CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OPTIMIZACIÓN INTELIGENTE

5° "A"

PROYECTO PARTE 1: RECOCIDO SIMULADO

Profesor: Aurora Torres Soto

Alumnos:

Espinoza Sánchez Joel Alejandro

Gómez Garza Dariana

González Arenas Fernando Francisco

Fecha de Entrega: Aguascalientes, Ags.**, 23** de noviembre de 2020

Índice

Introducción

1. Introducción a la optimización

La optimización, dentro y fuera del ámbito de las ciencias computacionales, es un conjunto de métodos que se aplican diariamente; en la opinión de Paredes (2019, p. 1) “la teoría de optimización clásica o programación matemática está constituida por un conjunto de resultados y métodos analíticos y numéricos enfocados a encontrar e identificar al mejor candidato de entre una colección de alternativas”. Paredes afirma que un problema de optimización es generalmente un problema de decisión.

Chapra, en su obra *“Métodos numéricos para ingenieros”* (2007, p. 353) también concuerda con Paredes y propone que “la localización de raíces y la optimización están relacionadas en el sentido de que ambas involucran valores iniciales y la búsqueda de un punto en una función”. Incluso ilustra la diferencia principal de ambos procesos en la figura 1.

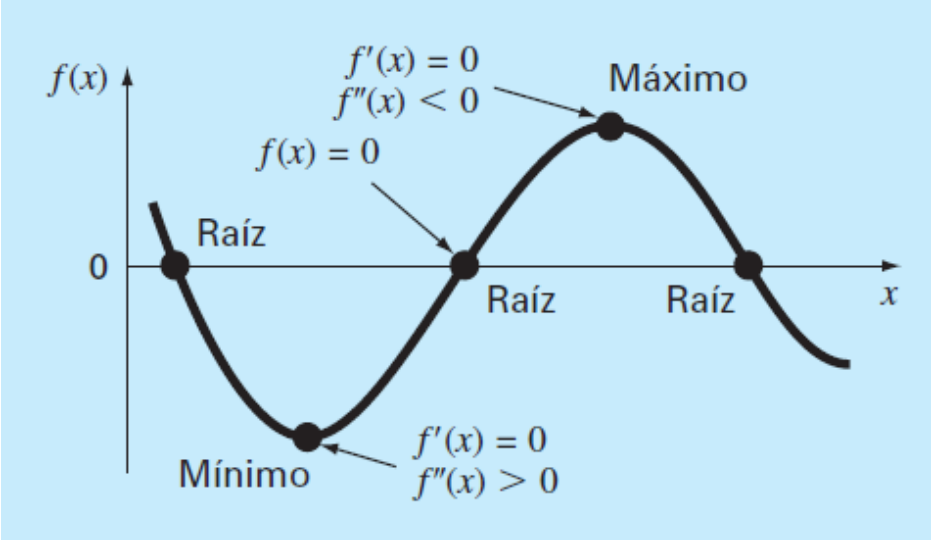


Figura 1: La diferencia de un problema de localización de raíces y un problema de optimización. Recuperada de: *Métodos numéricos para ingenieros* (Chapra, 2007, p.353).

Como una definición muy acertada del término que se quiere usar, podemos encontrar que, en términos de la investigación operativa, estos métodos de optimización “se enfocan en determinar la política a seguir para maximizar o minimizar la respuesta del sistema. Dicha respuesta, en general, es un indicador del tipo ‘Costo', ‘Producción’, ‘Ganancia’, etc., la cual es una función de la política seleccionada. Dicha respuesta se denomina objetivo, y la función asociada se llama función objetivo.” (Baquela, 2013, p. 13) y esta forma de conceptualizar los problemas de optimización será la que se usará para el desarrollo de los próximos ejes del presente documento.

Paredes menciona una cuestión de alta importancia en su definición de las técnicas de optimización que se debe a las cuestiones históricas que menciona Chapra sobre la solución de problemas de optimización, pues Chapra trata los métodos para resolver estas cuestiones como aquellos que se realizaron sin computadora y su historia y los métodos que se realizan con computadora.

Para tratar los métodos sin computadora, Chapra menciona que la solución de máximos y mínimos en funciones del cálculo en matemáticas era la manera en la que todo estudiante de ciencias e ingeniería realizaba estos procedimientos sin la computadora, mediante las bases de Bernoulli, Euler, Lagrange y otros que establecieron los fundamentos del cálculo de variaciones, el cual trata con la minimización de funciones, así como el método de multiplicadores de Lagrange que se desarrolló para optimizar problemas con restricciones (2007, p. 354).

Sin embargo, cuando se propusieron los métodos con el uso de una computadora, entre algunos de los principales problemas que existían puede destacarse el cómo se supone que una computadora derivaría funciones de cálculo, otra cuesión era que si la computadora encuentra un óptimo local, cómo sabría que ese punto de la función es el óptimo global de la función; por último hay que mencionar si merecería la pena evaluar de manera exhaustiva todo el espacio de posibilidades, es decir, el dominio de la función en su completitud. Este puede ser un problema, pues la mayoría de las funciones poseen como dominio el conjunto de todos los números reales, el cual es un espacio infinito de números.

Para poder resolver, sobre todo la última problemática planteada, se han diseñado distintos algoritmos para su aplicación en las ciencias computacionales que permiten encontrar soluciones que son buenas, pese a que no son las mejores y es entonces cuando surge el concepto de los algoritmos heurísticos para resolver estas problemáticas.

2. Las metaheurísticas

Según Taha (2012, p. 351) una heurística “está diseñada para encontrar buenas soluciones aproximadas a problemas combinatorios difíciles que de lo contrario no pueden resolverse mediante los algoritmos de optimización disponibles”. Taha también menciona que una heurística es una técnica de búsqueda directa que utiliza reglas favorables prácticas para localizar aquellas soluciones que son favorables.

Taha menciona también que la ventaja del uso de estas herramientas es que en general determina buenas soluciones con rapidez utilizando reglas de solución simple. la desventaja es que la calidad de la solución, en relación a la óptima normalmente se desconoce.

Para resolver el problema mediante estas estrategias, es necesario modelarlo matemáticamente, es decir, tendremos que expresarlo en términos matemáticos, esto requiere “identificar y definir las variables que están implicadas en dicho

problema” (Paredes, 2019, p. 1). Una vez que se identifiquen las variables, es posible expresar una función para la que tendremos que encontrar el mejor valor.

Paredes menciona basándose en un ejemplo de minimización de volumen de un cuerpo los elementos fundamentales de los problemas de optimización; él define tres importantes módulos: las variables de decisión, las restricciones y la función objetivo (2019, p. 2).

1) *Variables de decisión:* El primer elemento clave al formular estos problemas consiste en la selección de variables independientes que sean adecuadas para caracterizar los posibles diseños candidatos y las condiciones del sistema y su funcionamiento.
   Normalmente se eligen como variables de decisión aquellas que tienen un impacto significativo en la función objetivo. Paredes realiza una convención en la que si se eligen n variables, se representen mediante vectores columna de $\mathbb{R}^n$:
   $$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix}$$
   O vectores fila:
   $$x^T = (x_1 \quad x_2 \quad \cdots \quad x_n)$$
2) *Restricciones:* A su vez, también se necesita establecer mediante ecuaciones o inecuaciones las relaciones que existen entre las variables de decisión; esto puede expresarse matemáticamente mediante restricciones de igualdad de la forma:
   $$h(x) = h(x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n) = 0$$
   También pueden realizarse restricciones de desigualdad, que tendrían la forma siguiente:
   $$h(x) = h(x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n) \leq 0$$
   Donde $h: A \subseteq \mathbb{R}^n \to \mathbb{R}$ es una función real de variables reales definida sobre un conjunto de números reales A.
3) *Función objetivo:* Ésta permite determinar los mejores valores para las variables de decisión. Paredes (2019, p. 3) menciona que es muy importante determinar los mejores valores para las variables de decisión, pues plantear una función objetivo que cumpla con esta característica, enriquecerá a la metaheurística para elegir entre el conjunto de las variables de decisión.

Para plantear los elementos anteriores será necesario usar algún procedimiento planteado y aplicarse como algoritmo heurístico, es por ello que estas bases se enfocarán al desarrollo del algoritmo conocido como el recocido simulado.

2.1 El Recocido Simulado

El recocido simulado es una técnica que “escapa del atrapamiento en un óptimo local utilizando una condición de probabilidad que acepta o rechaza un movimiento inferior” (Taha, 2012, p. 365).

Está inspirado en el proceso metalúrgico que es el recocido, el cual consiste en calentar un metal a altas temperaturas y después observar el cambio de sus propiedades al dejarlo enfriar lentamente, por ello el nombre, recocido simulado, ya que se busca aparentar la condición de temperaturas altas al “mejorar alguna de sus propiedades de un objeto y que estos actos causen cambios en su descripción” (Vázquez, 1994, p. 26).

En el documento *“Práctica 5: Recocido Simulado”* podemos encontrar que el recocido simulado “nace a partir de una fuerte analogía entre los elementos que constituyen un problema de optimización y el proceso de recocido” (Torres, 2020, p. 1) los cuales son los siguientes:

- Las soluciones del problema son equivalentes a los estados del sistema físico.
- El costo de una solución es equivalente a la energía de un estado.
- Un parámetro de control juega el papel de la temperatura

Otra característica importante del recocido simulado mencionada en el documento *“Práctica 5: Recocido Simulado”* es que se puede ver como una iteración del algoritmo de Metrópolis, que es el mecanismo que permite la aceptación de soluciones cuando no mejoran a la actual:

1) Dado un estado actual i con energía $E(i)$.
2) Se genera un nuevo estado j con un mecanismo de perturbación de i.
3) Se calcula la energía de j, $E(j)$.
4) Si $E(j) - E(i) \leq 0$, entonces se acepta el estado j (si se va a minimizar).
5) Si no, se aceptará j con probabilidad:

$$P = e^{\frac{E(i)-E(j)}{K_B T}}$$

Asimismo, hay que mencionar la necesidad de añadir un programa de enfriamiento para que el algoritmo sea capaz de iniciar en una temperatura muy alta disminuirla a medida que se avanza en el proceso de optimización.

El algoritmo siguiente representa la versión conocida como monotónica, pues la temperatura siempre va en descenso. En este algoritmo se deben manejar los siguientes parámetros:

- T_0: Es la temperatura inicial.
- α: Es el factor de enfriamiento.

- ρ: Es el factor de reducción de iteraciones.
- K: Es el número de iteraciones a cierta T.

Cabe destacar que para hacer uso del recocido simulado se deben establecer:

1) Un punto de partida para la búsqueda.
2) Un mecanismo de perturbación.
3) Un programa de enfriamiento.

El algoritmo presenta el pseudocódigo observado en la figura 2:

```
Generar i = i0  // (Solución inicial)
T = T0
Inicializar parámetros // (α, ρ, K, k=0)
Mientras ( No se alcance la condición de PARO) hacer
        Mientras  (k < K)  hacer     // k: contador de iteraciones
                Generar j en N(i)
                Si (E(j) – E(i) < 0)      //Minimización
                        i = j
                Si no
                Generar un número r al azar
                    Si (r < exp [(E(i) – E(j))/T])   //  Minimización
                        i = j
                k = k + 1
        Fin del mientras (interno)
T = αT
K = ρK
k = 0
Si T=Tfin   PARO   //Criterio de paro por temperatura
Fin del mientras (externo)
Mostrar i, E(i)
```

Figura 2: Pseudocódigo del recocido simulado.
Recuperada de: *Optimización Inteligente. Práctica 5: Recocido Simulado* (Torres, 2020, p. 2).

Las bases del recocido simulado se verán aplicadas en un problema de matemáticas discretas el cual incluye el uso de grafos.

3. Preliminares de grafos y la biblioteca TSPLIB

3.1 Grafos

Una definición de un grafo G ya establecida "consiste en un conjunto V de vértices (o nodos) y un conjunto E de aristas (o arcos) tal que cada arista $e \in E$ se asocia con un par no ordenado de vértices" (Johnsonbaugh, 2005, p. 320).

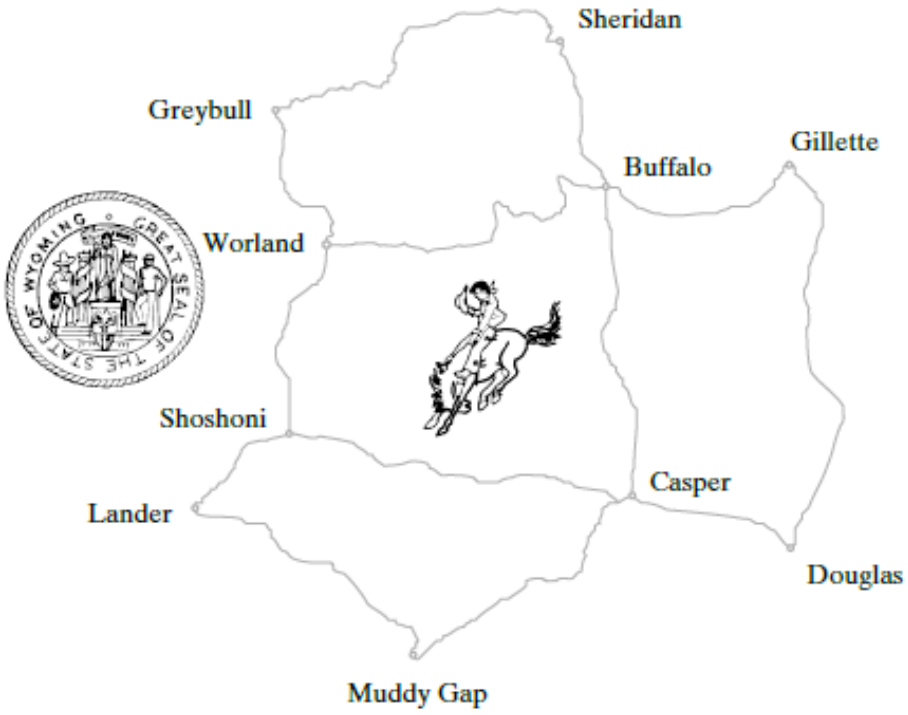


Figura 3: Parte del sistema de carreteras de Wyoming.
Recuperada de: *Matemáticas Discretas* (Johnsonbaugh, 2005, p. 319).

Johnsonbaugh nos permite ver mediante un ejemplo la aplicación de los grafos, donde podemos tener un mapa con carreteras de Wyoming como el que se aprecia en la figura 3 y el cual es posible modelar como un grafo a gusto propio, Johnsonbaugh propone dos modelos de grafos encontrados en la figura 4.

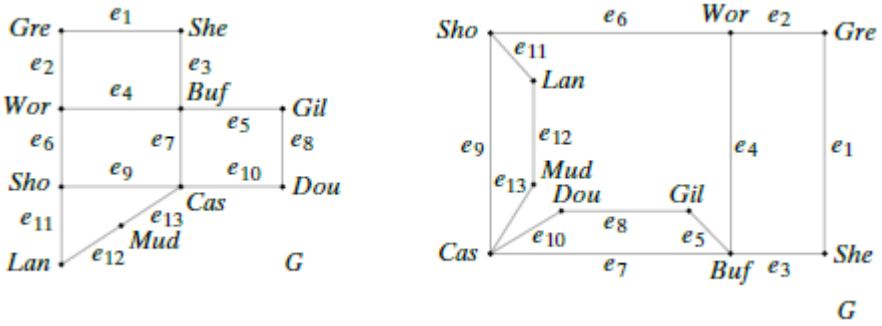


Figura 4: Modelos de grafos para el sistema de carreteras de la figura 3.
Recuperada de: *Matemáticas Discretas* (Johnsonbaugh, 2005, p. 319).

Observemos que sin importar la construcción del grafo, los propuestos en la figura 4 consisten del conjunto de vértices:

$$V = \{Gre, She, Wor, Buf, Gil, Sho, Cas, Dou, Lan, Mud\}$$

Y el conjunto de aristas:

$$E = \{e_1, e_2, \dots, e_{13}\}$$

Modelando un grafo de ciudades se pueden plantear problemas de recorrido de ciudades a los cuales Johnsonbaugh se refiere como trayectorias o rutas, donde puede proponerse una trayectoria estándar y ahora, con los conocimientos del recocido simulado, realizar mecanismos de perturbación, tales como los que apenas señala que son el subviaje inverso o la inversión de dos nodos aleatorios, que serán usados como mecanismos de perturbación del recocido simulado en estos problemas en situaciones ya planteadas por la biblioteca TSPLIB.

3.2 La biblioteca TSPLIB

Reinelt en su documento que explica la biblioteca TSPLIB menciona que es una biblioteca de unas instancias bases para el TSP (*Travelling Salesman Problem* o en español el Problema del Agente Viajero) y problemas relacionados de varios recursos y de distintos tipos.

En la biblioteca TSPLIB se encuentran disponibles instancias de las siguientes clasificaciones de problemas:

- Problemas del agente viajero simétricos.
- Problemas de ciclos hamiltonianos.
- Problemas del agente viajero asimétricos.
- Problemas de ordenamiento secuencial.
- Problemas de ruteo de vehículos capacitados.

Dentro de los problemas TSP simétricos, se habrán elegido dos instancias para trabajar el algoritmo del recocido simulado con las bases heurísticas mencionadas previamente y se expondrán las soluciones desarrolladas en ANSI C a los problemas elegidos.

Descripción de la solución

La problemática presentada abordaba dos instancias de la TSPLIB las cuales se pueden encontrar en su mismo repositorio bajo los nombres `gr21.tsp` y `kroD100.tsp` (véanse anexos 1 y 2 para revisar el contenido de ambos archivos).

1. La instancia `gr21.tsp`

La instancia `gr21.tsp` es un problema llamado el problema de Groetschel de 21 ciudades que contiene 21 nodos para trabajar. El tipo de peso de las aristas se encuentra explícito, es decir que la conexión entre dos nodos arbitrarios v_1 y v_2 tendremos el valor del costo que se requiere al pasar de uno al otro y como es un problema simétrico, también viceversa.

Sin embargo, el formato de los pesos de las aristas se encuentra dado en su forma de matriz triangular inferior, por lo que, denotemos como $e_1, e_2, e_3, \ldots$ a los pesos otorgados por el documento, éstos se verían representados en formato matricial de la siguiente manera:

$$E = \begin{bmatrix} e_1 & 0 & 0 & \cdots & 0 \\ e_2 & e_3 & 0 & \cdots & 0 \\ e_4 & e_5 & e_6 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e_k & e_{k+1} & e_{k+2} & \cdots & e_n \end{bmatrix}$$

Para modelar correctamente la matriz, de acuerdo a la situación del programa, donde se busca que toda la matriz sea una matriz de adyacencia indicando el peso de cada nodo, era necesario también rellenar la matriz triangular superior (a excepción de la diagonal, que ya estaba cubierta), debido a que se trata de un problema simétrico, es decir, dados dos elementos $n, m \in V$ y por las propiedades de las matrices, podemos representar el conjunto faltante de pesos usando la matriz transpuesta de E (E^T) de modo que el peso que se guarda en la matriz en el elemento a_{mn} será el mismo que encontraremos en el elemento a_{nm}.

A esta matriz de adyacencia entre los pesos de cada par de nodos, dentro del modelado de la solución la llamaremos **`graph`** y la definiremos como:

$$graph = \begin{bmatrix} P(v_1, v_1) & P(v_1, v_2) & \cdots & P(v_1, v_{21}) \\ P(v_2, v_1) & P(v_2, v_2) & \cdots & P(v_2, v_{21}) \\ \vdots & \vdots & \ddots & \vdots \\ P(v_{21}, v_1) & P(v_{21}, v_2) & \cdots & P(v_{21}, v_{21}) \end{bmatrix}$$

Donde $P(v_n, v_m)$ denota al peso que existe de viajar del vértice n al vértice m.

También, para este caso llamaremos como **qv** a la cantidad de vértices (quantity of vertexes). Para este caso, $qv = 21$.

2. La instancia `kroD100.tsp`

La instancia `kroD100.tsp` es un problema con el nombre del problema D de Krolak/Felts/Nelson de 100 ciudades que contiene 100 nodos para trabajar. El tipo de peso de las aristas se deberá calcular, ya que se tienen distancias euclidianas en dos dimensiones, esto quiere decir que el formato de los pesos en las aristas estará implícito dentro del documento, pues se da la lista de los 100 nodos numerados junto con dos columnas más; hay que interpretar estos datos como par de coordenadas en el plano, es decir, en un espacio bidimensional.

Para modelar este problema, entonces, en un principio no se puede construir la estructura `graph`, primeramente se extraerán los datos y se guardarán en un módulo temporal al que llamaremos **`list`**, constituido por el número del nodo, y las coordenadas en el plano de dicho nodo, para los 100 nodos.

Definimos a `list` de la siguiente manera:

$$list = \begin{bmatrix} v_1 & x_1 & y_1 \\ v_2 & x_2 & y_2 \\ v_3 & x_3 & y_3 \\ \vdots & \vdots & \vdots \\ v_{100} & x_{100} & y_{100} \end{bmatrix}$$

Es decir, la primera columna indicará de cuál vértice se trata, la segunda columna serán las respectivas posiciones en una dimensión del plano y la tercera columna serán las posiciones correspondientes a la otra dimensión del plano.

Construimos entonces a **`graph`** como (véase anexo 3 para ver la deducción completa del cálculo de cada elemento de `graph`):

$$\begin{bmatrix} \sqrt{(x_1-x_1)^2+(y_1-y_1)^2} & \sqrt{(x_1-x_2)^2+(y_1-y_2)^2} & \cdots & \sqrt{(x_1-x_{100})^2+(y_1-y_{100})^2} \\ \sqrt{(x_2-x_1)^2+(y_2-y_1)^2} & \sqrt{(x_2-x_2)^2+(y_2-y_2)^2} & \cdots & \sqrt{(x_2-x_{100})^2+(y_2-y_{100})^2} \\ \sqrt{(x_3-x_1)^2+(y_3-y_1)^2} & \sqrt{(x_3-x_2)^2+(y_3-y_2)^2} & \cdots & \sqrt{(x_3-x_{100})^2+(y_3-y_{100})^2} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{(x_{100}-x_1)^2+(y_{100}-y_1)^2} & \sqrt{(x_{100}-x_2)^2+(y_{100}-y_2)^2} & \cdots & \sqrt{(x_{100}-x_{100})^2+(y_{100}-y_{100})^2} \end{bmatrix}$$

Al igual que la instancia anterior, ésta también plantea un problema simétrico respecto a las aristas.

Ahora, en esta instancia, denotaremos como **`qv`** igualmente a la cantidad de vértices de `graph`, por lo que para esta instancia, $qv = 100$.

3. Resolviendo un problema arbitrario

Sin importar cuál de las dos instancias anteriores se tomen (incluso si se quiere personalizar un problema), analizaremos de manera general el problema para tratar de resolver de manera arbitraria problemas heurísticos aplicados a grafos mediante el recocido simulado.

Entonces, sea `qv` la cantidad de vértices que el grafo tiene y `graph` la matriz de adyacencia de dimensión $qv \times qv$ la cual contiene los pesos de viajar a través de los grafos; el primer paso para modelar formalmente la solución será con una lista que simbolice la trayectoria que se hará a través de los nodos, la cual denominaremos como **`tour`**. Esta estructura será un vector que dimensión $qv + 1$ definida como:

$$tour = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_{qv} \\ t_1 \end{bmatrix}$$

Donde cada elemento t_i de `tour` podrá tomar algún valor del conjunto $\{1,2,3,\dots,qv\}$ sin repetición, que simbolizará el número del nodo de la trayectoria, así el primer elemento de `tour` representa el nodo de comienzo de la trayectoria y el último elemento de `tour` será el nodo de finalización de la trayectoria y por la definición de `tour` tanto el primer elemento como el último de `tour` deben ser iguales.

Debido a la natrualeza del recocido simulado que se explicó anteriormente, se necesitarán dos alternativas de soluciones i y j; para este caso, análogamente `tour` sería la solución i y se construirá una nueva solución j la cual se llamará en esta forma de modelado como `newTour`, construida a partir de una perturbación de `tour`.

Para explicar el modelado completo del problema, se introducirán las variables faltantes que se deben mencionar, éstas son las siguientes:

- `T`: Es la temperatura.
- T_{fin}: Es la temperatura final.
- `α`: Es el factor de enfriamiento.
- `ρ`: Es el factor de reducción de iteraciones.
- `K`: Es el número de iteraciones a cierta `T`.
- `qv`: Ya fue mencionada y es la cantidad de vértices que tendrá el grafo.
- `r1`: Es un número aleatorio.
- `r2`: Es un número aleatorio.
- `r3`: Es un número aleatorio.

A partir de estas variables, se construyen las estructuras ya explicadas `tour` y `newTour` que tendrán asociadas dos variables para evaluar la energía, que son `ftour` y `fnewTour` respectivamente. La energía será el costo del viaje total de `tour` y `newTour`.

Asimismo se construye otra estructura que denotaremos como `availableV`, este será un vector de dimensión `qv` el cuál estará llenado de unos o ceros. Cada posición del vector simboliza a un respectivo nodo del grafo y si la posición contiene un 1 significa que para la generación de una trayectoria, es posible tomar dicho nodo; al tomarlo, el valor cambiará a 0, por lo que las posiciones con 0 significa que no se puede tomar dicho nodo del grafo.

El algoritmo comienza al definir las variables `T`, T_{fin}, `α`, `ρ` y `K`; el ajuste de parámetros es muy importante para que el funcionamiento del recocido simulado sea mejor. Se definirán con mayor formalidad todos los parámetros en la descripción de la herramienta, pero a continuación se presenta un acercamiento al dominio que presenta cada variable.

La primera variable a definir es la temperatura **`T`**, la cual se definirá al principio del procedimiento como la temperatura inicial. Este parámetro puede ser un número real positivo cualquiera. El siguiente a ajustar será uno muy importante, pues ayudará al experimento a parar, éste es $\mathbf{T_{fin}}$; su propósito es establecer la temperatura final del recocido simulado, el cual también puede calibrarse como cualquier número real positivo con la única condición que:

$$T_{fin} < T$$

Ahora, también se tiene el parámetro $\boldsymbol{\alpha}$. Su objetivo es ser un factor para reducir la temperatura, pues cuando la comparación entre los viajes `tour` y `newTour` se haga un número determinado de veces, la temperatura disminuirá y según su ajuste, disminuirá con mayor o menor velocidad hacia T_{fin}. Debido a su naturaleza, este parámetro está definido para todos los números reales que se encuentran en el intervalo: $(0,1)$.

Se mencionó previamente que la comparación de trayectorias se realizaría un número determinado de veces; este está dado por la variable **`K`**, la cual será la cantidad de iteraciones a cierta temperatura. Este parámetro puede ajustarse con cualquier valor entero positivo. Así como la temperatura, la variable `K` también tendrá un factor de reducción de iteraciones denotado por la letra griega $\boldsymbol{\rho}$. El propósito es similar al de α, pero usando las iteraciones, es decir, reducirá la cantidad de iteraciones al terminar una evaluación completa, por lo que se puede configurar con cualquier valor real dentro del intervalo: $(0,1)$.

Una vez que se termina el ajuste de dichos parámetros, escogemos una trayectoria inicial que cumpla con las restricciones donde todos los nodos estén conectados de forma lógica; asimismo calculamos su peso total el cual definimos como `ftour`.

Para generar un nuevo viaje, definido como `newTour`, tomamos el recorrido inicial y con un mecanismo de perturbación, modificamos ligeramente el trayecto definido en `tour` el cual también tiene que validarse como lógico y que esté conectado de forma permitida. En este caso podemos utilizar una técnica conocida como el subviaje inverso que se realiza tomando 2 posiciones aleatorias del recorrido y el pequeño recorrido formado entre estas dos posiciones se invertirá (claramente deberá validarse el nuevo camino o de lo contrario, deberá generarse otro).

Así por ejemplo, si se tiene un vector `tour` del tipo:

$$tour = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad \cdots \quad t_{qv} \quad t_1]$$

Y si definimos que:

$$1 < n < m \leq qv$$

Donde n y m serían números para seleccionar aleatoriamente dos nodos, es decir:

$$tour = [t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_{n-1} \quad t_n \quad t_{n+1} \quad \cdots \quad t_{m-1} \quad t_m \quad t_{m+1} \quad \cdots \quad t_{qv} \quad t_1]$$

Entonces se tendría un subviaje:

$$subtour = [t_n \quad t_{n+1} \quad \cdots \quad t_{m-1} \quad t_m]$$

Y este mismo se invertirá:

$$subtour\_inv = [t_m \quad t_{m-1} \quad \cdots \quad t_{n+1} \quad t_n]$$

Así, se insertará a la trayectoria principal y éste sería el producto del mecanismo de perturbación del subviaje inverso:

$$newTour = [t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_{n-1} \quad t_m \quad t_{m-1} \quad \cdots \quad t_{n+1} \quad t_n \quad t_{m+1} \quad \cdots \quad t_{qv} \quad t_1]$$

Otro método de perturbación que se puede utilizar es el método de elección e inversión de dos nodos, que consiste en elegir 2 nodos al azar y cambiarlos de posición. Los mecanismos de perturbación, además de hacer una vía diferente, deben de asegurarse de que ésta es válida, es decir, que los nodos escogidos estén conectados y sea posible llegar al nodo final. Si esto no es así, debe repetirse este paso hasta que el viaje sea válido.

Nuevamente se retoma el ejemplo suponiendo si se tiene un vector `tour` del tipo:

$$tour = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad \cdots \quad t_{qv} \quad t_1]$$

Otra vez definimos que:

$$1 < n < m \leq qv$$

Donde n y m serían números para seleccionar aleatoriamente dos nodos, es decir:

$$tour = [t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_{n-1} \quad t_n \quad t_{n+1} \quad \cdots \quad t_{m-1} \quad t_m \quad t_{m+1} \quad \cdots \quad t_{qv} \quad t_1]$$

De este modo, este mecanismo consistiría en construir un nuevo viaje que resulte en el siguiente:

$$newTour = [t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_{n-1} \quad t_m \quad t_{n+1} \quad \cdots \quad t_{m-1} \quad t_n \quad t_{m+1} \quad \cdots \quad t_{qv} \quad t_1]$$

Aunque al modelarlo, se busque elegir entre uno de estos dos mecanismos por todo un experimento, también se quiere buscar distintas soluciones, lo que incluye cambiar los nodos de inicio y fin; para ello, también se propuso otro parámetro el cual denotamos como **p**. Este parámetro es una probabilidad definido en el intervalo $[0,1)$ el cual representa la probabilidad en la cual se elegiría mejor cambiar los nodos de inicio y fin por sobre cualquiera de los dos mecanismos de perturbación seleccionado para experimentar y explicado previamente.

La forma de trabajo de este cambio es simple, pues consiste en generar número n tal que:

$$1 < n \leq qv$$

De modo que si definimos un vector `tour` del tipo:

$$tour = [t_1 \quad t_2 \quad t_3 \quad \cdots \quad t_n \quad \cdots \quad t_{qv} \quad t_1]$$

La operación a realizar produciría una estructura `newTour` de la siguiente forma:

$$newTour = [t_n \quad t_2 \quad t_3 \quad \cdots \quad t_1 \quad \cdots \quad t_{qv} \quad t_n]$$

Una vez teniendo el nuevo trayecto `newTour`, debemos obtener el peso total de él, denominado como `fnewTour`. Al terminar el proceso anterior se minimizará `tour`, si el peso de `newTour` es menor al peso de `tour`, es decir si $fnewTour < ftour$, se evaluará `newTour` como el camino actual y ahora el camino de `newTour` será `tour`; por lo tanto, `tour` se descartará. Por el contrario, si el peso de `newTour` es mayor a `tour` se someterá a la probabilidad de Boltzmann, es decir, se aceptará `newTour` como el viaje a evaluar bajo la probabilidad:

$$e^{\frac{newTour - tour}{T}}$$

De no aceptarse, `tour` se mantendrá como el recorrido de evaluación principal. El proceso se repetirá evaluando `tour` y su peso hasta la minimización, K veces. Se obtiene una nueva temperatura multiplicando αT, y también se actualiza K multiplicando ρK. Este proceso se realiza mientras T sea mayor a T_{fin}.

Y ahora con estos procedimientos planteados, se llevaron estas ideas a programarlo en lenguaje C para obtener de manera más sencilla los datos, y los resultados, así como el uso de la herramienta se comentan a continuación.

Descripción de la herramienta

La herramienta posee el nombre *“ProyectoP1 JoelDarianaFer”* como primer anexo externo al documento, también pueden presentarse como anexos las dos instancias cuyos nombres son *“gr21.tsp.txt”* y *“kroD100.tsp.txt”*.

Puede abrirse el código fuente al compilar y ejecutar el producto de este código, tendremos la pantalla de menú inicial para el programa; en él podemos escoger primeramente si se trabajará con alguna de las dos instancias o si se prefiere trabajar con un grafo personalizado, la imagen del menú puede apreciarse en la figura 5.

```
============================== RECOCIDO SIMULADO ==============================

--------------------------------
|  Calibración del algoritmo:
|  Seleccione la instancia a trabajar:
|    0: Grafo Personalizado
|    1: Instancia pequeña (gr21.tsp)
|    2: Instancia grande (kroD100.tsp)
|
```

Figura 5: Menú inicial del programa *“ProyectoP1 JoelDarianaFer”*.

Es importante mencionar que tanto en este menú como en todos los ajustes de variables, es necesario tener en mente los posibles valores que cada opción puede recibir, y aunque ya se han mencionado los dominios de algunas variables, aquí se recopilará toda la información, pues por ejemplo, este primer menú sólo admitirá como valores de entrada el conjunto: $\{0,1,2\}$ y puede verse en la figura 5 que elegir la opción 0 significaría tomar un grafo personalizado, tomar la opción 1 sería cargar la instancia `gr21.tsp` y la opción 2 llevará a cargar la instancia `kroD100.tsp`.

Una vez se presione la tecla “`ENTER`”, se accederá a un menú diferente en el que se calibrará el algoritmo. Se obtendrán opciones de personalización distintas según se haya elegido previamente la instancia de trabajo. Si se eligió analizar un grafo personalizado, las opciones serán las presentadas en la figura 6:

```
--------------------------------
|  Calibración del algoritmo:
|  Seleccione algún número si desea cambiar su valor (o 0 para continuar):
|  0: Listo. Continuar
|  1: T0 (Temperatura inicial): 140.0000
|  2: Tfin (Temperatura final): 10.0000
|  3: alpha (Factor de enfriamiento): 0.5000
|  4: rho (Factor de reducción de iteraciones): 0.5000
|  5: K (Número de iteraciones en T): 30
|  6: autom (Modo de acción): 1
|  7: disturbance (Modo en el que operará el mecanismo de perturbación): 1
|  8: p (Probabilidad de perturbar los nodos de inicio y fin): 0.5000
|  9: qv (Cantidad de vértices que posee el grafo): 7
|
```

Figura 6: Menú de ajuste de parámetros primarios para grafos personalizados del programa *“ProyectoP1 JoelDarianaFer”*.

Las opciones ya poseen unos valores prefijados que son sugerencia estándar del programa para ejecutar bajo dichos ajustes. En este menú se podrán elegir las opciones del 1 al 9 para modificar cada opción o 0 para continuar con el programa; y al elegir algún parámetro, el programa pedirá un nuevo valor para dicha variable, como puede observarse en la figura 7.

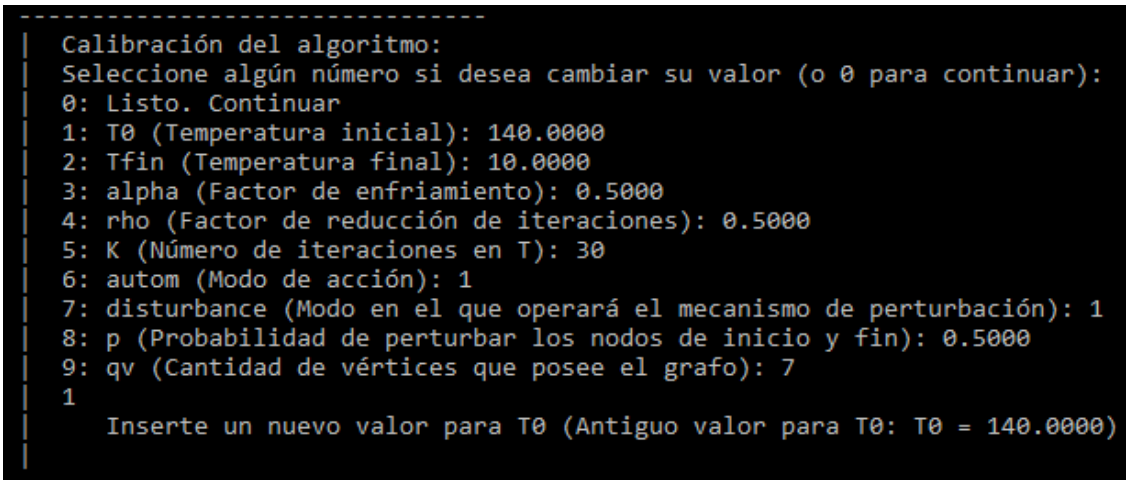


Figura 7: Menú de ajuste de parámetros primarios y ejemplo de personalización de valor para la temperatura incial del programa *"ProyectoP1 JoelDarianaFer"*.

Recordemos los parámetros:

- T_0: Simboliza la temperatura con la que se iniciará la ejecución del programa. Recordemos que su valor puede ser cualquier número real positivo.
- T_{fin}: Será la temperatura en la que el programa terminará de evaluar el experimento. Su valor puede ser cualquier número real positivo con la única restricción que debe ser menor a T_0.
- `alpha`: Refiriéndose a α, es el factor de reducción de temperatura que afectará cada fin de evaluación. Este parámetro puede definirse como cualquier número real dentro del intervalo $(0,1)$.
- `rho`: Refiriéndose a ρ, será el factor de reducción de iteraciones. Puede ajustarse este valor, al igual que α, como cualquier número real dentro del intervalo $(0,1)$.
- `K`: Serán las iteraciones que se harán a cierta temperatura. Puede calibrarse la variable como cualquier número entero positivo, pero como en el proceso esta variable podrá tomar valores que incluyan decimales, también se podría definir usando cualquier número real positivo, pero las iteraciones así las hará hasta alcanzar al número entero más próximo por exceso.
- `autom`: Es una variable de personalización para el usuario. Esta se agregó en un principio para crear una opción de despliegue de información para desarrollador y otra para usuario, sin embargo, ambas podría desearlas el usuario, por lo que se mantuvo. Esta variable acepta únicamente valores dentro del conjunto $\{0,1\}$ y quiere decir si se llevará a cabo en modo

automático o no. Si se le asigna el valor de 1, el procedimiento se realizará sin detenerse hasta el final, pero al asignarle el valor de 0 hará lo mismo pero esperando al usuario a que dé "`ENTER`" y pueda detenerse y analizar qué ocurrió a cada paso del algoritmo.

- `disturbance`: Esta variable está pensada como personalización del algoritmo a gusto del usuario. Perturbación en inglés es disturbance, pues esta variable acepta entradas dentro del conjunto $\{1,2\}$ dando opciones para elegir entre dos mecanismos de perturbación que se alcanzaron a implementar. Si se carga esta opción con el valor 1, entonces el mecanismo de perturbación estándar que usará será el subviaje inverso, pero si se elige la opción 2, hará por defecto la elección e inversión de dos nodos aleatorios.
- `p`: Esta variable permite agregar un mecanismo de perturbación secundario que consiste en cambiar de ciudad de inicio y fin con cierta probabilidad, por ello, esta variable está definida dentro del intervalo $[0,1)$ pues si esta probabilidad es aceptada, en lugar de perturbar por el modo seleccionado en `disturbance`, se optará por cambiar de ciudad de inicio y fin. Como se menciona, es opcional y al dejarla en 0, nunca se realizarán cambios de las ciudades de arranque y finalización; pero se busca que no sean valores tan altos para que el programa también pueda explorar opciones, ya sea por subviaje inverso o elección e inversión de nodos.
- `qv`: Es la cantidad de vértices que tendrá el grafo personalizado. Evidentemente sólo puede aceptar valores enteros positivos.

Se hace mención que se sigan los dominios cuidadosamente, porque si no se siguen los dominios de cada variable, el programa dejará de funcionar, pues no se desarrolló un método para no permitirle al usuario avanzar si alguno de estos valores es inválido.

Una vez que se ajustan estos valores, se accederá a otro menú para calibrar las estructuras que se generan alrededor del valor de `qv` como se ve en la figura 8.

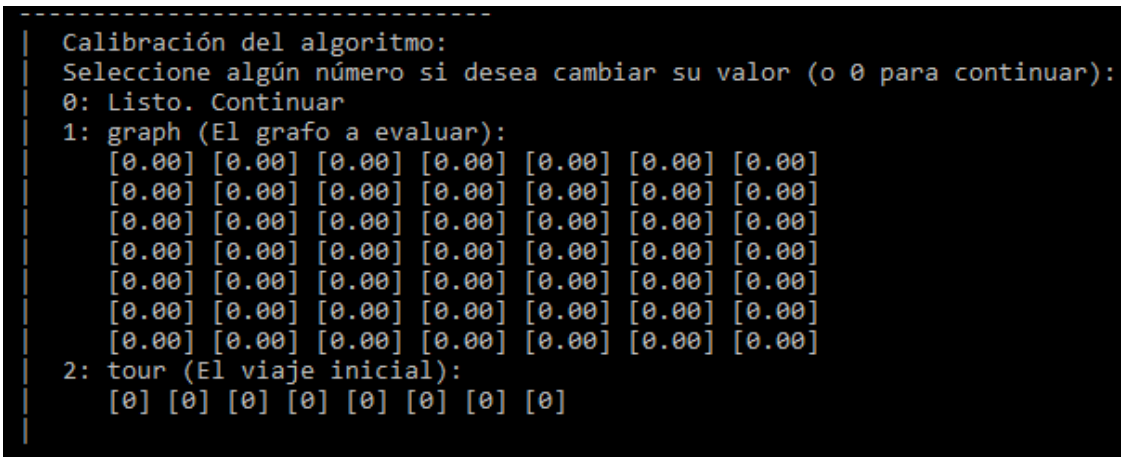


Figura 8: Menú de ajuste de estructuras del programa *"ProyectoP1 JoelDarianaFer"*.

El programa comienza pensando que se llena lógicamente sobre todo la estructura `tour` ya que `graph` es completamente personalizable, pero `tour` tiene que cumplir que el viaje sea válido, por lo que nuevamente se le pide al usuario ingresar un trayecto en el que existan las aristas que se insertan a la estructura.

Una vez que terminan de calibrarse los arreglos de este menú, se procederá a comenzar el recocido simulado, pero previo a ello, se explicarán brevemente las diferencias al elegir cualquier de las instanciasde los archivos.

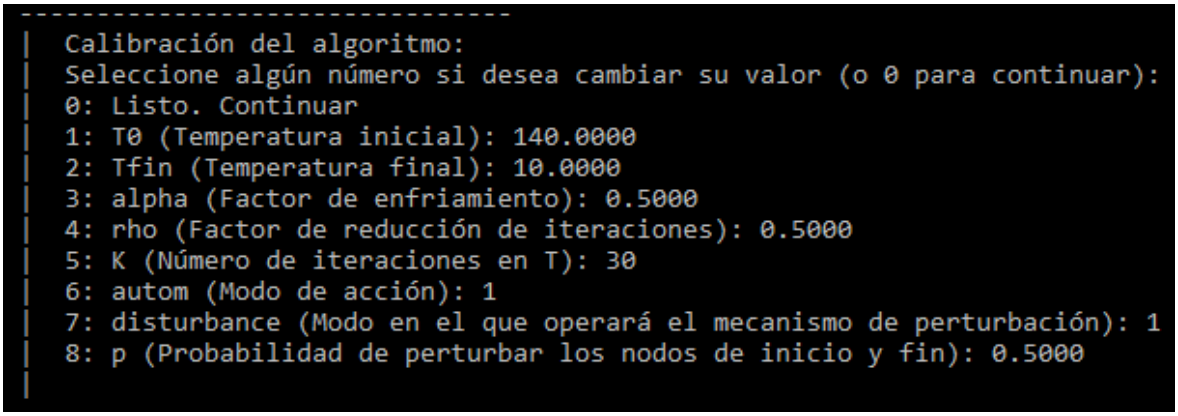


Figura 9: Menú de ajuste de parámetros primarios para cualquiera de las dos instancias almacenadas en archivos del programa *"ProyectoP1 JoelDarianaFer"*.

El único cambio en el menú de parámetros, como puede verse en la figura 9, primarios es que la opción de modificar la cantidad de vértices ya no aparece y no se modifica, pues ya está cargado al leer el respectivo archivo, por ello tampoco se da el acceso al menú de personalización de estructuras, ya que el programa extrae esta información directamente del archivo que se le indica.

En este caso, se muestra un poco de información del archivo, el grafo que genera y comienza el procedimiento del recocido simulado.

Hay que realizar una aclaración con respecto a los archivos, pues como primer requerimiento, se pide que la herramiento y los archivos de las instancias se encuentren en el mismo directorio, ya que no localizará los archivos si estos se encuentran en una carpeta distinta. También hay que hacer énfasis en que probablemente se haya descargado de diferente manera los archivos de la biblioteca TSPLIB, ya que estos archivos, por naturaleza son extensión `.tsp`, sin embargo, se descargaron y probaron con una extensión agregada `.txt` por lo que el programa puede arrojar un error de lectura de archivo y es altamente probable que sea por este motivo. Lo único que debe hacerse es eliminar en las líneas indicadas por la figura 10 la extensión `.txt` y la lectura se realizará normal, es decir, modificar la instrucción del programa que no lea un archivo `gr21.tsp.txt`, sino que lea `gr21.tsp`.

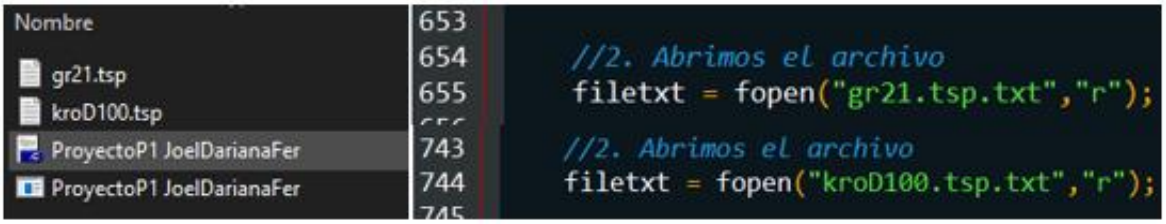


Figura 10: Consideraciones importantes en relación a los archivos para el funcionamiento correcto de la herramienta *"ProyectoP1 JoelDarianaFer"*.

Después de este procedimiento, todas las instancias llevarán a cabo el mismo trabajo, el cual cambiará según los valores configurados para dichas instancias.

Como un buen ajuste de parámetros, empíricamente se encontraron los siguientes ajustes (véase anexo 4 para mayor información del rendimiento):

- $T_0 = 300.$
- $T_{fin} = 5.$
- $alpha = 0.85.$
- $rho = 0.95.$
- $K = 200.$
- $autom = 1.$
- $disturbance = 1.$
- $p = 0.2.$

Conclusiones

Conclusión general: Ñe.

Joel Alejandro Espinoza Sánchez: Ñe.

Dariana Gómez Garza: Ñe.

Fernando Francisco González Arenas: Ñe.

Bibliografía

- Baquela, E. (2013). *Optimización Matemática con R. Volumen I: Introducción al modelado y resolución de problemas.* Madrid: Bubok Publishing.
- Chapra, S. (2007). *Métodos numéricos para ingenieros.* Michigan: McGraw Hill.
- Johnsonbaugh, R. (2005). *Matemáticas discretas.* Chicago: Pearson.
- Paredes, S. (2019). *Fundamentos de Optimización.* Cartagena: Editorial Universidad Politécnica de Cartagena.
- Reinelt, G. (1995). *TSPLIB 95.* Heidelberg: Editorial Universität Heidelberg.
- Taha, H. (2012). *Investigación de operaciones.* Arkansas: Pearson.
- Torres, A. (2020). *Optimización Inteligente. Práctica 5: Recocido Simulado.* México: Universidad Autónoma de Aguascalientes.
- Vázquez, M. (1994). *Recocido simulado: Un nuevo algoritmo para la optimación de estructuras.* Madrid: Editorial Escuela Técnica de Arquitectura de Madrid.

Anexos

Anexo 1: Contenido del documento `gr21.tsp`.

```
NAME: gr21
TYPE: TSP
COMMENT: 21-city problem (Groetschel)
DIMENSION: 21
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: LOWER_DIAG_ROW
EDGE_WEIGHT_SECTION
     0   510     0   635   355     0    91   415   605     0
   385   585   390   350     0   155   475   495   120   240
     0   110   480   570    78   320    96     0   130   500
   540    97   285    36    29     0   490   605   295   460
   120   350   425   390     0   370   320   700   280   590
   365   350   370   625     0   155   380   640    63   430
   200   160   175   535   240     0    68   440   575    27
   320    91    48    67   430   300    90     0   610   360
   705   520   835   605   590   610   865   250   480   545
     0   655   235   585   555   750   615   625   645   775
   285   515   585   190     0   480    81   435   380   575
   440   455   465   600   245   345   415   295   170     0
   265   480   420   235   125   125   200   165   230   475
   310   205   715   650   475     0   255   440   755   235
   650   370   320   350   680   150   175   265   400   435
   385   485     0   450   270   625   345   660   430   420
   440   690    77   310   380   180   215   190   545   225
     0   170   445   750   160   495   265   220   240   600
   235   125   170   485   525   405   375    87   315     0
   240   290   590   140   480   255   205   220   515   150
   100   170   390   425   255   395   205   220   155     0
   380   140   495   280   480   340   350   370   505   185
   240   310   345   280   105   380   280   165   305   150
     0
EOF
```

Anexo 2: Contenido del documento `kroD100.tsp`.

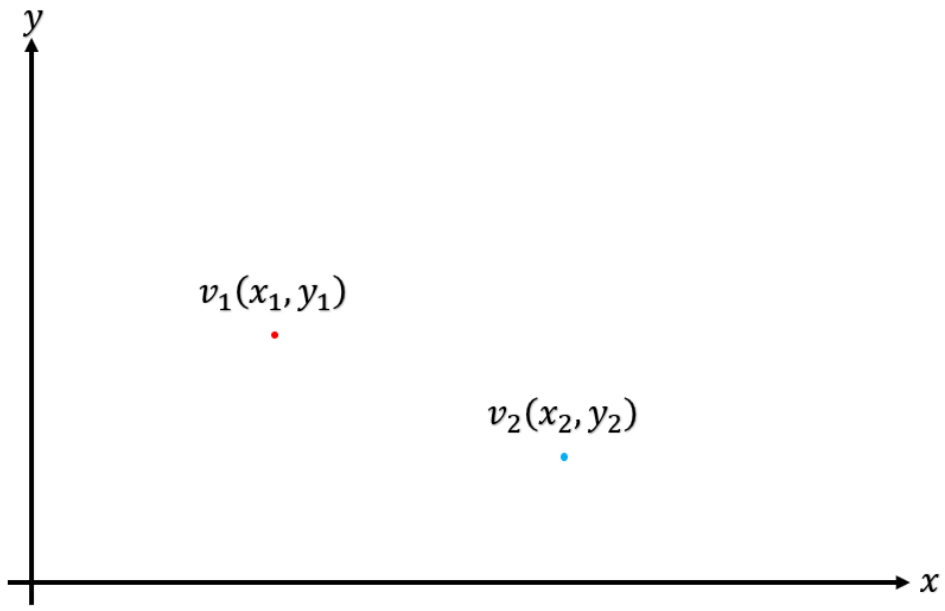
Por cuestiones de formato, se decidió desplegar el módulo `NODE_COORD_SECTION` en 3 columnas, aclarando que en el documento, ésta no es su presentación.

```
NAME: kroD100
TYPE: TSP
COMMENT: 100-city problem D (Krolak/Felts/Nelson)
DIMENSION: 100
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 2995 264                35 547 25                 69 2223 990
2 202 233                 36 3373 1902              70 3868 697
3 981 848                 37 460 267                71 1541 354
4 1346 408                38 3060 781               72 2374 1944
5 781 670                 39 1828 456               73 1962 389
6 1009 1001               40 1021 962               74 3007 1524
7 2927 1777               41 2347 388               75 3220 1945
8 2982 949                42 3535 1112              76 2356 1568
9 555 1121                43 1529 581               77 1604 706
10 464 1302               44 1203 385               78 2028 1736
11 3452 637               45 1787 1902              79 2581 121
12 571 1982               46 2740 1101              80 2221 1578
13 2656 128               47 555 1753               81 2944 632
14 1623 1723              48 47 363                 82 1082 1561
15 2067 694               49 3935 540               83 997 942
16 1725 927               50 3062 329               84 2334 523
17 3600 459               51 387 199                85 1264 1090
18 1109 1196              52 2901 920               86 1699 1294
19 366 339                53 931 512                87 235 1059
20 778 1282               54 1766 692               88 2592 248
21 386 1616               55 401 980                89 3642 699
22 3918 1217              56 149 1629               90 3599 514
23 3332 1049              57 2214 1977              91 1766 678
24 2597 349               58 3805 1619              92 240 619
25 811 1295               59 1179 969               93 1272 246
26 241 1069               60 1017 333               94 3503 301
27 2658 360               61 2834 1512              95 80 1533
28 394 1944               62 634 294                96 1677 1238
29 3786 1862              63 1819 814               97 3766 154
30 264 36                 64 1393 859               98 3946 459
31 2050 1833              65 1768 1578              99 1994 1852
32 3538 125               66 3023 871               100 278 165
33 1646 1817              67 3248 1906
34 2993 624               68 1632 1742
EOF
```

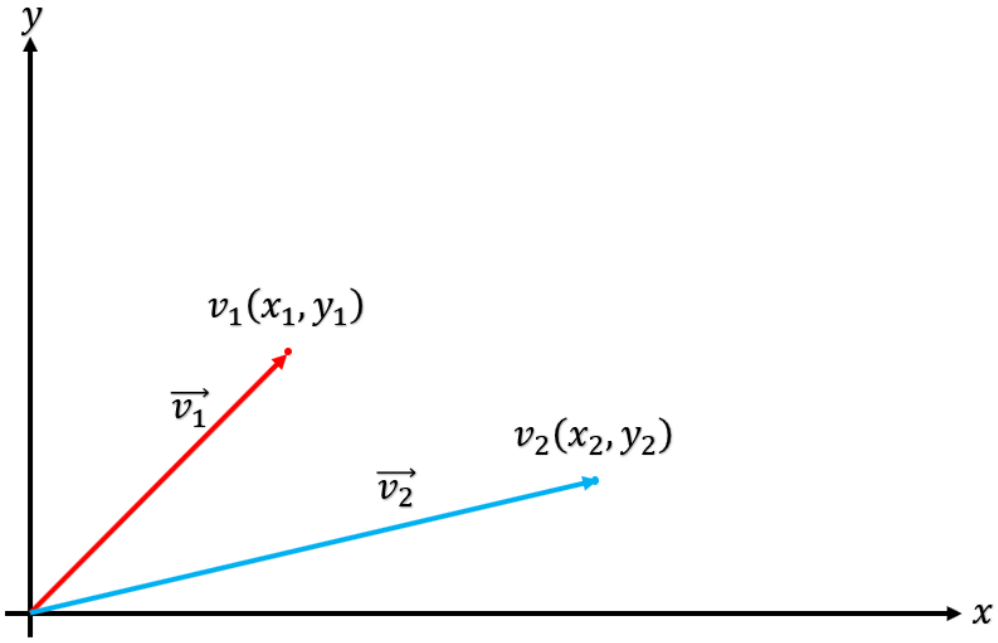
Anexo 3: Deducción de la forma de construir `graph` para la instancia `kroD100.tsp`

Debido a la naturaleza de `list`, planteemos que los puntos se encuentran distribuidos en el plano cartesiano, como ejemplo, tomemos los vértices v_1, v_2 con puntos (x_1, y_1), (x_2, y_2) respectivamente.

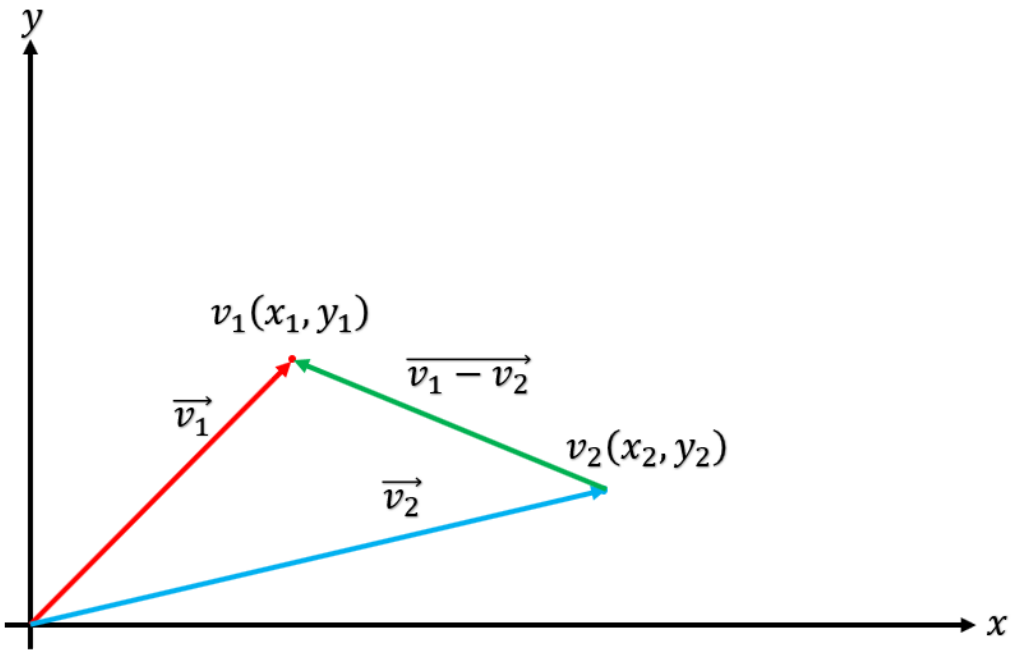
Supongamos la siguiente distribución:



Podemos plantear las coordenadas de éstos como vectores que parten del origen hasta sus coordenadas:



Es entonces gracias a esta vista que podemos plantearnos una resta de los vectores para calcular la distancia entre ambos puntos:



Realmente no es importante si planteamos $\overrightarrow{v_1 - v_2}$ o $\overrightarrow{v_2 - v_1}$, pues el cálculo relevante es el módulo de este vector, el cuál será el mismo sin importar cómo se tome el orden de la diferencia.

Recordemos que la diferencia de vectores se trata de la siguiente manera:

$$\overrightarrow{v_1 - v_2} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \end{bmatrix}$$

Finalmente, calculamos el módulo de un vector en $\mathbb{R}^2$ mediante la siguiente fórmula:

$$|\vec{u}|^2 = u_x^2 + u_y^2$$

Así entonces, tenemos:

$$|\overrightarrow{v_1 - v_2}|^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

$$\therefore |\overrightarrow{v_1 - v_2}| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Generalizando entonces la fórmula para dos vértices v_n, v_m en el conjunto de `list`:

$$|\overrightarrow{v_n - v_m}| = \sqrt{(x_n - x_m)^2 + (y_n - y_m)^2}$$

Anexo 4: Análisis de rendimiento

Según la biblioteca TSPLIB, la instancia `gr21.tsp` tiene un valor mínimo encontrado que genera un costo de 2707, puede observarse que el programa obtiene las siguientes soluciones:

```
  El tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 8, 7, 4, 20] genera un costo de: 3120.0000
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [20, 11, 0, 16, 9, 5, 6, 1, 12, 17, 19, 13, 14, 15, 2, 21, 18, 10, 8, 7, 4, 20] genera un costo de: 3375.0000
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.4200 >= 0.0000)

  El tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 8, 7, 4, 20] genera un costo de: 3120.0000
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [20, 11, 12, 1, 6, 5, 9, 16, 2, 15, 14, 13, 19, 17, 0, 21, 18, 10, 8, 7, 4, 20] genera un costo de: 3735.0000
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.5000 >= 0.0000)

  El tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 8, 7, 4, 20] genera un costo de: 3120.0000
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 4, 7, 8, 20] genera un costo de: 3110.0000
    Hemos escogido newTour (fnewTour < ftour)

  El tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 4, 7, 8, 20] genera un costo de: 3110.0000
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [20, 11, 12, 1, 6, 5, 9, 16, 15, 14, 13, 19, 17, 0, 2, 21, 18, 10, 4, 7, 8, 20] genera un costo de: 3779.0000
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.3300 >= 0.0000)

  El tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 4, 7, 8, 20] genera un costo de: 3110.0000
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 4, 8, 7, 20] genera un costo de: 3114.0000
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.9300 >= 0.4606)

  El tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 21, 18, 10, 4, 7, 8, 20] genera un costo de: 3110.0000
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [20, 11, 12, 1, 6, 5, 9, 16, 0, 17, 19, 13, 14, 15, 2, 8, 7, 4, 10, 18, 21, 20] genera un costo de: 3400.0000
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.8800 >= 0.0000)



¿Desea repetir el código?
0. No
1. Sí
```

Observamos aquí que la última iteración dejó al óptimo con valor de 3110 (ya que también generó un valor de 3400 pero más adelante se menciona que fue descartado).

Otra ejecución produjo el resultado siguiente:

```
  El tour [18, 10, 14, 15, 21, 2, 3, 0, 19, 4, 1, 7, 8, 6, 16, 9, 5, 12, 11, 20, 13, 18] genera un costo de: 3253.0000
  Ocurrió una perturbación en los nodos de inicio y fin del tour
  El nuevo tour [1, 10, 14, 15, 21, 2, 3, 0, 19, 4, 18, 7, 8, 6, 16, 9, 5, 12, 11, 20, 13, 1] genera un costo de: 4540.0000
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.0200 >= 0.0000)



¿Desea repetir el código?
0. No
1. Sí
```

Observamos entonces que encontró un costo de 3253 para esta ejecución.

En la instancia `kroD100.tsp`, la biblioteca TSPLIB registra un viaje óptimo de 21294 y observemos:

```
  El tour [28, 12, 47, 14, 65, 68, 80, 33, 78, 31, 99, 96, 63, 91, 86, 76, 23, 70, 89, 22, 42, 72, 45, 21, 95, 56, 25, 20, 10, 82, 50, 32, 97, 98, 17, 94, 38, 79, 88,
39, 73, 60, 46, 57, 36, 58, 29, 74, 61, 7, 75, 67, 84, 15, 53, 4, 71, 77, 100, 51, 24, 40, 18, 9, 83, 3, 5, 35, 48, 2, 30, 62, 37, 6, 54, 8, 52, 81, 27, 1, 49, 13, 90,
 11, 34, 66, 59, 85, 44, 93, 43, 41, 16, 64, 69, 19, 92, 55, 26, 87, 28] genera un costo de: 60632.8047
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [28, 12, 47, 14, 65, 68, 80, 33, 78, 31, 99, 96, 63, 91, 86, 76, 23, 70, 89, 22, 42, 72, 45, 21, 95, 56, 25, 20, 10, 82, 50, 32, 97, 98, 17, 94, 38, 79
, 88, 39, 73, 60, 46, 57, 36, 58, 29, 74, 61, 7, 75, 67, 84, 15, 53, 4, 71, 77, 100, 51, 24, 40, 18, 9, 83, 3, 5, 35, 48, 2, 30, 62, 37, 6, 54, 8, 52, 81, 27, 1, 49, 1
3, 90, 11, 85, 59, 66, 34, 44, 93, 43, 41, 16, 64, 69, 19, 92, 55, 26, 87, 28] genera un costo de: 63506.2773
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.9600 >= 0.0000)

  El tour [28, 12, 47, 14, 65, 68, 80, 33, 78, 31, 99, 96, 63, 91, 86, 76, 23, 70, 89, 22, 42, 72, 45, 21, 95, 56, 25, 20, 10, 82, 50, 32, 97, 98, 17, 94, 38, 79, 88,
39, 73, 60, 46, 57, 36, 58, 29, 74, 61, 7, 75, 67, 84, 15, 53, 4, 71, 77, 100, 51, 24, 40, 18, 9, 83, 3, 5, 35, 48, 2, 30, 62, 37, 6, 54, 8, 52, 81, 27, 1, 49, 13, 90,
 11, 34, 66, 59, 85, 44, 93, 43, 41, 16, 64, 69, 19, 92, 55, 26, 87, 28] genera un costo de: 60632.8047
  Ocurrió una perturbación en los nodos de inicio y fin del tour
  El nuevo tour [28, 12, 47, 14, 65, 68, 80, 33, 78, 31, 99, 96, 63, 91, 86, 76, 23, 70, 89, 22, 42, 72, 45, 21, 95, 56, 25, 20, 10, 82, 50, 32, 97, 98, 17, 94, 38, 79
, 88, 39, 73, 60, 46, 57, 36, 58, 29, 74, 61, 7, 75, 67, 84, 15, 53, 4, 71, 77, 100, 51, 24, 40, 18, 9, 83, 3, 5, 35, 48, 2, 30, 62, 37, 6, 54, 8, 52, 81, 27, 1, 49, 1
3, 90, 11, 34, 66, 59, 85, 44, 93, 43, 41, 16, 64, 69, 19, 92, 55, 26, 87, 28] genera un costo de: 60632.8047
    No hemos escogido aún (fnewTour >= ftour)
      Hemos escogido newTour por probabilidad de Boltzmann (r3 = 0.7300 < 1.0000)



¿Desea repetir el código?
```

Tuvimos un costo de 60632.8047 para esta ejecución.

```
  El tour [15, 13, 89, 61, 84, 88, 79, 41, 27, 73, 46, 98, 90, 1, 81, 52, 8, 67, 29, 66, 24, 32, 60, 100, 30, 3, 54, 77, 62, 37, 43, 39, 91, 64, 58, 23, 50, 11, 42, 70
, 97, 94, 49, 75, 31, 78, 33, 57, 25, 47, 26, 2, 51, 53, 63, 65, 80, 69, 76, 28, 21, 12, 20, 44, 40, 92, 95, 87, 48, 19, 35, 96, 38, 17, 86, 59, 9, 18, 72, 74, 36, 7,
22, 34, 71, 93, 5, 55, 56, 10, 82, 68, 14, 45, 99, 4, 83, 6, 85, 16, 15] genera un costo de: 73083.7500
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [15, 13, 89, 61, 84, 88, 79, 41, 27, 73, 46, 98, 90, 1, 81, 52, 8, 67, 29, 66, 24, 32, 60, 100, 30, 3, 54, 77, 62, 37, 43, 39, 91, 64, 58, 23, 50, 11,
42, 70, 97, 94, 49, 75, 31, 78, 33, 57, 25, 47, 26, 2, 51, 53, 63, 65, 80, 69, 76, 28, 21, 12, 87, 95, 92, 40, 44, 20, 48, 19, 35, 96, 38, 17, 86, 59, 9, 18, 72, 74, 3
6, 7, 22, 34, 71, 93, 5, 55, 56, 10, 82, 68, 14, 45, 99, 4, 83, 6, 85, 16, 15] genera un costo de: 73789.3594
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.2200 >= 0.0000)

  El tour [15, 13, 89, 61, 84, 88, 79, 41, 27, 73, 46, 98, 90, 1, 81, 52, 8, 67, 29, 66, 24, 32, 60, 100, 30, 3, 54, 77, 62, 37, 43, 39, 91, 64, 58, 23, 50, 11, 42, 70
, 97, 94, 49, 75, 31, 78, 33, 57, 25, 47, 26, 2, 51, 53, 63, 65, 80, 69, 76, 28, 21, 12, 20, 44, 40, 92, 95, 87, 48, 19, 35, 96, 38, 17, 86, 59, 9, 18, 72, 74, 36, 7,
22, 34, 71, 93, 5, 55, 56, 10, 82, 68, 14, 45, 99, 4, 83, 6, 85, 16, 15] genera un costo de: 73083.7500
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [15, 13, 89, 61, 84, 88, 79, 41, 27, 73, 46, 98, 90, 1, 81, 52, 8, 67, 29, 66, 24, 32, 60, 100, 30, 75, 49, 94, 97, 70, 42, 11, 50, 23, 58, 64, 91, 39,
 43, 37, 62, 77, 54, 3, 31, 78, 33, 57, 25, 47, 26, 2, 51, 53, 63, 65, 80, 69, 76, 28, 21, 12, 20, 44, 40, 92, 95, 87, 48, 19, 35, 96, 38, 17, 86, 59, 9, 18, 72, 74, 3
6, 7, 22, 34, 71, 93, 5, 55, 56, 10, 82, 68, 14, 45, 99, 4, 83, 6, 85, 16, 15] genera un costo de: 75797.5938
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.7400 >= 0.0000)

  El tour [15, 13, 89, 61, 84, 88, 79, 41, 27, 73, 46, 98, 90, 1, 81, 52, 8, 67, 29, 66, 24, 32, 60, 100, 30, 3, 54, 77, 62, 37, 43, 39, 91, 64, 58, 23, 50, 11, 42, 70
, 97, 94, 49, 75, 31, 78, 33, 57, 25, 47, 26, 2, 51, 53, 63, 65, 80, 69, 76, 28, 21, 12, 20, 44, 40, 92, 95, 87, 48, 19, 35, 96, 38, 17, 86, 59, 9, 18, 72, 74, 36, 7,
22, 34, 71, 93, 5, 55, 56, 10, 82, 68, 14, 45, 99, 4, 83, 6, 85, 16, 15] genera un costo de: 73083.7500
  Ocurrió una perturbación del tipo subviaje inverso
  El nuevo tour [15, 13, 89, 61, 84, 88, 79, 41, 27, 73, 46, 98, 90, 1, 81, 52, 8, 67, 29, 66, 24, 32, 60, 100, 30, 3, 54, 77, 62, 37, 43, 39, 91, 64, 58, 23, 50, 11,
42, 70, 97, 94, 49, 75, 31, 78, 33, 57, 25, 47, 26, 2, 51, 53, 63, 65, 80, 69, 76, 28, 21, 12, 20, 44, 40, 92, 95, 87, 48, 19, 35, 96, 38, 17, 86, 59, 9, 18, 72, 74, 3
6, 7, 22, 34, 71, 93, 5, 55, 56, 10, 82, 16, 85, 6, 83, 4, 99, 45, 14, 68, 15] genera un costo de: 74128.6016
    No hemos escogido aún (fnewTour >= ftour)
      Hemos descartado newTour por probabilidad de Boltzmann (r3 = 0.8800 >= 0.0000)
```

Ahora el costo generado en esta ejecución fue de 74128.6016.

Con parámetros ajustados cerca de los siguientes establecidos:

- $T_0 = 300.$
- $T_{fin} = 5.$
- $alpha = 0.85.$
- $rho = 0.95.$
- $K = 200.$
- $autom = 1.$
- $disturbance = 1.$
- $p = 0.2.$