Universidad Nacional Autónoma de México

Facultad de Ciencias

Heurísticas de optimización combinatoria

Proyecto 1: Recocido simulado

Semestre 2026-1

Alumno:

Medina Peralta Joaquín—— 320202513

Fecha de Entrega: 21 de septiembre de 2025

Introducción

Para este proyecto introductorio a las heuristicas de optimización combinatoria, se decidió implementar el recocido simulado por umbrales para resolver el problema del agente viajero, siendo un clásico problema de teoría de gráficas y un problema NP-completo, es decir, no existe un algoritmo que lo resuelva de manera óptima en un tiempo razonable en instancias de ciudades grandes.

De este modo, se estará trabajando en instancias de tamaño 40 y 150, donde el tamaño de la gráfica es 40! y 150! respectivamente, haciendo usando el algoritmo de dijkstra con complejidad de $O(|E|log(|V|))$, se estaría tomando el tiempo de la edad del universo para poder resolverse.

Por lo que, usando la heuristica propuesta, podremos obtener trayectorias con resultados óptimos en poco tiempo pero sin asegurar que esa solución al problema sea la mejor para esa instancia de ciudades. Por último, la implementación de la heuristica será realizado en el lenguaje de programación `Rust` por su ventaja del uso de vectores y su velocidad para acceder en memoria estos elementos.

Problema

Para este proyecto, se obtendrá soluciones factibles para el problema del agente viajero (*TSP* por sus siglas en ingles). Es un problema común en teoría de gráficas en donde se busca un ciclo hamiltoniano que pase por n ciudades (en nuestro será una trayectoria hamiltoniana), considerado un problema con complejidad `NP` completo al tener la característica de que el número de posibles soluciones crece exponencialmente con el número de nodos (por ejemplo el algoritmo de Dijkstra). (Espinosa Téllez et al., 2016)

De este modo, se define el conjunto finito de n ciudades $V = \{1, 2, \ldots, n\}$ y un conjunto de caminos que unen cada una de las ciudades dada por $(v_i, v_j) \in E$. Por lo que cada par de ciudades pueden estar comunicadas o no y su distancia se define como $C_{i,j}$ y una variable binaria $x_{i,j}$ que indica si existe el camino de ir a una ciudad v_i a v_j, por lo que tenemos la función objetivo dada por: (Espinosa Téllez et al., 2016)

$$T(V) = min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{i,j}$$

Sujeto a las restricciones para garantizar que se sale de cada ciudad exactamente una vez:

$$\sum_{i=1, i \neq j}^{n} x_{i,j} = 1 \ \forall j = 1 \ldots n$$

$$\sum_{j=1, i \neq j}^{n} x_{i,j} = 1 \ \forall i = 1 \ldots n$$

Por lo tanto, buscamos el recorrido que pase por todas las ciudades solo una vez. Y de esta manera, podríamos verificar todas las permutaciones de las ciudades hasta encontrar la mejor

solución, pero esto nos da una complejidad de $O(n!)$ por lo que en una instancia de 40 y de 150, no es posible por fuerza bruta.

A su vez, consideremos que estaremos trabajando con ciudades del mundo real, por lo que encontrar el costo $C_{i,j}$ de dos ciudades es por medio de su distancia dada por sus coordenadas, tendremos que: (Villamizar de la Hoz, 2025)

$$d(v,u) = 2 \cdot r \cdot \arcsin(\sqrt{A, 1-A})$$
$$A = \sin^2(\frac{\phi_v - \phi_u}{2}) + \cos(\phi_v) \cdot \cos(\phi_u) \cdot \sin^2(\frac{\lambda_v - \lambda_u}{2})$$

Donde:

- r es el radio de la esfera (para la Tierra $r \approx 6371$ km.
- ϕ_v, ϕ_u son las latitudes de las ciudades v, u en radianes.
- λ_v, λ_u son las longitudes de las ciudades v, u en radianes.

De este modo, las ciudades en el mapa y sus conexiones estarán representadas por una gráfica ponderada $G(E,V), E \subset V \times V$, con la función de peso que representará la distancia entre cada par de ciudades.

Así, sea $S \subset V$, donde S es la instancia de TSP que se quiere resolver. Entonces usaremos una gráfica completa $G_S(V_S, E_S)$ donde $V_S = S$ y $E_S = \{(u,v) | u, v \in S \text{ y } u \neq v\}$, con la función de peso aumentada $w_S : E_S \to \mathbf{R}^+$ definida como:

$$w_S(u,v) = \begin{cases} d(u,v), & \text{si } (u,v) \in E, \\ d(u,v) \times \text{máx}_d(S), & \text{en otro caso.} \end{cases}$$

Donde:

- $\text{máx}_d(S)$ es la distancia máxima de S donde $\text{máx}\{d(u,v) | u, v \in S \text{ y } (u,v) \in E\}$

De este modo este diseño propuesto (Peláez Valdés, 2025), alentará al sistema a descartar estas aristas que no estén en G para encontrar soluciones factibles al tener un peso mucho mayor que las aristas en E, por lo que podremos comparar las evaluaciones a soluciones factibles de instancias de TSP diferentes. Así, proponemos la siguiente función de costo de una permutación $P = v_{p(1)}, \ldots, v_{p(k)}$ con $k = |S|$ como:

$$f(P) = \frac{\sum_{i=2}^{k} w_S(v_{p(i-1)}, v_{p(i)})}{\mathcal{N}(S)}$$

Donde:

- Para cada par no ordenado $u, v \in S$, si $(u,v) \in E$, se agrega a una lista L la cual se ordena del mayor a menor, entonces sea la sublista L de los primeros $|S|-1$ elementos, entonces:

$$\mathcal{N}(S) = \sum_{d \in L'} d$$

Una última consideración del problema, tendremos que considerar el concepto de vecino de una solución, de este modo consideremos dos instancias P y P' de los elementos de S, entonces diremos que $P = v_{p(1)}, \ldots, v_{p(k)}$ es un vecino de $P' = v_{p'(1)}, \ldots, v_{p'(k)}$ si y solo si, existen dos índices $1 \leq s,t \leq k$, $s \neq t$, tales que:

- $v_{p(i)} = v_{p'(i)}$ si $i \notin \{s, t\}$
- $v_{p(s)} = v_{p'(t)}$ y $v_{p(t)} = v_{p'(s)}$

Heurística

Para esto, usaremos una variante de la heurística del *recocido simulado*, la cual fue propuesta en 1989 por **Gunter Dueck** y **Tobias Scheuer**, donde toma base de la idea de la metalurgia y de la termodinámica cuando el hierro fundido es enfriado muy despacio, tiende a solidificarse en una estructura de energía mínima. (Pérez de Vargas Moreno, 2014)

La idea general es hacer una búsqueda local, donde al principio teniendo una solución x donde sea su vecindario el espacio de búsqueda y un vecino x' se va aceptando soluciones no factibles para poder el espacio mientras que la temperatura disminuye, para posteriormente al tener una temperatura baja, solo se aceptarán soluciones factibles.

Para poder aceptar una solución, tendremos que el umbral de aceptación T_i que define el máximo empeoramiento $f(x') - f(x)$ aceptable. Entonces si esta diferencia es mayor que el umbral, es rechazada, mientras que si es menor a la actual, será sustituida por la nueva. A su vez, tendremos que existe la mejor solución encontrada hasta ahora denotada por x_{best}, si la solución x' tiene menor costo que x_{best} entonces es reemplazada por está misma.

Otra consideración a tomar en cuenta, será una α que será el factor de decrecimiento definido como $\alpha \in (0, 1)$ de nuestra temperatura donde un valor más cercano a 1 hará que el sistema se enfríe muy lentamente encontrando muy buenas soluciones pero tardando bastante, mientras que más cercano al 0, para todo lo contrario. Así, tendremos que:

$$T_{i+1} = \alpha T_i$$

De este modo, haremos una aceptación por un tamaño de lote denotado como $L \in \mathrm{N}^+$, donde el algoritmo no determinista para calcular un lote está dada por (Peláez Valdés, 2025):

Algoritmo 1 Algoritmo para calcular un lote

```
1: procedure CalculaLote(T, s)                    ▷ Temperatura T, solución s
2:     c ← 0
3:     r ← 0.0
4:     while c < L do
5:         Construir una solución candidata s'
6:         if f(s') ≤ f(s) + T then
7:             s ← s'                                  ▷ Aceptamos el candidato
8:             c ← c + 1
9:             r ← r + f(s')
10:        end if
11:    end while
12:    Salida: r/L, s   ▷ Promedio de las soluciones aceptadas y última solución aceptada
13: end procedure
```

Consideremos un vale ϵ dada por $\epsilon \in [0, \infty)$ junto con nuestro factor de enfriamiento α. Entonces tendremos nuestro procedimiento de aceptación por umbrales como (Peláez Valdés, 2025):

Algoritmo 2 Algoritmo para calcular un lote

```
1: procedure AceptacionUmbrales(T, s)          ▷ Temperatura T, solución inicial s
2:     p ← 0
3:     while T < ε do
4:         q ← ∞
5:         while p ≤ q do
6:             q ← p
7:             p, s ←CALCULALOTE(T, s)
8:         end while
9:         T ← αT
10:    end while
11:    Salida: s                                       ▷ Mejor solución aceptada
12: end procedure
```

De esta manera, tendremos el algoritmo general definido para el problema de TSP por aceptación por umbrales.

Diseño de la solución

Tecnologías a usar

En este proyecto se utilizaron las siguientes tecnologías y librerías, seleccionadas por sus ventajas específicas en el desarrollo de aplicaciones concurrentes, seguras y eficientes en Rust.

- **Rust:** Lenguaje de programación moderno orientado a sistemas, conocido por su *memory safety* sin recolector de basura y alto rendimiento (Rust Foundation, s.f.).
- **Tokio:** Runtime asíncrono para Rust que permite manejar tareas concurrentes de forma eficiente mediante *async/await* (Tokio Project Contributors, s.f.).
- **dotenvy:** Librería para cargar variables de entorno desde archivos '.env' en Rust (dotenvy Developers, s.f.). Facilita la configuración de la aplicación, permitiendo separar parámetros sensibles o específicos del entorno del código fuente.
- **SQLx:** Librería para acceso a bases de datos en Rust con soporte para SQL estático y validación en tiempo de compilación (SQLx Project Contributors, s.f.). Proporciona seguridad y eficiencia al interactuar con bases de datos relacionales, garantizando que las consultas sean correctas antes de ejecutar la aplicación.
- **Chrono:** Librería para manejo de fechas y tiempos en Rust (Chrono Developers, s.f.). Permite manipular zonas horarias, formatear fechas y calcular intervalos de tiempo de manera precisa, útil para registros de eventos, logs y control de caducidades.

Implementación del sistema

De este modo, tendremos la siguiente estructura para el proyecto con la comunicación entre los distintos paquetes, usando la programación orientada a objetos para la heurística, el manejo de la base de datos por medio de los controladores, los modelos y algunas clases para generar la gráfica de costo con número de soluciones aceptadas, leer la entrada del programa y escribir los reportes:

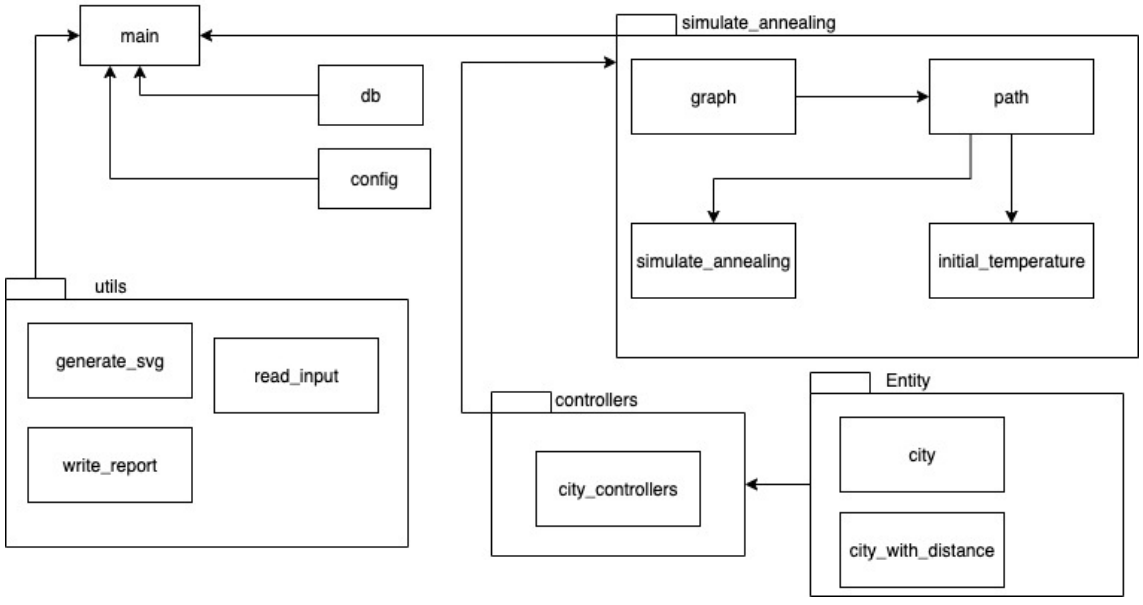

Figura 1: Estructura para la implementación del proyecto.

Experimentación

Para esto, se genero un archivo `.env` por el cual tiene como formato:

```
INITIAL_TEMPERATURE=1000.0
COOLING_RATE=0.95
SIZE_LOTE=4000
DATABASE_URL=sqlite:./tests/sql/tsp.bin
SQL_PATH=./data/tsp.sql
E_S=0.00001
E_P=0.00001
PERCENTAGE=0.65
LIMIT=1000
N=2000
```

Donde:

- INITIAL_TEMPERATURE: es la temperatura inicial del sistema.
- COOLING_RATE: factor de enfriamiento del sistema.
- SIZE_LOTE: tamaño del lote.
- DATABASE_URL: dirección en memoria de la base de datos.
- SQL_PATH: dirección en memoria del archivo .sql si es necesario construir la base de datos.
- E_S: la ϵ de nuestro sistema para el recocido simulado
- E_P: la ϵ de nuestro sistema para el calculo de la temperatura inicial.
- PERCENTAGE: porcentaje de aceptados para la temperatura inicial.
- LIMIT: el factor para calcular el limite dado por SIZE_LOTE$\times$LIMIT.
- N: el número de iteraciones N para calcular vecinos al calcular la temperatura inicial.

Para la instancia de 150 ciudades se mantiene la información del .env, mientras que la instancia de 40 ciudades, se cambio

En este sentido, para la instancia de 40 ciudades, se encontró que la semilla 21 generó una solución con un costo de 0.303441 en un tiempo de 12.587.

De este modo, se genero la experimentación de 4000 semillas obteniendo la siguiente tabla con los mejores 5 resultados de la experimentación junto con su tiempo de ejecución para la instancia de 150 ciudades:

Tabla 1: Resultados de ejecución por semilla

Semilla	**Tiempo (s)**	**Costo**
3750	229.802	0.141812
3078	216.332	0.142119
1444	214.348	0.144047
1264	198.625	0.144625
882	126.807	0.145168

A su vez, se realizó la experimentación en el siguiente equipo con las características dadas por:

```
Arquitectura:                            x86_64
modo(s) de operacion de las CPUs:        32-bit, 64-bit
Address sizes:                           39 bits physical, 48 bits virtual
Orden de los bytes:                      Little Endian
CPU(s):                                  8
Lista de la(s) CPU(s) en linea:          0-7
ID de fabricante:                        GenuineIntel
Nombre del modelo:                       Intel(R) Core(TM) i5-8250U CPU @
   1.60GHz
Familia de CPU:                          6
Modelo:                                  142
Hilo(s) de procesamiento por nucleo:     2
Nucleo(s) por socket:                    4
Socket(s)                                1
Revision:                                10
CPU(s) scaling MHz:                      22%
CPU MHz max.:                            3400,0000
CPU MHz min.:                            400,0000
BogoMIPS:                                3600,000
Modo(s) NUMA:                            1
CPU(s) del nodo NUMA 0:                  0-7
```

Resultados

Como podemos observar en las siguientes gráficas, se obtuvieron los siguientes resultados de nuestro sistema con las 5 semillas para la instancia de 150, con esto podemos darnos cuenta que es necesario modificar los parámetros.

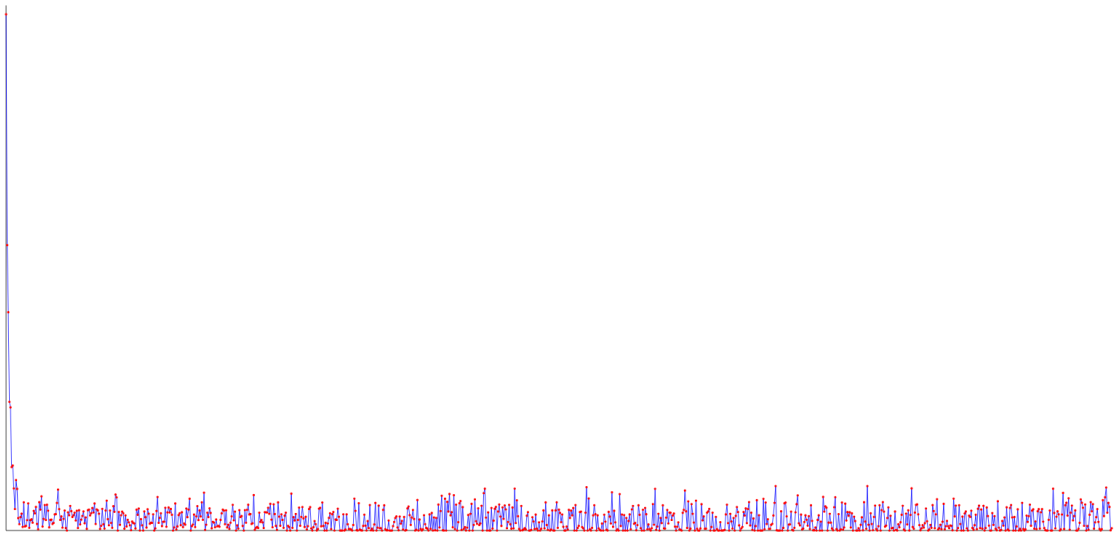

Figura 2: Resultado con la semilla 882

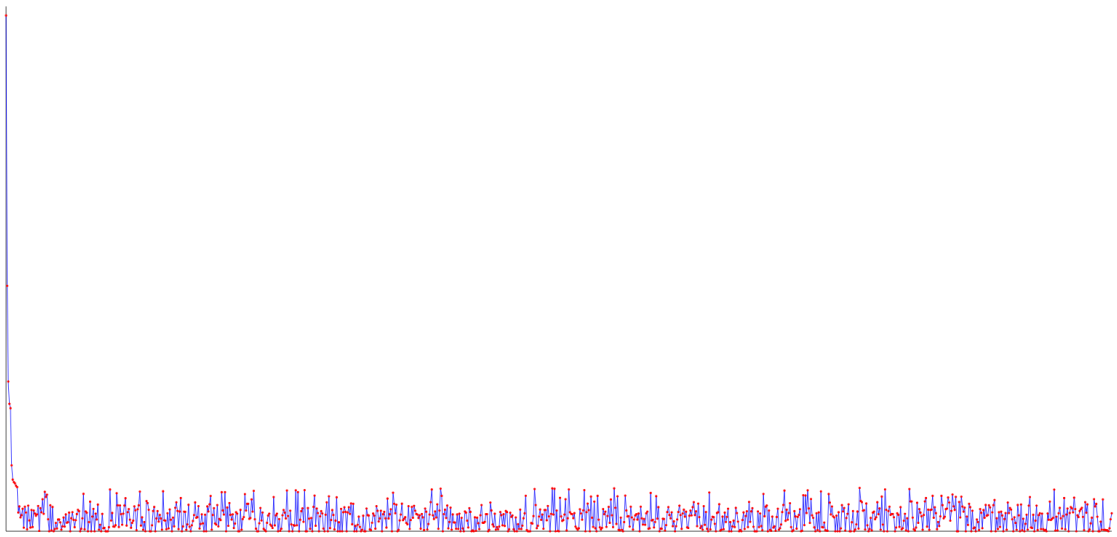

Figura 3: Resultado con la semilla 1264

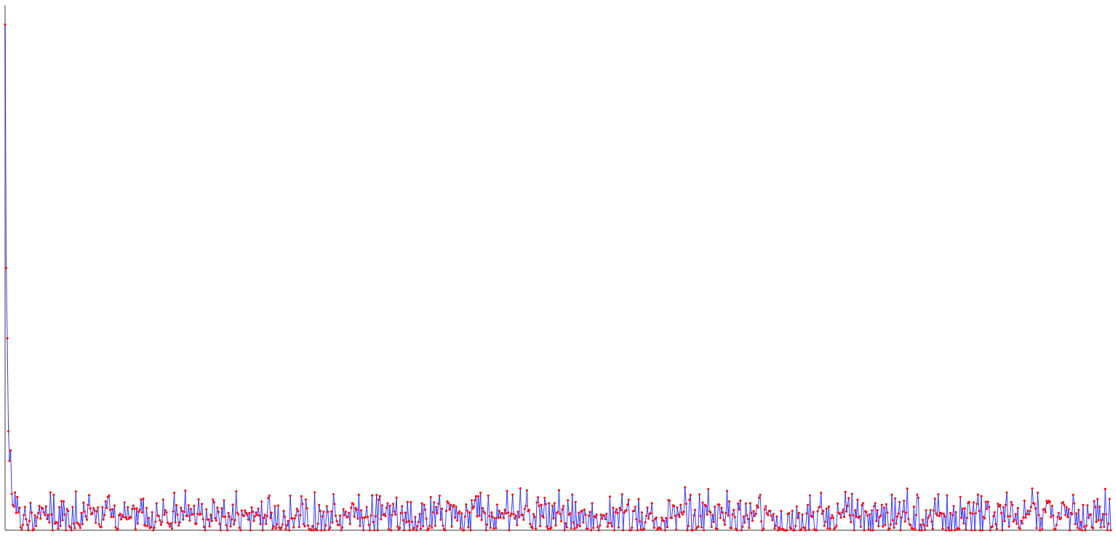

Figura 4: Resultado con la semilla 1444

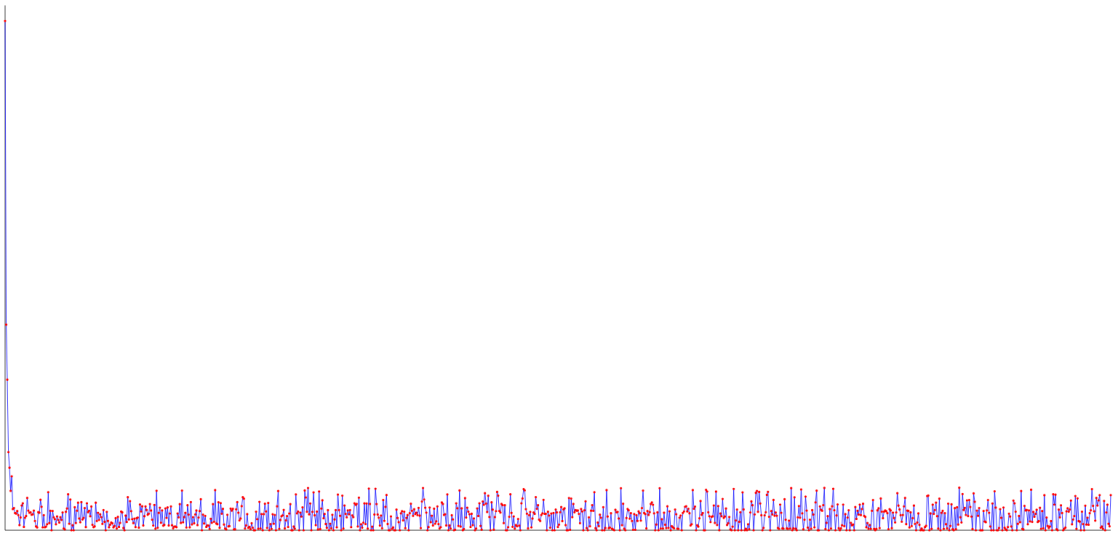

Figura 5: Resultado con la semilla 3078

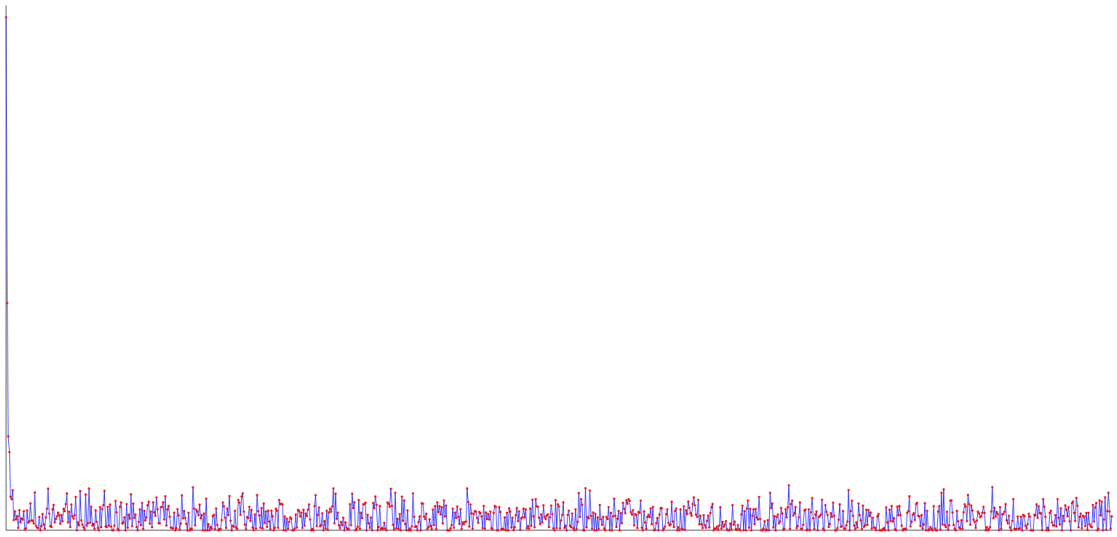

Figura 6: Resultado con la semilla 3750

Para la instancia de 40 ciudades con la semilla 21, se encontró la siguiente:

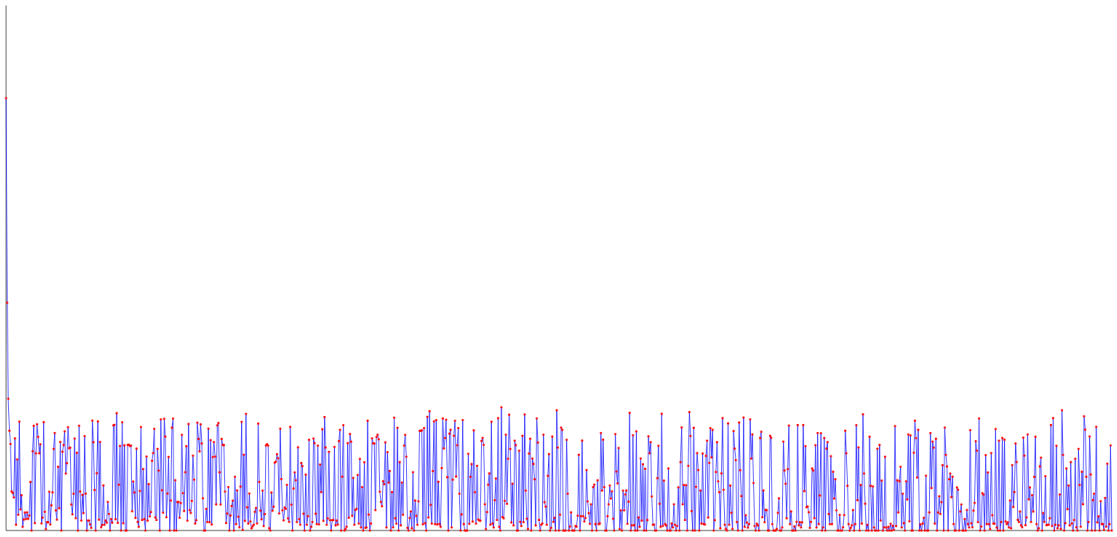

Figura 7: Resultado con la semilla 21

Conclusión

Podemos observar que la implementación de nuestro sistema es bastante lento, esto debido a que al momento de calcular el número de aceptados, tenemos un limite bastante alto pero que nos ayuda a obtener mejores soluciones. De este modo, se debe de modificar los demás parámetros para disminuir el número de iteraciones y el tiempo.

Tendríamos que revisar en la implementación del código, cuales son los lugares en donde nuestro sistema se esta ralentizando o ver la manera de encontrar otra diseño. De este modo, para posterior mantenimiento se intentará hacerlo para poder mejorar los tiempos del presente documento.

Referencias

Chrono Developers. (s.f.). Chrono: Date and time library for Rust [Accedido: 2025-09-21]. https://crates.io/crates/chrono

dotenvy Developers. (s.f.). dotenvy: Load environment variables in Rust [Accedido: 2025-09-21]. https://crates.io/crates/dotenvy

Espinosa Téllez, E. G., Sánchez Rodríguez, O., & Orlando Bernal, J. (2016). Problema del agente viajero [Accedido: 20 de septiembre de 2025]. *Ingenciencia*, *1*(2), 57-65. https://revistas.ucentral.edu.co/index.php/Ingeciencia/issue/view/28

Peláez Valdés, C. (2025, agosto). Seminario: Heurísticas de Optimización Combinatoria [Accedido: 21 de septiembre de 2025].

Pérez de Vargas Moreno, B. (2014, diciembre). *Resolución del problema del viajante de comercio (TSP) y su variante con ventanas de tiempo (TSPTW) usando métodos heurísticos de búsqueda local* [Trabajo de fin de grado en Ingeniería en Organización Industrial]. https://core.ac.uk/outputs/211096990

Rust Foundation. (s.f.). The Rust Programming Language [Accedido: 2025-09-21]. https://www.rust-lang.org/

SQLx Project Contributors. (s.f.). SQLx: Async SQL toolkit for Rust [Accedido: 2025-09-21]. https://crates.io/crates/sqlx

Tokio Project Contributors. (s.f.). Tokio: An asynchronous runtime for Rust [Accedido: 2025-09-21]. https://tokio.rs/

Villamizar de la Hoz, A. (2025, marzo). La distancia de Haversine: Una herramienta clave para el aprendizaje automático con datos geoespaciales [Accedido: 21 de septiembre de 2025]. https://www.linkedin.com/pulse/la-distancia-de-haversine-una-herramienta-clave-para-alfonso-5jhse/