Problema de Enrutamiento de Vehículos (VRP)

Alan Marcel Arenas Venegas, José Armando Equihua Antonio

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS, (CUCEI, UDG)

alan.arenas0363@alumnos.udg.mx

jose.equihua4167@alumnos.udg.mx

Este proyecto se abordará la solución a un problema de enrutamiento de vehículos y como fue implementado en un algoritmo para ello, explicando los pasos que tuvimos que hacer para poder llegar a un resultado optimo en el algoritmo. El enrutamiento de vehículos o VRP es un problema típicamente planteado para la optimización de las rutas de transporte y vehículos asociados a ellas.

Proyecto de Algoritmos, problema de enrutamiento de vehículos, Código de algoritmo.

I. Introducción

El problema de enrutamiento de vehículos (VRP, por sus siglas en inglés) es un problema de optimización combinatoria y de programación de entero qué pregunta "¿Cuál es el conjunto óptimo de rutas para una flota de vehículos que debe satisfacer las demandas de un conjunto dado de clientes?". Es una generalización del conocido Problema del Viajante (TSP, por sus siglas en inglés).

El modelo clásico del problema de ruteo vehicular describe el diseño de rutas donde a partir de un depósito del que sale cada vehículo y al que tiene que regresar, luego de visitar una sola vez a los clientes para satisfacer su demanda conocida, sin violar las restricciones de capacidad de carga de los vehículos, distancia máxima recorrida por éstos, y respetando el horario de trabajo: todo ello con el fin de buscar el costo mínimo.

II. Frontend

Empezando con la realización del proyecto comenzado con la realización del Frontend del mismo, que en este caso se optó por utilizar una herramienta ya usada por nosotros, pero en una versión mejorada que fue el CustomTkinter, el cual este no da un control similar al tkinter original, pero con la posibilidad de darnos un diseño más bonito en las ventanas emergentes que se van realizando

A. CustomTkinter

Como se mencionó con anterioridad, el CustomTkinter nos ayudo a darle un aspecto más limpio y bonito a nuestra interfaz principal, lo cual no es algo relevante para el funcionamiento del algoritmo principal pero es algo que se debe de tomar en cuenta en un aspecto más vistoso e interactivo, y esta librería nos proporcionaba un diseño mas limpio y vistoso al usuario.

Como se mencionó, lo primero que se empezó haciendo en el proyecto fue esta parte, primero que nada, investigando el cómo manipular esta librería para poder logra el diseño que nosotros queríamos, después, instalar en VisualS lo necesario para poder utilizar esta librería y poder importarlas de manera correcta.

```
import customtkinter as ctk
import tkinterDnD
```

Complementando a la librería ya comentada, también se instalo y utilizo otra librería de Tkinter que es la DnD la cual esta lo que nos proporciona es un soporte para arrastrar y soltar objetos dentro de una sola aplicación, dentro de la misma ventana o entre ventanas.

Después de eso ya se empezó a desarrollar en código lo que vendría siento ya el diseño del menú de nuestro proyecto, que como se comentó, la sintaxis que se usa ara esta versión customisada del tkinter es muy similar a la que se utiliza con el normal.

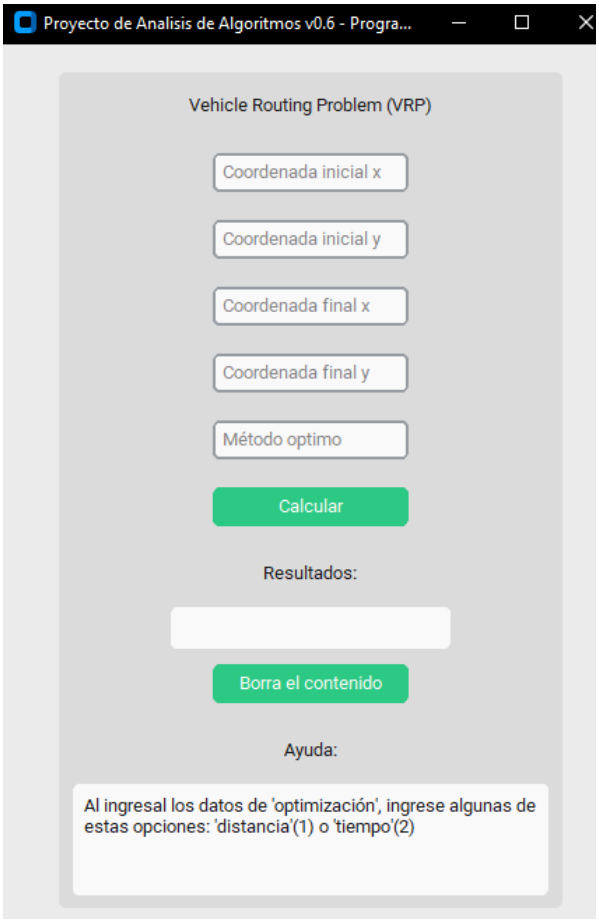
Lo que mas se destaca de esta parte es, aparte del desarrollo do los colores de lo que vendría siendo todo el diseño, son la declaración de las celdas donde el usuario pondrá las coordenadas que se le pedirán para hacer el calculo que el algoritmo ara.

```
entry1 = ctk.CTkEntry(master=mainFrame, placeholder_text="Coordenada inicial x")
entry1.pack(pady=10, padx=10)

entry2 = ctk.CTkEntry(master=mainFrame, placeholder_text="Coordenada inicial y")
entry2.pack(pady=10, padx=10)

entry3 = ctk.CTkEntry(master=mainFrame, placeholder_text="Coordenada final x")
entry3.pack(pady=10, padx=10)

entry4 = ctk.CTkEntry(master=mainFrame, placeholder_text="Coordenada final y")
entry4.pack(pady=10, padx=10)

entry5 = ctk.CTkEntry(master=mainFrame, placeholder_text="Método optimo")
entry5.pack(pady=10, padx=10)

button_1 = ctk.CTkButton(text="Calcular", master=mainFrame, command=buttonCallback)
button_1.pack(pady=10, padx=10)
```

También, otro parte a destacar de este apartado del Frontend es el apartado de la declaración de los botones y los cuadros de texto que se utilizaran para nuestro algoritmo, que en este caso son muy sencillos.

```
label2 = ctk.CTkLabel(text="Resultados:", master=mainFrame, justify=ctk.RIGHT)
label2.pack(pady=10, padx=10)
txtBxRes = ctk.CTkTextbox(master=mainFrame, height=1) #* Textbox de los resultados
txtBxRes.pack(padx=10)

label3 = ctk.CTkLabel(text="Ayuda:", master=mainFrame, justify=ctk.RIGHT)
label3.pack(pady=10, padx=10)
txtBxHelp = ctk.CTkTextbox(master=mainFrame, height=80, width=500) #* Textbox de ayuda al usuario
txtBxHelp.pack(padx=10)
txtBxHelp.insert("0.0", "Al ingresal los datos de 'optimización', ingrese algunas de estas opciones: 'distancia' o 'tiempo'")
```

Ya como parte final, en la siguiente imagen se muestra como quedo la ventana principal con el diseño que se implemento gracias a las librerías mencionadas



III. Backend

Para el apartado principal, que es el algoritmo ya en si, el funcionamiento de todo el proyecto para la solución del problema principal comienza en este apartado.

El algoritmo principal prácticamente es una implementación básica de un algoritmo de Dijkstra para encontrar la distancia más corta en un grafo, utilizando la biblioteca NumPy y la cola de prioridad de la biblioteca heapq.

Y para entrar un poquito más en contexto, ¿Que es el Dijkstra? Bueno pues el algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista.

Una vez sabiendo esta información ahora si comenzamos con el código. Primero, se importan las bibliotecas necesarias que es este caso fueron utilizadas las bibliotecas “numpy” como np para crear y manipular matrices y “heapq” como hpq para gestionar la cola de prioridad.

```
import numpy as np
import heapq as hpq
```

Después se define una clase para representar un grafo no dirigido con métodos para añadir aristas y calcular las distancias más cortas desde un nodo fuente utilizando el algoritmo de Dijkstra. El constructor de la clase Graph inicializa el grafo con un número dado de vértices, creando una matriz de adyacencia con ceros.

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = np.zeros((vertices, vertices))

    def addLimite(self, src, dest, weight):
        self.graph[src][dest] = weight
        self.graph[dest][src] = weight

    def metDijkstra(self, src):
        dist = [float('inf')] * self.V
        dist[src] = 0
        visitado = [False] * self.V
        pq = [(0, src)]

        while pq:
            d, u = hpq.heappop(pq)
            visitado[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and not visitado[v] and dist[v] > dist[u] + self.graph[u][v]:
                    dist[v] = dist[u] + self.graph[u][v]
                    hpq.heappush(pq, (dist[v], v))

        return dist
```

Se este apartado utilizamos un método que añade una arista bidireccional entre dos nodos con un peso dado. Y también otro método que calcula las distancias más cortas desde el nodo fuente a todos los otros nodos del grafo. Utiliza una cola de prioridad para seleccionar el siguiente nodo con la distancia mínima y actualiza las distancias a los nodos adyacentes si se encuentra un camino más corto.

En la función main, como se muestra en la siguiente imagen, se solicitan al usuario las coordenadas de dos puntos (inicio y

final) y se crea un grafo con dos nodos. Luego, se añade una arista entre estos dos nodos con un peso calculado como la distancia euclidiana entre los puntos. También se pregunta al usuario si desea optimizar según la distancia o el tiempo.

Se ejecuta el algoritmo de Dijkstra desde el nodo inicial y se imprime la distancia mínima o el tiempo estimado según la opción de optimización seleccionada. Si el usuario elige optimizar por tiempo, se asume que la velocidad es de 60 km/h para la conversión de distancia a tiempo.

```
def main():
    #? Entradas de texto de los puntos que debe de tomar en cuenta el usuario para resolve
    xInicio = float(input("Punto x inicial: "))
    yInicio = float(input("Punto y inicial: "))
    xFinal = float(input("Punto x final: "))
    yFinal = float(input("Punto y final: "))

    #? Opción de optimización para resolver el problema
    opti = input("Criterio (distancia, tiempo): ")

    #? Creación del grafo y añadir los aristas con los pesos correspondientes
    g = Graph(2)
    g.addLimite(0, 1, np.sqrt((xFinal - xInicio)**2 + (yFinal - yInicio)**2))

    #? Ejecutar la función de Dijkstra
    distCorta = g.metDijkstra(0)

    if opti.lower() == "distancia":
        print("La distancia mínimo entre los puntos es: ", distCorta[1], "mts")

    elif opti.lower() == "tiempo":
        tiempo = distCorta[1] / 60 #*Supongamos que va a 60km/hr
        print("El tiempo minimo es de: ", tiempo, "hrs")
    else:
        print("Error")
```

Y por último en este apartado, en el último If que se muestra en la siguiente imagen, este se asegura que la función main se ejecute solo si el script se ejecuta directamente, no cuando se importa como módulo.

```
if __name__ == "__main__":
    main()
```

Como añadido, en este proyecto hubo una prueba fallida que se intento realizar con ayudad de la fuerza bruta, pero al menos en nuestra forma de implementación, no lográbamos llegar al resultado que se quería ya que este se nos complicó demasiado y decidimos irnos por otro tipo de método

IV. Resultados Obtenidos del Proyecto

El resultado al que se llego con este proyecto fue bueno ya que este soluciona el problema planteado del proyecto como tal pero se sabe que esto pudo mejorar mucho mas si se hubiera implementado de manera correcta otro técnicas en el algoritmo aparte de la planteada, pero al final el resultado fue satisfactorio y se resolvió.

V. Conclusiones

En conclusión, este proyecto no fue del todo complicado, solo complejo pero mas que nada en aspectos en los cuales nosotros mismos nos complicábamos, pero todo salió como se pidió en un principio, la utilización del tkinter custom le dio un toque bueno a las vista del proyecto y la forma con la que fue realizado el algoritmo principal resulto factible para la solución del problema.

Referencias

[1] python-tkdnd. (2021, 31 julio). PyPI. https://pypi.org/project/python-tkdnd/

[2] tkinter — Python interface to Tcl/Tk. (s. f.). Python Documentation. https://docs.python.org/es/3/library/tkinter.html

[3] customtkinter. (2021, 7 mayo). PyPI. https://pypi.org/project/customtkinter/0.3/