Métodos numéricos 2, Universidad del Valle

Traveling Salesman Problem

Javier Mejía-18638
José López - 181045

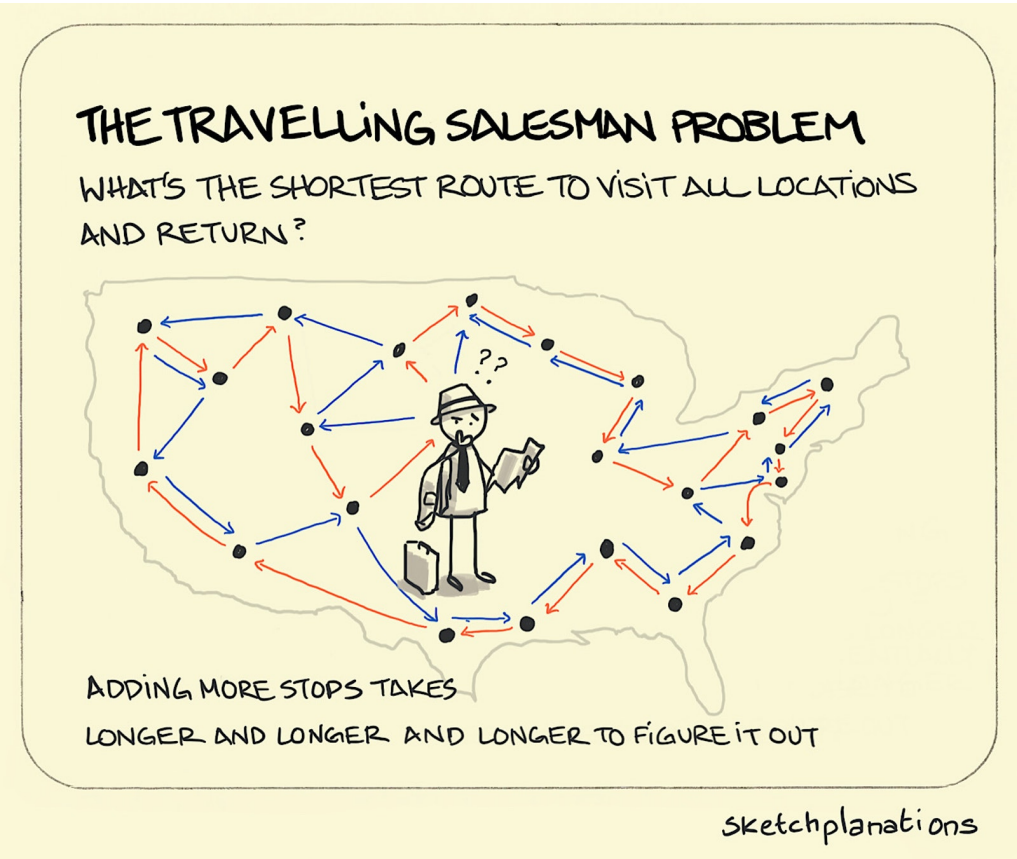
UVG
UNIVERSIDAD DEL VALLE DE GUATEMALA

November 22, 2021

Contents

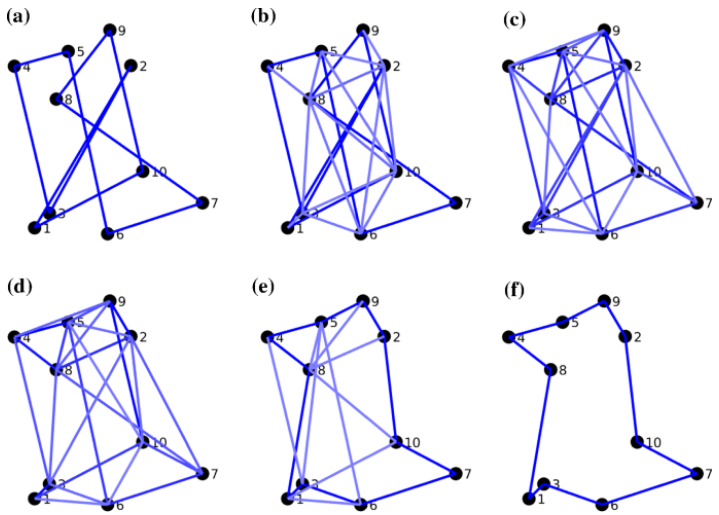
El Problema del Viajero (TSP)

Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?



Condiciones del Problema

- Comenzando desde la primer ciudad se debe recorrer el resto de ciudades exactamente una vez antes de regresar al punto de partida.
- La distancia entre las ciudades es conocida y se supone que es la misma en ambas direcciones.
- Solo se pueden utilizar las aristas brindadas en el problema (opcional).
- se trabaja con la distancia euclideada (en esta aproximación del problema).



Función objetivo del problema

Dado que se quiere obtener la distancia mínima tal que se recorren todos los puntos exactamente una vez, podemos describir la función objetivo de la siguiente manera. Considere el caso de N puntos, y sea $d(i,j)$ la distancia directa entre las ciudades i y j entonces se busca minimizar lo siguiente:

$$min\left\{z=\sum_{i=1}^{N}\sum_{j=1}^{N}x(i,j)d(i,j)\right\}$$

sujeto a las restricciones:

$$\sum_{j=1}^{N}x_{ij}=1,\quad i=1,...,N \qquad \sum_{i=1}^{N}x_{ij}=1,\quad j=1,..,N$$

con $x_{ij}\in\{0,1\}$ y con $i\neq j$

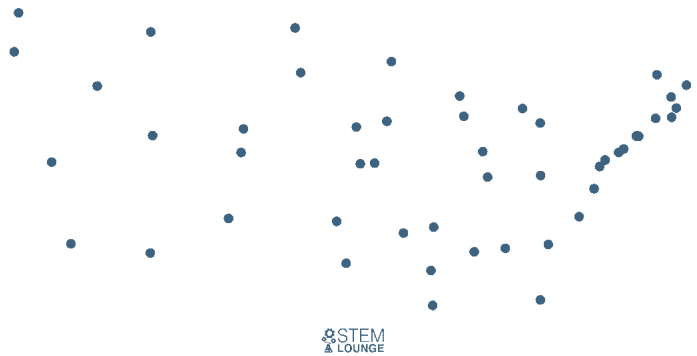
¿Qué hacer con un problema NP?

Una de las soluciones más comunes que se puede llegar a pensar para resolver el problema es probar todas las posibles combinaciones de ciudades que hay. Sin embargo, como hemos estudiado en el curso, esto es un problema NP porque es $O(n!)$.
En este caso, lo que se utiliza es un **PTAS (Esquema de aproximación en tiempo polinomial)**. Un algoritmo que calcule la solución en tiempo polinomial y que además, haga una buena aproximación de la misma.

Algoritmo Greedy

Algoritmo

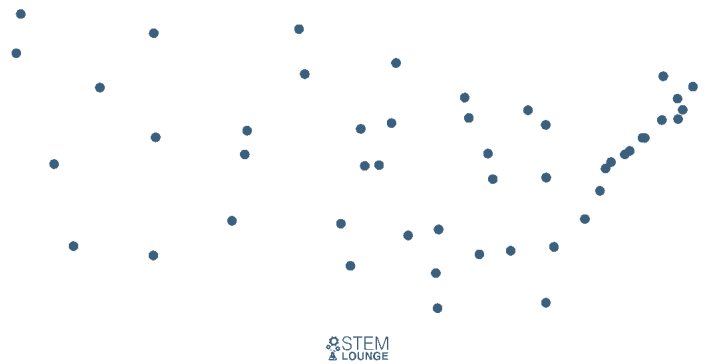
Es una estrategía de búsqueda que consiste en elegir la opción óptima en cada paso local. En nuestro caso, toma un subconjunto de puntos, y toma la arista de distancia más corta.



Algoritmo Nearest Neighbor

Algoritmo

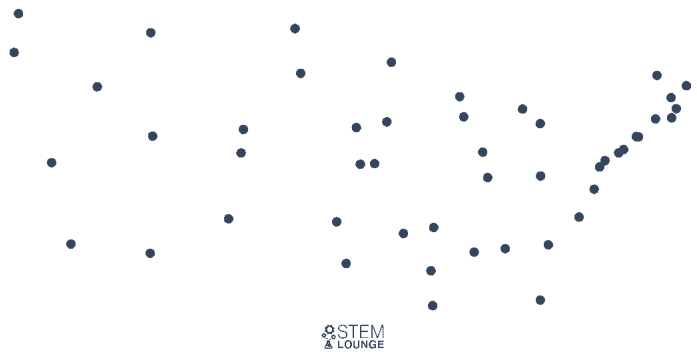
- Selecciona un punto aleatorio.
- Descubre la arista de menor distancia que lleva al siguiente punto.
- repite este proceso hasta haber pasado por todos los vértices.



Algoritmo 2 - opt

Algoritmo

Este es un algoritmo complementario para otros métodos de solución. Se complementa bien con el Nearest Neighbors y algoritmo recocido simulado. Consiste en intercambiar las aristas que se cruzan, de esta forma se reduce la distancia, el proceso converge en un grafo plano. (Es específico para el TSP).



Algoritmo genético: explicación

Este algoritmo busca simular el proceso de codificación genética. Se busca evolucionar cromosomas a traves de las iteraciones llamadas generaciones. En todas las generaciones los cromosomas son evaluados con una métrica de aptitud. Luego se aplican operaciones genéticas repetidamente.

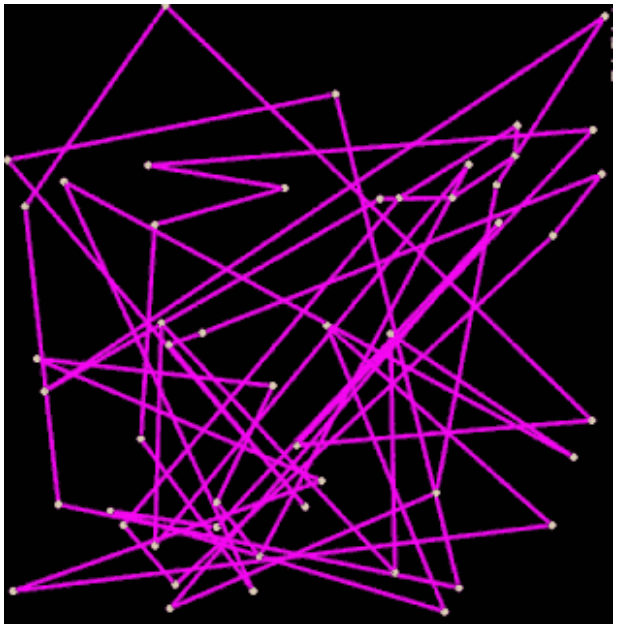
Algoritmo

- Inicialización: se genera aleatoriamente una población inicial.
- Evaluación: a cada uno de los cromosomas de esta población se le aplciará la función de aptitud.
- Criterio de paro: generalmente se coloca un máximo de generaciones o un máximo de generaciones sin mejora. Mientras el algoritmo no termine se aplican operadores genéticos.

Algoritmo Genético: demostración

Operadores Genéticos

- Selección: basandose en la aptitud de los cromosomas se selecciona a los que serán cruzados.
- Cruce: genera una nueva generación tomando como antecesores a la población con mejor apitutd.
- mutación: modifica al azar parte de los cromosomas.



Algoritmo genético: en nuestro caso

Diccionario

- Gen: es un punto representado con sus coordeandas (x, y)
- Cromosoma: son las rutas entre puntos.
- Función de aptitud: es la función que determina que tan buena es una ruta, es decir, que tan corta es:

$$f(x) = 1/x$$

  donde x representa la distancia de la ruta.
- Población: es la colección de todas las posibles conexiones que existen entre los n puntos.

Algoritmo de Christofides: explicación

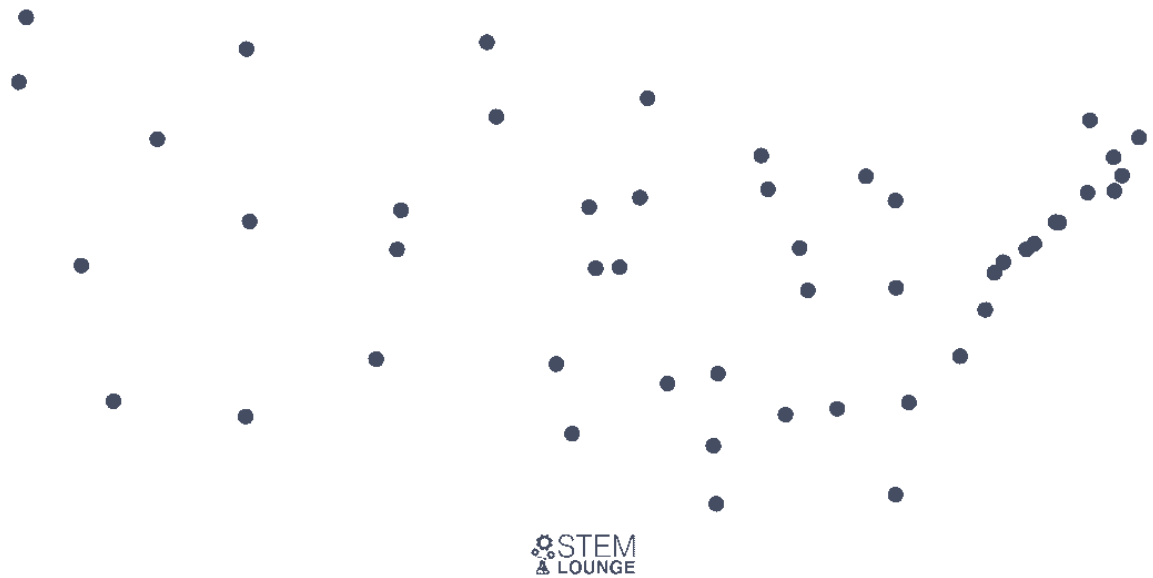
El algoritmo de Cristofides parte de la idea de que un arbol que cubre todos los vértices del grafo es casi un camino. Los pasos grandes son:

1. Calcular un Minimum spanning tree (MST) del grafo en cuestión. Esto se puede hacer en tiempo $O(n^2 \log n)$ usando el algoritmo de Prims, un algoritmo greedy que conecta todas las rutas más cortas (Djikstra). La longitud total del árbol es $cost(MST)$.
2. Dentro del grafo definido por el árbol, identificar los vértices de grado impar. Aseguramos que hay un número par de vértices de grado impar.

Algoritmo de Christofides: explicación

3. Conectar los vértices de grado impar con su "perfect match". La suma de longitudes de los perfect matches es $cost(M)$. Esto se puede hacer en $O(n^2)$
4. Hallar un circuito Euleriano en el grafo definido por el árbol.
5. Convertir ese circuito en un Hamiltoniano eliminando repeticiones.

Algoritmo de Cristofides: ejemplo



El Algoritmo de Cristofides es 3/2

Sea (V, E) un grafo conexo, y C el circuito hamiltoniano resultante del algoritmo de Cristofides. Sea $C*$ la solución óptima del TSP en (V, E), entonces se asegura que:

$$cost(C) \leq \frac{3}{2} cost(C*)$$

Demostración:
Nótese que la longitud de la solución óptima no puede ser menor que la longitud de un MST en (V, E), pues en ese caso el árbol no sería minimal en longitud. Entonces

$$cost(MST) \leq cost(C*)$$

Continuación

Por otro lado, nótese que $cost(M)$ es máximo en el caso en el que todos los vértices del MST tienen grado impar. En este caso, los matching constituyen la mitad del paseo, y dado que son perfect matches

$$cost(M) \leq \frac{1}{2} cost(C*)$$

Luego, notamos que

$$cost(C) \leq cost(MST) + cost(M) \leq \frac{3}{2} cost(C*)$$

Código y demostración en Python

Veamos paso a paso cómo funciona el algoritmo.

Comparación de tiempos

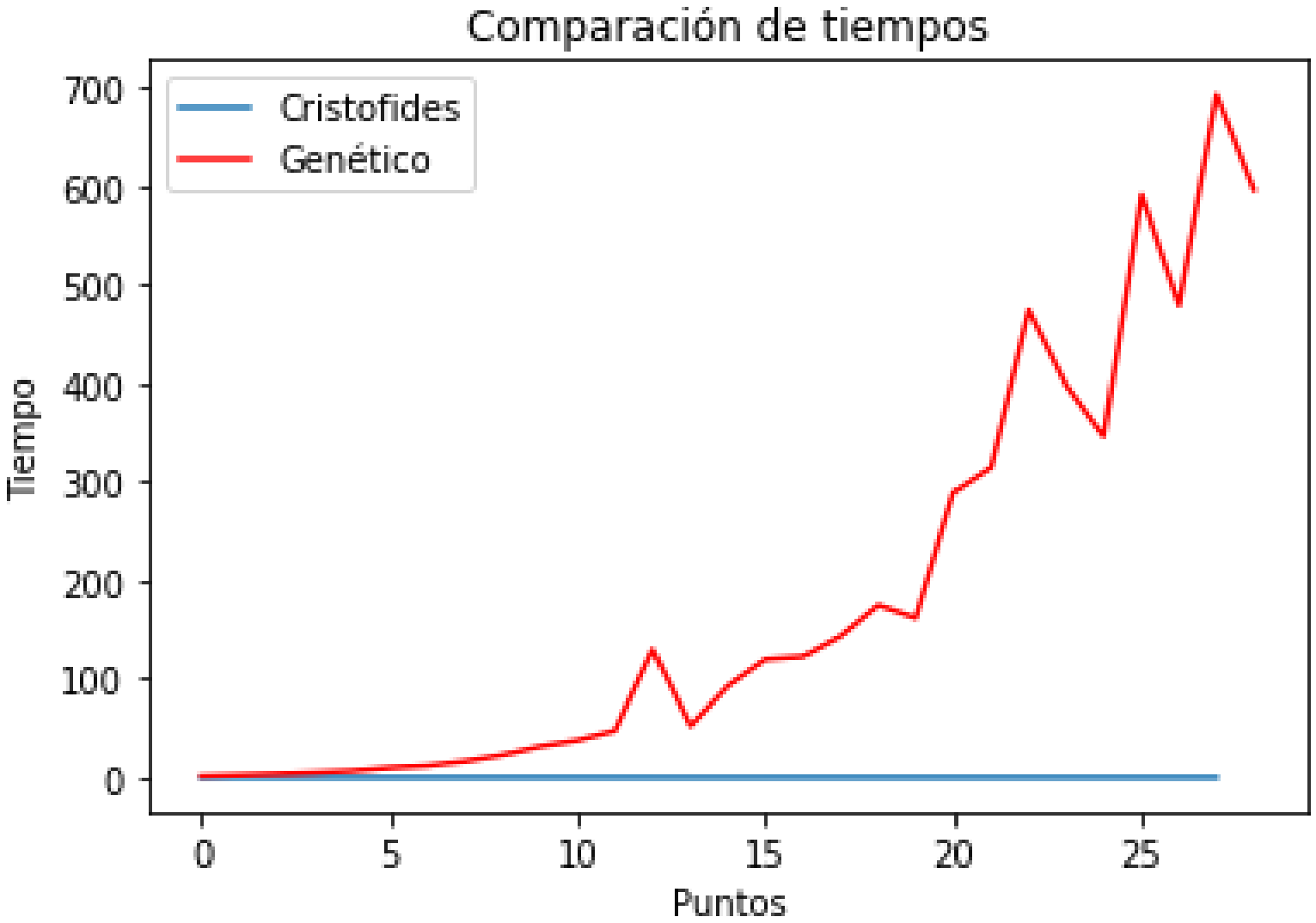


Figure: Comparación de tiempos de solución

Comparación de costo

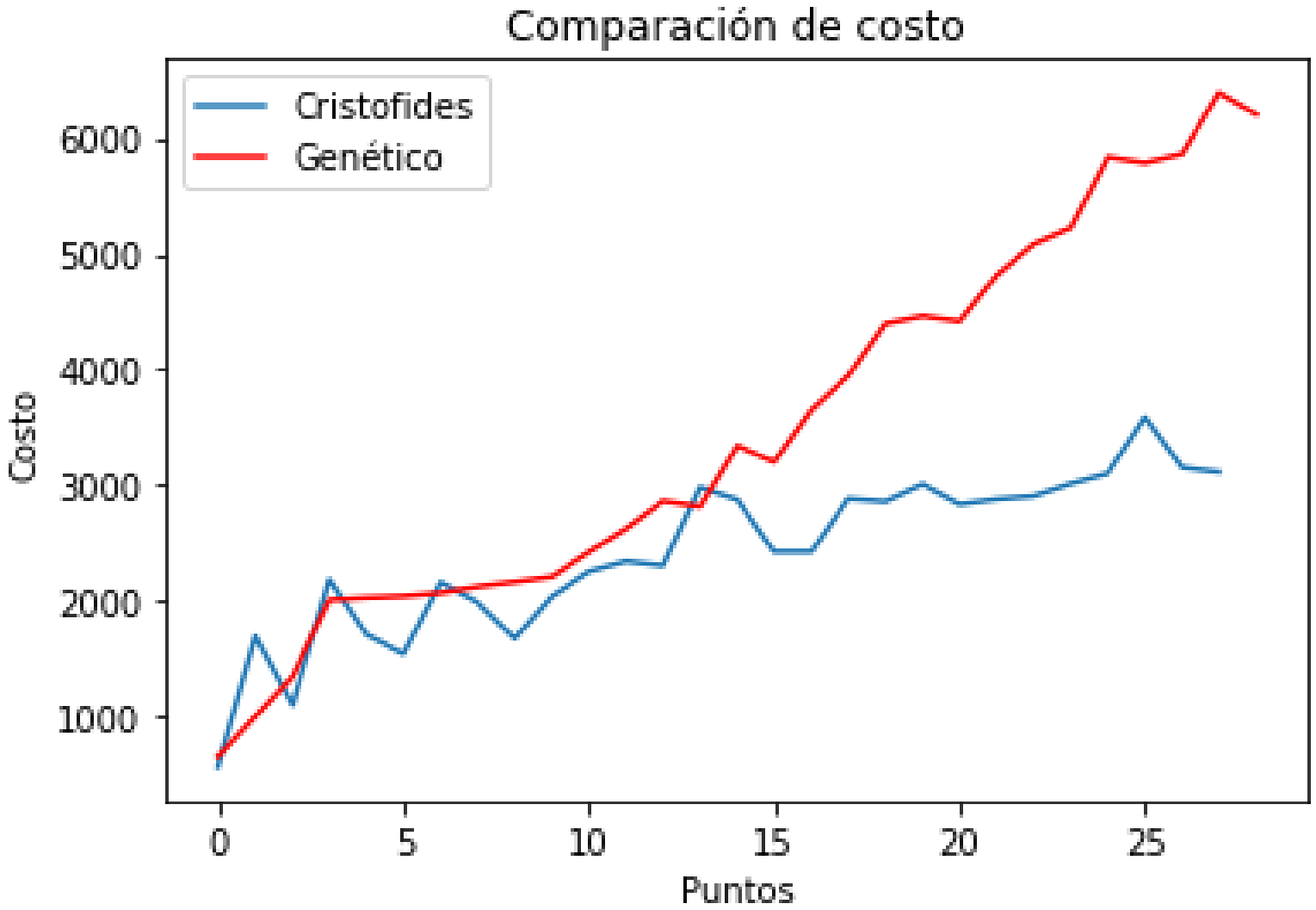


Figure: Comparación de costo de solución

Conclusiones

- Cuando trabajamos con problemas de tan alta complejidad es importante definir cuál es la meta de nuestra solución, si queremos resolver el problema completamente, o solo aproximarlo en un tiempo razonable.
- El algoritmo genético toma mucho más tiempo en resolver el problema, o en converger a un mínimo local mientras más se aumenta el número de puntos.

Conclusiones

- A pesar de que el algoritmo de Christodifes es una aproximación al problema, tiene mejores resultados cuando se utilizan muchos puntos. Esto se debe a que el criterio de convergencia del algoritmo genético solo le permite converger a una solución subóptima.
- En comparación, el algoritmo de Christofides es más eficiente que el algoritmo genético. Esto se debe principalmente a que el algoritmo genético es un problema NP - Hard, mientras que el algoritmo de Christofides si se puede llevar a tiempo polinomial.

Referencias

- Lawrence, W. (2021). 11 animated algorithms for the traveling salesman problem. Extraído de: https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/
- Korte, B., Vygen, J. (2008). Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics) (4th ed.). Springer.
- Trevisan, L. (2011). Combinatorial Optimization: Exact and Approximate Algorithms. Stanford University.