Instituto Tecnológico Superior de Misantla

ANÁLISIS DE ALGORITMOS DE LA RUTA MÁS CORTA EN UN GRAFO QUE REPRESENTA LA CIUDAD DE MARTÍNEZ DE LA TORRE, VERACRUZ

TESIS

QUE PARA OBTENER EL GRADO DE:

Maestro en Sistemas Computacionales

Presenta:

I.S.C. HUGO LUCAS ALVARADO

Director:
DR. EDDY SÁNCHEZ DE LA CRUZ

Co-Director:
DR. RAJESH ROSAN BISWAL

Misantla, Veracruz Noviembre de 2018

Al ángel que guía mi camino:
Á. A. L. A.

Agradecimientos

A mis maestros, que han contribuido a mi formación con sus enseñanzas, su ejemplo y su vocación.

A Dios, por la familia que me ha dado.

Un agradecimiento especial al Dr. Eddy. Su compromiso, motivación y sobre todo paciencia para con sus alumnos, son dignos de admiración.

A todos ellos que de manera directa o indirecta han colaborado para ayudarme a llegar hasta aquí.

¡Muchas gracias a todos!

Resumen

Se presenta una solución al problema de encontrar la ruta óptima en la ciudad de Martínez de la Torre, Veracruz, utilizando diversos algoritmos tradicionales que resuelven el problema de la ruta más corta. Se contempla también la manipulación de parámetros adicionales que representan ciertas restricciones en la elección de la ruta. El problema de la ruta más corta es un caso específico de problemas de optimización y se puede abordar utilizando Teoría de grafos. Desde un punto de vista computacional, los grafos son estructuras de datos que pueden ser tratadas como matrices, tablas o listas. Por otro lado, diversas organizaciones en Martínez de la Torre, enfrentan el problema de hallar la ruta óptima para reducir tiempos y costos, y para ofrecer un mejor servicio a sus clientes. En la presente investigación se aborda el problema de hallar la ruta óptima y se implementa la solución en la ciudad de Martínez de la Torre. Específicamente se analiza la construcción de las estructuras de datos que permiten la representación del grafo de la ciudad en la memoria de la computadora y, se implementan los algoritmos de Dijkstra, Bellman-Ford y Floyd-Warshall. Un grafo $G = (V, E)$ con $|V| = 3267$ y $|E| = 8966$ ha sido construido para representar la red de calles de la ciudad mencionada. Los tiempos de ejecución ponen de manifiesto que el algoritmo de Dijkstra es es el más eficiente de las opciones analizadas, en segundo lugar se encuentra el algoritmo Bellman-Ford y en la posición final Floyd-Warshall. Sin embargo, cada uno de los algoritmos presenta ventajas y desventajas en diversas aplicaciones.

Abstract

We present a solution to the problem of finding the optimal route in the Martinez de la Torre, Veracruz city, using several known algorithms that solve the shortest route problem. The manipulation of additional parameters that represent certain restrictions in the choice of the route is also contemplated. The shortest route problem is a specific case of optimization problems and can be addressed using graph theory. From a computational point of view, graphs are data structures that can be treated as matrices, tables or lists. On the other hand, several organizations in Martínez de la Torre, face the problem of finding the optimal route to reduce times and costs, and to offer a better service to their customers. In the present investigation the problem of finding the optimal route is addressed and the solution is implemented in Martínez de la Torre city. Specifically, the construction of data structures that allow the representation of the city graph in computer memory and the Dijkstra, Bellman-Ford and Floyd-Warshall algorithms is analyzed. A graph $G = (V, E)$ with $|V| = 3267$ and $|E| = 8966$ has been constructed to represent the network of streets in the city before mentioned. The execution times show that the Dijkstra agorhythm is the most efficient of the analyzed options, second place is the Bellman-Ford algorithm and in the final position Floyd-Warshall. However, each of the algorithms presents advantages and disadvantages in various applications.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

CAPÍTULO 1

GENERALIDADES

1.1 Introducción

Los problemas de optimización están presentes en industrias diversas como los bancos, la educación, los transportes de carga, entre otros. Muchos problemas de optimización importantes se analizan mejor por medio de una representación gráfica (grafos) o de red. Algunos modelos específicos de red son: problema de la trayectoria más corta, problemas de flujo máximo, problemas de árbol de expansión mínima, etc. [19]

Es de interés, para fines de la presente investigación, describir el problema de la trayectoria más corta. También conocido como el problema de la ruta más corta, consiste en encontrar la trayectoria de longitud mínima del nodo a a cualquier otro nodo en una red. La solución aquí descrita utiliza grafos y los algoritmos Dijkstra (D), Bellman-Ford (BF) y Floyd-Warshall (FW).

El primer artículo de teoría de grafos fue de Leonhard Euler en 1736. En éste se presentaba una teoría general que incluía una solución al ahora conocido problema de los puentes de Königsberg. Sin embargo, no fue sino hasta 1920 que surgió un interés sostenido, amplio e intenso en teoría de grafos [8]. Seguramente el reciente interés en la materia se debe a su aplicabilidad en muchas áreas, incluyendo ciencias de la computación, química, investigación de operaciones, ingeniería eléctrica, lingüística y economía.

Este trabajo de investigación atiende el caso de la ciudad de Martínez de la Torre, Veracruz. Resulta de especial interés debido a que en la ciudad se presentan múltiples problemas de tráfico vehicular cuya solución está relacionada con el problema de la ruta más corta. Se implementan y analizan diversos algoritmos de la ruta más corta sobre un grafo que representa las calles de la ciudad.

1.2 Descripción de problema

En la actualidad son útiles los sistemas que brindan, de manera precisa, la mejor ruta en un área geográfica determinada. Existen diversas necesidades que requieren este tipo de sistemas, por mencionar algunas: redes de distribución, programas de vialidad, redes de telecomunicaciones, redes eléctricas, entre otras.

En la ciudad de Martínez de la Torre, municipio de Veracruz, suelen presentarse situaciones como: gastos excesivos en la distribución de productos por parte de las empacadoras de cítricos, incapacidad de satisfacer la demanda de reparto de Gas LP, incongruencias en el consumo de combustible de vehículos de distribución, servicio ineficiente de taxis, entre otros problemas de vialidad; estos casos tienen un denominador común: el problema de encontrar la ruta óptima. Al resolver este problema es posible tener un control más preciso tanto de tiempos como de costos. Por otro lado, se dispone de herramientas cartográficas avanzadas como Google Maps, que proporcionan información útil para construir un sistema que permita determinar la mejor ruta.

1.3 Justificación

Las organizaciones que presentan la problemática de optimizar rutas requieren una solución. Seguramente al atender dicha problemática será necesario "partir de cero", debido a que no se dispone de una herramienta exclusiva para encontrar la mejor ruta en un área geográfica específica. El presente está enfocado a la ciudad de Martínez de la Torre, Veracruz. Con esta aportación diversos sectores resultan beneficiados, ya que a partir de entonces, se dispone de resultados que resultan de utilidad para la búsqueda de soluciones que se enfoquen en atender el problema de la ruta más corta en la ciudad.

Por otra parte, es importante mencionar que los algoritmos que se analizan en esta investigación, son de utilidad para hallar la ruta más corta en un grafo que representa las calles de Martínez de la Torre. Sin embargo, no solo es importante la distancia, también existen otros factores a considerar al elegir la mejor ruta con la finalidad de reducir costos y tiempos; es necesario entonces, agregar parámetros que pueden ser contemplados en la elección de la ruta óptima. Ejemplos de estos parámetros son: tramos cortados en calles por reparación (u otro factor), altura máxima permitida para ciertos vehículos, encharcamientos ocasionados por lluvias, entre otros. Resulta obvio que la presencia de estos factores influye en la elección de la ruta debido a que restringen el tránsito vehicular por determinadas calles (aristas). En adelante nos referimos a estos factores como **"parámetros adicionales"** (PA).

Martínez de la Torre, Veracruz a pesar de ser una ciudad relativamente pequeña, presenta graves problemas de congestionamiento vial. Entre otras causas, quizá la razón principal es el cuello de botella que se ocasiona al cruzar el puente 1, principal vía de comunicación, que conecta Martínez de la Torre con Villa Independencia, una localidad vecina con una mancha urbana bastante importante para la ciudad. Según datos del INEGI, en 2010, la cantidad de habitantes de las localidades de Martínez de la Torre e Independencia es de $60,074$ y $15,297$ respectivamente. La Figura 1.1 muestra la ubicación del puente mencionado.

El tráfico vehicular ha llegado a ser uno de los problemas más importantes que enfrentan las ciudades en las últimas décadas. Esta problemática está presente en diversos ámbitos. El tráfico lento provoca un incremento en las emisiones de gases contaminantes, un elevado consumo de combustible y perdida de tiempo en atascos de tráfico afectan la economía, los retardos en el desplazamiento de vehículos de emergencia pueden cobrar vidas. Además, problemas de salud como estrés, ansiedad, depresión y otros; son algunos de los efectos causados por la congestión vehicular [9]. Martínez de la Torre, no es la excepción. El presente trabajo, así como aquellas investigaciones que contribuyan a la búsqueda de soluciones para hacer frente a esta problemática, son de relevancia para el bienestar social.

Si bien es cierto que existen en el mercado sistemas de planificación de rutas, estos presentan elevados costos o no se adaptan fácilmente a las necesidades de las

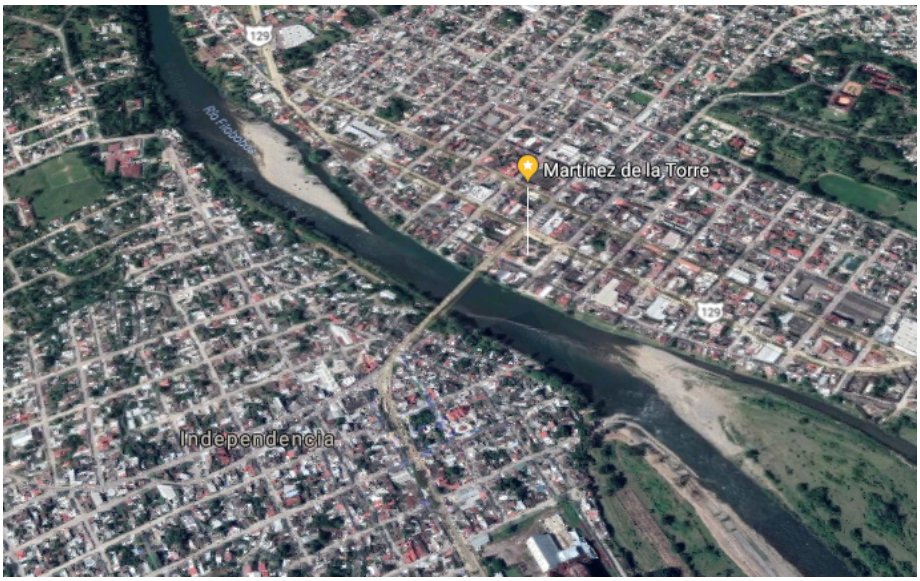

Figure 1.1: Puente 1 de Martínez de la Torre, Ver.

organizaciones que los requieren. Por otro lado, es posible desarrollar software a la medida para este fin utilizando APIs existentes. Sin embargo, el uso de estas APIs también puede tener un costo y no siempre satisface las necesidades completamente. Tal es el caso de la API Google Maps. Además de tener un costo, requiere de actualizaciones por parte de los usuarios acerca de información relevante para la generación de rutas. Por ejemplo, un tramo cortado que impide la circulación por alguna vialidad no será considerado a menos que un usuario (o autoridad correspondiente) lleve a cabo cierta notificación. En ese sentido un sistema de generación de rutas local es de utilidad. El presente trabajo contempla la construcción de las estructuras de datos que representan el grafo de la ciudad y la implementación y análisis de algoritmos sobre dicho grafo.

Una adecuada planificación de rutas por parte de las diversas empresas de la región, puede contribuir al desahogo vehicular que presenta la ciudad, además de representar una ventaja competitiva debido a que permite mayor control en cuanto a reducción de tiempos y costos.

Debido a lo anteriormente mencionado resulta de interés aplicar herramientas computacionales y teoría de grafos para resolver esté problema de optimización en la ciudad de Martínez de la Torre, Veracruz.

1.4 Objetivos

1.4.1 Objetivo general

Analizar algoritmos que permitan encontrar rutas óptimas contemplando restricciones impuestas por parámetros adicionales a la distancia, así como construir estructuras de datos que permitan la representación formal de un grafo de la ciudad de Martínez de la Torre, Veracruz para su manipulación por computadora.

1.4.2 Objetivos específicos

- Construir un grafo de una zona de interés para realizar pruebas iniciales.
- Crear un grafo que represente en su totalidad la ciudad de Martínez de la Torre, Veracruz.
- Implementar los algoritmos de Dijkstra, Bellman-Ford y Floyd-Warshall sobre el grafo creado para determinar la ruta más corta.
- Determinar y manipular parámetros adicionales que se deben considerar en la elección de la ruta más corta.
- Analizar los resultados.

1.5 Hipótesis

H1: *Los algoritmos de Dijkstra, Bellman-Ford y Floyd-Warshall son apropiados para determinar la ruta más corta en un grafo que representa las calles de la ciudad de Martínez de la Torre, Veracruz, contemplando restricciones impuestas por parámetros adicionales a la distancia entre nodos.*

H2: *Es posible representar la red de calles de la ciudad de Martínez de la Torre, Veracruz, mediante estructuras de datos que representen un grafo para su manipulación por computadora.*

1.6 Propuesta de solución

En esta investigación se pretende construir un grafo que represente la ciudad de Martínez de la Torre, Veracruz e implementar los algoritmos de Dijkstra, Bellman-Ford y Floyd-Warshall contemplando parámetros adicionales a la distancia, para determinar la ruta más corta. Es importante mencionar que la implementación de los algoritmos mencionados en este proyecto son de utilidad para hallar la ruta más corta, sin embargo, no solo es importante la distancia, también existen otros parámetros a considerar al elegir la mejor ruta con la finalidad de reducir costos y tiempos, es necesario entonces, manipularlos mediante los algoritmos; dichos parámetros son: el sentido de las calles, encharcamientos graves durante temporada de lluvias, acceso restringido al transporte pesado, altura máxima permitida de vehículos, flujo vehicular, amplitud de las calles, normas locales de tránsito, nivel de inseguridad, entre otros. La optimización de las rutas representa, en la ciudad de Martínez de la Torre, un beneficio para diversos sectores. De esta manera los resultados obtenidos en el presente, significan una aportación a la sociedad.

1.7 Estructura de la tesis

El presente documento está organizado en seis capítulos. El Capítulo 1 presenta el problema atendido y la propuesta de solución, así como los objetivos e hipótesis. En el Capítulo 2 se mencionan algunos conceptos teóricos y el estado del arte. Los materiales y métodos utilizados se describen en el Capítulo 3. El Capítulo 4 expone los experimentos realizados. El análisis de resultados se presenta en el Capítulo 5. Finalmente, en el Capítulo 6 se mencionan las conclusiones y trabajo futuro.

CAPÍTULO 2

MARCO TEÓRICO

2.1 El problema de la ruta más corta

Los problemas de optimización están presentes en diversos ámbitos y consisten en minimizar o maximizar el valor de una variable. La investigación de operaciones estudia la amplia categoría de problemas de optimización. Muchos de ellos se pueden modelar, entre otras técnicas, mediante una representación gráfica o de red. El problema de la ruta más corta es un modelo de red que pertenece tanto a la investigación de operaciones como a la teoría de grafos.
La teoría de grafos es una rama importante de las matemáticas modernas. Su nacimiento se le atribuye al matemático suizo Leonhard Euler en el año de 1736. Se encarga del estudio de las propiedades y características de los grafos, y es considerada piedra angular para la fundamentación matemática en varias áreas de las ciencias de la computación. Además, también son de gran utilidad para la representación de circuitos eléctricos, en problemas de trayecto óptimo y para modelar redes de carreteras. Más aun, los grafos pueden utilizarse en áreas como las ciencias sociales, la lingüística, las ciencias físicas, las ciencias económicas, y en diversas ingenierías, entre otras [18].
Encontrar la trayectoria más corta, es un problema clásico en la teoría de grafos conocido como el problema de la ruta más corta, para el cual se han propuesto muchas soluciones diferentes. Este problema consiste en encontrar el camino más corto entre dos vértices en un grafo cuyas aristas tienen un peso (grafo ponderado).

En este trabajo de investigación se ha construido un grafo de la red de calles de la ciudad de Martínez de la Torre, Veracruz. En el grafo, los nodos representan intersecciones y límites de calle, mientras que las aristas representan las calles que conducen de un nodo a otro contemplando el sentido de las mismas. Los pesos de las aristas representan la distancia (en metros) entre nodos.

2.2 Grafos

Diversos conceptos relacionados con grafos, como dígrafos, árboles, árboles binarios, árboles de expansión, entre otros; se utilizan en muchas áreas de las matemáticas, la computación, la inteligencia artificial y la investigación de operaciones. Los grafos son una generalización de los árboles, estructuras de datos compuestas por vértices y las conexiones entre ellos capaces de representar diversas situaciones y sucesos. Son ampliamente estudiados en estructuras de datos y matemáticas discretas. Antes de conocer los algoritmos para resolver el problema de la ruta más corta, se presentan algunos conceptos importantes relacionados con la aplicación de grafos en el tema que se aborda en esta investigación [4].

2.2.1 Grafo simple

Un grafo simple $G = (V, E)$ consiste en un conjunto no vacío V de vértices (o nodos) y un conjunto E de aristas, siendo cada arista un conjunto de dos vértices de V. Una arista de grafo simple tiene la forma (v_i, v_j) y para una arista como ésta, $(v_i, v_j) = (v_j, v_i)$. $|V|$ denota el número de vértices y $|E|$ denota el número de aristas. Se dice que una arista e en un grafo que se asocia con el par de vértices u y w es incidente sobre u y w, y se dice que u y w son incidentes sobre e y son vértices adyacentes. El grado de un vértice v , $deg(v)$, es el número de aristas incidentes con v.

2.2.2 Grafo dirigido o dígrafo

Se compone de un conjunto no vacío V de vértices y un conjunto E de aristas (también llamadas arcos), donde cada arista es un par de vértices de V. En un dígrafo, cada arista es de la forma (v_i, v_j) y en este caso, $(v_i, v_j) \neq (v_j, v_i)$.

2.2.3 Multígrafo

Es un grafo en el cual dos vértices pueden unirse por múltiples aristas, denominadas aristas paralelas. Un multígrafo $G = (V, E, f)$ se compone de un conjunto de vértices V, un conjunto de aristas E, una función $f : E \to v_i, v_j : v_i, v_j \in V$ y $v_i \neq v_j$.

2.2.4 Grafo ponderado

Un grafo con números en las aristas se llama grafo ponderado. Si la arista e se etiqueta k, se dice que el peso de la arista e es k. Según el contexto en el cual se usan estos grafos, el número asignado a las aristas se conoce como peso, costo, distancia, longitud o de alguna otra manera. En un grafo ponderado, la longitud de la ruta es la suma de los pesos de las aristas en la ruta.

2.2.5 Trayectoria (o camino)

Sean v_0 y v_n vértices en un grafo. Una trayectoria de v_0 a v_n es una sucesión alternante de de $n + 1$ vértices y n aristas que comienza en el vértice v_0 y termina en el vértice v_n,

$$(v_0, e_1, v_1, e_2, v_2, ..., v_{n-1}, e_n, v_n),$$

donde la arista e_1 es incidente sobre los vértices v_{i_1} y v_i para $i = 1, ..., n$. Otra forma de denotar una trayectoria es $v_0, v_1, v_2, ..., v_{n-1}, v_n$.

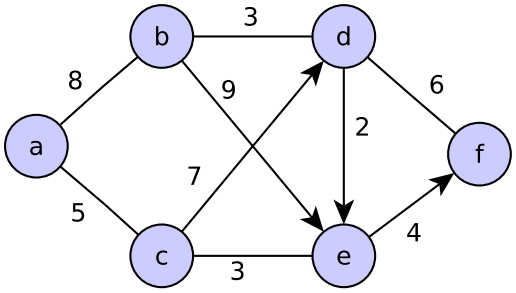

Figure 2.1: Ejemplo de grafo dirigido ponderado

2.2.6 Grafo denso

Se dice que un grafo G con m vértices y n aristas es denso cuando $m = O(n^2)$ y disperso, cuando $m = O(n)$ o inclusive $O(nlog(n))$. La distinción entre grafos densos y dispersos es relativamente vaga, decimos en términos prácticos que es denso cuando $|E|$ es cercana a $|V|^2$ y disperso cuando $|E|$ es cercana a $|V|$.

2.2.7 Grafo completo

Un grafo se llama grafo completo y se denota por K_n si para cada par de vértices existe exactamente una arista que los conecta.

2.3 Representación de grafos

La representación habitual de un grafo es de manera gráfica como el ejemplo de la Figura 2.1, el cual se trata de un dígrafo ponderado. Por razones de simplicidad las líneas sin flecha representan aristas en ambos sentidos con el mismo peso. De manera computacional, los grafos se pueden representar mediante listas de adyacencia y matrices de adyacencia e incidencia. La elección de un grafo de estas características, que corresponde con el grafo generado en la presente investigación, servirá para definir las distintas formas de representar grafos [1, 12].

2.3.1 Lista de adyacencia

Especifica todos los vértices adyacentes a cada vértice del grafo. Esta lista puede implementarse como tabla o como lista enlazada. Si el grafo G es disperso entonces la matriz de adyacencia (descrita en el siguiente punto) A de G contiene muchos ceros; por tanto, se desperdicia bastante espacio de memoria. En consecuencia cuando G es

disperso, G suele representarse en memoria por medio de una lista enlazada, también denominada lista de adyacencia.
La lista de adyacencia correspondiente al grafo G de la Figura 2.1 se ilustra en la Tabla 2.1. Aquí se muestra cada vértice v en G seguido por una lista de los nodos adyacentes a v (lista de adyacencia).Cada arista de G corresponde a un vértice v único en una lista de adyacencia y viceversa. Por lo tanto, G tiene 14 aristas como 14 vértices hay en las listas de adyacencia. Cabe recordar que, en nuestro grafo de ejemplo, una arista representada por una línea sin punta de flecha indica dos aristas en sentidos opuestos, ambas con el mismo peso.

Tabla 2.1: Ejemplo de lista de adyacencia

Vértice	Lista de adyacencia
a	b, c
b	a, d, e
c	a, d, e
d	b, e, f
e	c, f
f	d

2.3.2 Matriz de adyacencia

Una matriz de adyacencia de un grafo $G = (V, E)$ es una matriz binaria $|V|x|V|$ tal que cada entrada

$$a_{ij} = \begin{cases} 1 \text{ si existe una arista } (v_i, v_j) \\ 0 \text{ en caso contrario} \end{cases} \tag{2.1}$$

La Tabla 2.2 muestra la matriz de adyacencia del grafo G de la Figura 2.1. Se trata de una matriz cuadrada binaria. Tanto el número de filas como de columnas está determinado por la cantidad de vértices v de G. Cada elemento a_{ij} representa una arista cuyo origen es el vértice v_i y destino v_j. Como puede observarse existen 14 celdas marcadas con 1, cada una corresponde a una arista e del grafo G.

Tabla 2.2: Ejemplo de matriz de adyacencia

	a	b	c	d	e	f
a	0	1	1	0	0	0
b	1	0	0	1	1	0
c	1	0	0	1	1	0
d	0	1	0	0	1	1
e	0	0	1	0	0	1
f	0	0	0	1	0	0

2.3.3 Matriz de pesos

También conocida como matriz de distancias, una matriz de pesos $G = (V, E)$, donde G es un grafo ponderado, es una matriz $|V|x|V|$ tal que cada entrada

$$a_{ij} = \begin{cases} \text{w si existe una arista } (v_i, v_j) \\ \text{-1 en caso contrario} \end{cases} \tag{2.2}$$

Por convención, y para fines del presente trabajo, se ha utilizado -1 no como un valor numérico sino como un símbolo que indica la ausencia de arista (v_i, v_j). w representa el peso de la arista (v_i, v_j). La Tabla 2.3 muestra la matriz de pesos correspondiente al grafo de la Figura 2.1.

Tabla 2.3: Ejemplo de matriz de pesos

	a	b	c	d	e	f
a	0	8	5	0	0	0
b	8	0	0	3	9	0
c	5	0	0	7	3	0
d	0	3	0	0	2	6
e	0	0	3	0	0	4
f	0	0	0	6	0	0

2.3.4 Matriz de incidencia

Una matriz de incidencia del grafo $G = (V, E)$ es una matriz $|V|x|E|$ tal que

$$a_{ij} = \begin{cases} 1 \text{ si la arista } e_j \text{ es incidente con el vértice } v_i \\ 0 \text{ en caso contrario} \end{cases} \tag{2.3}$$

En un grafo dirigido $G = (V, E)$, para cualquier arista $e = (v_i, v_j)$ se dice que e es incidente en los vértices v_i y v_j, los cuales son sus vértices extremos, v_i es adyacente hacia v_j, mientras que v_j es adyacente desde v_i. Además, el vértice v_i es el vértice origen (o fuente) y v_j el vértice destino (o terminal).
Una matriz de incidencia de un grafo dirigido $G = (V, E)$ es una matriz $|V|x|E|$ tal que

$$a_{ij} = \begin{cases} 1 \text{ si la arista } e_j \text{ es incidente con el vértice } v_i \text{ y “sale de él”} \\ \text{-1 si la arista } e_j \text{ es incidente con el vértice } v_i \text{ y “llega hacia él”} \\ 0 \text{ si la arista } e_j \text{ no incide en el vértice } v_i \end{cases} \tag{2.4}$$

Continuando con nuestro grafo de ejemplo de la Figura 2.1 la matriz de incidencia correspondiente se muestra en la Tabla 2.4. Aquí una arista $e = (v_i, v_j)$ está abreviada de la forma $v_i v_j$.

Tabla 2.4: Ejemplo de matriz de incidencia

	ab	ac	ba	bd	be	ca	cd	ce	db	de	df	ec	ef	fd
a	1	1	-1	0	0	-1	0	0	0	0	0	0	0	0
b	-1	0	1	1	1	0	0	0	-1	0	0	0	0	0
c	0	-1	0	0	0	1	1	1	0	0	0	-1	0	0
d	0	0	0	-1	0	0	-1	0	1	1	1	0	0	-1
e	0	0	0	0	-1	0	0	-1	0	-1	0	1	1	0
f	0	0	0	0	0	0	0	0	0	0	-1	0	-1	1

Cabe preguntarse cual es la mejor representación de un grafo. Depende del problema a resolver. Si se trata de procesar vértices adyacentes al vértice v, es recomendable una lista de adyacencia. Por otra parte, si la tarea consiste en insertar o eliminar un vértice adyacente a v, una matriz es mejor opción. También, decimos que, las matrices de adyacencia se usan cuando el grafo G es denso, y las listas enlazadas suelen usarse cuando G es disperso.

2.4 Fórmula de Haversine

El problema de hallar la distancia entre dos puntos geográficos, se reduce a un problema de geometría conocido como longitud del círculo máximo, el cual permite hallar la distancia entre dos puntos sobre la superficie de una esfera. Hay múltiples opciones para realizar el cálculo, explicaremos la fórmula de Haversine, una de las más confiables y utilizada por la API Google Maps [14].
El método de Haversine tiene sus orígenes en la navegación marítima. Permite calcular la distancia entre dos puntos, dadas sus coordenadas geográficas. Sean (ϕ_1, λ_1) y (ϕ_2, λ_2) las coordenadas geográficas de los puntos 1 y 2 respectivamente, la fórmula de Haversine es:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\theta_2 - \theta_1}{2}\right) + \cos(\theta_1)\cos(\theta_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{2.5}$$

donde d es la distancia, θ_1, θ_2 y λ_1, λ_2 se refieren a la latitud y a la longitud, expresadas en radianes, de los puntos 1 y 2 respectivamente. r es el radio de la tierra $(6,378,137$ metros$)$.
Las coordenadas geográficas obtenidas de Google Maps, denominadas coordenadas GPS, proporcionan la latitud y longitud en grados decimales. Para realizar la conversión a radianes es necesario multiplicar por la constante $\pi/180$.

2.5 Estado del arte

A continuación se presenta información acerca de investigaciones que abordan la problemática de la optimización de rutas, así como las aplicaciones y mejoras realizadas al algoritmo de Dijkstra:
Edsger W. Dijkstra, resolvió dos problemas relacionados con grafos, uno de ellos consiste en encontrar la ruta de longitud mínima total entre dos nodos dados P y Q. La solución brindada es ampliamente utilizada y es conocido como algoritmo de Dijkstra [3].
Wang Shu-Xi, analizó el algoritmo de Dijkstra e identificó algunos aspectos que no estaban considerados en el problema original, con base en dicho análisis propuso mejoras al algoritmo y lo validó mediante experimentos. Las mejoras del algoritmo permiten: controlar el mecanismo de salida y evita entrar en un ciclo infinito en caso de existir vértices sin conexión, guardar información acerca de todos los vértices adyacentes y no solo del vértice anterior en la ruta, y que dos (o más) vértices sean etiquetados de manera simultanea al encontrar la ruta más corta [16].
Strehler et al., identificaron, entre otras cosas, la necesidad de determinar la ruta más corta y planificación de viajes para su utilización en vehículos eléctricos e híbridos, ya que es necesario regular la velocidad y el alcance, así como elegir paradas para la recarga del vehículo de manera estratégica, lo anterior debido a que el tiempo requerido para la recarga es mayor que los vehículos que utilizan combustibles fósiles. En el estudio mencionado desarrollaron un modelo apropiado para encontrar la ruta más corta para ese tipo de vehículos, basándose en el algoritmo de Dijkstra y considerando que una ruta puede contener ciclos, que conducen hacia una estación de recarga. En el estudio también consideraron los casos en que una estación de recarga pueda estar ocupada por otro vehículo [17].
Galán et al., manifiesta que aunque es posible calcular la ruta más corta mediante el algoritmo de Dijkstra, en situaciones reales está no siempre es la elegida. Los autores presentan un nuevo algoritmo denominado PEDA (Probabilistic Extension of Dijkstra's Algorithm), como una extensión al algoritmo de Dijkstra en el cual introducen cambios probabilísticos en el peso de las aristas y también en las decisiones al elegir el camino más corto. Como un ejemplo de aplicación presentan ATISMART*, un modelo que proporciona simulaciones de flujo de tráfico en ciudades inteligentes, utilizando señales inteligentes y el algoritmo PEDA. Cuando PEDA es aplicado al flujo de tráfico se obtienen simulaciones más realistas, en contraste con el modelo ATISMART, que utiliza el algoritmo de Dijkstra y por lo tanto satura aquellas vías que pertenecen a la ruta más corta, dejando algunas zonas del mapa casi vacías [9].
Zou et al. utilizaron un algoritmo basado en los algoritmos de Dijkstra y Bellman-Ford para planificación de rutas en sistemas de transportación aplicado a una red de sensores inalámbricos. Cada nodo en la red utiliza el algoritmo de Dijkstra y el algoritmo Bellman-Ford es utilizado para calcular la ruta más corta de manera global

en la red debido a que es posible su implementación de manera distribuida [20].
Hussain y Bingcai, presentan un nuevo método para seleccionar un nodo de tipo *clusterhead* (que gestiona la comunicación dentro de un cluster) en una red ad-hoc vehicular (VANET) utilizando los algoritmos K-Means y Floyd-Warshall. El algoritmo Floyd-Warshall calcula la distancia más corta de todos los pares para cada vehículo dentro del cluster definido. Un vehículo con la distancia promedio más corta en el cluster es elegido como el *clusterhead*. El algoritmo selecciona un vehículo centralizado, por lo tanto el tiempo de estabilidad mejora significativamente. Mediante una simulación demuestran que superan los algoritmos contemporáneos en términos de la media de la relación entre la señal pico y el ruido, el rango de transmisión, la conectividad promedio y la duración promedio del *clusterhead* [7].
Lozano *et al.*, proponen un método de solución exacta para el problema de la ruta más corta restringida (CSP) que es capaz de manejar redes grandes en un tiempo razonable. CSP es el problema de la ruta más corta sujeto a restricciones y fue utilizado para resolver eficazmente un problema de programación de turnos de actividades múltiples y un problema de diseño de rutas de transporte rápido de autobuses. El algoritmo propuesto, denominado algoritmo de impulsos, está basado en el algoritmo de Dijkstra para determinar la ruta más corta [11].
Guzmán *et al.*, presentan la generación de rutas óptimas entre dos nodos de un grafo de visibilidad que permite guiar el curso de un robot móvil. Utilizan y comparan dos algoritmos de búsqueda: el algoritmo de Dijkstra y el algoritmo genético. Ambos métodos arrojan la misma ruta óptima. Sin embargo, el algoritmo de Dijkstra tuvo un gasto computacional mucho menor que el algoritmo genético. El algoritmo genético no alcanza a demostrar su utilidad en su totalidad ya que son pocas las trayectorias subóptimas que tiene el ejemplo abordado. Este algoritmo puede arrojar mejores resultados cuando no se asocia con grafos, es decir, cuando tenga la disponibilidad de puntos posibles para encontrar la ruta óptima [6].
Karakatic y Podgorelec, realizaron un estudio acerca de algoritmos genéticos diseñados para resolver el problema del enrutamiento de vehículos de múltiples paradas (MD-VRP). Presentan fortalezas y debilidades de los métodos, operadores y entornos específicos utilizados para resolver este tipo de problemas. Los algoritmos genéticos (GA), pertenecientes al grupo de los metaheurísticos, son comparadas con otros enfoques, tanto exactos como heurísticos. Mientras los algoritmos exactos resuelven de manera eficiente problemas pequeños, existen dificultades para resolver problemas especiales de las variantes VRP. Los algoritmos metaheurísticos representan una alternativa. La principal ventaja de los GA sobre otros algoritmos metaheurísticos es el rendimiento y el resultado final sobre limitaciones de tiempo y de poder de cómputo, a la vez que se obtienen soluciones competitivas [10].

CAPÍTULO 3

MATERIALES Y MÉTODO

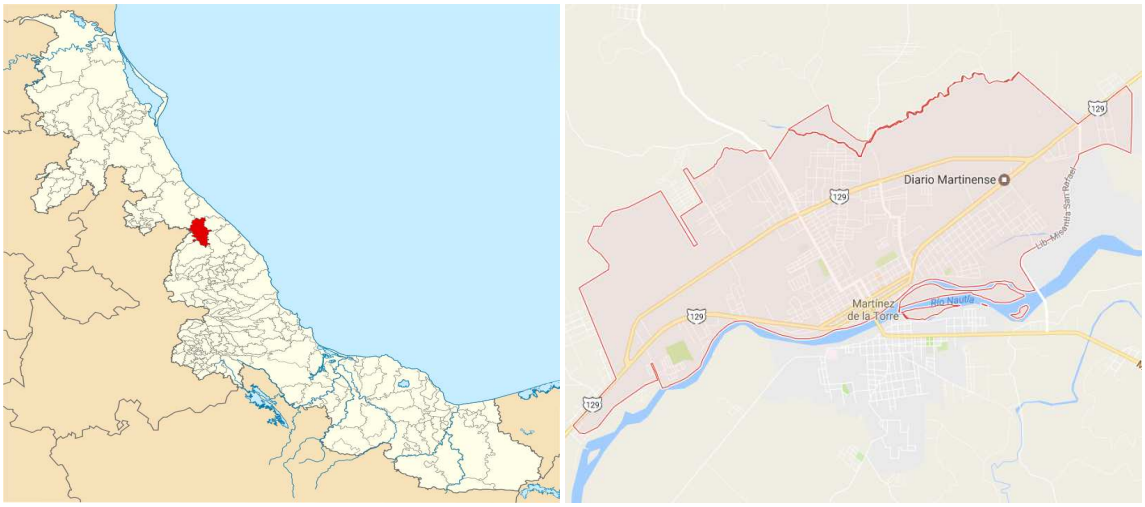

Figure 3.1: (a) Ubicación de Martínez de la Torre, Veracruz y (b) Mapa de la ciudad

3.1 Materiales cartográficos

Para obtener la información cartográfica se usó Google Maps[1], un servidor de aplicaciones web que pertenece a Alphabet Inc. Ofrece imágenes de mapas desplazables, así como fotografías por satélite del mundo. Además, permite obtener información sobre un lugar específico, medir la distancia entre dos puntos, consultar información sobre el tráfico vehicular, el transporte público, las rutas, entre otras. Para la presente investigación se utilizó esta herramienta para obtener información de la ciudad de Martínez de la Torre, Veracruz, a partir de la cual se construye el grafo.
Según la Encuesta Intercensal del 2015 realizada por el INEGI, el municipio de Martínez de la Torre cuenta con una población total de 110 415 habitantes (1.4% de la población estatal), una densidad de población de 277.0 hab/km^2 y una superficie de 815.13 km^2 (0.6% del territorio estatal), estos datos nos ayudan a entender que en la ciudad existe una población considerable, lo que conlleva al congestionamiento vial. En la Figura 3.1 a) se muestra la ubicación de Martínez de la Torre en el estado de Veracruz y en la Figura 3.1 b) se aprecia el mapa de la ciudad. Para realizar las pruebas iniciales se ha seleccionado la zona de interés que muestra la Figura 3.2, en la misma imagen aparece resaltado el grafo generado. En el grafo, los nodos representan las intersecciones o extremos de calles (numerados en negro, de 0 a 37) y los pesos de las aristas representan la distancia (en metros) entre los nodos (numerados en rojo).

[1]https://www.google.com.mx/maps

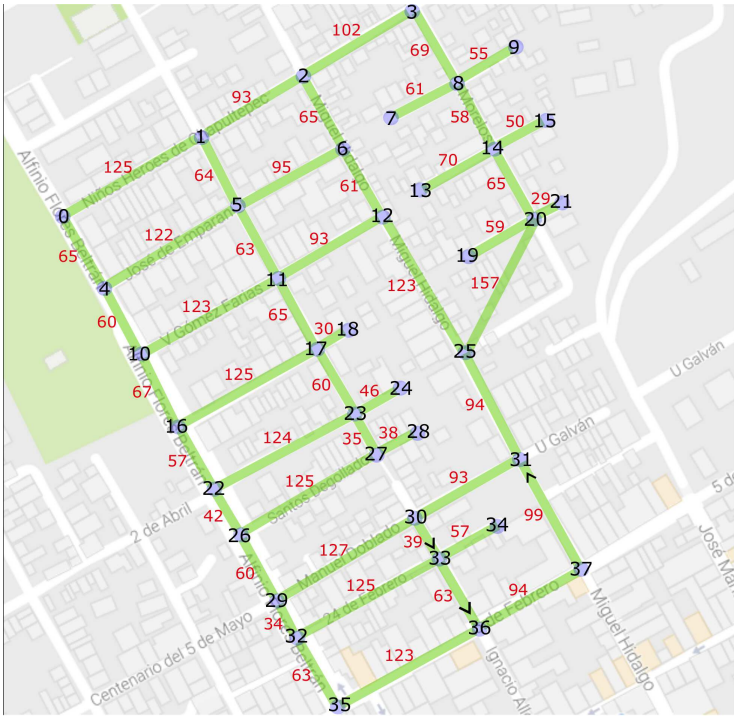

Figure 3.2: Área de interés: Colonia Melchor Ocampo

3.2 Herramientas de programación

Para implementar el algoritmo de Dijkstra con la información del grafo de la región seleccionada, se utilizó el lenguaje de programación Java con NetBeans IDE 8.2. El programa lee un archivo de texto que contiene la matriz de adyacencia del grafo, implementa el algoritmo y encuentra la ruta más corta entre dos puntos. Se utiliza programación orientada a objetos, de esta manera los vértices y aristas son objetos capaces de almacenar diversos atributos o características.
Para la construcción del grafo completo se ha utilizado la librería Java Client for Google Maps Services, que contiene las APIs necesarias para trabajar con direcciones, distancias, rutas, geolocalizaciones, mapas, etc. Permite trabajar con los servicios web de Google Maps. La Google Maps JavaScript API se utilizó para seccionar el mapa mediante el uso de polígonos y polilíneas. La API Apache POI se uso para el manejo de archivos de Microsoft Excel que contienen información del Grafo.

3.3 Método (Fase 1). Grafo del área de interés

El presente trabajo de investigación se ha manejado en dos fases. La primera fase consiste en construir el grafo de una sección que representa un área de interés de

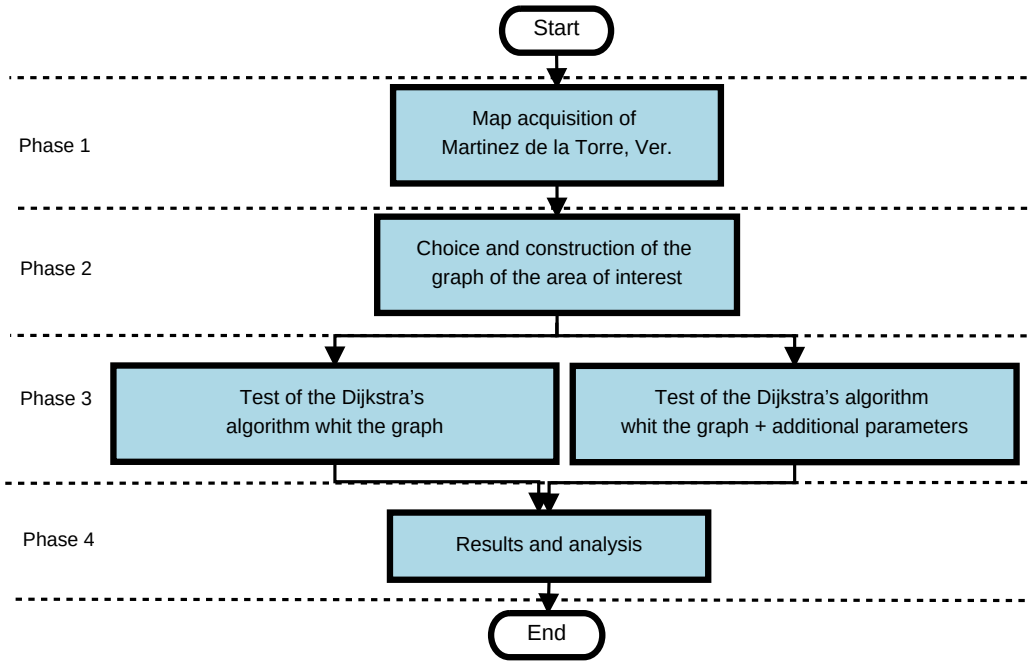

Figure 3.3: Metodología utilizada [13]

la ciudad, para posteriormente realizar pruebas con el algoritmo de Dijkstra. En la segunda fase se aborda la construcción del grafo la ciudad completa y se experimenta, además del algoritmo de Dijkstra, con los algoritmos Bellman-Ford y Floyd-Warshall. La Figura 3.3 muestra el diagrama de flujo de la metodología a seguir. El proceso consta de cuatro fases.

Como se ha mencionado, el modelo se pretende desarrollar para la ciudad de Martínez de la Torre, por ello, y con la finalidad de tener un panorama visual del problema, en la fase uno se adquirió el mapa de la ciudad. En la segunda fase se eligió una zona de interés dentro de la ciudad de Martínez de la Torre, se obtuvieron los datos básicos como son intersecciones (nodos) y distancia entre ellos (pesos de las aristas), se creó una representación gráfica (grafo) y finalmente se representó mediante una matriz de adyacencia. En la fase tres se llevó a cabo la programación del algoritmo en Java, se utiliza en primer lugar, una versión ampliada del algoritmo de Dijkstra que permite encontrar tanto la distancia de la ruta más corta (suma de los pesos) como la ruta más corta (secuencia de vértices que forman la ruta); y en segundo lugar, las adaptaciones necesarias para manejar parámetros adicionales. En esta fase también se realizaron las pruebas sobre dichos algoritmos. En la fase cuatro se obtienen y analizan los resultados, para determinar el correcto funcionamiento se comprueban los datos de manera manual y también se comparan resultados con los arrojados por la herramienta Google Maps.

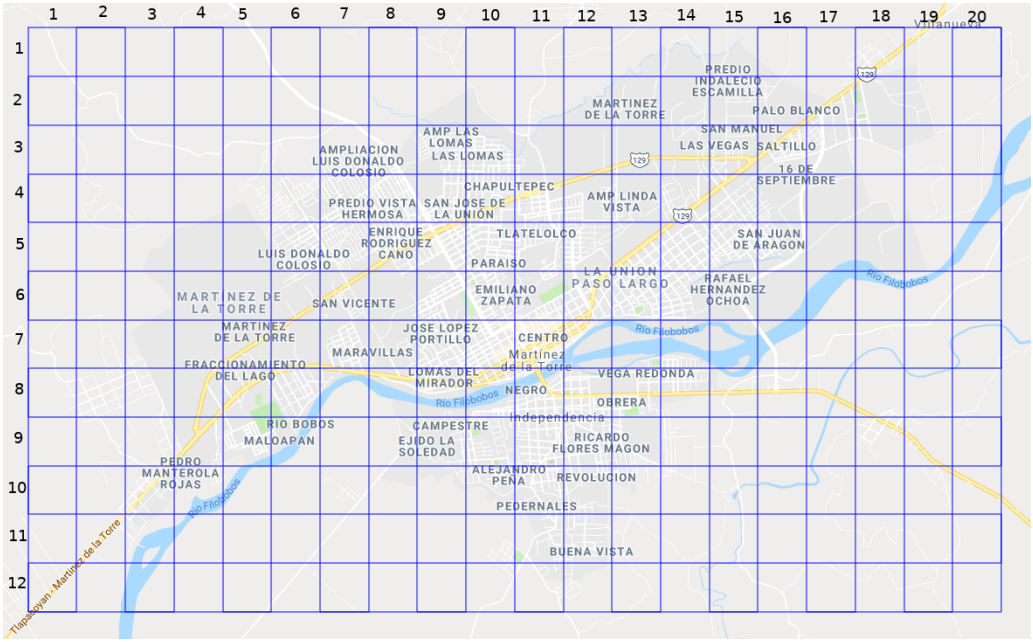

Figure 3.4: Mapa de Martínez de la Torre seccionado

3.4 Método (Fase 2). Grafo completo de la ciudad

3.4.1 Seccionamiento del mapa

Un área rectangular de $10x6km$ ($60km^2$) ha sido contemplada para construir el grafo de la ciudad de Martínez de la Torre. Se utiliza un mapa de Google Maps que corresponde con esas medidas y que abarca la ciudad en su totalidad. Para una mejor manipulación de la información y para facilitar la elaboración del grafo, el mapa ha sido seccionado en áreas de $500x500m$ ($0.25km^2$). Las distancias se han calculado mediante la fórmula de Haversine y las coordenadas geográficas se han estimado mediante el método del punto medio. Con ayuda de la Google Maps JavaScript API se han construido secciones rectangulares mediante polilíneas que se agregan a un mapa en forma de capa. El resultado final es un mapa dividido en 240 secciones en forma de malla como se muestra en la Figura 3.4.

3.4.2 Construcción de grafos individuales

La información obtenida de Google Maps son los nodos (intersecciones y límites de calles), las ubicaciones, las distancias y los nodos adyacentes. Para facilitar esta tarea, las distancias entre nodos adyacentes se calculan a partir de las coordenadas geográficas mediante la API DistanceMatrix de Google Maps. Los nodos de cada sección son etiquetados de manera temporal. Los grafos individuales se representan

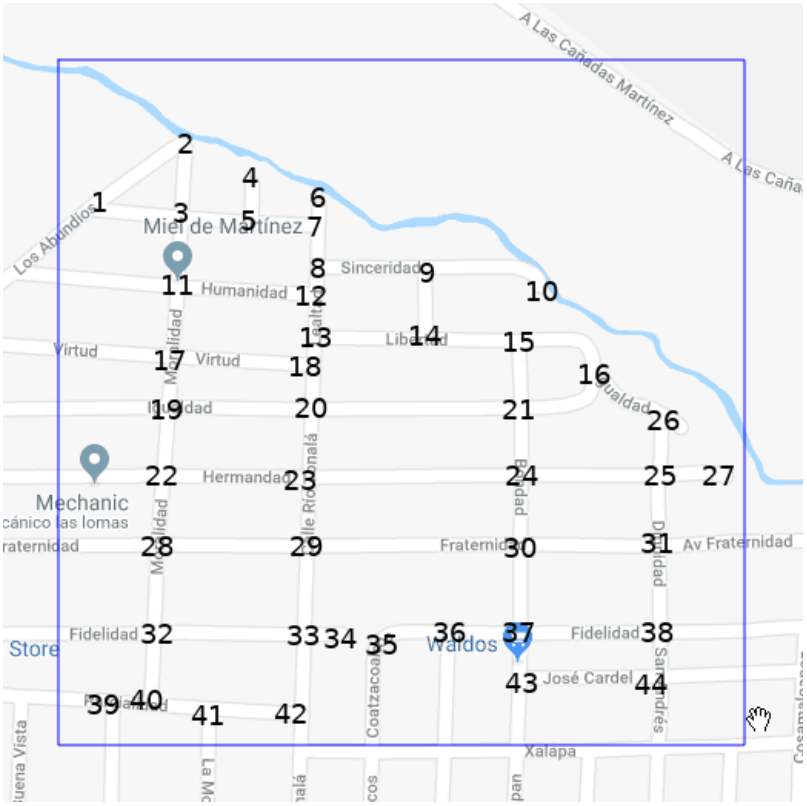

Figure 3.5: Ejemplo de sección con nodos etiquetados

mediante listas de adyacencia y se registran en un documento de Microsoft Excel que permitirá crear el grafo completo posteriormente. La Figura 3.5 muestra un ejemplo de una sección con los nodos etiquetados.

3.4.3 Construcción del grafo completo

Mediante un programa realizado en Java se genera de manera dinámica el grafo completo. Este define nodos y los etiqueta nuevamente para construir, a partir de los grafos individuales, una lista de adyacencia y una matriz de distancias globales. El resultado final es un grafo que representa la red de calles y carreteras de la región que ilustra la imagen 3.4 y que corresponde con la zona urbana y alrededores de la ciudad de Martínez de la Torre, Veracruz. Los detalles y la representación del grafo obtenido se describen en el Capítulo 4.

3.4.4 Incorporación de parámetros adicionales

Como ya se ha mencionado anteriormente, los PA contemplados en el presente trabajo son aquellos factores que restringen el tránsito vehicular por determinadas aristas (calles). Ejemplos de estos factores son: altura máxima permitida, acceso restringido a transporte pesado, amplitud de las calles, tramos cortados por reparación o tráfico detenido, encharcamientos, entre otros. Cabe mencionar que queda fuera del alcance de este trabajo de investigación la recopilación de información acerca de tales factores en la ciudad de Martínez de la Torre. En lugar de eso, nos enfocamos en adaptar y probar los algoritmos para la manipulación de dichos parámetros.
Es importante señalar que una solución alternativa al manejo de los PA consiste en alterar la topología de la red, específicamente, agregando o eliminando aristas. Sin embargo, esta solución no parece conveniente debido a que genera un grafo que no corresponde con la red de calles de la ciudad en la realidad. Por ejemplo, si asumimos que la arista (a, b) está bloqueada por tráfico detenido, entonces en determinado momento (mientras permanece el bloqueo) una ruta alterna deberá ser generada a pesar de que existe una calle que comunica los nodos a y b. Más tarde, cuando se reanude la circulación, dicha arista formará parte de la ruta generada. Lo anterior se habrá conseguido sin haber alterado la naturaleza del grafo.
La manipulación de la lista de adyacencia y la matriz de distancias del grafo, permite tratar a los nodos y las aristas como objetos. Así la información adicional se puede agregar fácilmente como atributos de los objetos correspondientes. De la misma forma se incorpora información de los PA que se utilizan para determinar la ruta óptima. Con fines ilustrativos acerca de las restricciones que imponen los parámetros adicionales, se han contemplado algunos como ejemplo, a saber: altura máxima, tramos cortados por reparación y calles inundadas. La presencia o ausencia de dichas características modifica la ruta final obtenida. En el Capítulo 4 se detalla el manejo de PA.

3.4.5 Implementación de algoritmos de la ruta más corta

Con la finalidad de comparar resultados se han utilizado de manera adicional al algoritmo de Dijkstra, los algoritmos Bellman-Ford y Floyd-Warshall, para resolver el problema de la ruta más corta. Además, para ejemplificar los resultados obtenidos contemplando parámetros adicionales a la distancia, se ha contemplado también determinar la ruta más corta utilizando la API de Google Maps. La descripción detallada de la manipulación de PA y la ejecución de los algoritmos de la ruta más corta se aborda en el Capítulo 4.

Algoritmos de la ruta más corta

El problema de la ruta más corta es un clásico problema de optimización ampliamente estudiado y con múltiples aplicaciones en diversas áreas. En los grafos ponderados con frecuencia se desea encontrar la ruta más corta entre dos vértices dados. Para resolver este problema se ha propuesto un gran número de soluciones diferentes. En relación con el presente trabajo de investigación se han analizado los algoritmos existentes que son capaces de resolver el problema aquí planteado, a saber: Dijkstra, Bellman-Ford y Floyd-Warshall [8] y [2].

- **Algoritmo de Dijkstra**

  El algoritmo de Dijkstra resuelve con eficiencia el problema de la ruta más corta. Para la aplicación del algoritmo de Dijkstra se asume que los pesos son números positivos y que se quiere encontrar la ruta más corta del vértice a al vértice z. Sea $L(v)$ la etiqueta del vértice v. En cualquier punto, algunos vértices tienen etiquetas temporales y el resto son permanentes. Sea T el conjunto de vértices que tienen etiquetas temporales. Si $L(v)$ es la etiqueta permanente del vértice v, entonces $L(v)$ es la longitud de una ruta más corta de a a v. Al inicio, todos los vértices tienen etiquetas temporales. Cada iteración del algoritmo cambia el estado de una etiqueta de temporal a permanente; el algoritmo termina cuando z recibe una etiqueta permanente [8].

  El algoritmo 1 encuentra la longitud de una ruta más corta del vértice a al vértice z en un grafo ponderado conexo. El peso de la arista (i, j) es $w(i, j) > 0$, y la etiqueta del vértice es $L(x)$. Al terminar, $L(z)$ es la longitud de la ruta más corta de a a z.

  Versión extendida del algoritmo de Dijkstra. En la mayoría de las aplicaciones, además de encontrar la longitud de la ruta más corta, también se desea identificar la ruta más corta. Una ligera modificación al algoritmo 1 permite conseguirlo. Consiste en agregar a $L(v)$, el nombre del vértice previo en la ruta (predecesor), es decir, la etiqueta del vértice v es $L(v), v_a$, donde $L(v)$ es la longitud mínima de a a v, y v_a es el predecesor. Esto asegura por completo, que z (el nodo destino) tiene una etiqueta que indica la longitud mínima desde a y es posible moverse hacia atrás en las etiquetas para encontrar la ruta más corta. En adelante, nos referimos a esta modificación como la *versión extendida del algoritmo de Dijkstra*.

  Complejidad algorítmica. El algoritmo de Dijkstra, tiene en el peor de los casos, complejidad $O(n^2)$. La línea 5 se ejecuta $O(n)$ veces. Dentro del ciclo "while", la línea 9 toma un tiempo $O(n)$. El cuerpo del ciclo "for" (línea 12) toma un tiempo $O(n)$. Como las líneas 9 y 12 están anidadas dentro del ciclo

Algoritmo 1 Algoritmo de la ruta más corta de Dijkstra

Entrada: Un grafo ponderado conexo en el que todos los pesos son positivos; vértices a a z

Salida: $L(z)$, la longitud de la ruta más corta de a a z

```
1: DIJKSTRA(w,a,z,L)
2: {
3:    L(a) = 0
4:    for todos los vértices x ≠ a do
5:       L(x) = ∞
6:    end for
7:    T = Conjunto de todos los vértices  //T es el conjunto de todos los vértices
      cuyas distancias más cortas desde a no se han encontrado
8:    while z ∈ T do
9:       seleccionar v ∈ T con L(v) mínimo
10:      T = T − v
11:      for  cada x ∈ T adyacente a v do
12:         L(x) = min{L(x), L(v), +w(v, x)}
13:      end for
14:   end while
15: }
```

"while" (línea 8) que toma un tiempo $O(n)$, el tiempo total para las líneas 9 y 12 es $O(n^2)$. Entonces el algoritmo de Dijkstra corre en un tiempo $O(n^2)$

- **Algoritmo de Bellman-Ford**

  El algoritmo Bellman-Ford resuelve el problema de la ruta más corta en el caso general en el cual los pesos de las aristas pueden ser negativos. Dado un grafo dirigido ponderado $G = (V, E)$ con un nodo origen s y una función peso $w : E \to R$, el algoritmo Bellman-Ford retorna un valor booleano que indica si existe o no un ciclo de peso negativo que es accesible desde el nodo origen. Si tal ciclo existe, el algoritmo indica que no hay solución. Si no existe tal ciclo, el algoritmo genera las rutas más cortas y su pesos.

  El algoritmo relaja las aristas, disminuyendo progresivamente una estimación $v.d$ sobre los pesos de una ruta más corta desde el nodo origen s al vértice $v \in V$ hasta que se consiga el peso actual de ruta más corta $\delta(s, v)$. El algoritmo retorna TRUE si y solo si el grafo no contiene ciclos de peso negativo que son accesibles desde el origen.

Algoritmo 2 Algoritmo Bellman-Ford

Entrada: Un grafo ponderado dirigido $G = (V, E)$,una función de pesos $w : E \to R$ y un nodo origen s

Salida: Un valor booleano que indica si existe o no solución.

```
1: BELLMAN-FORD(G, w, s)
2: {
3:    for each vertex v ∈ V do
4:       d = ∞
5:       π = NULL
6:    end for
7:    d = 0
8:    for i := 1 to |V| − 1 do
9:       for each edge (u.v) ∈ G.E do
10:         RELAX(u,v,w);
11:      end for
12:   end for
13:   for each edge (u, v) ∈ G.E do
14:      if v.d > u.d + w(u, v) then
15:         return FALSE
16:      end if
17:   end for
18:   return TRUE
19: }
```

El proceso de **relajación** de una arista $(u.v)$ consiste en probar si podemos mejorar la ruta más corta para llegar a v a partir de u, si es así, se actualiza d y π. Un paso de relajación puede decrementar el valor de la ruta más corta estimada d y actualizar el atributo predecesor de v, π. El algoritmo 3 presenta un paso de relajación sobre la arista (u, v).

Algoritmo 3 Paso de relajación

Entrada: arista (u, v) y el peso $w(u, v)$

Salida: Posible actualización de la ruta más corta estimada d y del predecesor π.

```
1: RELAX(u,v,w)
2: {
3:    if v.d > u.d + w(u, v) then
4:       v.d = u.d + w(u, v)
5:       v.π = u
6:    end if
7: }
```

Complejidad algorítmica. El algoritmo Bellman-Ford se ejecuta en un tiempo $O(VE)$, ya que la inicialización en las líneas 3-7 toma un tiempo $O(V)$, cada una de las $|V| - 1$ iteraciones sobre las aristas en las líneas 8-12 toma un tiempo $O(E)$, y el ciclo for de las líneas 13-17 se ejecuta en tiempo $O(E)$.

- **Algoritmo de Floyd-Warshall**

  El algoritmo Floyd-Warshall encuentra la ruta más corta entre todos los pares de vértices de un dígrafo ponderado con pesos no negativos. La representación natural para un grafo en el algoritmo Floyd-Warshall es una matriz de adyacencia. Se asume que tenemos un grafo dirigido $G = (V, E)$ y que los vértices V, son numerados $1, 2, ..., n$. Más aun, tenemos una matriz $C[i, j]$ que nos informa el peso de la arista (i, j). Si no hay arista $C[i, j]$, entonces asumimos que $C[i, j]$ es infinito. Aunque en el grafo puede haber aristas de peso negativo , nosotros asumimos que no existen ciclos de peso negativo. Para la naturaleza de la presente investigación asumimos que todos pesos son no negativos [5].

  El algoritmo 4 calcula la matriz A, donde $A[i, j]$ es la longitud más corta del vértice i al vértice j.

  Complejidad algorítmica. El tiempo de ejecución del algoritmo Floyd-Warshall está determinado por los tres ciclos "for" anidados en las líneas 11-19. El algoritmo Floyd-Warshall calcula la matriz de distancias de las rutas más cortas entre todos los pares de vértices del grafo $G = (V, E)$ en un tiempo $O(n^3)$ y un espacio $O(n^2)$.

Algoritmo 4 Algoritmo Floyd-Warshall

Entrada: Un grafo ponderado conexo $G = (V, E)$, donde $v = 1, 2, ..., n$ y una matriz de pesos $C[i, j]$.

Salida: Un matriz de distancias $A[1..n, 1..n]$ donde $A[i, j]$es la distancia de la ruta más corta de i a j

```
1:  FloydWarshall(G)
2:  {
3:     for i := 1 to n do
4:        for j := 1 to n do
5:           A[i, j] := C[i, j];
6:        end for
7:     end for
8:     for i := 1 to n do
9:        A[i, i] := 0;
10:    end for
11:    for k := 1 to n do
12:       for i := 1 to n do
13:          for j := 1 to n do
14:             if A[i, k] + A[k, j] < A[i, j] then
15:                A[i, j] := A[i, k] + A[k, j];
16:             end if
17:          end for
18:       end for
19:    end for
20: }
```

CAPÍTULO 4

EXPERIMENTOS

4.1 Preliminares

Para los experimentos realizados se utilizó un equipo de cómputo con las siguientes especificaciones: Laptop marca Lenovo, modelo YOGA 510-14IKB, sistema operativo Windows 10 Home Single Language de 64 bits, procesador Intel(R) Core(TM) i7-7500U CPU 2.70 GHz 2.90 GHz, RAM 8.00 GB, Unidad de almacenamiento HDD 1.00 TB y procesador x64. De manera estratégica se han elegido rutas que pudieran comprometer al algoritmo o que ponen de manifiesto el manejo de diversos parámetros, específicamente aquellas que pudieran tener más de una ruta más corta, calles de un solo sentido y calles que presentan caos vial en horas pico.
La Tabla 4.1, muestra la distancia total (suma de los pesos) y los vértices recorridos de la ruta más corta entre los nodos origen y destino.

Tabla 4.1: Rutas generadas por el algoritmo de Dijkstra

No.	Src	Dst	Dist	Path (sin parámetros)	$Dist_p$	Path (con parámetros)
1	31	37	289	31-30-33-36-37	289	31-30-33-36-37
2	0	37	665	0-4-10-16-22-26-29-32-35-36-37	850	0-1-2-6-12-25-31-30-33-36-37
3	0	4	65	0-4	311	0-1-5-4
4	0	35	448	0-4-10-16-22-26-29-32-35	879	0-1-2-6-12-25-31-30-33-36-35
5	35	0	448	35-32-29-26-22-16-10-4-0	877	35-36-37-31-25-12-6-2-1-0
6	3	35	760	3-8-14-20-25-31-30-29-32-35	761	3-8-14-20-25-31-30-33-36-35
7	27	30	312	27-26-29-30	563	27-23-17-11-12-25-31-30
8	12	25	415	12-25-20-14-13	415	12-25-20-14-13

Se alcanzaron los resultados esperados, pues se encontró la ruta más corta en una sección de la ciudad considerando otros parámetros aparte de la distancia. El haber elegido una región específica para la implementación de este proyecto, permitió contemplar parámetros que los sistemas existentes no han considerado. Estos resultados han sido reportados en [13].

4.2 Finales

4.2.1 Recursos hardware y software utilizados

Tanto para la construcción de las estructuras de datos que representan el archivo, como para la programación y análisis de algoritmos se ha utilizado un equipo de cómputo con las siguientes características: Laptop marca Lenovo, modelo YOGA 510-14IKB, sistema operativo Manjaro Linux (Versión de Kernel 4.9.96-1-MANJARO) de 64 bits, procesador Intel(R) Core(TM) i7-7500U CPU 2.70 GHz 2.90 GHz, RAM 8.00 GB, Unidad de almacenamiento SSD 480 GB y procesador x64.

4.2.2 Características del grafo

El grafo resultante que representa la totalidad de calles de la ciudad de Martínez de la Torre, cuenta con las características que se describen a continuación:

- *Dirigido.* El sentido de las calles define la dirección de las aristas. Sea a y b dos nodos(intersección o limite de calle) conectados por una arista (calle) e . Las aristas (a, b) y (b, a) existen si la calle es doble sentido, en caso contrario solo una arista existe para calles de un solo sentido. Lo anterior define un grafo dirigido o dígrafo.

- *Ponderado.* El peso de las aristas corresponde a la distancia en metros entre un par de nodos. Esta condición define un grafo ponderado.

- *Sin pesos negativos.* El grafo no contiene pesos negativos debido a que los pesos son distancias.

- *Disperso.* Cada nodo tiene como máximo cuatro nodos adyacentes, por tanto el número de aristas es lejano al número máximo de aristas. Sea m el número de vértices y n el número de aristas se cumple que $m = O(n)$. Se trata entonces de un grafo disperso (no denso).

- *No es multígrafo.* Se presentan pocos casos de pares de nodos conectados por más de una arista. En estos casos, se ha agregado un nodo intermedio para descartar esta condición y de esta manera evitar modificaciones a los algoritmos utilizados que incrementan el costo computacional.

El grafo final está formado por un total de 3227 nodos ($|V| = 3227$). Resulta indispensable utilizar una lista de adyacencia y una matriz distancias (o de pesos) de manera conveniente para mejorar el rendimiento de los algoritmos. Una lista de adyacencia resulta útil cuando se analizan los nodos adyacentes, mientras que una matriz de distancias es requerida para consultar los pesos de las aristas.

4.2.3 Vértices, aristas y pesos

Los vértices del grafo representan intersecciones o límites de calles. Por ejemplo, en la Figura 4.1 los vértices b y c son intersecciones mientras que los vértices a y d corresponden a límites de calle. Las aristas son las calles que unen dichos vértices considerando el sentido de las mismas. Algunas de las aristas de la Figura 4.1 son: (a, b), (b, a), (b, c), (c, d) y (d, c). Aquí, la flecha en las calles define el sentido y una calle sin flecha es ambos sentidos.
Un caso especial se presenta cuando dos nodos están comunicados por dos calles (múltiples aristas), lo cual genera un multígrafo que obliga a modificar los algoritmos

Figure 4.1: Vértices: intersecciones y límites de calles

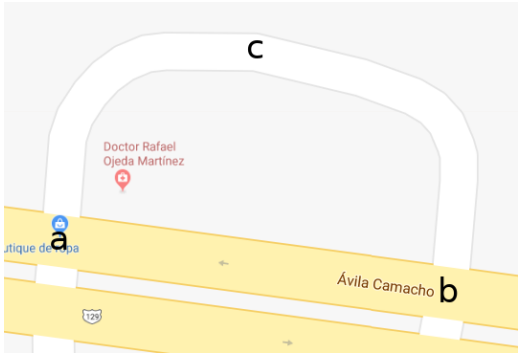

Figure 4.2: Vértice que elimina aristas paralelas

añadiendo un costo computacional. La solución para evitar esto es agregar un tercer nodo de manera conveniente como se ilustra en la Figura 4.2. Aquí, sin considerar el nodo c, los nodos b y a están conectados por dos aristas (paralelas). Al agregar c, se consigue eliminar una de las aristas paralelas quedando las siguientes: (a, c), (c, a), (b, c), (c, b) y (b, a).

Los pesos de las aristas han sido calculados de dos maneras: mediante la fórmula de Haversine y por medio de la API DistanceMatrix. La fórmula de Haversine (descrita en 2.4) calcula la distancia "en línea recta"[1] entre dos puntos geográficos (nodos). La desventaja de este método es que la distancia calculada entre nodos conectados por calles o carreteras que no son líneas rectas, es una aproximación lejana a la realidad. La segunda opción utiliza la API DistanceMatrix, un servicio que proporciona tiempos y distancias de viaje para un origen y un destino. La API retorna información calculada mediante la API Google Maps. Resulta evidente

[1]Estrictamente, se conoce como la distancia del círculo máximo entre dos puntos de un globo[15]

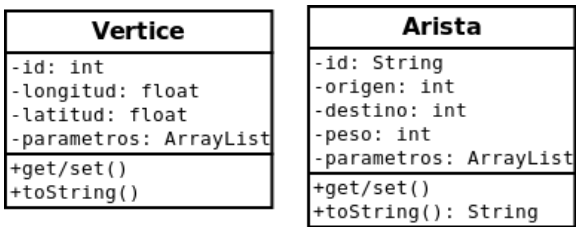

Figure 4.3: Diagrama de las clases Vertice y Arista

que la opción más apropiada es la segunda, sin embargo, tiene como desventaja las restricciones y costos impuestos por Google.
Se ha construido una matriz para cada una de las opciones descritas anteriormente. En ambos casos, los pesos de las aristas son indicados en metros.

4.2.4 Programación

Los elementos base para la construcción de las estructuras de datos que representan el grafo en la memoria de la computadora son los vértices y aristas. Un vértice se define por un identificador o nombre, las coordenadas GPS (longitud y latitud) y una lista de parámetros. Una arista está formada por un identificador, un nodo origen, un nodo destino, el peso o distancia y una lista de parámetros. La Figura 4.3 muestra los diagramas de las clases **Vertice** y **Arista**. En ambos diagramas utilizamos, a manera de abreviatura, la expresión *get/set()* para indicar que existen los correspondientes métodos *get* y *set* por cada uno de los atributos de la clase.
Un arreglo bidimensional de enteros representa la matriz de pesos de nuestro grafo ponderado. Se trata de una matriz $A[i, j]$ cuadrada de 3267x3267. Los índices corresponden con el identificador de cada uno de los vértices. Cada elemento a_{ij} puede ser, o bien -1 si no existe la arista (vi_i, v_j), o bien la distancia en metros de la arista (i, j).

La Figura 4.4 muestra la matriz de pesos final, que representa el grafo ponderado de la ciudad de Martínez de la Torre, Veracruz. Aquí, cuando $A[i, j] = -1$ no existe la arista (v_i, v_j). Por otro lado, cuando la arista existe, $A[i, j] = w$, donde w es la distancia en metros entre los vértices v_i y v_j.
La consulta de los pesos de las aristas resulta directa utilizando la matriz de pesos. Para la arista (v_i, v_j), $w_{ij} = A[i, j]$. Sin embargo, al tratarse de un grafo disperso se provoca un desperdicio en la memoria de la computadora. También, para consultar los pesos de todos los vértices adyacentes al vértice v_i, se debe iterar de 0 a $|V| - 1$ como lo indica el algoritmo 5. Lo anterior, solo para encontrar como máximo un total de 4 vértices adyacentes, requiere un tiempo $O(V)$.

Vértice origen \ Vértice destino	0	1	2	3	4	5	6	7	8	9	10	11	...	3266
0	-1	66	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	...	$a_{0,3266}$
1	66	-1	95	-1	642	-1	-1	-1	-1	-1	-1	-1	...	$a_{1,3266}$
2	-1	95	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	...	$a_{2,3266}$
3	-1	-1	-1	-1	64	-1	-1	-1	-1	-1	-1	-1	...	$a_{3,3266}$
4	-1	642	-1	64	-1	321	-1	-1	-1	-1	-1	-1	...	$a_{4,3266}$
5	-1	-1	-1	-1	321	-1	139	-1	-1	-1	-1	-1	...	$a_{5,3266}$
6	-1	-1	-1	-1	-1	139	-1	-1	-1	-1	-1	-1	...	$a_{6,3266}$
7	-1	-1	-1	-1	-1	-1	-1	-1	394	-1	-1	-1	...	$a_{7,3266}$
8	-1	-1	-1	-1	-1	-1	-1	394	-1	213	-1	-1	...	$a_{8,3266}$
9	-1	-1	-1	-1	-1	-1	-1	-1	213	-1	-1	-1	...	$a_{9,3266}$
10	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	...	$a_{10,3266}$
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	...	$a_{1,3266}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮
3266	$a_{3266,0}$	$a_{3266,1}$	$a_{3266,2}$	$a_{3266,3}$	$a_{3266,4}$	$a_{3266,5}$	$a_{3266,6}$	$a_{3266,7}$	$a_{3266,8}$	$a_{3266,9}$	$a_{3266,10}$	$a_{3266,11}$	...	$a_{3266,3266}$

Figure 4.4: Matriz de distancias final

Algoritmo 5 Iteración para consultar pesos de aristas adyacentes

```
1: for j := 0 to |V| − 1 do
2:     if A[i, j]! = −1 then
3:         w_ij = A[i, j]
4:     end if
5: end for
```

Una solución más apropiada que elimina las desventajas de la matriz de pesos descritas anteriormente, es la representación del grafo mediante una lista de adyacencia. La Figura 4.5 muestra la lista de adyacencia que representa el grafo. Utiliza un arreglo de tamaño 3267 para almacenar los vértices. Cada vértice apunta hacia una lista enlazada que almacena los nodos adyacentes y que tendrá como máximo 4 elementos. Esto garantiza que se reserva exactamente la cantidad de memoria requerida. Además, permite consultar nodos adyacentes de manera directa eliminando el costo computacional que implica el uso de la matriz de pesos descrito en el algoritmo 5.
Los algoritmos Dijkstra, Bellman-Ford y Floyd-Warshall se han programado en Java. El grafo final, representado por la matriz de pesos, se almacena para su lectura en un archivo CSV. Cada vez que el programa se ejecuta se crean en memoria las estructuras de datos descritas anteriormente mediante la lectura de dicho archivo.

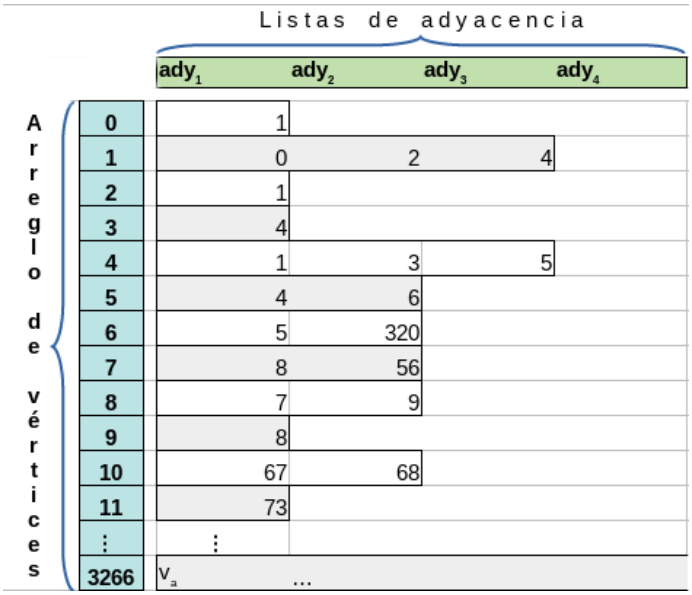

Figure 4.5: Lista de adyacencia

4.2.5 Manipulación de parámetros adicionales

Está claro que para determinar la ruta óptima no es suficiente calcular la ruta más corta en términos de distancia. Existen múltiples factores o atributos, además de la distancia, que influyen en la selección de la ruta óptima. Hemos denominado a estos factores parámetros adicionales. Los PA pueden pertenecer a las aristas o a los vértices. Podemos identificar claramente dos grupos de parámetros: a) aquellos que impiden la circulación a través de un vértice o de una arista y, b) aquellos que modifican el peso de una arista.

Nos referimos como **parámetros booleanos** al primer grupo. Algunos ejemplos son: tramos cortados, encharcamientos durante temporada de lluvias que impiden la circulación, altura máxima de vehículos, acceso restringido al transporte pesado, amplitud de las calles, entre otros. Denominamos **parámetros numéricos** al segundo grupo. Forman parte de este grupo: el flujo vehicular, nivel de inseguridad, inclinación de las calles o carreteras, tiempo de espera en semáforos, cantidad de topes, cruces peatonales (por ejemplo en escuelas), etc.

En el presente trabajo de investigación se aborda la manipulación de los parámetros booleanos. Para explicar porque este tipo de parámetros impiden la circulación a través de la arista a la que pertenecen, tomaremos como ejemplos dos de ellos: tramo cortado y altura máxima. Se puede encontrar un tramo cortado por reparación, bloqueado por algún accidente o por manifestación, entre otras causas. Un ejemplo de restricción por altura máxima puede tratarse de una calle que pasa debajo de un puente. Tal es el caso de la Avenida Pedro Belli a la altura del Palacio Municipal

Figure 4.6: Altura máxima para circulación en Av. Pedro Belli

que permite la circulación debajo del puente a vehículos con una altura no mayor a 2.2 m. como muestra la Figura 4.6.
Este tipo de parámetros representan restricciones que permiten o impiden la circulación a determinados vehículos. De ahí que se denominen parámetros booleanos.
Modificar el grafo para eliminar la arista no representa la naturaleza del problema, dado que la arista existe aunque la circulación, bajo ciertas condiciones, no sea posible. Por lo tanto, para la manipulación de cada parámetro booleano se verifica la disponibilidad de la arista antes de determinar la ruta más corta en cada paso de relajación de los algoritmos. Cabe mencionar que el paso de relajación (mostrado en 3) es utilizado en los tres algoritmos: Dijkstra, Bellman-Ford y Floyd-Warshall. La solución se muestra en el algoritmo 6. Aquí la función $estaAristaDisponible()$ retorna un valor booleano y se encarga de verificar la disponibilidad de la vialidad (arista) de acuerdo a los parámetros contemplados. Ya que cada algoritmo establece $w = \infty$ para indicar que no hay arista entre ciertos nodos, hemos realizado la misma asignación para indicar que a pesar de que hay arista, esta no se encuentra disponible.

Algoritmo 6 Paso de relajación con verificación de disponibilidad de la arista

Entrada: arista (u, v) y el peso $w(u, v)$
Salida: Posible actualización de la ruta más corta estimada d y del predecesor π.

```
1: RELAX(u,v,w)
2: {
3:    if ¬estaAristaDisponible(u, v) then
4:       w = ∞
5:    end if
6:    if v.d > u.d + w(u, v) then
7:       v.d = u.d + w(u, v)
8:       v.π = u
9:    end if
10: }
```

CAPÍTULO 5

RESULTADOS Y ANÁLISIS

5.1 Rutas analizadas

Como se propuso inicialmente, la construcción del grafo y la manipulación de diversos parámetros mediante algoritmos de la ruta más corta, permiten encontrar la ruta óptima en un entorno local en la ciudad de Martínez de la Torre.
La Tabla 5.1 muestra información de diversas rutas con las cuales se pusieron a prueba los algoritmos. La columna *Ruta* es el número que identifica a la ruta. Las columnas *Origen* y *Destino* se refieren al ID (en el grafo) de los nodos origen y destino respectivamente. Las columnas *O(Latitud, Longitud)* y *D(Latitud, Longitud)* indican las coordenadas GPS de los puntos origen y destino.

Tabla 5.1: Detalle de las rutas de prueba

Ruta	Origen	O(Latitud, Longitud)	Destino	D(Latitud, Longitud)
R1	1696	(20.066832,-97.059322)	1719	(20.065563,-97.057522)
R2	2113	(20.061416,-97.053959)	2358	(20.060086,-97.056364)
R3	195	(20.081004,-97.069385)	2392	(20.058190,-97.051965)
R4	53	(20.085665,-97.098550)	151	(20.081458,-97.081019)
R5	477	(20.078131,-97.079507)	392	(20.080620,-97.032064)
R6	2904	(20.050179,-97.093920)	2200	(20.060544,-97.086212)
R7	169	(20.079601,-97.075671)	2594	(20.055116,-97.084513)
R8	348	(20.082197,-97.035893)	1841	(20.069135,-97.041891)
R9	2537	(20.058625,-97.030857)	1188	(20.073574,-97.064780)
R10	2579	(20.054482,-97.087911)	2537	(20.058625,-97.030857)
R11	1236	(20.070124,-97.062706)	1634	(20.068539,-97.061707)
R12	1737	(20.065261,-97.056234)	2094	(20.064876,-97.054982)

5.2 Tiempos de ejecución

La Tabla 5.2 muestra los tiempos de ejecución, en milisegundos, de cada uno de los algoritmos. Para cada algoritmo se ha obtenido el tiempo promedio de un total de 10 ejecuciones por ruta. Se han ejecutado los tres algoritmos para cada ruta de las mencionadas en la Tabla 5.1. La columna *Ruta* indica el número de ruta. Los algoritmos se han ejecutado sobre ambas matrices de pesos: la obtenida mediante la formula de Haversine (MPH) y la que se obtuvo mediante la API DistanceMatrix de Google Maps (MPG). Las columnas *Distancia(H)* y *Distancia(G)* corresponden a la distancia (en metros) obtenida utilizando las matrices MPH y MPG, respectivamente. La columna *Nodos* se refiere al total de nodos que forman la ruta. Las columnas T-D, T-BF y T-FW indican los tiempos promedio de las ejecuciones de los algoritmos de Dijkstra, Bellman-Ford y Floyd-Warshall respectivamente.
Debido a que ambas matrices son del mismo orden, los tiempos promedio de ejecución solo se han obtenido con la matriz MPG, ya que presenta mayor precisión en las distancias.

Tabla 5.2: Tiempos de ejecución de los algoritmos

Ruta	Distancia(H)	Distancia(G)	Nodos	T-D	T-BF	T-FW
R1	312	313	4	3	20373	62993
R2	292	291	4	2	20350	61653
R3	3415	3547	51	20	38762	61828
R4	1892	2222	2	1	20434	62569
R5	6125	6236	73	25	20057	66613
R6	1654	1663	21	5	19965	66065
R7	3937	3961	44	20	38988	63887
R8	2037	2041	29	7	39073	65108
R9	4625	4790	60	23	48605	65698
R10	7025	7027	76	27	48005	63311
R11	205	206	8	3	48707	62905
R12	199	199	3	3	49096	63002

Los resultados mostrados en la Tabla 5.2 se visualizan en la Figura 5.1. El eje horizontal indica las rutas y el eje vertical muestra los tiempos de ejecución en milisegundos. Nótese que no se visualizan las barras de tiempos del algoritmo de Dijkstra. Esto no es un error, por el contrario pone de manifiesto la inmensa diferencia de tiempo en comparación con los algoritmos Bellman-Ford y Floyd-Warshall. Los tiempos del algoritmo FW se mantienen constantes relativamente, debido a que en cada ejecución se calculan las rutas más cortas de todos los pares de nodos en el grafo. Cabe recordar que una primera ejecución del algoritmo FW sería suficiente para calcular la matriz de rutas y la matriz de distancias de las rutas más cortas de todos los pares de nodos, de tal manera una vez construidas dichas matrices las consultas de rutas y distancias serian directas. Esto representa una solución apropiada para aplicaciones que no presentan continuamente cambios en la topología del grafo. Por su parte, el algoritmo de BF será de utilidad cuando el grafo contiene pesos negativos.

5.3 Métodos relevantes

En general, la programación de los algoritmos esta definida por tres métodos:

1. *CrearMatriz (CM)*. Este método crea un arreglo bidimensional que representa la matriz de distancias. La información es leída de un archivo CSV que contiene MPG.

2. *EjecutarAlgoritmo (EA)*. Se encarga de la ejecución del algoritmo que calcula la ruta más corta y la distancia. Para el caso de los algoritmos Dijkstra y Bellman-Ford el resultado está contenido en un objeto "ArrayList". En el caso del algoritmo Floyd-Warshall las rutas más cortas y las distancias entre todos los pares de nodos se almacenan en arreglos bidimensionales.

Figure 5.1: Tiempos de ejecución de los algoritmos

Tabla 5.3: Tiempos de ejecución por métodos

Metodo	Dijkstra	Bellman-Ford	Floyd-Warshall
CrearMatriz	6,572 (94.43%)	7,019 (9.82%)	7,766 (4.79%)
EjecutarAlgoritmo	89.6 (1.29%)	64,279 (89.89%)	154,004 (94.99%)
GenerarRuta	4.56 (0.07%)	0.518 (0.00%)	9.55 (0.01%)
Otros	293.84 (4.22%)	213.482 (0.30%)	350.45 (0.22%)

3. *GenerarRuta (GR).* Se encarga de la generación de las rutas y distancias a partir de las lectura de las estructuras de datos que se generan en los métodos "EjecutarAlgoritmo".

La Tabla 5.3 muestra los tiempos de ejecución de cada uno de los métodos, para los programas que implementan los algoritmos. Para tal medición se utilizó la herramienta "Profile Project" de NetBeans. De las rutas analizadas se ha utilizado $R10$ que tiene mayor cantidad de nodos, mayor distancia y uno de los mayores tiempos de ejecución, de acuerdo a los resultados mostrados en la Tabla 5.2.
La Figura 5.2 muestra la relación porcentual de cada uno de los métodos para cada algoritmo de manera independiente. Entre paréntesis se muestra el tiempo total (en milisegundos) de la ejecución del programa para calcular la ruta.
La Figura 5.3 toma los datos de la Tabla 5.3 para representar una comparación de los porcentajes obtenidos en la ejecución de los métodos entre los diferentes programas. Aquí podemos observar que el programa que implementa el algoritmo de Dijkstra invierte la mayor parte de su tiempo de ejecución en la construcción de la matriz y la generación de la ruta. En contraste, los programas que implementan BF y FB invierten la mayoría de tiempo en la ejecución del algoritmo. Nuevamente, los resultados también ponen de manifiesto la enorme diferencia entre el tiempo de

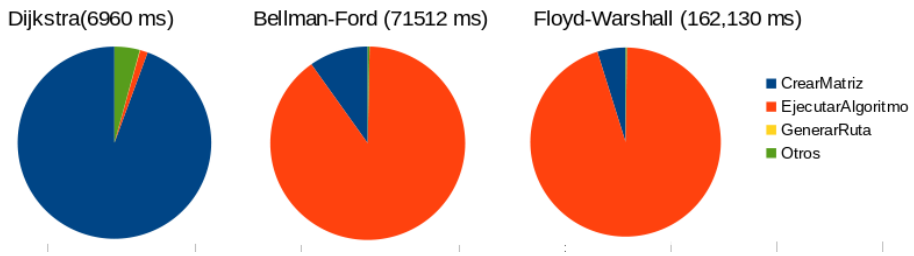

Figure 5.2: Tiempos de ejecución por métodos (relación porcentual)

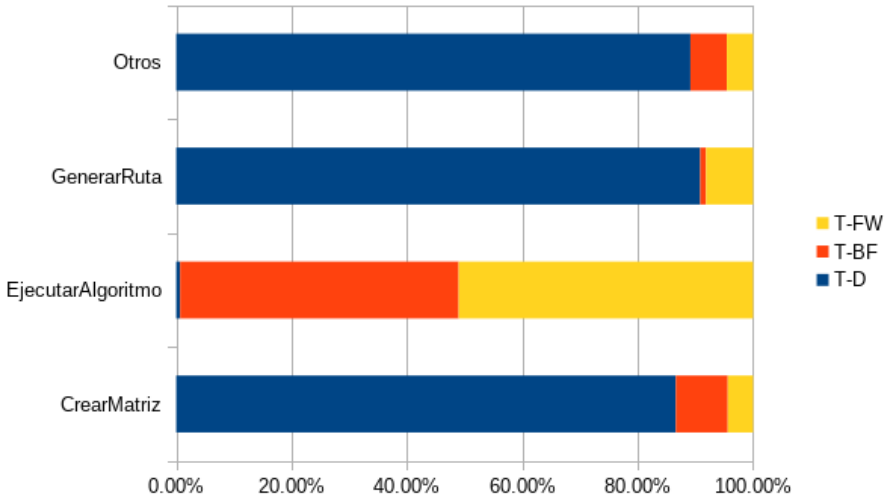

Figure 5.3: Comparativa tiempos de ejecución (métodos y programas)

ejecución del algoritmo de Dijkstra en contraste con BF y FW.

5.4 Generación de rutas locales

Maps, la aplicación oficial de Google, es una de las más utilizadas por los viajeros. Proporciona mucha información y su uso es ideal cuando viajamos por una ciudad desconocida. Sin embargo, en ocasiones la información proporcionada no se encuentra actualizada y esta situación permanece hasta que algún usuario la reporta (si es que sucede). Tal es el caso del tramo cortado en el Boulevard Alfinio Flores (Figura 5.4), desde el cruce con la calle Lic. Manuel Mateos, y hasta el cruce con la calle Francisco Javier Miranda, dos cuadras donde la circulación está interrumpida por encontrarse en reparación. El día 31 de mayo de 2018 se realizó una generación de ruta a través de la aplicación Maps y la ruta obtenida sugiere la circulación a través de dicho tramo. Esto se debe, como se mencionó anteriormente, a que no se reporta de manera oportuna la información necesaria. Es lógico que se presenten este tipo

Figure 5.4: Tramo cortado por reparación

Tabla 5.4: Rutas obtenidas en tramo en reparación: Blvd. Alfinio Flores

Origen	Destino	D. Maps	R. Maps	D. Maps	R. Solución propia
1	7	300 m.	1-2-3-4-5-6-7	350 m.	1-8-9-11-7

de inconvenientes debido a la cobertura global de la aplicación. Utilizar Maps para configuración y generación de rutas, presenta tal desventaja.

La Figura 5.5 muestra la ruta (en azul) generada por Google Maps en el caso descrito anteriormente. Los tramos cortados por reparación corresponden a las aristas $(3, 4)$, $(4, 5)$ y $(5, 6)$. Como puede observarse, la ruta sugerida, permite el acceso a través de dichas aristas.

Con una cobertura local es más sencillo el control y la actualización de la información de la ciudad relacionada con aquellos factores que intervienen en la generación de rutas. La Tabla 5.4 muestra una comparación de la rutas generadas por Maps, en contraste con el resultado obtenido a través de la solución propuesta mediante la manipulación del PA "tramo cortado". Los nodos y rutas corresponden al grafo de la Figura 5.5.

La situación descrita anteriormente puede generar problemas derivados de la no actualización oportuna de la información. Por ejemplo, generación de rutas erróneas, cuellos de botella de tráfico, necesidad de tomar retornos como rutas alternas con un incremento del costo en la ruta, entre otros.

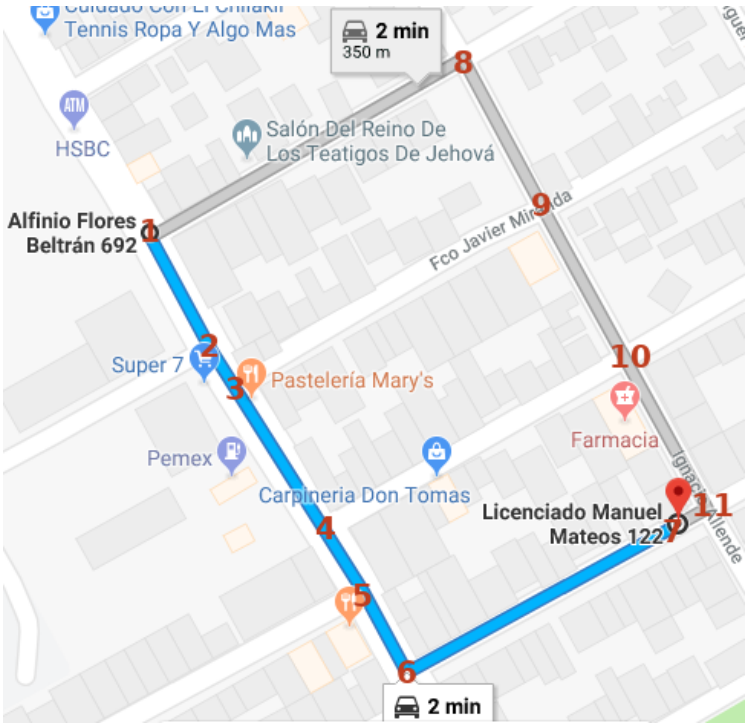

Figure 5.5: Ruta errónea (tramo cortado por reparación)

CAPÍTULO 6

CONCLUSIONES Y TRABAJO FUTURO

Con respecto a las hipótesis planteadas se concluye lo siguiente:
H1: Los algoritmos de Dijkstra, Bellman-Ford y Floyd-Warshall han sido utilizados para encontrar la ruta más corta y además, mediante la manipulación de PA a la distancia entre nodos, se ha conseguido determinar la ruta óptima. El algoritmo de Dijkstra es el más eficiente, sin embargo Floyd-Warshall también presenta ventajas en casos donde la topología del grafo no cambia constantemente. Bellman-Ford sera indispensable en caso de que existan pesos negativos en el grafo.
H2: Se ha logrado construir el grafo completo de la ciudad de Martínez de la Torre, Veracruz. Para la consulta de pesos de las aristas resulta conveniente el uso de una matriz de pesos, mientras que para determinar nodos adyacentes la mejor opción es una lista de adyacencia. Las estructuras de datos construidas se manipulan en un tiempo razonable mediante los algoritmos.
Los resultados obtenidos pueden ser utilizados para el desarrollo de un sistema computacional que resultará útil para las distintas organizaciones que enfrentan el problema de hallar la ruta óptima.
Diversas aplicaciones reales podrían implementarse tomando como base los resultados obtenidos en el presente trabajo de investigación y, de esta manera, representar una aplicación directa en diversos sectores de la sociedad. Las siguientes propuestas han de considerarse como trabajo futuro:

- Sistema local para generación de rutas para usuarios en general.
- Aplicación en redes de reparto y distribución locales.
- Sistema de apoyo a la logística en el servicio de taxis.

Para una mayor funcionalidad y para facilitar las condiciones para operar de manera dinámica, es posible también utilizar tecnología GPS.

Bibliografía

[1] Balabanian, N., Bickart, T. A., and Seshu, S. *Teoría de redes eléctricas*. Editorial REVERTÉ, S. A., 2007. 12

[2] Cormen, T. H., Leiserson, C. E., Rivest, R., and Stein, C. *Introduction to Algorithms*, third ed. Massachusetts Institute of Technology, 2009. 25

[3] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271. 16

[4] Drozdeck, A. *Estructuras de datos y algoritmos en Java*. Thomson, 2007. 10

[5] Gross, J., and Yellen, J. *Handbook of graph theory*. CRC Press, 2003. 28

[6] Guzmań, J., Arango, R., and Jimeneź, L. Buśqueda de la ruta oṕtima mediante los algoritmos: genet́ico y dijkstra utilizando mapas de visibilidad. *Scientia et Technica Año XVII* (2017). 17

[7] Hussain, I., and Bingcai, C. Cluster formation and cluster head selection approach for vehicle ad-hoc network (vanets) using k-means and floyd-warshall technique. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS 8*, 12 (2017), 11–15. 17

[8] Johnsonbaughc, R. *Matemáticas discretas*. Pearson Educación, 2005. 2, 25

[9] José L. Galán-García, Gabriel Aguilera-Venegas, M. A. G.-G. A new probabilistic extension of dijkstra's algorithm to simulate more realistic traffic flow in a smart city. *ELSEVIER* (2014). 4, 16

[10] Karakatic, S., and Podgorolec, V. A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing* (2014). 17

[11] Leonardo Lozano, A. L. M. On an exact method for the constrained shortest path problem. 17

[12] Lipschutz, S., and Lipson, M. *Matemáticas discretas.* McGraw-Hill/Interamericanda Editores, S.A. de C.V., 2009. 12

[13] Lucas, H., Sanchez, E., and Biswal, R. Transport routes optimization in martinez de la torre, veracruz city, using dijkstraś algorithm with an additional parameter. *Research in Computing Science* (2017). VII, 21, 31

[14] Purvis, M., Sambells, J., and Turner, C. *Beginning Google maps applications with PHP and Ajax.* Springer, 2006. 15

[15] Sanchez, J., and Canton, M. P. *Space Image Processing.* CRC Press LLC, 1999. 33

[16] Shu-Xi, W. The improved dijkstra's shortest path algorithm and its application. *IWIEE* (2012). 16

[17] Strehler, M., Merting, S., and Schwan, C. Energy-efficient shortest routes for electric and hybrid vehicles. *Transportation Research Part B: Methodological 103* (2017), 111–135. 16

[18] Villalpando, J. F., and García, A. *Matemáticas discretas, aplicaciones y ejercicios.* Grupo Editorial Patria, 2014. 10

[19] Winston, W. L. *Investigación de operaciones Aplicaciones y algoritmos.* CENGAGE Learning, 2005. 2

[20] Zou, B., Hu, J., Wang, Q., and Ke, G. A distributed shortest-path routing algorithm for transportation systems. 17