CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OPTIMIZACIÓN INTELIGENTE

5° "A"

PROYECTO PARTE 2: COLONIA DE HORMIGAS

Profesor: Aurora Torres Soto

Alumnos:

Espinoza Sánchez Joel Alejandro

Gómez Garza Dariana

González Arenas Fernando Francisco

Fecha de Entrega: Aguascalientes, Ags.**, 14** de diciembre de 2020

índice

Introducción

1. Introducción a la colonia de hormigas

La teoría de optimización por colonia de hormigas o och (Ant Colony Optimization, aco), fue introducida por Marco Dorigo en los inicios de 1990 como herramienta para la solución de problemas de optimización complejos (Dorigo y Gambardella, 1996, p. 3; Dorigo y Blum, 2005, p. 244).

La och pertenece a la clase de métodos heurísticos, los cuales son algoritmos aproximados utilizados para obtener soluciones lo suficientemente buenas a problemas complejos en una cantidad razonable de tiempo de cómputo.

La fuente de inspiración de la och es el comportamiento real de las hormigas. Estos insectos cuando están en búsqueda de la comida inicialmente exploran el área alrededor de su nido de una forma aleatoria. Tan pronto encuentran fuentes de alimentos, evalúan su cantidad y calidad, y llevan alguna parte de esta comida para su nido. Durante el regreso al nido, las hormigas depositan una sustancia química llamada feromona sobre el camino, la cual servirá de guía futura para que las demás encuentren los alimentos.

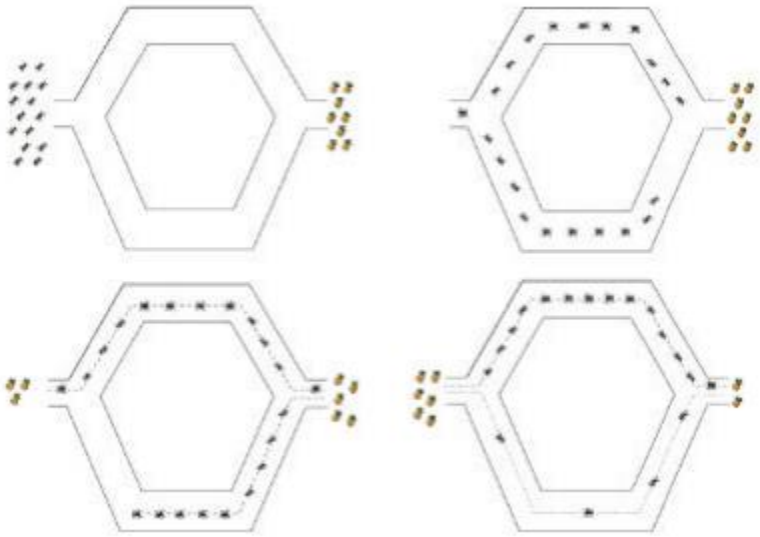


Figura 1: El comportamiento de la colonia termina por obtener el camino más corto entre dos puntos. Recuperada de: La metaheurística de optimización basada en colonias de hormigas: modelos y nuevos enfoques (Alonso et ál., 2004, p. 6).

La cantidad de dicha sustancia depositada dependerá de la cantidad y calidad de los alimentos. Diferentes estudios han demostrado que la comunicación de las hormigas a través de caminos con feromonas les permite encontrar las rutas más cortas entre su nido y las fuentes de alimentos (Alonso et ál., 2004, p. 4). Esta característica es ampliamente utilizada para la solución de problemas de

optimización que necesitan mejorar sustancialmente los tiempos de cómputo para la solución de una aplicación específica.

2. Teoría de grafos

Los grafos constituyen una herramienta básica para modelar fenómenos discretos, y son fundamentales para la comprensión de las estructuras de datos y el análisis de algoritmos.

Un grafo en matemáticas e informática es una generalización del concepto simple de un conjunto de puntos, llamados vértices. Con el desarrollo de las ciencias de la computación, han avanzado considerablemente los campos de investigación relacionados con la misma, como es el caso de la Inteligencia Artificial.

Una de las técnicas que esta estudia son los algoritmos basados en el comportamiento de las hormigas, que son métodos empleados en la solución de problemas complejos de búsqueda y optimización. Para estas soluciones se tiene en cuenta elementos de la teoría de grafos, fundamentalmente los grafos ponderados vinculados a la toma de decisiones.

Tomar decisiones es la actividad que en el ser humano manifiesta la capacidad de elegir diferentes opciones y llevar a cabo una acción como resultado del conocimiento que posee y de un proceso intelectual que involucra la reflexión y la proyección en el futuro de las consecuencias de la opción elegida. (Ramírez y Zacarías, 2007, p. 36).

El objetivo del trabajo es mostrar la vinculación del algoritmo optimización con colonia de hormigas aplicando la teoría 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).

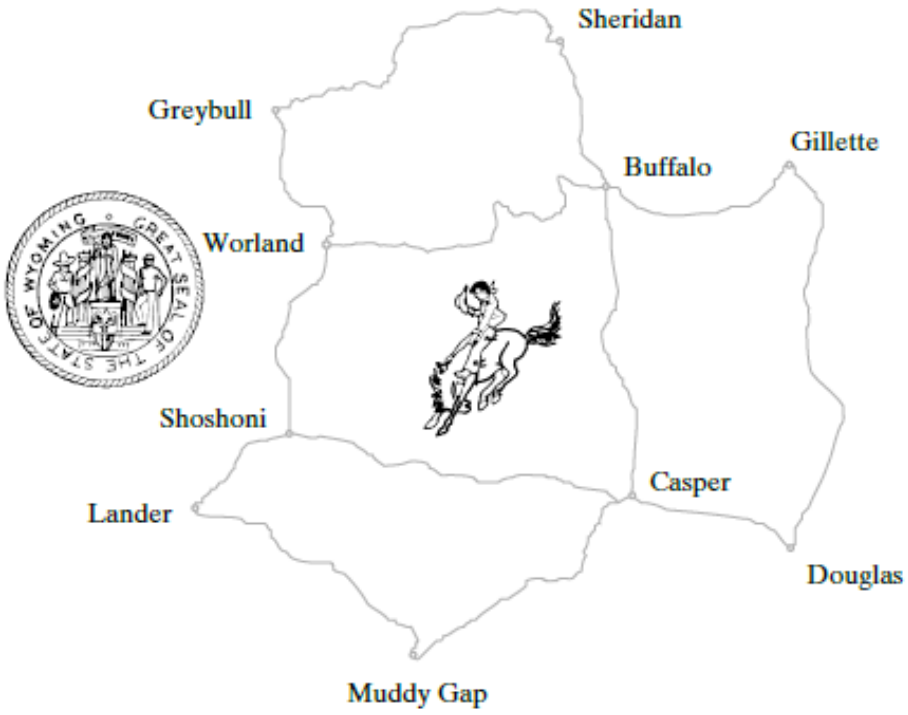


Figura 2: Parte del sistema de carreteras de Wyoming.
Recuperada de: *Matemáticas Discretas* (Johnsonbaugh, 2005, p. 319).

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 2 y el cual es posible modelar como un grafo a gusto propio, Johnsonbaugh propone dos modelos de grafos encontrados en la figura 3.

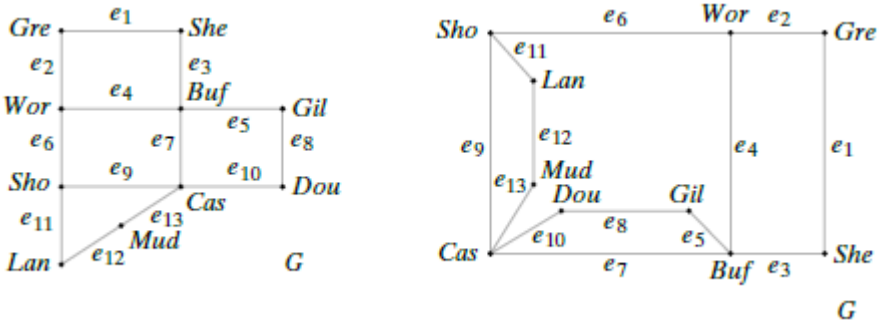


Figura 3: Modelos de grafos para el sistema de carreteras de la figura 2.
Recuperada de: *Matemáticas Discretas* (Johnsonbaugh, 2005, p. 319).

Observemos que sin importar la construcción del grafo, los propuestos en la figura 3 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 colonia de hormigas y grafos

Las hormigas poseen una característica muy peculiar que las diferencian de otros animales, son capaces de encontrar la vía más corta desde el hormiguero a una fuente de comida y viceversa, sin usar pistas visuales, también pueden adaptase a cambios en el ambiente. Esto es posible por el rastreo de la feromona, sustancia que ellas depositan mientras caminan. (Cobo y Serrano, 2010).

Los algoritmos de OCH utilizan agentes computacionales simples, en este caso hormigas artificiales, que trabajan de manera cooperativa y tienen comunicación mediante rastros de feromona artificial, simulando el comportamiento de una colonia de hormigas naturales. En este algoritmo se utiliza como ya se mencionaba anteriormente comunicación indirecta a través de la feromona, donde los caminos más cortos tienen una razón más elevada de crecimiento del valor de la feromona y las hormigas disponen de una preferencia probabilística por las rutas con valores altos de feromona (Bello, 2008).

También se deben de tener en cuenta otras características que hacen que las hormigas artificiales posen capacidades que no tienen las reales, pero que contribuyen a la resolución del problema, por ejemplo; cada hormiga es capaz de determinar qué tan lejos está de un estado, poseen información acerca de su ambiente y la utilizan al tomar decisiones y tienen memoria, la cual es necesaria para asegurar que se generen sólo soluciones factibles (Mendoza, 2001).

Para aplicar el algoritmo OCH es necesario establecer una secuencia de pasos, los cuales se indican a continuación.

a) Representar el problema mediante nodos.
b) Definir el significado de los rastros de feromona de una manera adecuada.
c) Poner pesos a la información heurística en cada nodo o arco.
d) Desarrollar algoritmos que permitan realizar optimizaciones locales.
e) Escoger un algoritmo OCH específico.
f) Refinar los parámetros del algoritmo de OCH.

3.3 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) ya comentados en el archivo "*Proyecto Parte 1*" vuelven a definirse a conitnuación.

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

Tomando cualquiera de las dos instancias anteriormente mencionadas analizaremos un problema de manera general y resolveremos de manera arbitraria problemas heurísticos aplicados a grafos mediante la colonia de hormigas.

Primeramente, necesitaremos obtener una matriz de distancias denominada `graph`, ésta guardará el valor que tiene cada arista al ir de un nodo `i` a otro nodo `j`. Una vez teniendo la matriz `graph` seguiremos obteniendo las siguientes matrices.

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

- `tau`: Valor de la feromona inicial.
- `Q`: Factor de influencia de la distancia.
- α: Valor de la influencia de las feromonas.
- β: Valor de la influencia de la información heurística.
- ρ: Factor de evaporación de 0 a 1.
- `qh`: Cantidad de hormigas.
- `qv`: Cantidad de vértices que tendrá el grafo.
- `r`: Es un número aleatorio de 0 a 1.
- `L`: Longitud del recorrido completo realizado por una hormiga.
- `d`: Distancias entre nodos

Entonces, sea `qv` la cantidad de vértices que el grafo tiene y `graph` la matriz de distancias de dimensión $qv \times qv$ la cual contiene los pesos de viajar a través de los nodos `i,j`.

$$graph = \begin{bmatrix} 0 & d_{12} & d_{13} & \cdots & d_{1n} \\ d_{21} & 0 & d_{23} & \cdots & d_{2n} \\ d_{31} & d_{32} & 0 & \cdots & d_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d_{n1} & d_{n2} & d_{n3} & \cdots & 0 \end{bmatrix}$$

El primer paso para modelar formalmente la solución será obtener la matriz de feromonas, en este caso la matriz `pheromone` de dimensión $qv \times qv$.

Ésta matriz se obtiene a partir del valor inicial `tau` que se le otorga a la feromona, de esta manera tendríamos el valor de `tau` en la matriz donde si hay adyacencia, de lo contrario se considerará 0, tendríamos lo siguiente:

$$pheromone = \begin{bmatrix} 0 & tau_{12} & tau_{13} & \cdots & tau_{1n} \\ tau_{21} & 0 & tau_{23} & \cdots & tau_{2n} \\ tau_{31} & tau_{32} & 0 & \cdots & tau_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ tau_{n1} & tau_{n2} & tau_{n3} & \cdots & 0 \end{bmatrix}$$

Una vez hecha la matriz `pheromone` necesitamos construir otra matriz, a ésta matriz la llamaremos `vision`. `vision` es la matriz de visibilidad que consiste en dividir $\frac{1}{d}$ que tiene cada nodo `i,j`. Se vería de la siguiente manera:

$$vision = \begin{bmatrix} 0 & \frac{1}{d_{12}} & \frac{1}{d_{13}} & \cdots & \frac{1}{d_{1n}} \\ \frac{1}{d_{21}} & 0 & \frac{1}{d_{23}} & \cdots & \frac{1}{d_{2n}} \\ \frac{1}{d_{31}} & \frac{1}{d_{32}} & 0 & \cdots & \frac{1}{d_{3n}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{d_{n1}} & \frac{1}{d_{n2}} & \frac{1}{d_{n3}} & \cdots & 0 \end{bmatrix}$$

Después de haber obtenido las matrices anteriores proseguimos con la creación de la matriz `tabu[qh][qv]` de dimensión $qh \times qv$, siendo `qh` la cantidad de hormigas que evalueremos, la matriz `tabu` debe tener como posicisión `[1][n]` el nodo donde se iniciará el recorrido. Una vez comenzando el algoritmo para complementar la lista `tabu` no se debe repetir ningún nodo recorrido por la misma hormiga.

Para cada hormiga se debe evaluar a qué nodos puede ir; una vez teniendo los nodos a los que puede viajar calcularemos el peso de cada nodo con la siguiente fórmula:

$$p_{ij}^{k}(t) = \frac{\left[tau_{ij}(t)\right]^{\alpha} \left[\eta_{ij}\right]^{\beta}}{\sum v\, j\ candidatos \left[tau_{ij}(t)\right]^{\alpha} \left[\eta_{ij}\right]^{\beta}}$$

Siendo $\eta = el\ valor\ de\ la\ visibilidad\ en\ i,j$

Al tener el peso de todos los nodos a los que puede ir la hormiga debemos obtener un número aleatorio `r` entre 0 y 1; si $r \leq p_{ij}^{k}$ optará por irse al nodo correspondiente a esa probabilidad.

Ya que tengamos a qué nodo siguiente irá la hormiga, se actualizará la matriz `tabu` y se seguirá el mismo proceso hasta terminar el recorrido de `n` hormigas y la matriz esté completamente llena.

Al terminar este proceso y tengamos todas las soluciones se actualizará los rastros de feromonas. Tomaremos la matriz `pheromone` y se colocará una nueva, la cantidad de feromona que se deposite en cada arco es proporcional a la distancia del recorrido completo encontrado por cada hormiga.

$$\Delta T_{ij}^{k} = \frac{Q}{L_k}$$

Finalmente se realiza la actualización de la matriz `pheromone` de la siguiente manera:

$$(1-\rho)tau_{ij} + \sum \Delta T_{ij}$$

Descripción de la herramienta

La herramienta se nombró como "*ProyectoP2 JoelDarianaFer*" como primer anexo externo al documento, también pueden presentarse como anexos las dos instancias que son *"gr21.tsp.txt"* y *"kroD100.tsp.txt"*.

Una vez se ejecute el archivo, se abrirá la pantalla de menú principal, la cual puede apreciarse en la figura 4.

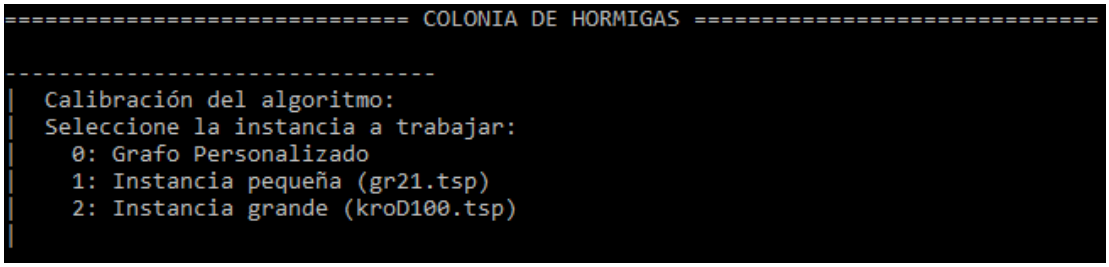


Figura 4: Menú inicial del programa *"ProyectoP2 JoelDarianaFer"*.

Nuevamente, así como en "*ProyectoP1 JoelDarianaFer*", 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 4 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`. Si no se ingresa alguna opción no válida, el programa no funcionará; el programa no tiene validado qué hacer si no recibe una opción correcta.

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 5:

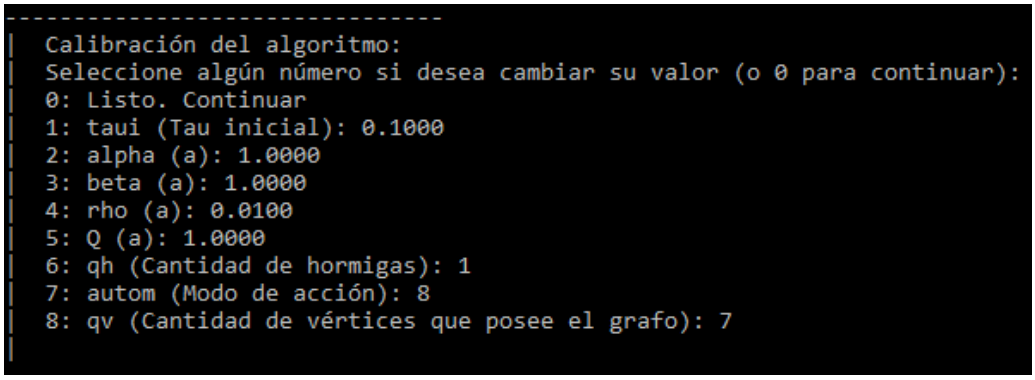


Figura 5: Menú de ajuste de parámetros primarios para grafos personalizados del programa *"ProyectoP2 JoelDarianaFer"*.

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 8 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.

Recordemos los parámetros:

- **`taui`: Valor para tau (τ) inicial.**
  Simboliza el valor de las feromonas al comienzo del algoritmo.
  Su valor puede ser cualquier número real positivo, pero se busca que sea un número muy cercano a 0.
- **`alpha`: Valor de influencia de las feromonas.**
  Será un valor que le proporcione mayor o menor dominio a las feromonas cuando la matriz de visibilidad se actualiza.
  Su valor puede ser cualquier número real positivo.
- **`beta`: Valor de influencia de la visibilidad.**
  Será un valor que le proporcione mayor o menor dominio a la visibilidad cuando la matriz de visibilidad se actualiza.
  Su valor puede ser cualquier número real positivo.
- **`rho`: Factor de evaporación de las feromonas.**
  Determinará qué tanto se eliminarán las feromonas después de una iteración.
  Su valor puede ser cualquier número real dentro del intervalo . $(0,1)$
- **`Q`: Valor de influencia del camino recorrido por cada hormiga.**
  Será un valor que regulará qué tanto afectará la distancia del camino recorrido por la hormiga.
  Su valor puede ser cualquier número real positivo.
- **`qh`: Cantidad de hormigas.**
  Serán las hormigas que se destinen a evaluar dentro del algoritmo.
  Su valor puede ser cualquier número entero positivo.
- **`autom`: Modo de acción.**
  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.
- **`qv`: Cantidad de vértices del grafo.**
  Determinará de forma personalizada cuántos vértices tendrá el grafo.
  Su valor puede ser cualquier número entero positivo.

Se hace mención nuevamente 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 no está dentro de las definiciones anteriores.

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 6.

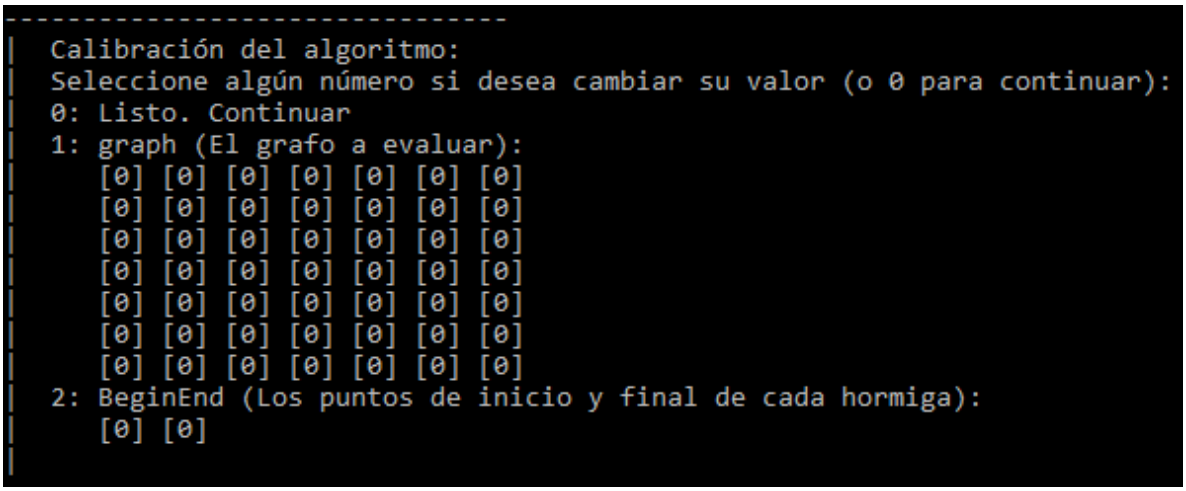


Figura 6: Menú de ajuste de estructuras del programa *"ProyectoP2 JoelDarianaFer"*.

El programa se modificó para que el punto de inicio sea el mismo que el punto final y forzosamente se recorriera todo el grafo, así como se validó una opción para evitar que una hormiga se atore en un grafo no completo.

Una vez que terminan de calibrarse los arreglos de este menú, se procederá a comenzar la colonia de hormigas, pero previo a ello, se explicarán brevemente las diferencias al elegir cualquier de las instancias de los archivos.

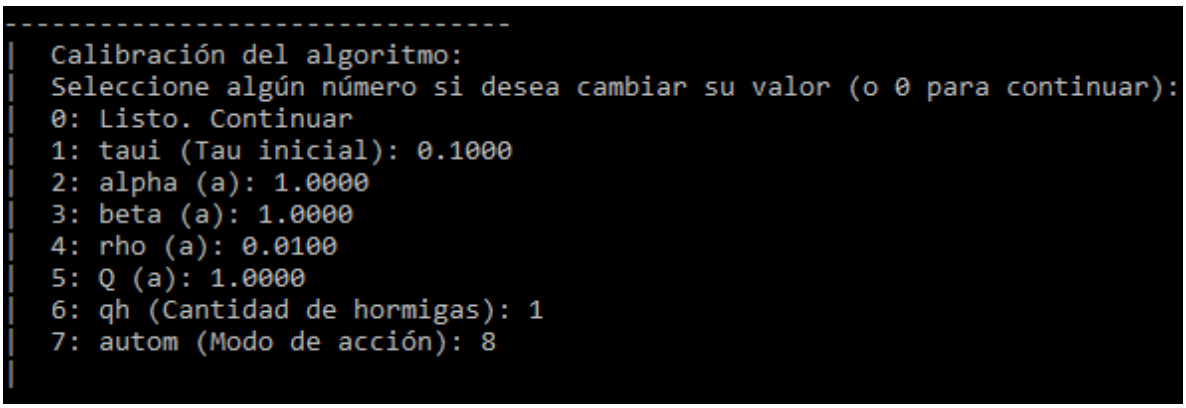


Figura 7: Menú de ajuste de parámetros primarios para cualquiera de las dos instancias almacenadas en archivos del programa *"ProyectoP2 JoelDarianaFer"*.

El único cambio en el menú de parámetros, como puede verse en la figura 7, 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.

Para utilizar el programa es importante tener en cuenta algunas cosas importantes para que su ejecución sea correcta. Hay que realizar una aclaración con respecto a los archivos, pues como primer requerimiento, se pide que la herramienta 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 en las funciones las funciones `setInstance21` y `setInstance100`, también encontradas en la figura 8 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` exactamente en las líneas:

```
filetxt = fopen("kroD100.tsp.txt","r");

filetxt = fopen("gr21.tsp.txt","r");
```

Así se enfatiza que únicamente debería retirarse la extensión `.txt` y sólo dejar el nombre hasta `.tsp`. Por el contrario, si los documentos tienen el sufijo `.txt`, no debe realizarse cambio alguno.

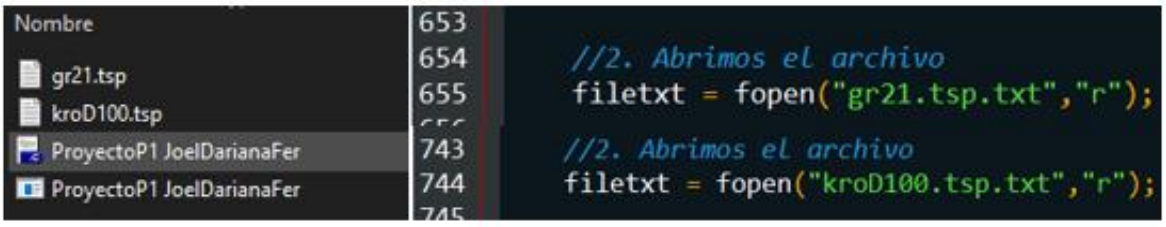


Figura 8: Consideraciones importantes en relación a los archivos para el funcionamiento correcto de la herramienta *"ProyectoP1 JoelDarianaFer"*.

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:

$taui = 0.1$	$rho = 0.2$
$alpha = 1$	$Q = 1$
$beta = 1$	$qh = 3$
$autom = 3$	

Conclusiones

Joel Alejandro Espinoza Sánchez: La colonia de hormigas es un mecanismo heurístico muy útil cuando la exploración simultánea puede ser un gran factor. La implementación de este algoritmo nos permitió profundizar en la colonia de dos o más hormigas, así como la modificación de la trayectoria del grafo, el cual debía recorrer completamente el grafo. Estas consideraciones nos permitieron ampliar las aplicaciones de la colonia de hormigas.

Asimismo, pudimos observar cómo se comportaba el algoritmo mediante la prueba de distintas configuraciones de variables, pues con menor factor de evaporación, las feromonas se mantenían más, sin embargo, de ser un factor notorio, las feromonas tendían a cero rápidamente. Así también los factores de influencia de las feromonas y la visualización jugaban un papel tan importante que al final se decidió mantener en una igualdad en uno, pues de esta manera afecarían de forma equivalente al procedimiento.

Dariana Gómez Garza:

A lo largo de este proyecto pudimos reforzar los conocimientos previos, ya que se tuvo que hacer un poco más de investigación para definir algunos aspectos del algoritmo. También nos dimos cuenta de cómo va creciendo la dificultad conforme el número de hormigas va aumentando en una colonia.

A mi parecer es un algoritmo muy sencillo de entender, lo único es que si es algo más tardado de hacer (manualmente) dependiendo la cantidad de hormigas que estarán en la colonia, ya que se debe hacer un recorrido diferente por cada hormiga y por lo tanto se debe realizar el mismo proceso una por una. A pesar de eso queda muy claro como se inspiraron al crear este algoritmo, ya que nos deja como ejemplo como es que las hormigas naturales, de la vida real, trabajan para llegar por ejemplo, a su comida que tienen almacenada.

Fernando Francisco González Arenas:

La colonia de hormigas es un algoritmo de optimización muy útil, el cual emula la organización de las hormigas biológicas para encontrar comida utilizando la ruta más corta. Esto lo hacen por medio de las feromonas que van dejando en el camino y que las otras hormigas de la colonia van siguiendo para encontrar el camino mas corto.

El algoritmo es muy interesante, ya que simula con formulas y variables la forma de actuar de las hormigas, como van dejando feromonas en el camino que les parece el mejor y así hacen que las otras hormigas (que en este caso son iteraciones del algoritmo), las sigan y se encuentre el objetivo deseado. Yo opino que este algoritmo nos podra ser útil en el futuro para problemas de optimización, ya sea en nuestra vida académica o laboral.

Bibliografía

- Alonso, S. et ál. (2004), La metaheurística de optimización basada en colonias de hormigas: modelos y nuevos enfoques, Granada, Departamento de Ciencias de la Computación e Inteligencia Artificial, E.T.S. Universidad de Granada.

- Ponce, J., Padilla, F. y Ochoa, C. (2006), Algoritmo de optimización con colonia de hormigas para el problema de la mochila, México, Universidad Autónoma de Aguascalientes.

- Dorigo, M. y Blum, C. (2005), “Ant Colony Optimization Theory: A Survey”,en Lecture Notes in Computer Science, vol. 344, pp. 243-278.

- Ramírez, J. A., Zacarías, H. (2007). Gestión del conocimiento e Innovación en la toma de decisiones en el abastecimiento de librerías. Tesis de Doctorado no publicada. Centro de Estudios Avanzados del IPN.México.

- Bello Pérez, R. (2 de julio de 2008). Teoría de conjuntos aproximados y colonia de hormigas en el contexto de la inteligencia artificial. (R. Millet Luaces, Entrevistador).

- Reinelt, G. (1994), The traveling salesman: computational solutions for TSP applications, Berlin, Springer-Verlag.

- Torres, A (2020) Optimización mediante colonia de hormigas. México: Universidad Autónoma de Aguascalientes.

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`.

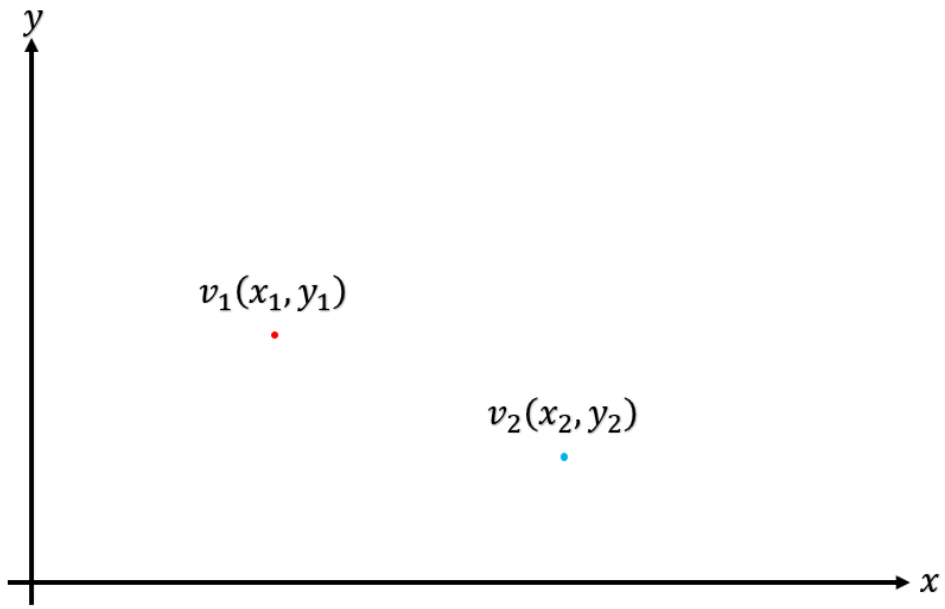
Por cuestiones de formato, se decidió desplegar el módulo `NODE_COORD_SECTION` en 3 columnas, aclarando que en el documento, ésta no es su presentación.

```
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
```

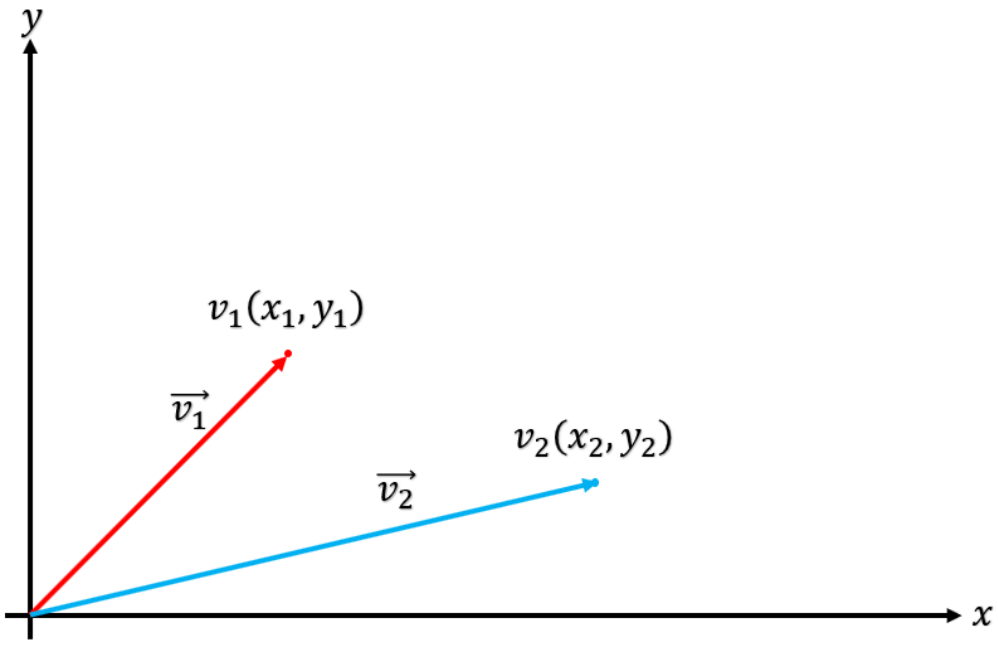
Anexo 3: Deducción de la forma de construir `graph` para la instancia `kroD100.tsp`

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.

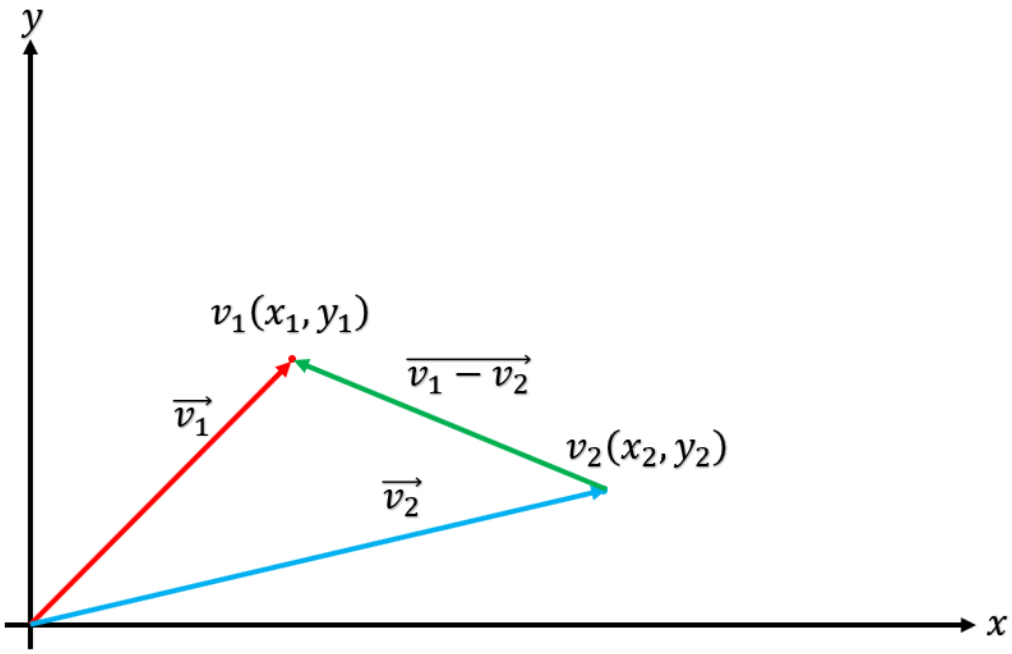
Supongamos la siguiente distribución:



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`:

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