Proyecto Longitudinal ALGABO

Santiago Aurrecochea
Sebastian Martinez
Agustín Toya

Noviembre 2024

1 Introducción

En este documento, abordamos el problema del agente viajero, conocido en inglés como TSP (Travelling Salesman Problem). Este problema, ampliamente estudiado en matemáticas y ciencias de la computación, plantea desafíos tanto teóricos como prácticos, con aplicaciones en logística, planificación de rutas y optimización de recursos.

El objetivo principal de este trabajo es explorar el TSP desde sus fundamentos hasta la implementación de una solución práctica basada en programación dinámica. En la primera parte, se presenta el contexto histórico del problema, su definición formal y la formulación matemática que permite estructurarlo de manera precisa. Posteriormente, se describe el algoritmo utilizado para resolverlo, con énfasis en su implementación y análisis.

Finalmente, se discuten las aplicaciones del TSP en el sector del turismo en Uruguay, mostrando cómo este problema teórico puede convertirse en una herramienta poderosa para la optimización de rutas turísticas y la generación de valor económico. A través de este análisis, se busca destacar la relevancia del TSP no solo como un problema matemático, sino también como un modelo aplicable a problemas reales que enfrentan diversas industrias en la actualidad.

1.1 Origen

El TSP tiene un origen incierto, con referencias tempranas desde prinicipios del siglo XIX en Alemania, donde se discutía la importancia de elegir recorridos óptimos para ahorrar tiempo, aunque sin un enfoque matemático. El problema fue definido como tal en 1800 por el matemático irlandés WR Hamilton y por el matemático británico Thomas Kirkman. Hamilton Icosian Game. El juego Icosain es un desafío conocido como icosian game, un juego matemático desarrollado en 1857 por William Rowan Hamilton. El objetivo del juego es dar con el recorrido de Hamilton por las aristas de un dodecaedro para visitar una y sólo una vez cada vértice y que el de llegada coincida con el de partida. Este desafío se comercializó como un tablero con agujeros en cada nodo del grafo

del dodecaedro y luego, fue distribuido en Europa en diversos formatos. Era un rompecabezas de recreo con base en la búsqueda de un ciclo de Hamilton. La forma general del TSP parece haber sido estudiado por primera vez por los matemáticos durante la década de 1930 en Viena y la Universidad de Harvard, en particular por Karl Menger. Posteriormente, en las décadas de 1950 y 1960, el problema ganó una creciente popularidad en los círculos científicos de Europa y Estados Unidos, impulsado por la Corporación RAND en Santa Mónica, que ofreció premios por avances en su resolución.

2 Definición

Dada una colección de ciudades y el costo de viaje entre cada par de ellas, el problema del agente viajero , consiste en encontrar la forma más optima de visitar todas las ciudades una sola vez y regresar al punto de partida.

2.1 Complejidad

Dado que n es el número de ciudades a visitar, el número total de caminos posibles diferentes que cubren todas las ciudades puede representarse como:

$$\frac{(n-1)!}{2}.$$

3 Formulación Matemática del Probelma TSP

Fijamos la ciudad de inicio x_1. Para un conjunto $S \subseteq \{x_2, \ldots, x_n\}$ y un vértice e tal que $e \neq x_1$ y $e \notin S$, definimos:

$$g(S,e) = \min \left\{ C(T) \mid \begin{array}{l} \text{- } T \text{ es un camino que cumple:} \\ \text{- } T \text{ inicia en } x_1, \\ \text{- } T \text{ termina en } e, \\ \text{- } T \text{ visita cada ciudad de } S \text{ exactamente una vez,} \\ \text{- } T \text{ no visita ninguna otra ciudad (fuera de } S \cup \{x_1, e\}). \end{array} \right\}$$

El objetivo del problema es **minimizar el costo total** de recorrer un ciclo que pase por todos los vértices del conjunto $\{x_1, \ldots, x_n\}$ y regrese al vértice de partida, después de haber visitado cada uno de los demás vértices exactamente una vez.

El objetivo se puede descomponer en:

- Comenzar en la ciudad x_1.
- Visitar todas las demás ciudades $\{x_2, x_3, \ldots, x_n\}$ exactamente una vez.
- Regresar a la ciudad inicial x_1.

Esto implica evaluar todas las posibles rutas que recorren las ciudades restantes y terminan en cada posible ciudad final e, para luego regresar a x_1, seleccionando la de menor costo total.

Matemáticamente, el costo total mínimo se expresa como:

$$\text{TSP} = \min \begin{Bmatrix} g([n] \setminus \{x_1, x_2\}, x_2) + C_{x_2,x_1}, \\ g([n] \setminus \{x_1, x_3\}, x_3) + C_{x_3,x_1}, \\ g([n] \setminus \{x_1, x_4\}, x_4) + C_{x_4,x_1}, \\ \vdots \\ g([n] \setminus \{x_1, e\}, e) + C_{e,1} \end{Bmatrix}.$$

donde:

- $g(S, e)$: Representa el costo mínimo de un camino que comienza en el vértice de inicio (es decir, x_1), visita todos los vértices en el conjunto S exactamente una vez y termina en el vértice e.
- $C_{e,1}$: Es el costo de regresar desde el vértice e al vértice inicial x_1.

4 Formulacion de la solucion para el TSP

Sea T^* el camino óptimo asociado al subproblema $g(S, e)$. Este camino es aquel que comienza en el vértice inicial x_1, visita exactamente una vez todos los vértices de S y termina en el vértice e, teniendo el menor costo posible entre todos los caminos que cumplen estas condiciones.

En T^*, sea w el vértice anterior a e. Es decir, w es el penúltimo vértice visitado en T^* antes de alcanzar e. La clave de esta construcción es que T^* se puede descomponer en dos partes: un subcamino óptimo que termina en w tras visitar todos los vértices de $S \setminus \{w\}$, y el último paso desde w hasta e.

El costo total del camino T^* puede expresarse como la suma del costo del subcamino hacia w, que es $g(S \setminus \{w\}, w)$, y el costo de la arista directa desde w a e, denotado como $C_{w,e}$. Así, se tiene:

$$l(T^*) = g(S \setminus \{w\}, w) + C_{w,e}.$$

Para que T^* sea óptimo, todos los subcaminos que lo componen también deben ser óptimos,si un subcamino no fuera óptimo, podríamos reemplazarlo por otro con menor costo, lo que haría que T^* dejara de ser el camino de menor costo posible, contradiciendo su definición. Por lo tanto, el subcamino que termina en w tras visitar los vértices de $S \setminus \{w\}$ debe ser óptimo para $g(S \setminus \{w\}, w)$.

Para encontrar $g(S, e)$, evaluamos todos los posibles vértices $w \in S$ como el vértice anterior a e. Para cada w, calculamos el costo total de terminar el camino en e pasando por w. Esto se expresa como:

$$g(S, e) = \min_{w \in S} \{g(S \setminus \{w\}, w) + C_{w,e}\}.$$

Cuando el conjunto S contiene un único vértice ($S = \{w\}$), el único camino posible es viajar directamente desde 1 hasta w. En este caso, el costo se reduce a:

$$g(S, w) = C_{1,w}.$$

Así, la solución al problema se construye recursivamente, descomponiendo cada camino óptimo en un subcamino óptimo hacia w más un paso hacia e, evaluando todos los posibles w y eligiendo el que minimice el costo total.

4.1 Solución Final

Una vez que se han calculado $g(S, e)$ para todos los subconjuntos $S \subseteq V - x_1$ y todos los nodos e, el costo total mínimo del ciclo que visita todas las ciudades y regresa al nodo inicial x_1 se obtiene como:

$$\text{Costo total óptimo} = \min_{e \in V - x_1} \{g(V - x_1, e) + C_{e,x_1}\}.$$

Donde:

- $g(V - x_1, e)$: Es el costo mínimo de visitar todas las ciudades comenzando en x_1 y terminando en e.
- C_{e,x_1}: Es el costo de regresar desde el nodo final e al nodo inicial x_1.

5 Algoritmo utilizado

Para este algoritmo representaremos el conjunto de las n ciudades como una secuencia de n bits, en que los bits en 1 significarían que la ciudad en esa posición ha sido recorrida.
Para esto utilizaremos 2 tablas, costo y predecesor, de tamaño $2^n * n$ que dadas las entradas costo[máscara][índice] y predecesor[máscara][índice] retornaran el costo mínimo de recorrer el subconjunto de ciudades de la máscara terminando en la ciudad índice y la ciudad anterior a indice en dicho recorrido respectivamente.
Por ejemplo: costo[01011][2] nos daría el costo mínimo de recorrer las ciudades 1, 2 y 4, terminando el recorrido en la segunda, y predecesor[01011][2] nos daría para ese recorrido la ciudad anterior a 2.

5.1 Input

ciudad_incial : numero natural
dist : matriz de distancias n^2

5.2 Inicialización

1: $n \leftarrow dist.length$
2: $costo[i][j] \leftarrow +\infty, \forall i \mid 0 \leq i < (1 << n) \wedge \forall j \mid 0 \leq j < n$

3: $predecesor[i][j] \leftarrow nulo, \forall i \mid 0 \leq i < (1 << n) \land \forall j \mid 0 \leq j < n$
4: $costo[1 << ciudad\_inicial][ciudad\_incial] \leftarrow 0$

5.3 Ejecución

1: **for** $recorrido \leftarrow 0$ hasta $1 << n$ **do**
2: **for** $ult\_ciudad \leftarrow 0$ hasta n **do**
3: **if** (ult_ciudad AND $recorrido$) $= 0$ **then** ▷ Si no fue recorrida
4: Ir a siguiente iteración
5: **end if**
6: **for** $sig\_ciudad \leftarrow 0$ hasta n **do**
7: **if** (sig_ciudad AND $recorrido$) $<> 0$ **then** ▷ Si ya fue recorrida
8: Ir a siguiente iteración
9: **end if**
10: $nuevo\_recorrido \leftarrow recorrido$ OR $(1 << sig\_ciudad)$
11: $dis \leftarrow costo[recorrido][ult\_ciudad] + dist[ult\_ciudad][sig\_ciudad]$
12: **if** $dis < costo[nuevo\_recorrido][sig\_ciudad]$ **then**
13: $costo[nuevo\_recorrido][sig\_ciudad] \leftarrow dis$
14: $predecesor[nuevo\_recorrido][sig\_ciudad] \leftarrow ult\_ciudad$
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: $min \leftarrow -\infty$
20: $ciudad\_final \leftarrow nulo$
21: $fin\_camino \leftarrow (1 << n) - 1$
22: **for** $ult\_ciudad \leftarrow 0$ hasta n **do**
23: $cost \leftarrow costo[fin\_camino][ult\_ciudad] + dist[ult\_ciudad][ciudad\_inicial]$
24: **if** $cost < min$ **then**
25: $min \leftarrow cost$
26: $ciudad\_final \leftarrow ult\_ciudad$
27: **end if**
28: **end for**
29: $tour \leftarrow [ciudad\_inicial]$
30: $recorrido \leftarrow fin\_camino$
31: $ult\_ciudad \leftarrow ciudad\_final$
32: **while** $recorrido <> 0$ **do**
33: $tour.add(ult\_ciudad)$
34: $nueva\_utl\_ciudad \leftarrow predecesor[recorrido][ult\_ciudad]$
35: $recorrido \leftarrow recorrido$ XOR $(1 << ult\_ciudad)$
36: $ult\_ciudad \leftarrow nueva\_ult\_ciudad$
37: **end while**
38: return $tour.invertir(), min$

5.4 Análisis de Tiempo y Espacio

Este algoritmo se ejecuta en un tiempo $O(n^2 * 2^n)$ y ocupa espacio $O(n * 2^n)$ en función a la cantidad de ciudaes de la entrada, representado como *dist.length*.

5.4.1 Cálculo de Tiempo

Las instrucciones 1 y 4 de la inicialización toman un tiempo O(1), las instrucciones 2 y 3 toman ambas un tiempo $O(n * 2^n)$, ya que para todos los subconjuntos de ciudades posibles (2^n) recorre todas las ciudades (n), por tanto la inicialización tiene tiempo $O(n * 2^n)$.

La ejecución se puede dividir en 3 partes, desde la instrucción 1 a la 18, desde la instrucción 22 a la 28 y desde la instruccion 32 a la 37; como son bucles no anidados, el tiempo de la ejecución sera el máximo entre el tiempo de estas 3 partes (ya que las instrucciones que se encuentran entre estas parte toman todas tiempo O(1)).

La instrucción 1 toma un tiempo $O(2^n)$ ya que recorre todos los subconjuntos de ciudades posibles y la instrucción 2 toma un tiempo $O(n)$ ya que recorre todas las ciudades. Las instrucciones 3 y 4 toman un tiempo O(1) y la instrucción 6 toma un tiempo $O(n)$ ya que recorre todas las ciudades. Las instrucciones 7, 8, 10, 11, 12, 13 y 14 toman un tiempo O(1). Por lo tanto, la parte 1 de la ejecución desde la instrucción 1 a 18 tiene un tiempo $O(2^n * n * n) = O(2^n * n^2)$.

La instrucción 22 toma un tiempo $O(n)$ ya que recorre todas las ciudades, y las instrucciones 23, 24, 25 y 26 toman un tiempo O(1). Por lo tanto, la parte 2 toma un tiempo $O(n)$.

La instrucción 32 toma un tiempo $O(n)$ ya que recorre la máscara completa que es equivalente a recorrer todas las ciudades, y las instrucciones 33, 34, 35 (esta permite que el while termine) y 36 toman un tiempo O(1). Por lo tanto, la parte 3 de la ejecución toma un tiempo $O(n)$.

Por último, el return tarda un tiempo $O(n)$ ya que tiene que dar vuelta el tour dado que se ingresan las ciudades del final al comienzo. Habiendo hecho esto podemos calcular el orden de tiempo del algoritmo igual a $max(O(n * 2^n), O(n^2 * 2^n), O(n)) = O(n^2 * 2^n)$.

5.4.2 Cálculo de Espacio

La variable n de la inicialización ocupa un espacio O(1) y tanto costo como predecesor ocupan un espacio $O(2^n * n)$ ya que tienen 2^n filas (representando todos los posibles subconjuntos de las ciudades) y n columnas (representando las posibles ciudades finales para ese subconjunto).

Las variables de las instrucciones 10, 11, 19, 20, 21, 23, 30, 31 y 34 ocupan todas espacio O(1) y la variable tour de la instrucción 29 ocupa espacio $O(n)$, ya que ocupa $n + 1$ ciudades que componen el ciclo óptimo.

Habiendo hecho esto podemos calcular el orden de espacio del algoritmo igual a $max(O(n * 2^n), O(n)) = O(n * 2^n)$.

6 El sector del turismo en Uruguay

El TSP, además de tener un carácter muy teórico, se consolida como una herramienta poderosa para resolver problemas reales. En el caso de Uruguay, el TSP puede ser aplicado para optimizar rutas turísticas, potenciando no solo la experiencia de los visitantes, sino también impulsando el desarrollo económico y la generación de empleo en el país.

El sector turístico es crucial para la economía uruguaya, representando el 5,4% del PIB en 2023, con picos del 7,5% en períodos previos a la pandemia. Además, genera una importante proporción de divisas cerca del 12% del total de exportaciones y el 45% de las exportaciones de servicios, posicionándolo como el cuarto mayor sector exportador. Por otro lado, emplea a más de 121.000 personas, lo que equivale al 7,3% del total de ocupados en el país.

Sin embargo, la distribución geográfica del turismo en Uruguay muestra desequilibrios importantes, con una concentración en la costa oceánica y un menor desarrollo en el interior del país.

Aplicar modelos basados en el TSP para diseñar rutas eficientes para visitar las ciudades del interior del pais y volver al punto de origen, podría promover un turismo mas integral y beneficios para el conjunto del pais.

References

[1] Nguyen, Q. N. (2020). *Traveling Salesman Problem and Bellman-Held-Karp Algorithm*.

[2] Fomin, F. V., Kratsch, D. (2010). *Exact Exponential Algorithms*. Springer.

[3] García Travieso, M. V. (2014). *Problema del Viajante de Comercio (TSP). Métodos exactos de resolución*. Trabajo de Fin de Grado en Matemáticas, Estadística e Investigación Operativa, Universidad de La Laguna.

[4] Davendra, D. (Ed.). (2010). *Traveling Salesman Problem, Theory and Applications*. Rijeka, Croatia: InTech. ISBN: 978-953-307-426-9.

[5] LaRusic, J. (s.f.). *A Heuristic for Solving the Bottleneck Traveling Salesman Problem*. Bachelor of Computer Science candidate, University of New Brunswick.

[6] Cámara Uruguaya de Turismo. (2024). *El sector turismo en Uruguay: Presente y futuro de cara a las elecciones nacionales 2024*. Informe final realizado con el apoyo de PwC. Montevideo, Uruguay.

[7] Wikipedia. (s.f.). *Travelling salesman problem*. Recuperado el 25 de noviembre de 2024, de https://en.wikipedia.org/wiki/Travelling$_s alesman_p roblem$.