Metodología de resolución al problema del viajante de comercio

Axel Lucas Gómez Caamaño†

† *Dto. Electrónica UTN FRBA*
gomezaxel.lucas@gmail.com

***Resumen*— Se propone una metodología e implementación en Python para una solución al problema del viajante de comercio. El método fue propuesto por Claudio Verrastro, el cual se basa en reglas que acotan el problema de forma convergente garantizando una expansión polinómica cúbica y la existencia de uno o más lazos en perjuicio de la calidad del recorrido obtenido.**

***Palabras Clave*— Viajante de comercio, Travelling Salesman Problem, Implementación en Python.**

I. INTRODUCCIÓN

Se desarrolla en Python una metodología para resolver el problema del viajante de comercio. Dicho problema plantea: "Se tienen N ciudades y sus costos de ir de una ciudad a otra. Encontrar el camino óptimo tal que un viajante pase por todas las ciudades una sola vez y, al terminar, vuelva al punto de partida inicial". [1]

En dicho procedimiento se tuvieron en cuenta las reglas enunciadas en "El Viajante de Comercio". [2]

Adicionalmente, se ejecutó al algoritmo con problemas de la biblioteca TSP LIB del tipo euclideanos en 2D con representación de cada punto en coordenadas (x,y).

II. IMPLEMENTACIÓN

A continuación, se describen las reglas del procedimiento propuesto y la descripción de cada una de ellas con el fin de contrastar y detallar su correspondiente implementación.

A. Regla 1

"Se seleccionará, como caminos candidatos a formar el recorrido óptimo, a los "dos" de menor costo (de los N-1) que acceden a cada nodo. Se supone que seleccionando estos caminos se podrá entrar y salir de cada nodo por los accesos más cortos, de esta manera, se reduce a 2.N el número de caminos a considerar. De los 2.N caminos seleccionados, se eliminan aquellos que están repetidos. Usando la propiedad **Cij** = **Cji**". [2, p. 2]

Se conformará la lista **CS** de caminos seleccionados. En la medida que se ejecute el algoritmo, dicha lista, tendrá un tamaño variable, usando la propiedad **Cij** = **Cji**, en donde **Cij** representa el costo desde la ciudad **i** hasta la ciudad **j** mientras que **Cji** se asocia al costo del camino contrario.

Inicialmente, se toman ciudades:

$$1, 2, 3, \ldots, n \qquad \text{con } n \in N$$

Se arma una matriz con los costos asociados entre ciudades.

Luego, se hallan los 2 menores costos de cada ciudad/nodo, se arma una matriz de 2 columnas por N filas. Tengo una lista de elementos, en el que cada elemento se compone de una lista ciudades origen y destino:

$$CS = \begin{matrix}[[origen1, destino1],\\ [origen1, destino2],\\ [origen2, destino3],\\ [origen2, destino4],\\ \ldots\ ]\end{matrix}$$

Cada fila de la matriz posee los caminos ordenados por ciudad ascendente (**Cij** tal que **i < j**) y contempla que no haya filas duplicadas.

B. Regla 2

"Computar **EAi** para todos los nodos. Para el cómputo del número de arcos (**NAi**) que concurren en un nodo, basta con contar las repeticiones de un mismo subíndice, en los costos de los caminos seleccionados por Regla 1 (ya sea en la posición **i** como en la **j**)". [2, p. 3]

Se calcula el exceso de arcos **EAi** de cada nodo (o ciudad) y el excedente de caminos **EC** como sigue:

$$EAi = NAi - 2, \quad \text{para todo } i \qquad (1)$$

$$EA = \text{Sum}(EAi) = 2\,EC \qquad (2)$$

EAi se almacenan en una lista cuyo índice corresponde con cada ciudad:

$$[\ EAi, EAj, EAk, EAl, \ldots, EAn] \qquad \text{con N nodos}$$

C. Regla 3

"Ordenar de mayor a menor la lista Caminos Seleccionados (**CS**) por la Regla 1. Ordenar de menor a mayor los subíndices de los **CS** de forma que para todo **Cij** tal que **i < j**". [2, p. 3]

Se ordena de mayor a menor la matriz de caminos seleccionados por la Regla 1 en función a su costo en orden descendente y se arma una lista:

$$[\ Cij, Ckl, Cmn, Cop, \ldots\ ]$$

con los costos asociados a la matriz ordenada de caminos seleccionados:

$$CS = \begin{matrix}[[i, j],\\ [k, l],\\ [m, n],\\ [o, p],\\ \ldots\ ]\end{matrix}$$

D. Regla 4

"Explorar la lista **CS** de mayor a menor y remover los caminos **Cij** de la lista **CS** que cumplan con **EAi>0** y **EAj>0**. Actualizar **EAi=EAi-1** y **EAj=EAj-1**, **EC=EC-1** después de cada eliminación. Si **EC=0** Terminar (verificar la existencia de más de un lazo)." [2, p. 3]

Se remueven los caminos que conectan ciudades con exceso de arcos, tal que dichos excesos, de las ciudades que lo comprenden, son mayores a 0 y se actualizan **EAi**, **EAj** y **EC**.

E. Regla 5

"Si **EC>0** entonces armar una lista **A** con todos los **Aii** con **EAi>1** y todos los **Aij** con **EAi>0** y **EAj>0** que tengan un camino puente que NO pertenezca a **CS** y el puente no es un Camino **Cii**." [2, p. 4]

Se unifica el puente a calcular con el par de caminos a reemplazar bajo el nombre Ahorro Puente (**AP**). De forma tal que al calcular **Aii** no será necesario validar que el **AP** haya sido considerado o que el mismo represente un camino **Cii**.

El pseudocódigo de la implementación de la Regla 5 es:

```
  obtener los nodos tal que EAi > 0
   por cada uno de dichos nodos se evalúa su exceso de arcos
    si es mayor a 1:
     hallar caminos que pasen por el nodo i
      tomar dichos caminos de a pares, por cada par evaluar:
       puentes elegidos = []
       si el puente resultante del par:
        1. existe en la lista CS => descartar
       sino:
        hallar su ahorro tal que Aij = Cij + Cik - Cjk
        agregar dicho Ahorro Puente a la lista de Ahorro Puentes con ahorro calculado
    si es mayor a 0:
     tomar otro nodo que tenga EAj > 0, en caso de encontrarlo:
      hallar caminos que pasen por el nodo i y el nodo j
      tomar dichos caminos de a pares, por cada par evaluar:
       puentes elegidos = []
        si el puente resultante del par:
        1. existe en la lista CS => descartar
        2. es un puente ya tomado => descartar
        3. es un camino Cii => descartar
       sino:
        hallar su ahorro tal que Aii = Cij + Cik - Cjk
        agregar dicho Ahorro Puente a la lista de Ahorro Puentes con ahorro calculado (para no tener que calcular el ahorro del mismo puente).
```

La salida de aplicar la Regla 5 consiste en una variable con el valor de **EC**, una matriz cuyas filas corresponden a los puentes de la lista **A** diagramados de la siguiente forma:

$$\text{Ahorro Puentes (AP)} = \begin{matrix}[[\ i, [k, l], i\ ], \\ [\ m, [n, o], p\ ], \\ \dots\ ]\end{matrix}$$

Siendo:

- **[i, [k, l], i]** el puente **kl** que reemplaza a los pares **ik** e **il**.
- **[m, [n, o], p]** el puente **no** que reemplaza a los pares **mn** y **op**.

Nota: se optó por la nomenclatura anteriormente mencionada para poder tener la información completa tanto del puente como de los pares a los que reemplaza, simplificando la implementación.

Adicionalmente, como salida de la Regla 5, se tiene una lista con los ahorros asociados a cada fila de la matriz de Ahorro Puentes:

[Aikli, Amnop, …]

Siendo:

- **Aikli** el ahorro **Akl** correspondiente al reemplazo de los pares **ik** y **li**.
- **Amnop** el ahorro **Ano** correspondiente a la sustitución de los pares **mn** y **op**.

Se diferencian el par a suceder y su puente debido a que es posible que exista un mismo puente que remueva a otros pares de caminos que deberán considerarse.

Por ejemplo: **Amnop** no es equivalente a **Apnom** ya que el puente **no** sustituye a los pares **mn**, **op** y **pn**, **om** respectivamente.

F. Regla 6

"Ordenar de mayor a menor la lista". [2, p. 4]

Se ordena de mayor a menor la matriz representada por la lista **A** en función a los ahorros anteriormente calculados.

G. Regla 7

"Remover el primer elemento de la lista **A** el **Aij** mayor. Aplicar el ahorro removiendo de la lista **CS** los caminos **Cik** y **Cjl** y agregando el puente **Ckl**. Actualizar **EAi=EAi-1** y **EAj=EAj-1**, **EC=EC-1** después de cada eliminación. Si **EC=0** Terminar (verificar la existencia de más de un lazo)". [2, p. 4]

H. Regla 8

"Aplicar recursivamente las Reglas 6 y 7 hasta que **EC=0**". [2, p. 4]

Entonces:

1. Se quita el primer elemento de la matriz **A**, la cual contiene el puente y los pares a remover.
2. Se suprime también el elemento de la lista de ahorros asociado a dicho puente.
3. Se actualiza **EC**.
4. Se recalcula la matriz A para evitar sustituciones de puentes inválidos. Este cálculo previene que un próximo reemplazo derive en una negación de la condición referida por la Regla 5. Para simplificar la implementación, la "actualización" de dicha matriz consiste en aplicar nuevamente la Regla 5, cuyos parámetros de entrada son:
   a. Lista de caminos seleccionados **CS** (luego de la sustitución).
   b. Lista de exceso de arcos **EAi** en cada nodo.
   c. **EC**.

Se aplica de forma recursiva las Reglas 5, 6 y 7 teniendo en cuenta la condición final **EC=0**.

A partir de dicha condición, se implementó una función que, basándose en la última lista de caminos

seleccionados **CS**, computa todos los lazos del recorrido solución cuyo funcionamiento implica:

1. Separar la lista **CS** (matriz):

```
        [[origen1, destino1],
         [origen2, destino2],
CS =     [origen3, destino3],
         [origen4, destino4],
         ... ]
```

en dos listas distintas:

a. Orígenes → primer elemento de cada fila:

```
              [origen1,
               origen2,
Orígenes =     origen3,        con N nodos
               origen4,
                 …,
               origenN]
```

b. Destinos → segundo elemento de cada fila:

```
              [destino1,
               destino2,
Destinos =     destino3,       con N nodos
               destino4,
                 …,
               destinoN]
```

2. Suprimir el primer elemento de la lista Orígenes, agregar dicha ciudad al recorrido solución.
3. Quitar el primer elemento de la lista Destinos, anexándolo al recorrido final y almacenándolo temporalmente.
4. Buscar la ciudad recientemente agregada en los elementos restantes de ambas listas.
5. Una vez encontrada, tomar su análoga desde la otra lista. Por ejemplo, si se encontró la ciudad en la lista Orígenes, tomar el elemento de mismo subíndice, pero de la lista Destinos (o viceversa).
6. Repetir los pasos 2 al 5 hasta que la ciudad a incorporar en el recorrido solución sea la misma que el primer elemento de dicho recorrido. La condición anteriormente mencionada implica que se formó un lazo. Deberá verificarse la existencia elementos en las listas reducidas de Orígenes y Destinos. En caso afirmativo, se deberá de forma recursiva los pasos 2 al 5 hasta encontrar todos los lazos del recorrido.
7. Si se acaban las ciudades en Orígenes y Destinos significa que no habrá más lazos, en tal caso el recorrido obtenido es la solución al problema.

La función devuelve todos los lazos encontrados como resultado y su costo total. En caso de obtenerse un solo lazo, tanto el recorrido como el costo obtenidos serán concluyentes. Por el contrario, el costo resultante no contemplará la transición entre lazos por lo que dicho valor no será el definitivo. En dicho caso será necesario averiguar la unión entre lazos para computar el costo total.

III. RESOLUCIÓN AL PROBLEMA CON 20 CIUDADES EQUIDISTANTES

Tomando como ejemplo **iia_fijo20.tsp** el cual consiste en una distribución equidistante con 20 ciudades a lo largo de una circunferencia de radio 1000 (Fig. 1).

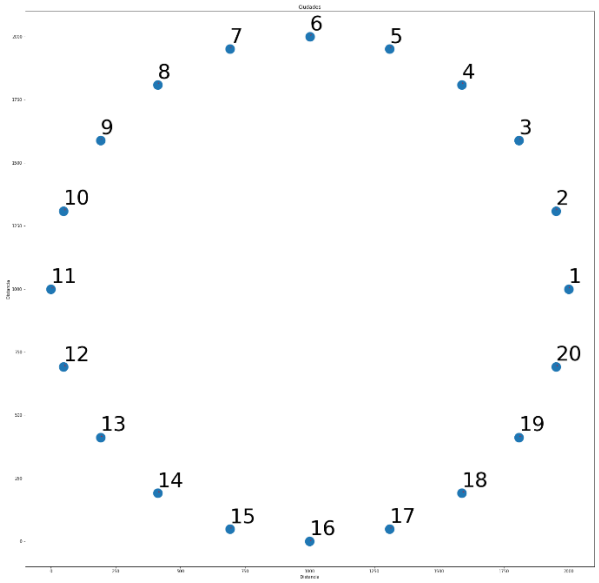

Fig. 1. Ciudades del problema iia_fijo20.tsp

En la Fig. 2 se visibilizan los caminos seleccionados (en rojo) referentes a la aplicación de las Reglas 1, 2 y 3 del algoritmo. La lista **CS** ordenada resulta:

[[1, 2], [1, 20], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [8, 9], [9, 10], [10, 11], [11, 12], [12, 13], [13, 14], [14, 15], [15, 16], [16, 17], [17, 18], [18, 19], [19, 20]]

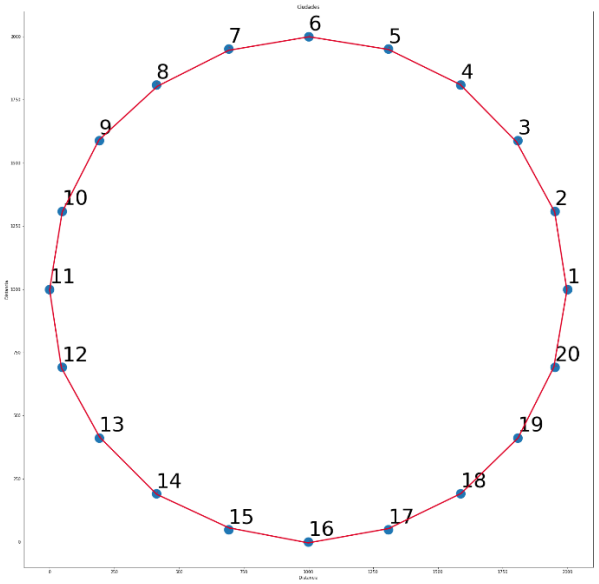

Fig. 2. Aplicación de las Reglas 1, 2 y 3 al problema iia_fijo20.tsp

Como **EC = 0** → no se deberán calcular los ahorros de los puentes ni reemplazar caminos ya que se prescinde de exceso de arcos para cualquiera de las ciudades.

Finalmente, se calcula el recorrido solución, el cual consiste de 1 solo lazo:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1]

IV. RESOLUCIÓN AL PROBLEMA CON 20 CIUDADES DE DISTANCIA VARIABLE

Tomando como ejemplo el problema **iia20.tsp** el cual consiste en la distribución aleatoria de 20 ciudades a lo largo de una circunferencia de radio 1000 (Fig. 3).

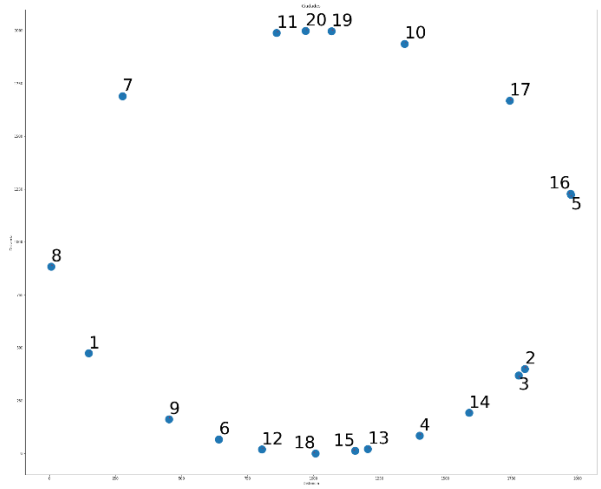

Fig. 3. Ciudades del problema iia20.tsp

En la Fig. 4 se visibilizan los caminos seleccionados (en rojo) referentes a las Reglas 1, 2 y 3 del algoritmo. La lista **CS** ordenada resulta:

[[1, 8], [1, 9], [2, 3], [2, 14], [3, 14], [4, 13], [4, 14], [5, 16], [5, 17], [6, 12], [6, 9], [7, 11], [7, 20], [8, 9], [9, 12], [10, 19], [10, 20], [11, 20], [11, 19], [12, 18], [13, 15], [13, 18], [15, 18], [16, 17], [10, 17], [19, 20]]

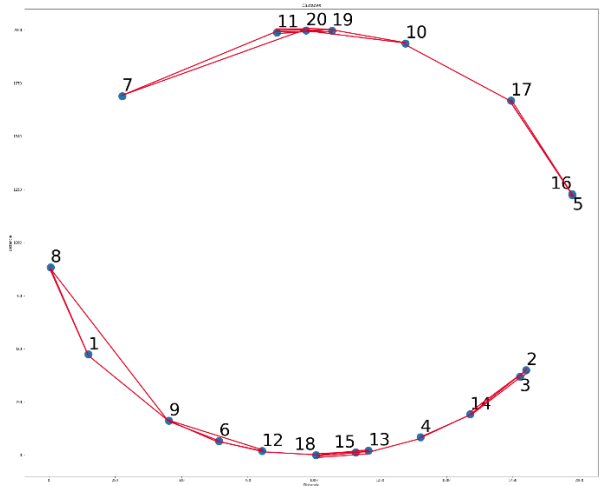

Fig. 4. Caminos seleccionados por las Reglas 1, 2 y 3 en el problema iia20.tsp

Al aplicar la Regla 4 se deben suprimir los caminos: [10, 17], [9, 12], [11, 19], [13, 18]; por lo que la lista **CS** ordenada (Fig. 5) resulta:

[[8, 9], [7, 20], [7, 11], [5, 17], [16, 17], [1, 9], [1, 8], [10, 20], [2, 14], [10, 19], [3, 14], [4, 14], [6, 9], [4, 13], [12, 18], [6, 12], [15, 18], [11, 20], [19, 20], [13, 15], [2, 3], [5, 16]]

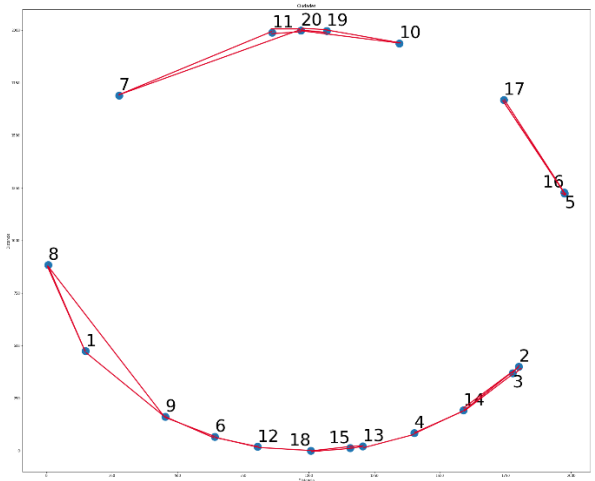

Fig. 5. Caminos resultantes de la Regla 4 del problema iia20.tsp

Debido a la aplicación de la Regla 5 se deben sustituir los caminos: [8, 9], [7, 20] por el puente [7, 8]; por lo que la lista **CS** ordenada (Fig. 6) resulta:

[[7, 11], [5, 17], [16, 17], [1, 9], [1, 8], [10, 20], [2, 14], [10, 19], [3, 14], [4, 14], [6, 9], [4, 13], [12, 18], [6, 12], [15, 18], [11, 20], [19, 20], [13, 15], [2, 3], [5, 16], [8, 7]]

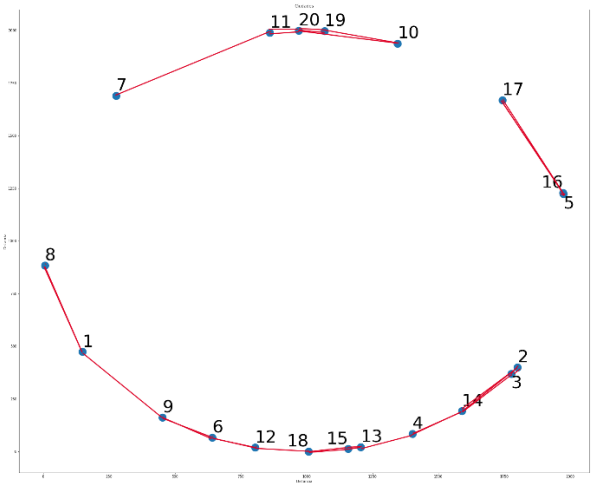

Fig. 6. Caminos resultantes de aplicar la Regla 5 (1° vez) al problema iia20.tsp

Al aplicar la Regla 5 nuevamente se deben reemplazar los caminos: [2, 14], [10, 20] por el puente [2, 10]; por lo que la lista **CS** ordenada (Fig. 7) resulta:

[[7, 11], [5, 17], [16, 17], [1, 9], [1, 8], [10, 19], [3, 14], [4, 14], [6, 9], [4, 13], [12, 18], [6, 12], [15, 18], [11, 20], [19, 20], [13, 15], [2, 3], [5, 16], [8, 7], [2, 10]]

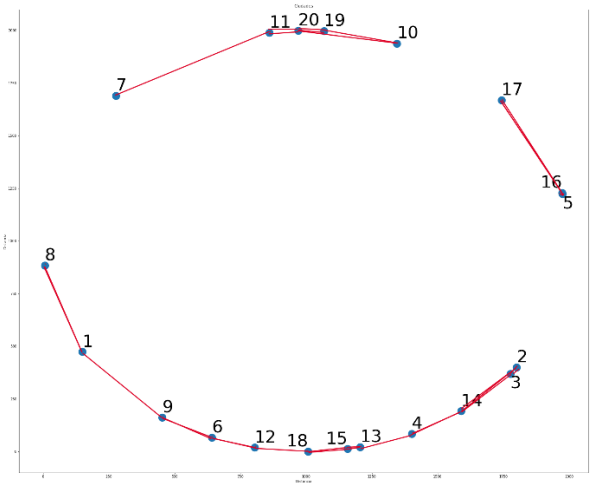

Fig. 7. Aplicación de la Regla 5 (2° vez) al problema iia20.tsp

Como **EC = 0** → se calcula el recorrido solución, el cual consiste de 2 lazos:

[1, 8, 7, 11, 20, 19, 10, 2, 3, 14, 4, 13, 15, 18, 12, 6, 9, 1] y **[5, 17, 16, 5]**

V. RESULTADOS OBTENIDOS

Se anexa el detalle de soluciones obtenidas por la ejecución del algoritmo para un conjunto de archivos pertenecientes a:

- Problemas enviados por los docentes. Nombres de archivo comienzan con **TSP_IN_** y **AGD_TSP_IN**.
- Problemas generados con una distribución aleatoria de ciudades sobre una circunferencia. Nombres de archivo **iiaN** con N número de ciudades.
- Problemas generados con una distribución equidistante de ciudades sobre una circunferencia. Nombres de archivo **iia_fijoN** con N número de ciudades.
- Problemas del conjunto TSP LIB 95 en formato euclidiano de 2 dimensiones. Nombres de archivo diversos.

VI. CONCLUSIONES FINALES

Una mejora considerable a agregar al algoritmo, tomando en cuenta los recursos a utilizar, consiste en considerar los elementos de la lista **A** (en regla 5) cuyo ahorro sea menor o igual al mayor ahorro anteriormente obtenido descartando el resto.

La implementación actual del método propuesto puede resolver problemas con 1000 ciudades en un tiempo aproximado de 5 horas, calculando todos los lazos que posea la solución, pero sin tener en cuenta el costo de transición entre lazos. Es por ello que, en el anexo, se detallan los resultados que evidencian la existencia de un valor resultante menor al costo óptimo del problema.

En comparación con la implementación de otros algoritmos de resolución de problemas TSP tal como los expuestos en el trabajo práctico "Algoritmos de búsqueda heurísticas y meta-heurísticas" [3] cabe recalcar lo siguiente:

- Tiempo: el algoritmo implementado es más rápido que "A*" [3, p. 4], pero más lento que "Recocido Simulado" [3, p. 9].
- Solución óptima del problema: el algoritmo implementado no asegura la solución óptima, como sí lo hace "A*" [3, p. 5].

Como propuesta a futuro, se deberá contemplar la unión de los lazos resultantes. Una posibilidad sería armar una lista con los dos menores costos de transición entre las ciudades de cada lazo y los nodos de los lazos restantes. Una vez completa la lista, tomar el menor costo y eliminar aquel o aquellos que unen lazos ya tenidos en cuenta. Finalmente, repetir el ciclo hasta que no queden lazos por unir.

REFERENCIAS

[1] «https://es.wikipedia.org/wiki/Problema_del_viajante,» [En línea].

[2] C. Verrastro, El Viajante de Comercio, 2022.

[3] A. Gómez Caamaño, Algoritmos de búsqueda heurísticas y meta-heurísticas, 2021.