolver problemas de optimización. Las metaheurísticas generan soluciones "aceptables" en un tiempo razonable para resolver problemas difíciles y complejos, sin garantizar que dichas soluciones sean óptimas [1].

Existen metaheurísticas basadas en la evolución natural de las especies o en el comportamiento sociales de los animales o insectos, las cuales se conocen como metaheurísticas bio-inspiradas o metaheurísticas basadas en población. Estas metaheurísticas se pueden agrupar en dos categorías: los algoritmos evolutivos y la inteligencia de enjambre (Swarm Intelligence). Dentro de la inteligencia de enjambre se encuentran PSO (Particle Swarm Optimization - Optimización por cúmulo de partículas) y ACO (Ant Colony Optimization - Optimización por colonia de hormigas) [10].

2.1 Algoritmos Evolutivos

Los algoritmos evolutivos están inspirados en la evolución natural de las especies, es decir, en el principio de supervivencia del más apto de Darwin. Los algoritmos evolutivos se basan en una población de individuos en los que se aplican de manera iterativa operadores genéticos como la selección y mutación en base a la aptitud o *fitness* de los individuos con el fin de evolucionar hacia nuevas generaciones de soluciones que incluyan a los individuos con los más altos valores de fitness [10]. En la categoría de algoritmos evolutivos se encuentran GAs (Genetic Algorithms - Algoritmos genéticos), ES (Evolution Strategies - Estrategias evolutivas), EP (Evolutionary Programming - Programación evolutiva) y GP (Genetic Programming - Programación genética) [1].

2.2 PSO (Particle Swarm Optimization - Optimización por cúmulo de partículas)

PSO es una metaheurística que simula el comportamiento social de las bandadas de pájaros o bancos de peces, la cual hoy en día se ha convertido en un método muy popular para resolver problemas de optimización multi-objetivo [9].

PSO consiste de soluciones candidatas llamadas partículas que se mueven o vuelan en el espacio de búsqueda. Una partícula tiene su propia posición y velocidad. Cada partícula se mueve de una posición a otra hacia el

óptimo global de acuerdo a la posición visitada por sí misma (pbest) y a la mejor posición visitada por el enjambre (gbest), este proceso se realiza de manera iterativa hasta encontrar el criterio de parada [1].

2.3 ACO (Ant Colony Optimization - Optimización por colonia de hormigas)

ACO simula el método de forrajeo de las hormigas las cuales utilizan un sistema de comunicación mediante rastros de feromonas para encontrar las rutas más cortas entre el nido y su comida [4]. A medida que las hormigas se mueven van dejando marcas de feromonas que forman un rastro en su trayectoria. La feromona es una sustancia olfativa y volátil. Cuanto mayor sea la cantidad de feromona en un camino particular, mayor será la probabilidad de que las hormigas seleccionen el camino [1].

Con lo anteriormente expuesto, hago mención que para el diseño de la API seleccionamos los algoritmos genéticos (GAs – Genetic Algorithms), al identificar, en base a la revisión téorica, que son los tipos de algoritmos evolutivos más populares, por lo que para este caso fue suficiente abordar estos algoritmos, aunque también se pueden usar las metaheurísticas ACO y PSO para resolver el problema del viajero [13].

3. PROBLEMAS DE OPTIMIZACIÓN

Un problema de optimización implica maximizar o minimizar una serie de funciones objetivo [12]. Matemáticamente, un problema de optimización con restricciones se puede formular de la siguiente manera:

Encontrar x que minimice o maximice $[f_0(x), f_1(x), \ldots, f_n(x)]$

Sujeto a $x \in S$

$g_j(x) \geq 0, j = 1, \ldots, J$

$h_K(x) = 0, k = 1, \ldots, K$

donde n es el número de objetivos a optimizar. S es el conjunto de las soluciones potenciales, J es el número de las restricciones de desigualdad expresadas de la forma $g_j(x) \geq 0$ y K es el número de restricciones de igualdad expresado en la forma $h_k(x) = 0$. En los problemas de optimización mono-objetivo (n=1) se debe encontrar el mejor valor (máximo o mínimo) de alguna función u objetivo. En cambio, los problemas de optimización multi-objetivo (n>1) son aquellos en los que se trata de optimizar simultáneamente varias funciones objetivo [7].

3.1 El óptimo de Pareto

El concepto de óptimo tiene sus raíces en el siglo XIX, fue propuesto inicialmente por Francis Ysidro Edgeworth y posteriormente fue generalizado por Vilfredo Pareto, conocido como *optimalidad de Pareto*. Un problema de optimización multi-objetivo implica optimizar una serie de objetivos simultáneamente los cuales entran en conflicto entre sí, es decir, la solución óptima de una función objetivo es diferente al de las otras. Al resolver tales problemas, con o sin restricciones, estos problemas dan lugar a un conjunto de soluciones, conocidas popularmente como soluciones óptimas de Pareto. Una solución es óptima de Pareto si no es posible mejorar un objetivo dado sin deteriorar al menos otro objetivo [1].

ISSN: 2007-4786

4. ARQUITECTURA PROPUESTA DE LA API RESTFUL

La API desarrollada en Java combina la potencia de Spring Boot Framework con la funcionalidad de las metaheurísticas de MOEA Framework, obteniendo como resultado un modelo de tres capas como aparece indicado en la Figura 1. Este modelo contiene dos módulos principales: el controlador REST que gestiona el acceso al API y la capa metaheurística que resuelve los problemas de optimización de rutas. La capa intermedia o capa de servicio maneja las peticiones entre el controlador y la capa metaheurística. Esta última capa mantiene el mecanismo de optimización mediante la implementación de los algoritmos evolutivos GA (Genetic Algorithm) y NSGA-II (Non-dominated Sorting Genetic Algorithm) de MOEA Framework [6].

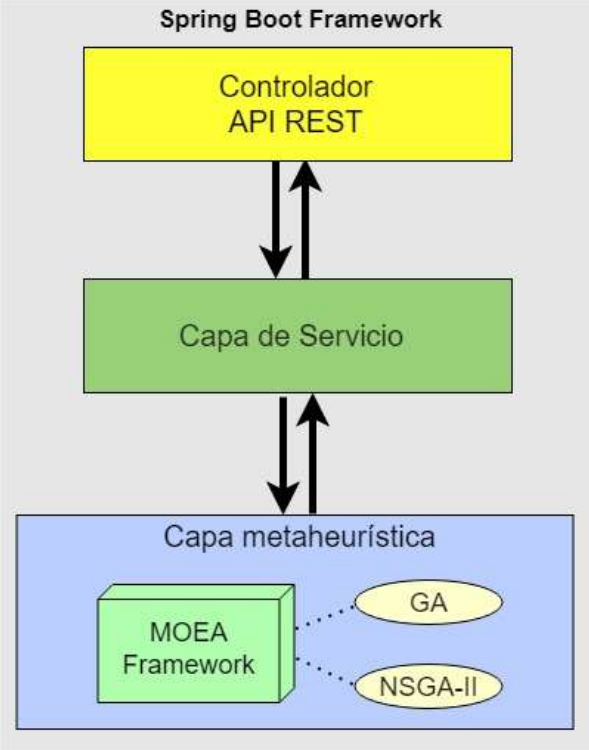

Figura 1. Modelo propuesto de la arquitectura de la API

Para el desarrollo de la API RESTful elegimos el lenguaje Java por su robustez y por la disponibilidad actual de los frameworks gratuitos para Java como Spring Boot y MOEA Framework. A continuación, se presenta un resumen de las tecnologías y herramientas que conforman la arquitectura de la API:

4.1 REST (Representational State Transfer –Transferencia de estado representacional)

REST es una arquitectura de software para desarrollo de servicios web. Mediante el protocolo http, el cliente consume un servicio en la web proporcionada por un servidor a través de los verbos http: GET, POST, PUT, PATCH y DELETE. El formato de intercambio de datos es JSON (Javascript Object Notation). El servicio es referenciado a través de un URI (Uniform Resource Identifier) el cual es un identificador de recurso único para cada servicio web de un API. Se denomina RESTful a la representación de un servicio REST.

4.2 Java

Java es un lenguaje de programación orientado a objetos, robusto, multiplataforma y ampliamente utilizado.

4.3 SPRING Boot

Spring Boot es un marco de trabajo gratuito para Java que utiliza el patrón de diseño MVC (modelo-vista-controlador). Permite un desarrollo de aplicaciones web fácil y rápido. Con este framework se puede construir un servicio REST para que los clientes lo puedan consumir a través de la web con los verbos http GET, POST, PUT, PATCH y DELETE.

4.4 Apache Maven

Maven es una herramienta gratuita para gestión de proyectos de software que se puede usar en una aplicación Web de Spring Boot como sistema de administración de dependencias.

4.5 Apache Tomcat

Tomcat es un contenedor de servlets gratuito para implementar aplicaciones desarrolladas en Java. Spring Boot contiene un servidor web Tomcat embebido.

4.6 MOEA Framework

MOEA framework es un marco de trabajo para Java que permite usar y probar diversas metaheurísticas del estado del arte para resolución de problemas de optimización multi-objetivo. Es una biblioteca gratuita y de código abierto, la cual se puede incorporar en alguna aplicación Java para utilizar las metaheurísticas del framework [6].

5.- FUNCIONAMIENTO DE LA API RESTFUL

5.1 Consumo del servicio web

Para el consumo del servicio web de la API se consideran los verbos http GET y POST. La forma general del URI es: *http://localhost:8080/api/evolutiva/<servicio>.* Donde <servicio> puede ser "*instancias*" o "*rutaoptima*" como viene indicado en *las secciones siguientes.*

5.1.1 Método GET

Con una petición GET se obtiene el listado de instancias TSP que se pueden optimizar con la API. La Figura 2 indica el URI correspondiente mediante el cliente POSTMAN:

ISSN: 2007-4786

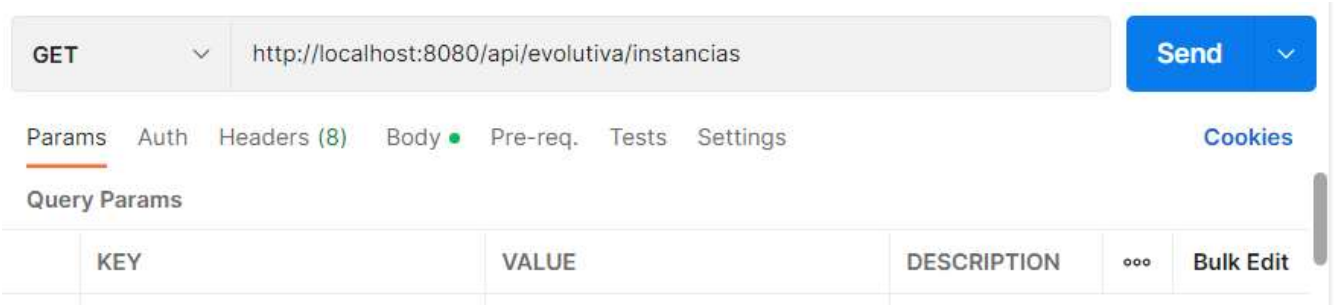

Figura 2. Petición GET para obtener lista de instancias TSP

Un fragmento del resultado de la petición GET anterior se muestra en la Figura 3.

Figura 3. Fragmento del resultado de la petición GET.

5.1.2 Método POST

Para obtener el cálculo de la ruta óptima de una instancia TSP se utiliza la petición POST con el URI correspondiente y un formato de entrada JSON que incluye como parámetros el nombre de la instancia TSPLIB y el número de evaluaciones de la función objetivo, como lo indica el ejemplo de la Figura 4 mediante el cliente POSTMAN:

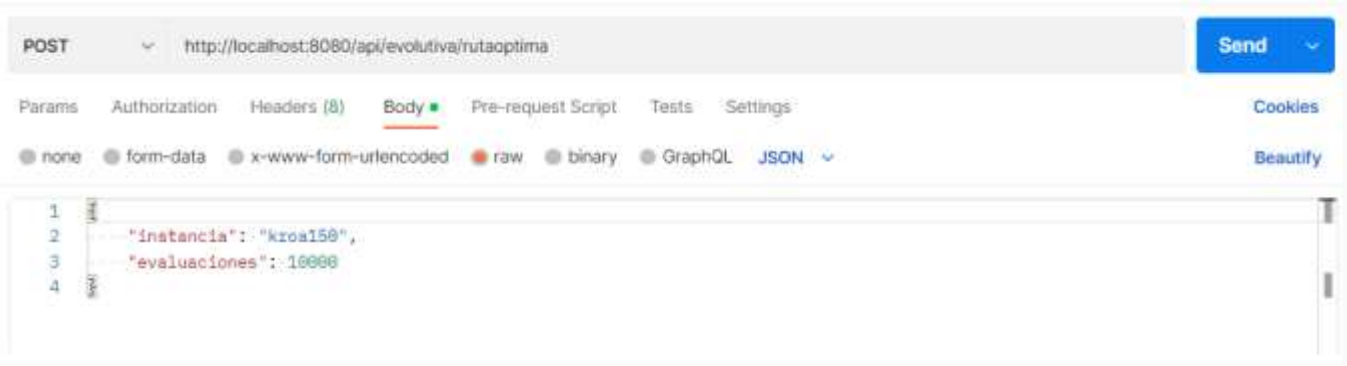

Figura 4. Petición POST para optimizar la ruta kroa150.tsp

El resultado de la petición anterior se muestra en la Figura 5.

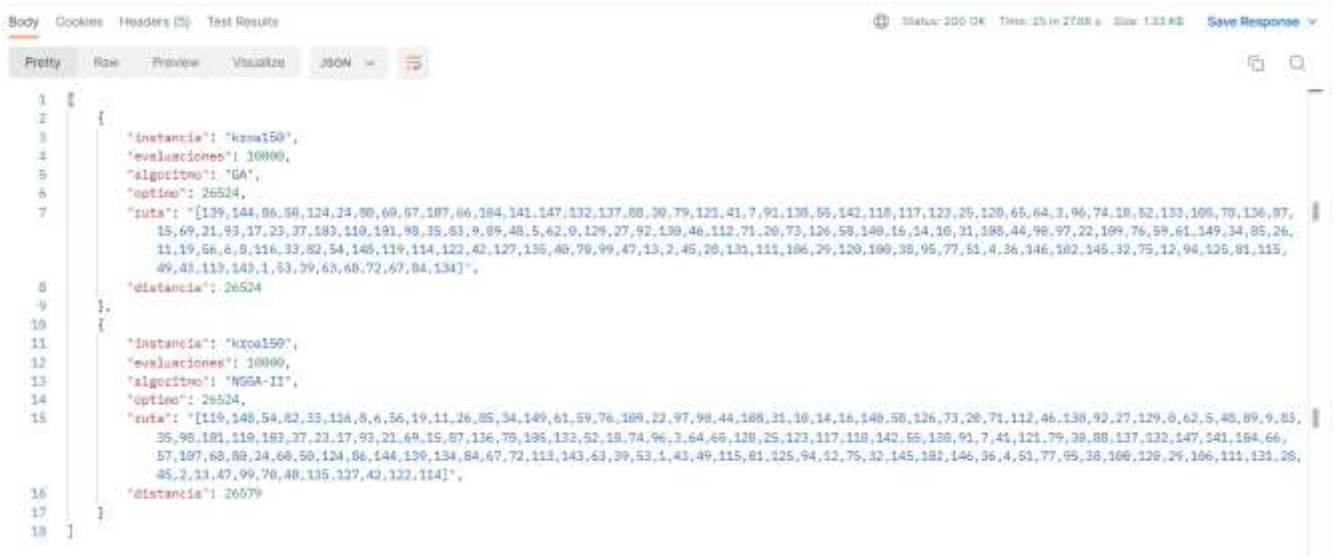

Figura 5. Resultado de la petición POST de la instancia kroa150

La clave *distancia* es la mínima distancia que el viajero recorre en su viaje y la clave *ruta* es la lista de las ciudades recorridas por el viajero según la instancia TSP especificada (kroa150). La clave "optimo" indica el valor óptimo de la ruta publicado en el sitio web TSPLIB [8], mostrado únicamente como referencia para comparación con el resultado de la metaheurística.

5.2 Manejo de errores

Para el manejo de errores en el uso de la API, se utilizan los siguientes códigos comunes en respuesta a alguna petición http:

200 Ok. Respuesta estándar para peticiones exitosas.
400 Petición incorrecta (Bad Request). La solicitud contiene sintaxis errónea en el formato JSON.
404 No encontrado (Not Found). Cuando el servidor web no encuentra la página o recurso solicitado.
405 Método no permitido (Method not allowed). Cuando se intenta una operación no permitida por el URI.

5.3 Mecanismo del algoritmo evolutivo

De forma general, los algoritmos evolutivos implementados en la API funcionan de la siguiente manera: como primer paso generan una población inicial que sería un conjunto de cromosomas. Para nuestra práctica, se considera que un cromosoma es una ruta y un gen significa una ciudad dentro de esa ruta. Posteriormente, se aplica a esta población los operadores genéticos de selección, cruza (crossover) y mutación (mutation), para generar nuevas soluciones o descendencia para las próximas generaciones. La cruza combina dos o más padres para crear una descendencia y la mutación involucra un sólo padre [6]. Se realiza un cálculo de fitness a cada individuo de la población para saber quién es el más apto [3]. De esta forma, las ciudades se recombinan entre rutas para elegir la solución más apta. Esto se hace de manera iterativa hasta alcanzar el criterio de parada predefinido como lo indica la Figura 6.

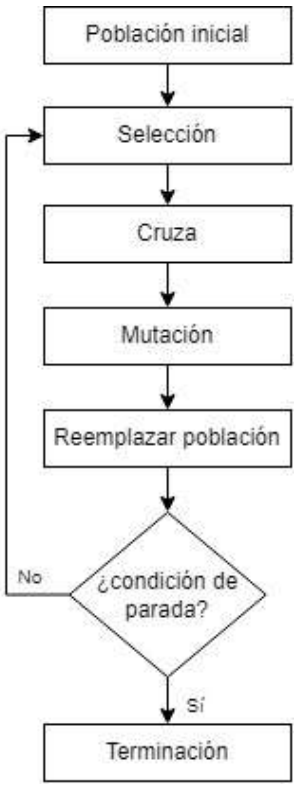

Figura 6. Diagrama de flujo del algoritmo evolutivo [6]

Por lo general, el usuario escoge el criterio de parada, el cual puede ser un número predeterminado de generaciones o puede ser tan pronto se encuentre una solución objetivo [12].

5.4 Programación con MOEA Framework

MOEA Framework incluye un gran número de algoritmos para resolver diversos problemas de optimización. A continuación, se muestra un extracto del código del mecanismo evolutivo de MOEA Framework para resolver el problema de optimización de ruta.

```
1  Problem problem = new TSPProblem(instance);
2  NondominatedPopulation executor = new Executor ()
3                                    .withAlgorithm("NSGA-II")
4                                    .withProblem(problem)
5                                    .withMaxEvaluations(10000)
6                                    .distributeOnAllCores()
7                                    .run();
```

En la línea 1 se crea el problema instanciando la clase TSPProblem que usa el parámetro instance, el cual es un archivo TSP de la Tabla 1. En la línea 2 se crea una instancia de la clase Executor. De la línea 3 a la 5, se especifica el nombre del algoritmo, el problema, el número de evaluaciones que va a tener la función objetivo. La línea 6 habilita el procesamiento en paralelo de las evaluaciones de la función objetivo en todos los núcleos de la computadora. Con la línea 7 se ejecuta el algoritmo [6].

Una vez finalizada la ejecución, se obtiene el resultado de la variable que es la ruta recorrida y el objetivo (la distancia mínima del viaje). La API convierte la respuesta en formato JSON y se devuelve al cliente, tal y como aparece en el ejemplo de la Figura 5.

6. RESULTADOS

Las pruebas realizadas en la API RESTful se usaron para resolver problemas TSP simétricos de una variable y un sólo objetivo. La variable es una permutación que define las ciudades visitadas por el agente viajero. El objetivo es minimizar la distancia total recorrida por el viajero. El primer paso del algoritmo evolutivo es, como ya vimos, generar una población inicial, la cual se obtiene a partir de la instancia seleccionada de la Tabla 1. El formato TSPLIB especifica la distancia del plano euclidiano entre cada par de ciudades, el tipo de instancia es EUC_2D (distancia euclidiana en 2D).

Tabla 1. Instancias de la librería TSPLIB [8]

Instancia	Ciudades	Tipo
berlin52	52	EUC_2D
bier127	127	EUC_2D
ch130	130	EUC_2D
ch150	150	EUC_2D
eil101	101	EUC_2D
kroa100	100	EUC_2D
kroa150	150	EUC_2D

Se utilizaron las metaheurísticas GA (Genetic Algorithm) y NSGA-II (Non-dominated Sorting Genetic Algorithm) de MOEA Framework para el mecanismo de optimización de la API. Todas las pruebas se realizaron en una computadora con procesador de cuatro núcleos y 8 Gb RAM en sistema operativo propietario de 64 bits. Para cada prueba se tomaron en cuenta 1,000 y 10,000 evaluaciones, se realizaron 2 corridas por instancia, el tamaño de la población está definida por el número de ciudades de la instancia TSPLIB. La tabla 2 muestra los resultados de las pruebas.

Tabla 2. Resultados de las pruebas con la API RESTful

Instancia	Ciudades	Evaluaciones	Óptimo	GA	NSGA-II
berlin52	52	1,000	7542	7542	7542
berlin52	52	10,000	7542	7542	7542
bier127	127	1,000	118282	119473	118882
bier127	127	10,000	118282	118664	118423
ch130.tsp	130	1000	6110	6239	6200
ch130.tsp	130	10,000	6110	6137	6110
ch150.tsp	150	1,000	6528	6645	6627
ch150.tsp	150	10,000	6528	6550	6528
eil101.tsp	101	1,000	629	642	642
eil101.tsp	101	10,000	629	635	630
kroa100.tsp	100	1,000	21282	21305	21319
kroa100.tsp	100	10,000	21282	21282	21282
kroa150.tsp	150	1,000	26524	26964	26905
kroa150.tsp	150	10,000	26524	26524	26579

7. DISCUSIÓN Y CONCLUSIONES

El presente trabajo se centró principalmente en el diseño de una API RESTful para la resolución de problemas TSP mediante algoritmos evolutivos. También, se resaltó la importancia de las metaheurísticas bio-inspiradas basadas en población en la solución de problemas de optimización, aportando una visión general de cada una de ellas. Se utilizó la implementación de los algoritmos evolutivos GA y NSGA-II de MOEA Framework como mecanismo de optimización de las rutas TSP. Como resultado, las metaheurísticas lograron encontrar soluciones factibles, en algunos casos encontraron el óptimo y en otros, un valor muy cercano al óptimo. Cabe resaltar que los algoritmos evolutivos utilizados en las pruebas son de naturaleza estocástica, usan reglas aleatorias durante la búsqueda, por lo que no siempre dan el mismo resultado cada vez que se ejecutan [1]. Todas las soluciones de las pruebas se generaron en tiempos computacionales cortos y en la práctica pueden resultar muy útiles para los responsables involucrados en la toma de decisiones.

Finalmente, mediante las metaheurísticas basadas en población se puede concluir que es posible optimizar cualquier ruta que se desee si se cuenta con la información de las distancias entre cada punto, comprobándose la utilidad y eficiencia de los algoritmos evolutivos en la resolución de problemas TSP. La API propuesta sienta las bases para resolver problemas de optimización multi-objetivo en diversas áreas de investigación; permite a investigadores, docentes, estudiantes o comunidades de desarrollo de tecnología realizar pruebas de optimización fáciles y gratuitas, ahorrando tiempo y esfuerzo; incentiva y aporta un diseño novedoso con tecnologías robustas, confiables y actualizadas, además de reducir la curva de aprendizaje en los temas tratados.

REFERENCIAS

[1] El-Ghazali Talbi (2009). Metaheuristics: from Design to Implementation. John Wiley & Sons, Inc.
[2] Nosheen Qamar, Nadeem Akhtar, Irfan Younas (2018). Comparative Analysis of Evolutionary Algorithms for Multi-Objective Travelling Salesman Problem. (IJACSA) International Journal of Advanced Computer Science and Applications.
[3] Ehsan Ahmadi, Gürsel A. Süer, Farah Al-Ogaili (2018) Solving Stochastic Shortest Distance Path Problem by Using Genetic Algorithms. Complex Adaptive Systems Conference with Theme: Cyber Physical Systems and Deep Learning, CAS 2018, 5 November – 7 November 2018, Chicago, Illinois, USA
[4] Ajith Abraham, He Guo, Hongbo Liu (2006). Swarm Intelligence: Foundations, Perspectives and Applications.
[5] Jihene Kaabi & Youssef Harrath (2019). Permutation rules and genetic algorithm to solve the travelling salesman problem. Arab Journal of Basic and Applied Sciences.
[6] David Hadka (2022). Beginner's Guide to the MOEA Framework. Obtenido de http://moeaframework.org/documentation.html el 27 Febrero 2023.
[7] Carlos Segura, Carlos A. Coello Coello, Gara Miranda, Coromoto León (2016). Using multi-objective evolutionary algorithms for single-objective constrained and unconstrained optimization.
[8] Gerhard Reinelt. TSPLIB. Obtenido de http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ el 24 Febrero 2023
[9] A.J. Nebro, J.J. Durillo, J. García-Nieto, C.A. Coello Coello, F. Luna and E. Alba (2009). SMPSO: A new PSO-based metaheuristic for multi-objective optimization.
[10] Harith Al-Sahaf, Ying Bi, Qi Chen, Andrew Lensen, Yi Mei, Yanan Sun, Binh Tran, Bing Xue and Mengjie Zhang (2019) A Survey on Evolutionary Machine Learning.
[11] Kusum Deep, Hadush Mebrahtu (2011). Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem.
[12] Kalyanmoy Deb (2011). Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction.
[13] Mohammad Reza Bonyadi, Mostafa Rahimi Azghadi, Hamed Shah-Hosseini (2008).Travelling Salesman Problem. Edited by Federico Greco.

Correo de autor: ***sergio_jsp@hotmail.com***