Universidad Católica de Temuco

Curso: Teoría de Grafos

Informe Técnico:

Problema del Viajante (TSP) sobre Grafos Completos

Fecha: 22 de noviembre de 2025

Integrantes del grupo:

Simón Cifuentes
Jecar Yañez

Índice

1. Introducción **2**

2. Datos de la instancia **2**

2.1. Matriz de distancias D 3

3. Metodología **4**

3.1. Representación del problema 4

3.2. Búsqueda exhaustiva 4

3.3. Heurística del Vecino Más Cercano (NN) 5

3.4. Sistema de visualización interactivo 5

4. Resultados **5**

4.1. Rutas obtenidas 5

4.2. Comparación cuantitativa 6

4.3. Ciclos finales 6

5. Análisis y discusión **8**

6. Conclusiones **8**

A. Fragmentos esenciales del código **9**

1. Introducción

El Problema del Viajante consiste en determinar un ciclo Hamiltoniano de costo mínimo que visite exactamente una vez cada nodo de un grafo y regrese al punto de partida. Desde la perspectiva de la teoría de grafos y la optimización combinatoria, el TSP es uno de los problemas más estudiados debido a su relevancia aplicada en logística, transporte, diseño de redes y planificación de rutas, y a la vez por su alta complejidad computacional: se trata de un problema NP-difícil.

En este trabajo se resuelve una instancia real del TSP, seleccionando siete ciudades de la zona centro–sur de Chile y construyendo el grafo completo K_7 cuyas aristas tienen pesos dados por distancias euclidianas sobre coordenadas geográficas (longitud, latitud). Se comparan dos enfoques de solución:

- **Búsqueda exhaustiva**: recorre sistemáticamente todos los ciclos Hamiltonianos posibles, garantizando así encontrar la solución óptima global.
- **Heurística del Vecino Más Cercano (Nearest Neighbor, NN)**: construye el ciclo de manera incremental, eligiendo en cada paso la ciudad no visitada más cercana a la ciudad actual. Es un método rápido, pero no garantiza optimalidad.

Adicionalmente se implementó un sistema de visualización interactivo que permite observar el proceso de ambos algoritmos: cómo la búsqueda exhaustiva recorre los ciclos y va actualizando la mejor ruta, y cómo la heurística NN construye el ciclo paso a paso, mostrando explícitamente la naturaleza *greedy* o miope del algoritmo.

El objetivo del informe es documentar la instancia considerada, describir la metodología implementada, presentar los resultados obtenidos y analizar cuantitativamente el compromiso entre optimalidad y eficiencia (trade-off) que se observa al comparar ambos enfoques.

Formulación matemática del TSP

Sea $G = (V, E)$ un grafo completo donde cada vértice $i \in V$ representa una ciudad y cada arista $(i, j) \in E$ tiene un costo d_{ij} correspondiente a la distancia entre las ciudades i y j. El Problema del Viajante consiste en encontrar un ciclo que visite cada vértice exactamente una vez y minimice la longitud total recorrida:

$$\min_{\pi} \sum_{k=1}^{n} d_{\pi_k, \pi_{k+1}}, \tag{1}$$

donde π es una permutación de las ciudades y se define $\pi_{n+1} = \pi_1$ para cerrar el ciclo.

2. Datos de la instancia

Se seleccionaron siete ciudades chilenas: Santiago, Valparaíso, Rancagua, Talca, Chillán, Concepción y Temuco. La Tabla 1 presenta sus coordenadas geográficas (latitud, longitud) obtenidas desde Google Maps.

Cuadro 1: Ciudades seleccionadas y sus coordenadas geográficas.

Ciudad	Latitud	Longitud
Santiago	-33.4489	-70.6693
Valparaíso	-33.0472	-71.6127
Rancagua	-34.1708	-70.7444
Talca	-35.4264	-71.6554
Chillán	-36.6066	-72.1034
Concepción	-36.8270	-73.0503
Temuco	-38.7359	-72.5904

Fuente: Google Maps, consulta realizada el 21 de noviembre de 2025.

2.1. Matriz de distancias D

La matriz de distancias $D = [d_{ij}]$ se construyó utilizando la distancia euclidiana entre pares de coordenadas (x_i, y_i) y (x_j, y_j) en el plano longitud–latitud:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{2}$$

La Tabla 2 muestra la matriz resultante (valores aproximados con cuatro decimales).

Cuadro 2: Matriz de distancias D (unidades euclidianas).

	Stgo	Valpo	Rgua	Talca	Chillán	Concep.	Temuco
Stgo	0.0000	1.0254	0.7258	2.2097	3.4681	4.1329	5.6252
Valpo	1.0254	0.0000	1.4200	2.3796	3.5931	4.0440	5.7721
Rgua	0.7258	1.4200	0.0000	1.5513	2.7893	3.5175	4.9242
Talca	2.2097	2.3796	1.5513	0.0000	1.2624	1.9767	3.4390
Chillán	3.4681	3.5931	2.7893	1.2624	0.0000	0.9722	2.1843
Concep.	4.1329	4.0440	3.5175	1.9767	0.9722	0.0000	1.9635
Temuco	5.6252	5.7721	4.9242	3.4390	2.1843	1.9635	0.0000

La Figura 1 muestra la distribución geográfica de las ciudades en el plano longitud–latitud.

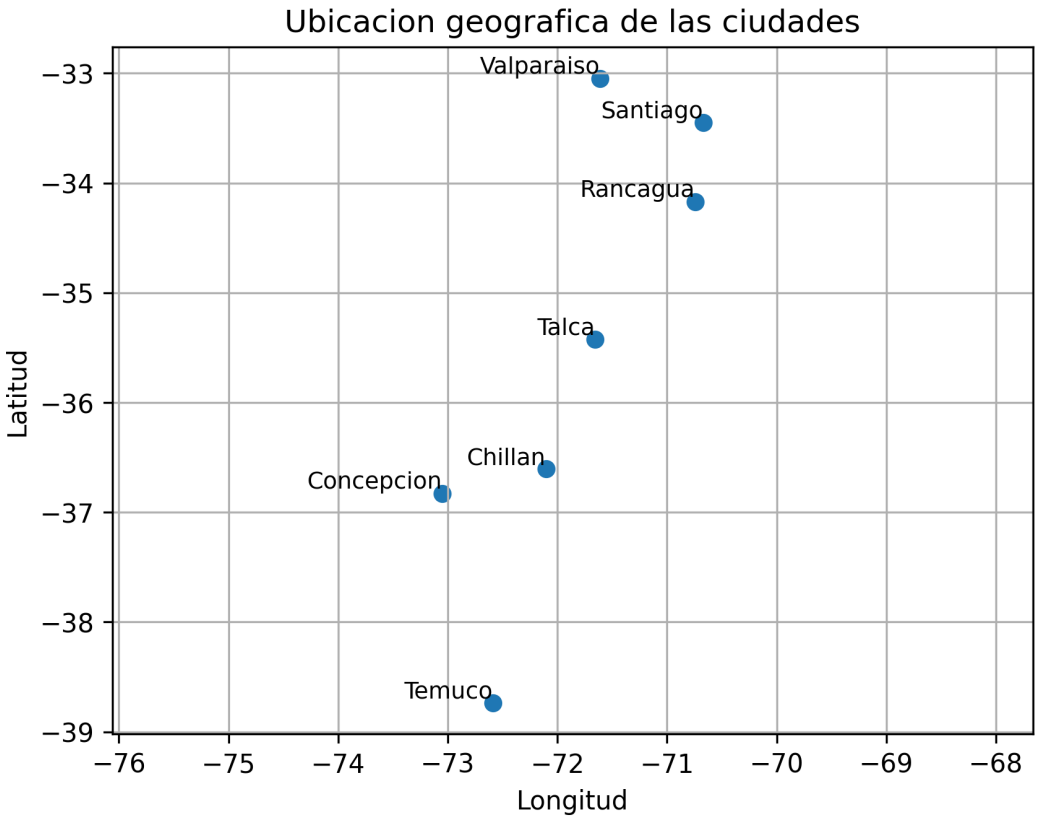

Figura 1: Distribución geográfica de las ciudades utilizadas en la instancia del TSP.

3. Metodología

3.1. Representación del problema

Cada ciudad se representa como un vértice de un grafo completo K_7. La distancia euclidiana entre las coordenadas de dos ciudades se utiliza como peso de la arista correspondiente. Una ruta factible del viajante se modela como un ciclo Hamiltoniano, es decir, una permutación de las siete ciudades que comienza y termina en la misma ciudad.

3.2. Búsqueda exhaustiva

El método exacto implementado consiste en enumerar todas las permutaciones posibles de las ciudades, fijando la ciudad de origen para evitar repeticiones cíclicas. Para $n = 7$ ciudades, el número de ciclos Hamiltonianos distintos es $(7-1)! = 720$.

Para cada permutación $\pi = (\pi_1, \ldots, \pi_7)$ se construye el ciclo cerrado $(\pi_1, \pi_2, \ldots, \pi_7, \pi_1)$ y se calcula su longitud total mediante la suma de las distancias $d_{\pi_k, \pi_{k+1}}$. El algoritmo mantiene de forma incremental la mejor ruta encontrada hasta el momento y su longitud asociada. Al finalizar la enumeración completa se obtiene el ciclo óptimo $\pi^\star$ con longitud mínima $L^\star$.

Este procedimiento garantiza encontrar el óptimo global, pero su complejidad computacional es factorial en n, por lo que se vuelve impracticable para instancias de mayor tamaño.

3.3. Heurística del Vecino Más Cercano (NN)

La heurística del Vecino Más Cercano comienza en una ciudad inicial (en este trabajo, Santiago) y mantiene un conjunto de ciudades no visitadas. En cada iteración:

1. Desde la ciudad actual se calculan las distancias a todas las ciudades no visitadas.
2. Se elige como siguiente ciudad aquella con distancia mínima.
3. Se actualiza la ciudad actual y se elimina la ciudad elegida del conjunto de no visitadas.

El proceso se repite hasta que todas las ciudades han sido visitadas y, finalmente, se regresa a la ciudad de origen para cerrar el ciclo. La complejidad del algoritmo es $O(n^2)$ debido a la búsqueda repetida del vecino más cercano, por lo que resulta mucho más eficiente que la búsqueda exhaustiva. Sin embargo, es una heurística *greedy*: toma decisiones puramente locales y no garantiza encontrar la solución óptima.

3.4. Sistema de visualización interactivo

La implementación se realizó en Python 3.12 utilizando las librerías estándar `math` e `itertools` para cálculos numéricos y permutaciones, y la librería `matplotlib` para la generación de gráficos.

Se desarrolló un visualizador interactivo que permite alternar entre dos modos:

- **Modo exhaustivo**: muestra en el plano el ciclo que se está evaluando en la iteración actual y, con una línea discontinua, el mejor ciclo encontrado hasta ese momento. Mediante botones de "Anterior", "Siguiente" y "Rápido", el usuario puede recorrer el proceso de búsqueda y observar cómo se converge gradualmente al óptimo global.
- **Modo NN**: muestra la construcción paso a paso de la ruta heurística. En cada paso se representa la ruta parcial y un panel de texto explica desde qué ciudad se parte, cuál fue la ciudad elegida, cuál era su distancia, y qué otras alternativas existían. De esta forma se hace explícita la naturaleza miope del algoritmo.

Adicionalmente se generan figuras estáticas para la ruta óptima y la ruta heurística, que se utilizan en las secciones de resultados.

4. Resultados

4.1. Rutas obtenidas

A partir del código desarrollado se obtuvieron las rutas que se presentan en la Tabla 3.

Cuadro 3: Comparación de las rutas óptima y heurística.

Método	Ruta (ciudades)
Búsqueda exhaustiva (óptimo)	Santiago → Valparaíso → Concepción → Temuco → Chillán → Talca → Rancagua → Santiago.
Heurística Vecino Más Cercano (NN)	Santiago → Rancagua → Valparaíso → Talca → Chillán → Concepción → Temuco → Santiago.

4.2. Comparación cuantitativa

La Tabla 4 resume las longitudes de los ciclos y los tiempos de ejecución medidos. Los valores corresponden al cálculo de la ruta; el tiempo adicional asociado a la generación y guardado de figuras no se considera en esta comparación.

Cuadro 4: Comparación numérica entre ambos métodos.

Métrica	Exhaustivo	NN
Longitud del ciclo	$L^\star = 12{,}7566$	$L_{\mathrm{NN}} = 14{,}3487$
Gap de optimalidad	—	$g = 12{,}48\,\%$
Tiempo de ejecución (s)	$t^\star \approx 0{,}0009$	$t_{\mathrm{NN}} \approx 0{,}000032$

El gap de optimalidad se calculó como

$$g = \frac{L_{\mathrm{NN}} - L^\star}{L^\star} \times 100 \approx 12{,}48\,\%. \tag{3}$$

4.3. Ciclos finales

La Figura 2 muestra el ciclo óptimo obtenido mediante búsqueda exhaustiva, mientras que la Figura 3 muestra la ruta obtenida por la heurística NN sobre las mismas coordenadas geográficas.

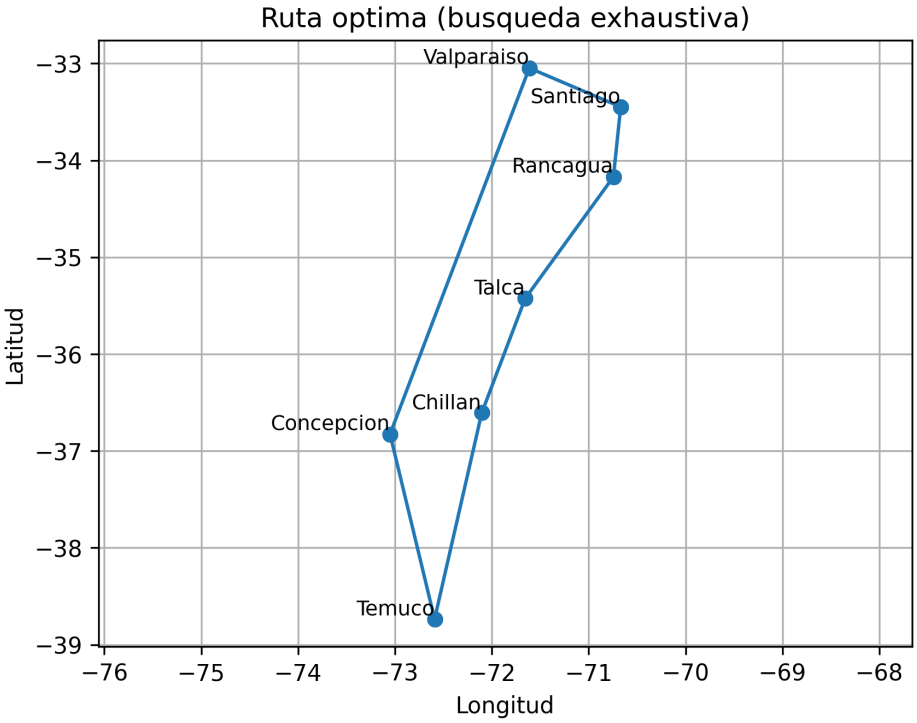

Figura 2: Ciclo óptimo $\pi^\star$ obtenido mediante búsqueda exhaustiva.

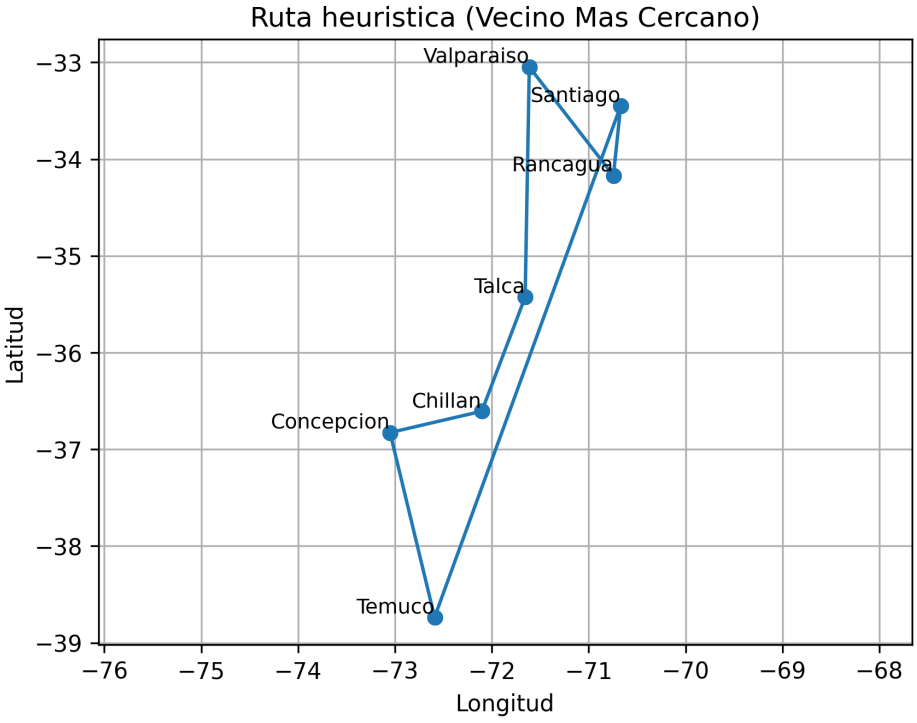

Figura 3: Ciclo heurístico π_{NN} obtenido mediante Vecino Más Cercano.

5. Análisis y discusión

La búsqueda exhaustiva evaluó los 720 ciclos Hamiltonianos posibles para la instancia con $n = 7$ ciudades y encontró un ciclo óptimo de longitud $L^\star = 12{,}7566$. La heurística del Vecino Más Cercano, en cambio, produjo una ruta de longitud $L_{\mathrm{NN}} = 14{,}3487$, es decir, aproximadamente un 12,48 % más larga que el óptimo.

Desde el punto de vista de la calidad de la solución, se trata de una diferencia apreciable: la heurística logra una ruta razonable, pero se aleja de manera clara del mejor ciclo posible. Las visualizaciones del proceso de NN muestran cómo en cada paso se privilegia la ciudad más cercana sin considerar el impacto global de esa decisión sobre el recorrido completo, lo que puede conducir a trayectorias subóptimas.

En cuanto a tiempos de ejecución, ambos métodos son extremadamente rápidos para $n = 7$, pero ya se observa que la heurística es aproximadamente 27,9 veces más rápida que la búsqueda exhaustiva. Esto anticipa su comportamiento asintótico: mientras que la heurística NN presenta una complejidad cuadrática $O(n^2)$, la búsqueda exhaustiva crece factorialmente con n, volviéndose impracticable para instancias de tamaño moderado.

Escalabilidad y limitaciones

Para $n = 7$ ciudades la búsqueda exhaustiva evalúa $(7-1)! = 720$ ciclos en tiempos despreciables. Sin embargo, si se aumentara a $n = 10$, el número de ciclos crecería a $(10 - 1)! = 362\,880$, y para $n = 12$ superaría los 39 millones de rutas posibles, lo que haría el método inviable en tiempos razonables en un computador doméstico si no se incorporan técnicas de poda o paralelización.

La heurística NN, en cambio, mantiene una complejidad cuadrática y escala mejor con n, pero su principal limitación es que puede quedarse en soluciones de mala calidad cuando la distribución espacial de las ciudades induce decisiones locales engañosas. En otras palabras, el algoritmo siempre toma la mejor decisión local disponible, pero no controla el efecto global de esas decisiones sobre el ciclo completo.

En resumen, el estudio pone de manifiesto el *trade-off* clásico del TSP:

- La búsqueda exhaustiva ofrece *certeza* (optimalidad garantizada) a costa de una *baja escalabilidad*.
- La heurística NN ofrece *rapidez* y *simplicidad* a costa de aceptar soluciones aproximadas cuyo gap puede ser significativo.

6. Conclusiones

En este trabajo se resolvió una instancia del Problema del Viajante sobre siete ciudades chilenas utilizando dos enfoques complementarios: un método exacto de búsqueda exhaustiva y la heurística constructiva del Vecino Más Cercano. La búsqueda exhaustiva encontró el ciclo óptimo de longitud $L^\star = 12{,}7566$, mientras que la heurística NN produjo una ruta un 12,48 % más larga, lo que refleja el compromiso entre precisión y eficiencia.

Las visualizaciones desarrolladas, tanto en forma de figuras estáticas como mediante un visualizador interactivo, permiten comprender el funcionamiento interno de ambos algoritmos: cómo la búsqueda exhaustiva explora sistemáticamente el espacio de soluciones y cómo la heurística NN toma decisiones puramente locales basadas en el vecino más cercano disponible.

En contextos reales donde el número de ciudades es grande, la búsqueda exhaustiva resulta impracticable, y métodos heurísticos como NN se vuelven la única alternativa viable, aunque con una posible pérdida de calidad de la solución. Como trabajo futuro se podría extender este estudio incorporando otras heurísticas y metaheurísticas (por ejemplo, 2-opt, recocido simulado o algoritmos genéticos) que mejoren la calidad de las rutas manteniendo tiempos de cómputo razonables.

Referencias

- Wikipedia. *Travelling salesman problem*. Disponible en: `https://en.wikipedia.org/wiki/Travelling_salesman_problem`
- Material del curso de Teoría de Grafos.
- Código fuente completo entregado junto con este informe en el archivo `tsp.py`.
- Repositorio del proyecto (código completo, figuras y animaciones): `https://github.com/SimonCifuentes/grafos`

A. Fragmentos esenciales del código

A continuación se muestran algunos fragmentos representativos del código utilizado para resolver la instancia del TSP. El archivo completo `tsp.py` incluye además la construcción de la matriz de distancias, la medición de tiempos y el sistema de visualización interactivo.

Cálculo de distancias y longitud de un ciclo

```
def distancia_2d(a, b):
    """Distancia euclidiana entre dos puntos 2D."""
    return math.hypot(a[0] - b[0], a[1] - b[1])

def construir_matriz_distancias(coords):
    """Matriz de distancias D[i][j] entre todas las ciudades."""
    n = len(coords)
    matriz = [[0.0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if i != j:
                matriz[i][j] = distancia_2d(coords[i], coords[j])
    return matriz
```

```
def longitud_ciclo(tour, D):
    """Longitud total de un ciclo Hamiltoniano cerrado."""
    total = 0.0
    for k in range(len(tour) - 1):
        i, j = tour[k], tour[k + 1]
        total += D[i][j]
    return total
```

Búsqueda exhaustiva

```
def tsp_exhaustivo_con_historia(D, origen=0):
    n = len(D)
    otras = [i for i in range(n) if i != origen]

    mejor_tour = None
    mejor_longitud = float("inf")
    historia = []

    inicio = time.perf_counter()

    for idx, perm in enumerate(itertools.permutations(otras), start=1):
        tour = (origen,) + perm + (origen,)
        L = longitud_ciclo(tour, D)

        if L < mejor_longitud:
            mejor_longitud = L
            mejor_tour = tour

        historia.append({
            "iter": idx,
            "tour": tour,
            "length": L,
            "best_tour": mejor_tour,
            "best_length": mejor_longitud,
        })

    tiempo = time.perf_counter() - inicio
    return mejor_tour, mejor_longitud, tiempo, historia
```

Heurística del Vecino Más Cercano

```
def tsp_vecino_mas_cercano_con_historia(D, origen=0):
```

```
n = len(D)
no_visitadas = set(range(n))
no_visitadas.remove(origen)

tour = [origen]
actual = origen
historia = []

# Paso 0
historia.append({
    "step": 0,
    "tour_partial": tuple(tour),
    "current": origen,
    "chosen": None,
    "length_partial": 0.0,
    "candidates": [],
})

inicio = time.perf_counter()
paso = 0

while no_visitadas:
    candidatos = sorted(
        ((j, D[actual][j]) for j in no_visitadas),
        key=lambda x: x[1],
    )
    siguiente = candidatos[0][0]

    tour.append(siguiente)
    no_visitadas.remove(siguiente)
    paso += 1

    tour_parcial = tour + [origen]
    L_parcial = longitud_ciclo(tour_parcial, D)

    historia.append({
        "step": paso,
        "tour_partial": tuple(tour_parcial),
        "current": actual,
        "chosen": siguiente,
        "length_partial": L_parcial,
        "candidates": candidatos,
```

```
    })

    actual = siguiente

tour.append(origen)
L = longitud_ciclo(tour, D)
tiempo = time.perf_counter() - inicio
return tuple(tour), L, tiempo, historia
```