Problema del Agente Viajero

Adrián González, Antonio Bueno, Álvaro López

Investigación de Operaciones

Profesora Dra. Mayra Núñez López

5 de febrero de 2025

Índice

1. Introducción

El problema del Agente Viajero es uno de los problemas más estudiados en matemáticas computacionales. Este proyecto tiene como objetivo resolver dicho problema para un conjunto de ciudades elegidas entre Qatar, Uruguay y Zimbabue, aplicando heurísticas, metaheurísticas y metodologías híbridas. Con ello, se busca encontrar la solución más eficiente en términos de tiempo y costo.

Datos

- n: Número de ciudades.
- d_{ij}: Costo (o distancia) entre las ciudades i y j, donde $d_{ij} \geq 0$ para $i, j = 1, \ldots, n$.

Variables de decisión

- $x_{ij} \in \{0, 1\}$:

$$x_{ij} = \begin{cases} 1, & \text{si el trayecto de } i \text{ a } j \text{ es parte del recorrido,} \\ 0, & \text{en caso contrario.} \end{cases}$$

- $u_i \in \mathbb{R}$: Orden relativo en el que la ciudad i es visitada en el recorrido, para $i = 1, \ldots, n$.

Modelo de Programación Lineal

Función objetivo

Minimizar el costo total del recorrido:

$$\text{Min } z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot x_{ij}.$$

Restricciones

1. Cada ciudad debe tener exactamente una salida:

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

2. Cada ciudad debe tener exactamente una entrada:

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

3. Restricciones de eliminación de subciclos (MTZ):

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \quad \forall i, j = 2, \ldots, n;\ i \neq j.$$

4. Restricción de dominio de las variables auxiliares:

$$1 \leq u_i \leq n \quad \forall i = 2, \ldots, n.$$

5. Restricción de simetría:

$$x_{ii} = 0 \quad \forall i = 1, \ldots, n.$$

Notas

- La restricción (3) MTZ asegura que no se formen subciclos que no incluyan todas las ciudades.
- El modelo asume que $u_1 = 1$ como ciudad inicial para simplificar.

Restricciones MTZ

Estas restricciones introducen variables auxiliares u_i, que representan el orden de visita de cada nodo.

- u_i: Orden de visita del nodo i, con $2 \leq u_i \leq n$ para $i \in \{2, 3, \ldots, n\}$.

la restricción (3) tiene la siguiente interpretación

- Si $x_{ij} = 1$, el nodo j sigue inmediatamente al nodo i, es decir, $u_i + 1 = u_j$.
- Si $x_{ij} = 0$, no se imponen restricciones entre u_i y u_j.

Estas restricciones aseguran que el recorrido sea único y no incluya subciclos, ya que los valores de u_i imponen un orden secuencial en los nodos.

2. Metodología

2.1. Justificación teórica

Se utilizarán heurísticas y metaheurísticas debido a la complejidad computacional del problema del Agente Viajero (TSP por sus siglas en inglés). Un método heurístico es un procedimiento que probablemente descubra una muy buena solución factible, pero no necesariamente una solución óptima para el problema específico que se esté considerando, en este caso, TSP. Por otro lado, las metaheurísticas son métodos de solución general que proporcionan una estructura general y pautas estratégicas para desarrollar un método heurístico específico para un tipo particular de problema. Ambos métodos permiten encontrar soluciones cercanas al óptimo en un tiempo razonable, lo que resulta crucial para resolver problemas que involucren un gran número de ciudades.

2.2. Algoritmo propuesto

Se implementará una heurística basada en la división de ciudades mediante clustering y una optimización local empleando el algoritmo Two-Opt, y luego una optimización con base en algoritmos genéticos. Para mejorar el rendimiento computacional, se utilizarán algoritmos como K-means para dividir el problema en subproblemas más manejables. Es importante mencionar que se probará la heuristica con y sin clusters para comparar el desempeño y demostrar cómo impacta la cantidad de ciudades en el problema.

2.2.1. Descripción de los algoritmos

- **Algoritmos elegidos:** Los algoritmos para resolver el problema de TSP que utilizaremos son los siguientes. Nuestra elección de dichos algoritmos se debe a que todos ellos reducen los costos computacionales y mejoran la calidad de las soluciones.

  - **Two-Opt** (*heurística*): Es un método de optimización local (búsqueda local) diseñado para mejorar las soluciones al TSP, eliminando iterativamente los cruces de ruta. El algoritmo funciona identificando dos aristas en un recorrido que, cuando se eliminan y se invierte el segmento intermedio, dan como resultado una ruta general más corta. Este proceso se repite hasta que no se puedan realizar más mejoras.
  - **Inserción voraz/codiciosa** (*heurística*): Es un tipo de algoritmo voraz. Un algoritmo voraz es un método de resolución de problemas que toma la decisión más beneficiosa en cada paso, buscando una solución óptima a nivel local con la esperanza de que estas decisiones conduzcan a una solución óptima a nivel global. Este enfoque es eficaz para problemas que presentan dos propiedades clave: la propiedad de elección voraz (es posible construir una solución óptima a nivel global tomando decisiones óptimas a nivel local) y una subestructura óptima (una solución óptima al problema contiene soluciones óptimas a sus subproblemas). Al tratarse de heurísticas, los algoritmos voraces no siempre garantizan la solución óptima para cada problema. Por ejemplo, en el TSP, la inserción voraz, que es una estrategia mediante la cual se selecciona la ciudad no visitada más cercana en cada paso, puede no dar como resultado el recorrido más corto posible.
  - **K-means** (*heurística*): El agrupamiento (*clustering*) en k-means es una heurística que agrupa datos no etiquetados en grupos distintos según la similitud de características. El objetivo de este algoritmo es dividir los datos en k grupos, minimizando la varianza dentro de cada grupo. Primero se seleccionan k centroides (usualmente son puntos al azar) de la muestra dada. Luego se asigna cada punto de datos al centroide más cercano, formando k grupos. Se actualizan los centroides calculando la media de todos los puntos en cada grupo, y este procedimiento se repite varias veces hasta que los centroides se estabilicen o se alcance una cantidad determinada de iteraciones.
  - **Algoritmo genético** (*metaheurística*): Los algoritmos genéticos son técnicas inspiradas en la selección natural y la genética. Mejoran iterativamente las soluciones a los problemas imitando los procesos evolutivos. Los componentes clave de

los algorítmos genéticos incluyen: población (un conjunto de soluciones potenciales), selección (elegir los individuos más aptos en función de una función de aptitud), cruce o recombinación (combinar partes de dos soluciones parentales para crear descendencia), mutación (introducir cambios aleatorios en los individuos para mantener la diversidad) y terminación (el algoritmo se detiene cuando se encuentra una solución satisfactoria o después de un número determinado de generaciones). Al ser metaheurísticas, los algorítmos genéticos son particularmente útiles para resolver problemas de optimización complejos con grandes espacios de búsqueda, como el TSP.

- **Parámetros:**
  - **Número de clústeres:** Determinado dinámicamente según el número de ciudades, ajustándose para evitar clusters vacíos.
  - **Tamaño de la población:** 100 individuos, para garantizar diversidad en la búsqueda genética.
  - **Número de generaciones:** 50, buscando un equilibrio entre tiempo de ejecución y calidad de la solución.
  - **Tasa de mutación:** 0.2, para evitar convergencia prematura.
  - **Tasa de élite:** 0.15, para preservar las mejores soluciones durante la evolución genética.

2.2.2. Pseudocódigo

Algorithm 1 Generación de la matriz de distancias

Inicializar n como el numero de ciudades
Crear un arreglo *coords* con las coordenadas de las ciudades
Crear una matriz *dist_matrix* donde cada entrada (i, j) es la distancia euclidiana entre ciudad i y ciudad j
Redondear los valores de *dist_matrix* y convertirlos a enteros
Return *dist_matrix*

Algorithm 2 Calculo de la longitud de un recorrido

Inicializar *total* como 0
for $i = 0, 1, \ldots,$ tamaño del tour $- 1$ **do**
    Añadir a *total* la distancia entre ciudad $tour[i]$ y $tour[i + 1]$
end for
Añadir la distancia desde la ultima ciudad al inicio del tour
Return *total*

Algorithm 3 Optimizar recorrido con Two-Opt

Inicializar n como el tamaño del tour
Inicializar *improved* como VERDADERO
while *improved* **do**
    Establecer *improved* como FALSO
    for $i = 1, 2, \ldots, n-2$ **do**
        for $j = i+1, \ldots, n-1$ **do**
            Calcular la distancia actual (*old_distance*) usando $tour[i-1]$, $tour[i]$, $tour[j]$ y $tour[j+1]$
            Calcular la nueva distancia (*new_distance*) intercambiando los arcos entre i y j
            if *new_distance* $<$ *old_distance* **then**
                Invertir la subsección $tour[i : j+1]$
                Establecer *improved* como VERDADERO
                Break
            end if
        end for
    end for
end while
Return *tour*

Algorithm 4 Recorrido inicial codicioso

Inicializar n como el tamaño del subconjunto de ciudades
Crear un conjunto *unvisited* con los índices de todas las ciudades excepto la primera
Inicializar *tour* con la ciudad inicial (indice 0)
while *unvisited* no este vacío **do**
    Seleccionar la ultima ciudad del tour
    Encontrar la ciudad mas cercana en *unvisited*
    Añadir esa ciudad al tour y eliminarla de *unvisited*
end while
Return *tour*

Algorithm 5 Resolución de un clúster

Crear un sub-TSP con las ciudades del cluster
Resolver el sub-TSP con solve, incrementando $recursion\_depth$ en 1
Convertir el recorrido resultante a indices globales
Return el recorrido global

Algorithm 6 Conexión de clústers

Inicializar $merged\_tour$ como un arreglo vacío
for cada tour en $cluster\_tours$ **do**
    Añadir todos los elementos del tour a $merged\_tour$
end for
Optimizar $merged\_tour$ usando _quick_two_opt
Return $merged\_tour$

Algorithm 7 Resolución completa

if $recursion\_depth \geq MAX\_RECURSION\_DEPTH$ **then**
    Return recorrido simple (orden natural de las ciudades)
end if
if el numero de ciudades $<$ $num\_clusters$ **then**
    Reducir $num\_clusters$ al número de ciudades
end if
Normalizar las coordenadas de las ciudades
Aplicar K-means para dividir las ciudades en $num\_clusters$ clústers
for cada clúster **do**
    Obtener los índices de las ciudades en el clúster
    Resolver el TSP del clúster con _solve_clúster
    Almacenar el tour resultante en $cluster\_tours$
end for
Conectar los tours de los clústers con _connect_clusters
Calcular la longitud del recorrido final
Return $final\_tour$ y $final\_length$

Algorithm 8 Algoritmo genético

Require: población inicial p generada mediante inserción voraz
Require: tasa de mutación mutation_rate, tasa de élite elite_rate, número de generaciones, generations

inicializar la mejor solución *best_tour* y *best_length* a partir de la población inicial
for cada generación **do**
    calcular la longitud de cada tour en p y ordenar por longitud (fitness)
    seleccionar los mejores $elite\_count = \text{elite\_rate} \cdot \text{size}(p)$ tours
    copiarlos a *new_population*
    while size(*new_population*) < size(p) **do**
        seleccionar *parent*1 mediante torneo en p
        seleccionar *parent*2 mediante torneo en p
        generar *child* aplicando el cruce parcialmente mapeado (pmx) a *parent*1 y *parent*2
        if random() < mutation_rate **then**
            aplicar quicktwoopt a *child*
        end if
        añadir *child* a *new_population*
    end while
    actualizar $p \leftarrow$ *new_population*
    obtener el mejor tour actual *current_best* en p y su longitud *current_best_length*
    if *current_best_length* < *best_length* **then**
        actualizar *best_tour* $\leftarrow$ *current_best* y *best_length* $\leftarrow$ *current_best_length*
        imprimir información: "generación", *generation*, "mejor distancia", *best_length*
    end if
end for
añadir el nodo inicial al final de *best_tour* para cerrar el ciclo
Return *best_tour*, *best_length*

2.3. Implementación

El código fuente desarrollado para resolver el problema se encuentra disponible en el siguiente repositorio:

https://github.com/Alvlopzam78/ProyectoFinal_IDO

3. Resultados

3.1. Instancias resueltas

Se resolvieron instancias para los siguientes países:

- Qatar: 194 ciudades.
- Uruguay: 734 ciudades.
- Zimbabwe: 929 ciudades.

3.2. Tablas de resultados

RUTAS	Algoritmo simple	Algoritmo con Clustering	Óptimo real
Qatar	9486	9849	9352
Uruguay	-	86612	79114
Zimbabwe	-	103228	95345

Cuadro 1: Comparación de las longitudes de las rutas.

RUTAS	% Diferencia simple	% Diferencia Clustering
Qatar	0.01	0.05
Uruguay	-	0.09
Zimbabwe	-	0.08

Cuadro 2: Diferencias con respecto a los óptimos.

TIEMPOS	Algoritmo simple	Algoritmo con Clustering
Qatar	108.05	2.63
Uruguay	-	41.84
Zimbabwe	-	127.42

Cuadro 3: Tiempos de ejecución para los algoritmos en segundos.

3.3. Gráficas de las rutas

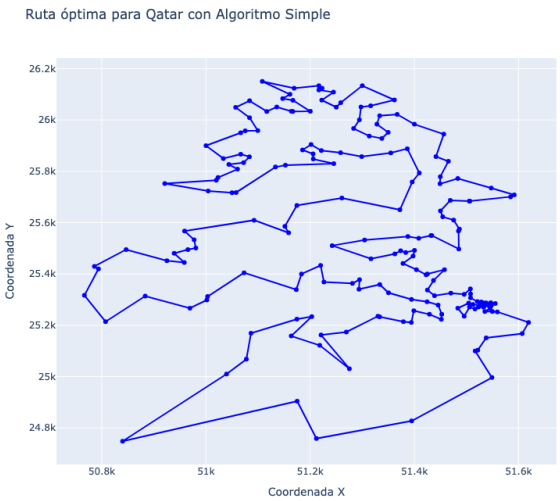

Figura 1: Ruta de Qatar con algoritmo simple.

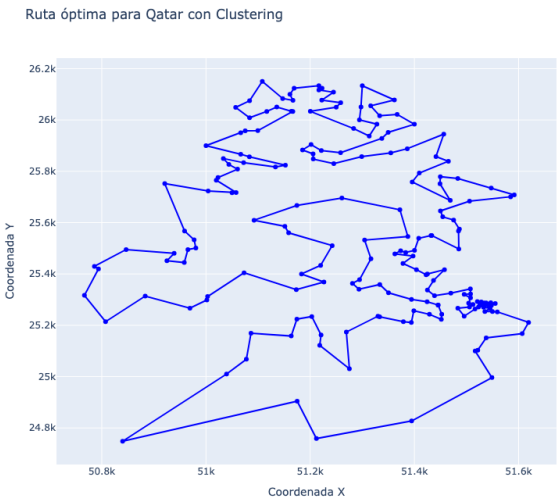

Figura 2: Ruta de Qatar con algoritmo con Clustering.

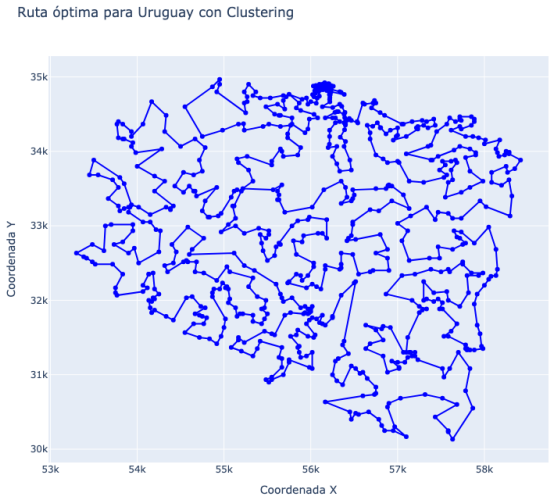

Figura 3: Ruta de Uruguay con algoritmo con Clustering.

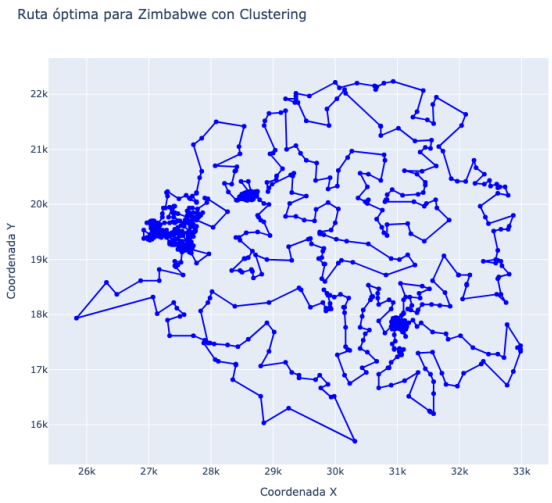

Figura 4: Ruta de Zimbabwe con algoritmo con Clustering.

4. Conclusiones

Este proyecto se ha centrado en la resolución del problema del Agente Viajero (TSP) mediante el uso de diversas heurísticas y metaheurísticas. A lo largo del proyecto, se implementó una combinación de algoritmos como Two-Opt, algoritmos genéticos y técnicas de Clustering (K-means) para abordar instancias de tamaño considerable, con el objetivo de encontrar soluciones aproximadas en tiempos computacionales que sean razonables.

Los resultados obtenidos muestran que la metodología propuesta logra obtener soluciones cercanas a las óptimas en las instancias de los tres países estudiados: Qatar, Uruguay y Zimbabwe. En las tablas presentadas se puede observar que los óptimos encontrados están dentro de un margen de error menor al 10 %. Con el problema de Qatar, sin utilizar técnicas de Clustering, llegamos a la solución óptima con 1 % de error, pero tomó más tiempo. Esto nos llevó a ver que era impensable utilizar la misma técnica para los problemas de Uruguay y Zimbabwe. El uso del algoritmo k-means ayudó a reducir la complejidad de dichos problemas de tal forma que se redujo el tiempo de ejecución sin sacrificar la calidad de las soluciones.

Se observó que, aunque las soluciones generadas son satisfactorias por su cercanía al óptimo real, aún existe un margen de mejora en términos de optimización. Como pasos a seguir, se recomienda explorar otros métodos que puedan ser implementados, así como evaluar valores alternativos a los parámetros utilizados en el modelo presentado. En resumen, los resultados obtenidos con las diferentes metodologías proporcionan una base sólida para continuar mejorando las soluciones a este problema, explorando nuevas heurísticas, metaheurísticas y combinaciones de métodos híbridos.

Referencias

[1] Taha, H. A. (2011). *Investigación de operaciones* (9ª ed.). Pearson.

[2] Hillier, F. S., & Lieberman, G. J. (2020). *Introducción a la investigación de operaciones* (11ª ed.). McGraw-Hill.