Universidad Autónoma de Querétaro
Facultad de Informática
Ingeniería en Computación

Modelado y Simulación de Sistemas de Transporte Público:
Un caso de estudio aplicado a la Ciudad de Querétaro

TESIS

que como parte de los requisitos para obtener el título de

Ingeniero en Computación

Presenta:
Edwin Paul Vega Escobedo

Dirigida por:
Dr. Arturo González Gutiérrez

SINODALES

Dr. Arturo González Gutiérrez _____
Presidente

Dr. Jaime Rangel Mondragón _____
Secretario

M. C. Fidel González Gutiérrez _____
Vocal

M. C. Guillermo Díaz Delgado _____
Suplente

Ing. Ernesto Ignacio Espinosa Chávez _____
Suplente

_____
M. C. Ruth Angélica Rico Hernández
Directora de la Facultad

Campus Juriquilla
Querétaro, Qro.
21 de Marzo del 2012
México

RESUMEN

El creciente nivel de complejidad de los modernos sistemas de transporte público (STP) hace que su utilización por el usuario del servicio de transporte sea cada vez más sofisticada. Para tratar con tal complejidad y hacer el uso de tales sistemas más eficiente, los STP pueden modelarse mediante grafos dirigidos conexos, los cuales conforman la estructura de datos subyacente sobre la que algoritmos sofisticados operan. Dichos algoritmos son diseñados para responder a peticiones de los usuarios que requieren planear recorridos, a fin de alcanzar un punto destino en el STP, desde un punto inicial, conforme a variados criterios de optimización y restricciones impuestas *a priori* por el usuario. A pesar de que la ciudad de Querétaro, en particular, ha experimentado un importante crecimiento poblacional en los últimos 20 años, no cuenta con un sistema de información completo, fácil de usar que soporte consultas para el uso eficiente de su STP. Por ello, en esta tesis se presentan tanto un modelo computacional basado en el lenguaje de programación funcional de alto nivel *Mathematica*, como un prototipo basado en Web en el contexto del ambiente provisto por *Google Earth* del STP de la ciudad de Querétaro, como un caso de estudio. Tanto el modelo como el prototipo permiten al usuario del servicio de transporte construir el itinerario óptimo como una sucesión de puntos, cada uno consistente en un sitio –a menudo una intersección de más de una ruta del STP- y una lista de identificadores de ruta del sistema que puede el usuario opcionalmente abordar para alcanzar el punto destino.

(**Palabras clave**: Algoritmo de Dijkstra, Teoría de grafos, Sistemas de Transporte Público, Planeación de Itinerarios)

SUMMARY

The increasing level of complexity of modern public transport systems (PTS) makes their use increasingly sophisticated. To deal with such complexity and make use of such systems more efficiently, the PTS can be modeled by connected and directed graphs, which conform the subjacent data structure on which sophisticated algorithms run. These algorithms are designed to respond queries from users who require planning trips, in order to reach a destination point from a starting point through the PTS, according to various optimization criteria and restrictions imposed by the user *a priori*. Although the city of Queretaro, in particular, has experienced significant population growth in the last 20 years, it lacks a comprehensive information system, easy to use that supports queries for efficient use of the STP. Thus, we present in this thesis a computational model based on the high level functional programming language of *Mathematica*, as well as a Web-based prototype in the context provided by the environment of Google Earth, as a case study applied to the city of Queretaro. The model and prototype allow the user to build the optimum itinerary as a sequence of points, each one consisting of a site –more often as an intersection of more than one routes of the STP- and a list of route identifiers of the system that the user can optionally takes to reach the destination point.

(**Key words**: Dijkstra algorithm, Graph Theory, Public Transport Systems, Traveling Planning)

AGRADECIMIENTOS

Expreso mi agradecimiento a mi director de tesis, el Dr. Arturo González Gutiérrez por el apoyo otorgado desde el principio y al haberme alentado a realizar este trabajo. A los profesores del Cuerpo Académico: Algoritmos, Optimización y Redes, el Dr. Jaime Rangel Mondragón y el M.C. Guillermo Díaz Delgado por sus minuciosas revisiones. Al M.C. Fidel González Gutiérrez y al compañero Pedro Ayala Elizarraráz, por su contribución al proyecto durante el Verano de la Ciencia 2011. Al Ing. Ernesto Ignacio Espinosa Chávez por sus asesorías en la elaboración del prototipo basado en web.

A mis padres, que siempre me han apoyado durante mi etapa de estudio y me han enseñado a ir más allá de mis objetivos.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

ACRÓNIMOS

Término	Definición
3G	*3rd generation mobile telecommunications*. Tercera generación de estándares para telecomunicaciones.
AJAX	*Asynchronous JavaScript and XML*. JavaScript asíncrono y XML.
API	*Aplication Programming Interface*. Interfaz de programación de aplicaciones.
CSS	*Cascading Style Sheets*. Hojas de estilo en cascada.
DOM	*Document Object Model*. Modelo de objetos del documento.
GPS	*Global Positioning System.* Sistema de Posicionamiento Global.
HTML	*HyperText Markup Language*. Lenguaje de marcado de hipertexto.
JSON	*JavaScript Object Notation*. Notación de objeto *JavaScript*.
JSP	*JavaServer Pages*. Páginas *JavaServer* .
KML	*Keyhole Markup Language*. Lenguaje de marcado de *Keyhole*.
OGC	*Open Geospatial Consortium.* Consorcio Geoespacial abierto.
STP	Servicio de Transporte Público.
SSC	Secretaría de Seguridad Ciudadana del Estado de Querétaro.
WGS84	*World Geodetic System 84*. Sistema Geodésico Mundial 1984.
WiFi	*Wi-Fi Alliance*. Alianza *Wi-Fi*.
XML	*Extensible Markup Language*. Lenguaje de marcado extensible.

GLOSARIO

Término	Definición
API	Conjunto de estructuras de datos, funciones y herramientas utilizadas por otras aplicaciones como una interfaz para comunicarse entre sí.
AJAX	Forma de realizar intercambio de datos del lado del cliente a un servidor
Cluster	Conjunto, cúmulo, grupo.
CSS	Hojas de estilo usadas para describir la presentación de un documento escrito en *HTML* o *XML*.
DOM	Estándar que define la manera de accesar y manipular elementos de un documento *HTML*.
HTML	Lenguaje para definir páginas web.
Interfaz	Conexión funcional entre dos sistemas independientes.
Java Servlet	Objeto que se ejecuta en un servidor Java.
JSON	Formato basado en texto para el intercambio de datos.
JSP	Tecnología basada en *Java* para la creación de contenido web dinámico.
KML	Tipo de archivo utilizado para desplegar datos geográficos.
OGC	Consorcio que define estándares abiertos para Sistemas de Información Geográfica.
Punto de ajuste	Puntos colocados en las calles curvas para el proceso de linealización de los recorridos.
Punto de inflexión	Punto donde la unidad de transporte hace un quiebre para incorporarse a otra calle o avenida.
Punto de intersección	Punto donde los recorridos de una o más rutas se intersectan.
Punto intermedio	Puntos marcados en tramos muy largos de avenidas, los cuales representan los sitios donde existe ya sea un ascenso o un descenso de usuarios. Evitan que el usuario tenga que llegar hasta el próximo punto de inflexión o intersección.

Smartphone	Teléfono inteligente. Dispositivo electrónico de telefonía celular que posee características y funciones similares a las de una computadora.
Tablet	Computadora portátil de pantalla táctil cuyo tamaño es un poco mayor al de un *smartphone*
Transporte privado	Servicio de transporte que no depende de horarios ni de rutas establecidas. Este tipo de transporte no necesariamente está disponible para todo público.
Transporte púbico	Servicio de transporte colectivo que está sujeto a rutas y horarios.
Traza de ruta	Recorrido descrito por una unidad de transporte sobre un mapa.
WGS84	Sistema estándar de coordenadas geográficas utilizado en cartografía, geodesia y navegación.
Wi-Fi	Organización comercial que certifica a los equipos que cumplen los estándares relacionados a redes locales inalámbricas.
XML	Lenguaje para la estructuración de datos en texto.

1. INTRODUCCIÓN

"El hombre está dispuesto siempre a negar todo aquello que no comprende."
Blaise Pascal(1623 - 1662)

Los problemas relacionados con la obtención de la ruta más corta están entre los más estudiados sobre optimización de flujo en redes, con interesantes aplicaciones en varios campos. Uno de estos campos son las redes de transporte.

El transporte público es el medio más usual para llevar a cabo el traslado de un gran número de personas de un lugar a otro dentro de una ciudad. La mayoría de las ocasiones el transporte público es elegido ya sea por su bajo costo comparado con adquirir y mantener un auto, ya sea por el bajo impacto ambiental. A pesar de sus ventajas, el Servicio de Transporte Público (STP) de la Ciudad de Querétaro es percibido por los usuarios como un sistema complejo, obsoleto e inflexible, en el cual no se cuenta con horarios definidos que establezcan la frecuencia de paso de cada ruta, ni con información detallada sobre los recorridos de las rutas.

De acuerdo a los datos publicados por el Centro Queretano de Recursos Naturales (Centro Queretano de Recursos Naturales, 2011), la Ciudad de Querétaro ha venido experimentando un crecimiento territorial muy pronunciado (Figura 1.1). Este crecimiento ha demandado grandes inversiones en infraestructura de vivienda y de servicios; particularmente, las vías de

comunicación para el transporte público y privado han requerido una particular atención.

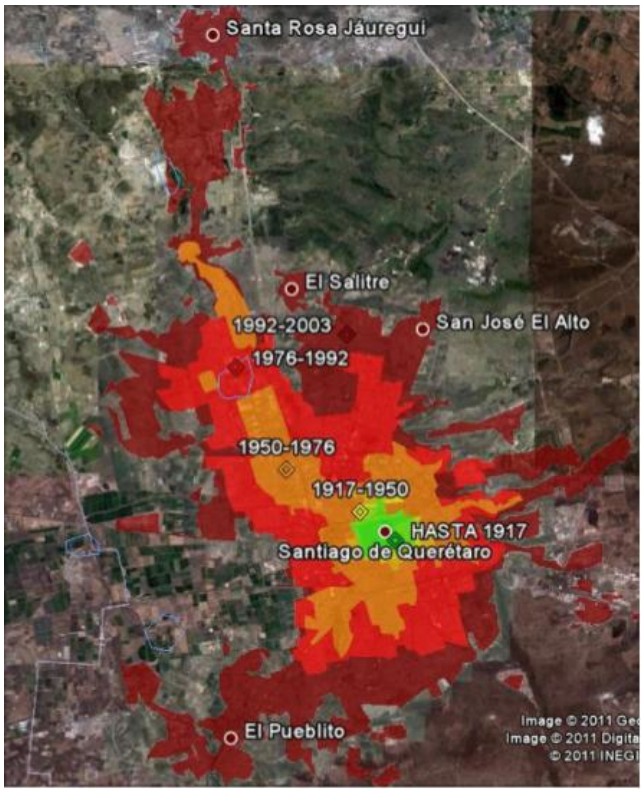


Figura 1.1. Crecimiento territorial en la Ciudad de Querétaro.

Los STP actuales cubren grandes regiones, son altamente complejos y, a menudo, difíciles de utilizar de forma efectiva. En años recientes hemos visto con gran beneplácito el surgimiento de aplicaciones computacionales que proveen mapas en línea, directorios, servicios basados en localización y servicios de planeación de itinerarios, las cuales intentan proporcionar la información necesaria a los usuarios. Sin embargo, estas aplicaciones de cómputo no siempre proveen una interfaz hombre-máquina apropiada. Parte de la dificultad al desarrollar tales sistemas completamente integrados es la naturaleza de los datos y la falta de un modelo coherente de datos que pueda ser usado efectivamente.

Conforme se ha ido dando respuesta a las necesidades de los usuarios del transporte público, el STP de la Ciudad de Querétaro, y su zona

metropolitana, se ha vuelto un sistema complejo; a tal grado que hasta hace un año consistía de más de 90 rutas que cubren principalmente la zona metropolitana de Querétaro, así como algunas de las poblaciones colindantes. Sin embargo, a pesar de dicha complejidad, hoy en día no se cuenta con un sistema de información que permita al usuario del STP obtener, de forma rápida y precisa, información básica acerca de los recorridos, horarios y puntos de ascenso y descenso de pasajeros. Más aún, no existe un sistema que informe al usuario el itinerario óptimo para transportarse de un punto de origen a un punto de destino en la ciudad.

En este trabajo se presume que es posible modelar el STP de la Ciudad de Querétaro mediante un grafo dirigido ponderado, y que los algoritmos para calcular rutas óptimas entre cada par de puntos de la red que represente al STP se pueden utilizar apropiadamente.

Para modelar el STP de la Ciudad de Querétaro, se realizó un levantamiento del recorrido geográfico de algunas de las principales rutas que lo conforman, las cuales fueron trazadas en el ambiente *Google Earth* (Google, Inc., 2011), debido a que brinda bastantes ventajas en cuanto al manejo eficiente de la información geográfica. La implementación de los algoritmos que se utilizan en este trabajo se llevó a cabo en el lenguaje *Mathematica*, el cual es un lenguaje de alto nivel basado en el paradigma de programación funcional. Dicho lenguaje ofrece un conjunto de funciones especializadas que implementan diversos algoritmos que permiten calcular las rutas óptimas entre cada par de puntos o vértices de un grafo. El prototipo del STP aplicado a la ciudad de Querétaro se hizo en *Mathematica* (Wolfram Research, Inc., 2011) versión 8.0.

De las funciones con las que cuenta *Mathematica*, la mayormente utilizada en este trabajo de tesis es la función *FindShortestPath*, la cual genera el itinerario óptimo con base en el modelo de la red de transporte. El itinerario es

creado conforme a la ruta óptima, que permite al usuario transitar desde un punto de origen al punto destino a través de la red.

El resto de este trabajo se estructura de la siguiente manera: en el Capítulo 2 se describen y evalúan diferentes sistemas basados completamente en web, cuyo objetivo es la planeación de itinerarios entre dos puntos en una ciudad. Las consultas realizadas en estos sistemas se aplican en algunas ciudades de nuestro país, teniendo como opciones rutas que hace uso de transporte público, privado y/o peatonal. La cobertura de cada sistema tiene diferentes regiones de México y medios de transporte; por ejemplo, algunos utilizan la red de carreteras, la red de tráfico vehicular, la red de calles y avenidas o el STP de algunas ciudades.

Posteriormente se expone brevemente, en el Capítulo 3, el estado actual de la red del STP de la Ciudad de Querétaro, junto con algunos de los problemas que presenta. Una etapa crucial de este trabajo, consiste en el levantamiento de información geográfica de las trazas de algunas de las rutas que conforman la red de transporte público, así como la elaboración del modelo computacional del sistema en el lenguaje *Mathematica*.

El Capítulo 4 inicia con una breve introducción a la Teoría de Grafos, remontándonos al planteamiento original de Euler en el siglo XVIII para resolver el problema de los siete puentes de Königsberg (Jungnickel, 2007). Enseguida se hace mención de algunos conceptos fundamentales de la Teoría de Grafos, para posteriormente dar paso al problema del cómputo de la ruta más corta entre dos puntos cualesquiera de un grafo, poniendo particular interés en el algoritmo clásico de Dijkstra, el cual se utiliza para llevar a cabo ruteos en la red del STP de la Ciudad de Querétaro. Particularmente se utiliza la función *FindShortestPath* de *Mathematica*.

El Capítulo 5 está dedicado a la implementación del modelo computacional del STP, como base de un sistema de información de rutas basado en web. Se describe brevemente el método para desarrollar la primera etapa de un prototipo experimental montado en web. Para desarrollar este prototipo se ha utilizado la librería *JGraphT* (Naveh, 2011) en la implementación del modelo del grafo del STP de la Ciudad de Querétaro, así como el lenguaje de programación *JavaScript* para crear la aplicación que implementa el algoritmo de Dijkstra y muestra la ruta calculada sobre un mapa de *Google Maps* (Google, Inc., 2011) mediante el uso de las herramientas *API* de éste último, generando así el itinerario a seguir por un usuario.

En el Capítulo 6 se presentan las conclusiones generales de este trabajo de tesis, así como también se exponen algunas líneas futuras de investigación, tanto para extender el prototipo desarrollado del STP de la Ciudad de Querétaro, como para su utilización como una poderosa herramienta para estudiar el comportamiento del STP bajo diferentes condiciones de operación.

Finalmente en el Anexo A se muestra el póster presentado al término del Verano de la Ciencia 2011 Región Centro, alusivo al modelado del STP. En el Anexo B se presenta el artículo titulado “Modelado de Redes de Transporte: Un caso aplicado a la Ciudad de Querétaro.” derivado parcialmente de este trabajo de tesis, presentado en el marco del primer Congreso Internacional de Movilidad Urbana Sustentable, que se llevó a cabo en el mes de enero del 2012 en la Ciudad de Querétaro. En el Anexo C se incluye el código desarrollado en el lenguaje *Mathematica*, mientras que en el Anexo D se presenta el código desarrollado para la implementación del prototipo del STP de la Ciudad de Querétaro. Por último, en el Anexo E se describen los pasos para instalar y ejecutar el prototipo. Por último,

2. SISTEMAS ACTUALES DE INFORMACIÓN DE RUTAS DE TRANSPORTE PÚBLICO BASADOS EN WEB

"Si no trabajas en problemas importantes, es probable que no hagas un trabajo importante."
-Richard Hamming (1915 - 1998)

Actualmente existen varios sitios web que responden a consultas similares a las que se busca responder mediante este trabajo de investigación. A lo largo del presente capítulo se mencionan estos sitios, describiendo su interfaz, las poblaciones a donde están orientados, los medios de transporte en los que se pueden hacer consultas y sus resultados. Sin embargo, algunos de estos sitios arrojan información que no resulta del todo útil para el usuario, por ejemplo al dar resultados de la ruta óptima que no respeta el sentido de las calles.

2.1. Sistemas similares basados en web

Hasta diciembre de 2011, en la Ciudad de Querétaro, no se cuenta con un servicio de información completo, detallado y de fácil operación. Ni las instancias gubernamentales (I. H. Vela, Comunicación personal, 3 de febrero, 2011) ni las empresas de transporte han desarrollado un sistema de consulta eficiente acerca del STP de la ciudad, cuyo servicio esté enfocado a los usuarios finales de ese servicio.

Existen varios sistemas que responden a solicitudes de usuarios respecto al cálculo de rutas óptimas que conectan dos puntos geográficos. Estos sistemas van de los de uso general a los de uso particular que consideran únicamente el

transporte público. Por ejemplo, entre los sistemas más generales se tienen el *Google Maps* (Google, Inc., 2011) y el *MapQuest* (MapQuest, Inc., 2011) los que, dados dos puntos por el usuario (origen y destino), calculan la ruta óptima a través del sistema de calles de una ciudad sin tomar en consideración los sentidos vehiculares. Aunque estos sistemas proporcionan varias opciones respecto a transportación pública y privada, no se tiene el mismo servicio para la Ciudad de Querétaro.

A nivel nacional se encuentran sistemas como el *Buscaturuta* (Buscaturuta.com, 2011) y el *ViaDF.org* (ViaDF.com.mx, 2011), que están orientados a dar servicio a los usuarios de nueve ciudades importantes del país: Ciudad de México, Guadalajara, Monterrey, Aguascalientes, Cancún, Puerto Vallarta, Saltillo, Torreón y Villahermosa; asimismo -en versión beta- proporcionan información para las ciudades de Mazatlán y Puebla. Estos sistemas ofrecen a los usuarios rutas eficientes haciendo uso de transportación motorizada pública y privada, así como los desplazamientos a pié que el usuario debe llevar a cabo para hacer las conexiones pertinentes. La Secretaría de Comunicaciones y Transportes también ha puesto en línea su sistema de *Rutas punto a punto* (Secretaría de Comunicaciones y Transportes, 2011), mediante el cual ofrece a los usuarios el ruteo óptimo a través del sistema carretero nacional.

Entre los sistemas de información que particularmente atienden preguntas de ruteo en la ciudad de Querétaro se encuentran *Rutero Online Querétaro* (Rutero Online, 2011) y un sistema provisto por la *Secretaría de Seguridad Ciudadana* (Secretaría de Seguridad Ciudadana, 2011). Los cuales despliegan información sobre el recorrido de las rutas del STP de la Ciudad de Querétaro. En el primero, dado un punto geográfico por el usuario, el sistema despliega la línea de transporte público más cercana, mientras que el segundo, despliega a petición del usuario una línea en particular. Sin embargo, todos los sistemas anteriormente, no responden a las necesidades del usuario de contar con un itinerario de la ruta óptima conformado por una secuencia de cruces de

calles donde una determinada línea del transporte público puede ser abordada, así como los puntos de transbordo necesarios para que el usuario arribe al sitio de destino.

A continuación se muestran cada uno de los sistemas antes mencionados, señalando sus características relevantes en términos de funcionalidad, así como las características deseables que apuntan al prototipo que se desarrollará en esta tesis.

2.1.1. Rutero Online Querétaro

La interfaz del sistema *Rutero Online Querétaro* (Figura 2.1) está dividida en dos partes. Del lado izquierdo despliega el mapa usando el ambiente de *Google Maps* y del lado derecho muestra la lista de las 81 rutas disponibles (de las más de 90 rutas que conforman el STP de la Ciudad de Querétaro) para ser seleccionadas por el usuario.

Un usuario hace *click* sobre el mapa para indicar su posición en la ciudad y el sistema despliega un marcador de posición mediante la silueta de un individuo -en el punto donde el usuario hace *click* con el mouse sobre el mapa- y una lista de las rutas que pasan “lo más cerca posible” a la ubicación donde está el marcador, sin que esto implique que tales rutas pasen exactamente por la posición señalada por el marcador. El sistema podría no desplegar ruta alguna cuando el punto marcado está “alejado” de algún punto cubierto por el STP. Esto ya denota una carencia del sistema. El usuario podría seleccionar directamente una de las rutas que pasan lo más cercano al marcador, mostrando la traza de dicha ruta en el mapa, tal como se muestra en la Figura 2.1. En este sistema es posible tener hasta dos marcadores de posición, -si se vuelve a hacer *click* dentro del mapa- el segundo marcador se despliega sobre el mapa en forma de una bandera, dando la idea de un punto de origen y un punto de destino (Figura 2.2).

Cada ruta se despliega en colores diferentes, de esta manera es posible distinguir el sentido de ida y el sentido de vuelta de su recorrido. En caso de que más de una ruta sea dibujada en el mapa, es posible diferenciar los sentidos de ambas rutas por sus colores. Además de lo anterior, ninguna otra información es proporcionada al usuario por este sistema.

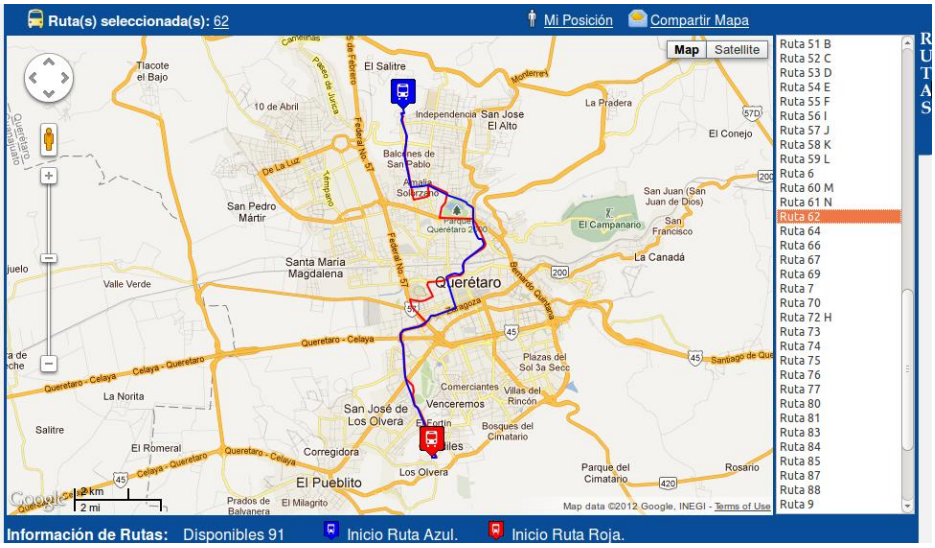


Figura 2.1. *Rutero Online* mostrando la ruta 62, que pasa cerca del punto señalado por el usuario en el mapa.

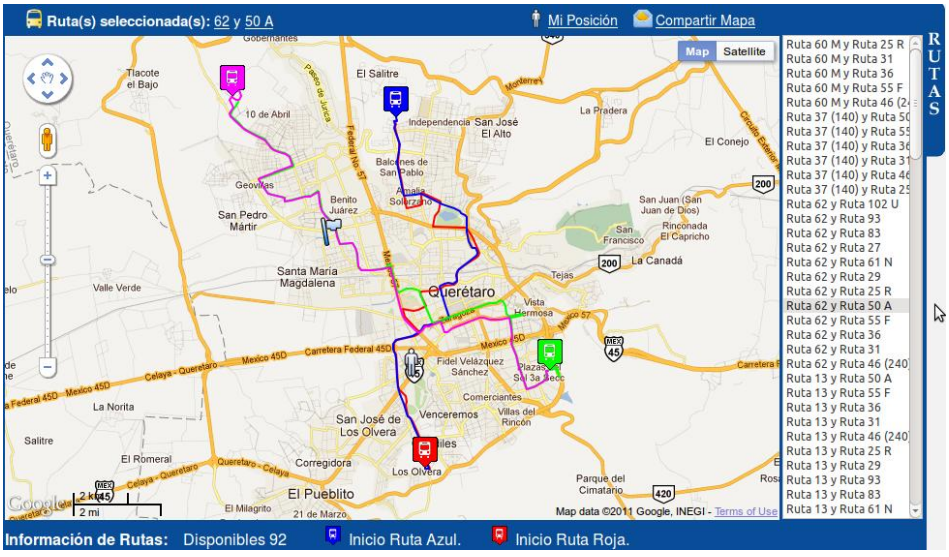


Figura 2.2. Funcionamiento del *Rutero Online* con dos marcadores.

2.1.2. Sistema de Información de Rutas del STP de la Ciudad de Querétaro de la Secretaría de Seguridad Ciudadana (SSC)

Este sitio web presenta una interfaz muy amigable para el usuario; sin embargo, carece de ciertas funcionalidades. Como se ilustra en la Figura 2.3, el sistema muestra una pantalla que enlista en su lado derecho solamente 86 rutas, y a su lado izquierdo despliega, sobre un mapa basado en *Google Maps*, las rutas marcadas. En algunos casos el sistema solamente es capaz de desplegar a lo más 25 rutas simultáneamente, a partir de esa cantidad, los recorridos dejan de dibujarse sobre el mapa o inclusive, algunos de los recorridos seleccionados previamente, simplemente desaparecen. Otra desventaja es de que al desplegarlas todas ellas con el mismo color café no se pueden identificar fácilmente; en la Figura 2.3 se muestran las rutas 10, 13, 20, 24, 27, 30, 37 y 40.

Por lo tanto, el sistema se reduce a informar sobre la traza de las rutas, con las desventajas mencionadas anteriormente y, más aún, no le permite al usuario construir su itinerario a través del sistema de transporte público dados los puntos de origen y destino.

El sistema también representa las carreteras estatales y las bases de transporte público –puntos de salida y llegada-, conforme a la simbología establecida en el mismo sitio.

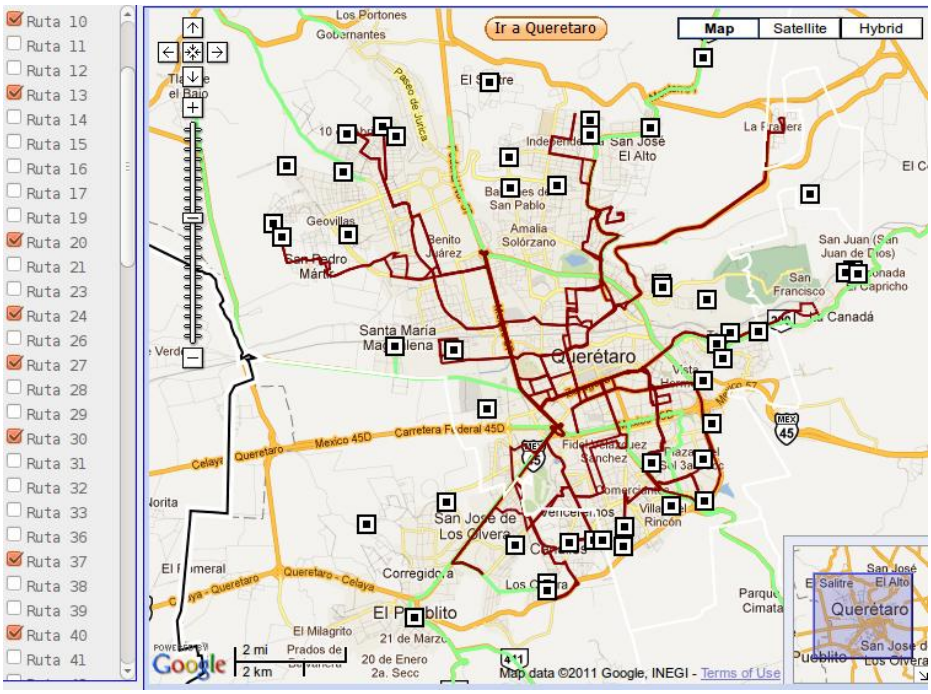


Figura 2.3. Sitio de la SSC mostrando el recorrido de ocho rutas.

2.1.3. Sistema de Información de la Secretaría de Comunicaciones y Transportes, "Rutas punto a punto"

Este sistema está orientado a la creación de la ruta más corta para un recorrido carretero entre dos ciudades del país, detallando en los resultados el nombre de las poblaciones por las que se atravesará, la entidad federativa, el número de carretera federal, longitud del tramo carretero, el tiempo que se tomará en recorrer ese tramo, número de casetas y el costo del peaje para cada una de ellas.

La interfaz principal (Figura 2.4) cuenta con instrucciones paso por paso para procesar la petición. En la primer parte se fija el estado y ciudad de origen, así como la ciudad y estado destino. En caso de que el usuario necesite indicar puntos intermedios en la ruta, -como otras poblaciones- se tiene la opción de agregar hasta dos puntos adicionales. En la siguiente sección, el usuario elige el tipo de vehículo que utilizará en su recorrido, con el objetivo de que el sistema muestre las tarifas correspondientes en cada caseta. Entre las opciones de

vehículos se encuentran automóviles, autobuses de dos a cuatro ejes, camiones de dos a nueve ejes, motocicletas y pick ups. El apartado siguiente (que no se muestra en la Figura 2.4) es opcional; el usuario llena el formulario sólo si desea conocer el gasto estimado de combustible, para lo cual debe proporcionar al sistema el tipo de vehículo, rendimiento estimado del vehículo (km/l), tipo de combustible y desplazamiento (cm^3). Proporcionar o no la información al sistema en esta parte no genera un cambio en el resultado del recorrido.

Al enviar el formulario completado al sistema, los resultados se muestran de forma gráfica en un mapa del país (Figura 2.5), con la ruta resaltada junto con las poblaciones que atraviesa el recorrido. Se anexa una tabla (Figura 2.6), donde cada renglón es un punto en la ruta; en sus campos se encuentran los datos correspondientes descritos anteriormente.

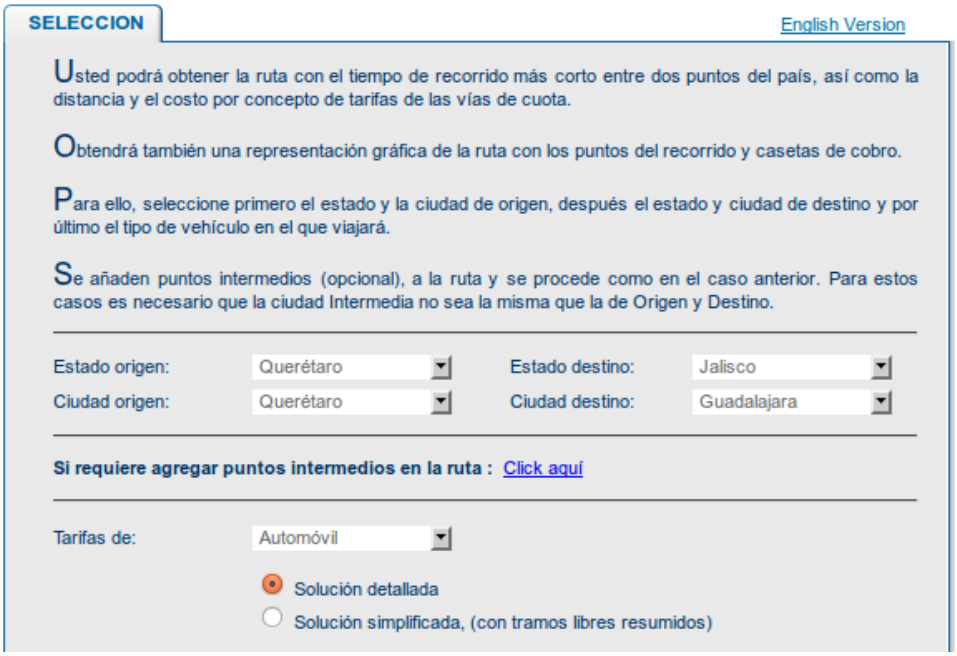


Figura 2.4. Formulario de inicio del sistema *Rutas punto a punto*.

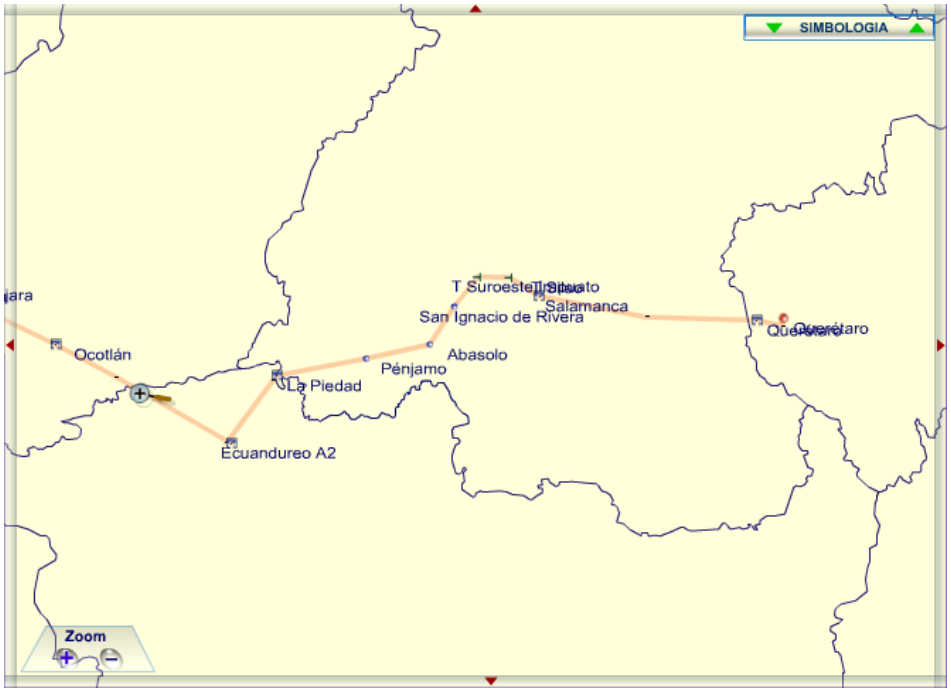


Figura 2.5. Resultado de la ruta en el mapa.

Nombre	Edo.	Carretera	Long.(km)	Tiempo(Hrs)	Caseta o puente	Automóvil
Querétaro Centro - Querétaro (Ent. T Villa del Pueblito)	Qro	Zona Urbana	2.000	00:03		
Querétaro (Ent. Villa del Pueblito) - Entronque Celaya	Gto	Mex 045D	44.500	00:24	Querétaro	63.0
Entronque Celaya - Entronque Silao	Gto	Mex 045D	50.750	00:27	Salamanca	63.0
Libramiento Sur Irapuato (Ent. Silao - Ent. Suroeste)	Gto	Mex 045D	8.656	00:04		
Entronque Suroeste Irapuato - San Ignacio de Rivera	Gto	Mex 110	12.328	00:07		
San Ignacio de Rivera - Abasolo	Gto	Mex 110	15.000	00:09		
Abasolo - Pénjamo	Gto	Mex 110	21.000	00:15		
Pénjamo - La Piedad	Mich	Mex 110	37.000	00:27	La Piedad	11.0
La Piedad - Entronque Ecuandureo	Mich	Mex 037	27.100	00:20		
Entronque Ecuandureo - Entronque La Barca	Jal	Mex 015D	42.060	00:23	Ecuandureo A2	65.0
Entronque La Barca - Entronque Guadalajara	Jal	Mex 015D	72.476	00:39	Ocotlán	111.0
Entronque Guadalajara - Entronque Tonalá	Jal	Mex 090D	13.860	00:07	La Joya	46.0
Entronque Tonalá - Gdl. (Ent. Av. Revolución)	Jal	Mex 090D	6.000	00:03		
Gdl. (Ent. Av. Revolución) - Guadalajara Centro	Jal	Zona Urbana	8.000	00:09		
		Totales	360.730	03:43		359.0

Figura 2.6. Itinerario resultante en el sistema *Rutas punto a punto*.

2.1.4. Google Maps (México, versión beta)

La interfaz minimalista con la que el sitio cuenta (Figura 2.7), facilita al usuario realizar una búsqueda. En el panel izquierdo la opción “Cómo llegar” es donde se lleva a cabo la consulta; por medio de dos iconos se indica al sistema el tipo de transporte que el usuario utilizará, teniendo como opciones “caminando” o

“usando el automóvil”. Debajo de estos íconos, se encuentran dos campos de texto, uno para la dirección de origen y otro para la dirección de destino. El usuario puede escribir directamente la dirección por calle, número y colonia. Sin embargo, no en todas las ocasiones se tiene la dirección deseada; los inconvenientes son que el sistema no reconoce todas las abreviaciones, colonias o calles. Existe otra alternativa por parte del sitio para no escribir la dirección, mediante un *click* directamente en el mapa, indicando de esta forma los puntos de origen y destino (Figura 2.8).

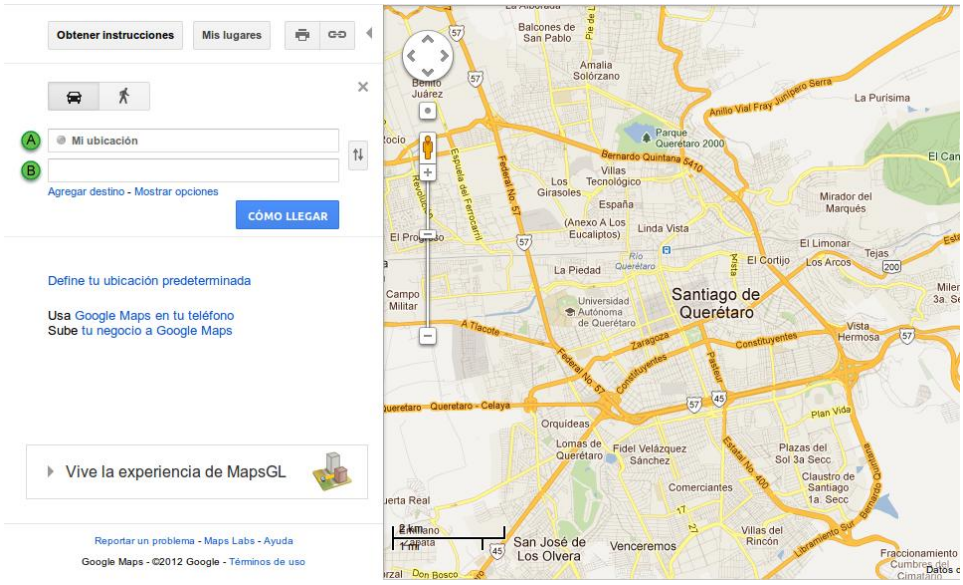


Figura 2.7. Interfaz principal de *Google Maps* al hacer una búsqueda.

Figura 2.8. Mapa del resultado de una ruta en *Google Maps*.

Se hizo la prueba dentro de la Ciudad de Querétaro, para conocer el resultado de la ruta entre dos puntos. Los resultados que el sitio arrojó fueron tres rutas distintas, teniendo como medio de transporte el automóvil. Con ambos puntos marcados en el mapa, automáticamente el sitio despliega las rutas sugeridas (en caso de tener más de dos) para ir de un punto a otro (Figura 2.9). Posteriormente se enlista una serie de pasos de cómo llegar del punto de origen al punto destino (Figura 2.10).

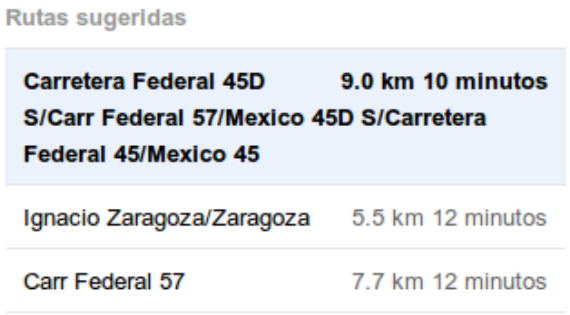


Figura 2.9. Distintas rutas sugeridas por *Google Maps*.

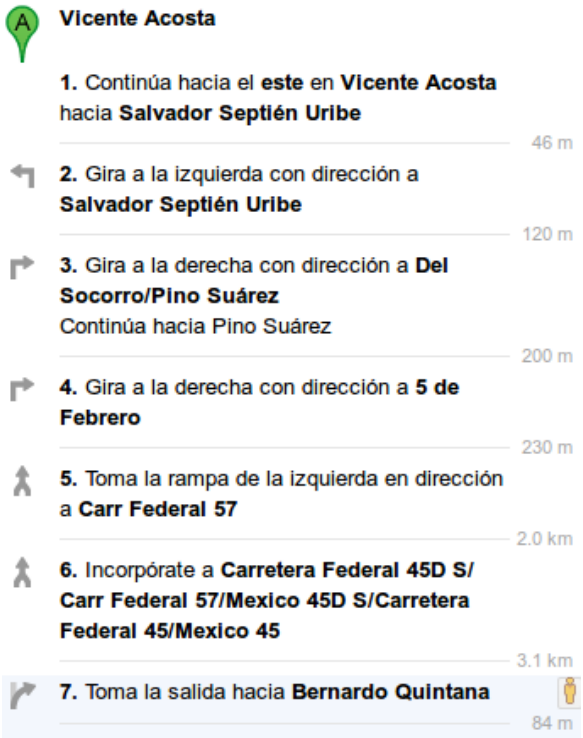


Figura 2.10. Parte de las instrucciones de *Google Maps* enlistadas.

Cada que la ruta hace un quiebre de calle o algún movimiento, se enlista como un paso más junto con su descripción. Otra de las ventajas que ofrece *Google Maps* es que en cada paso de la lista hay un ícono que representa la función "*Street View*", un servicio desarrollado por *Google* que, como su nombre lo dice, permite al usuario tener una vista de la calle tal y como si estuviera ahí parado (Figura 2.11).

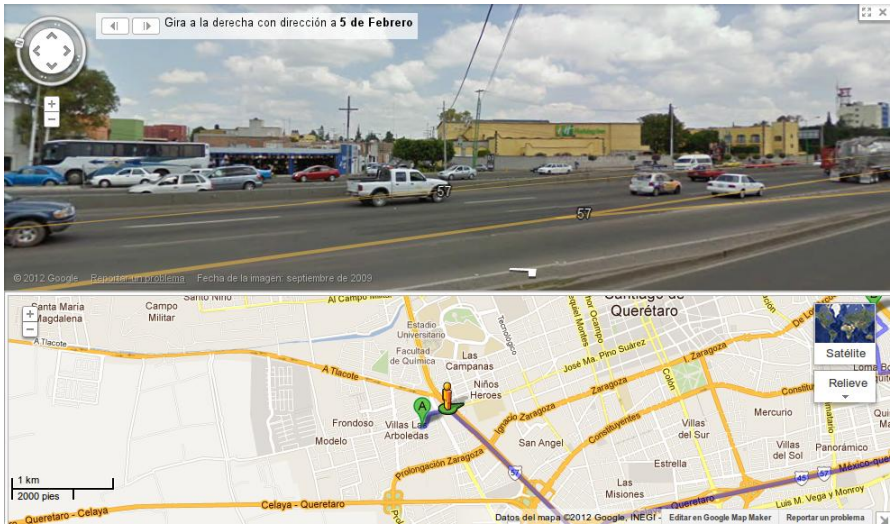


Figura 2.11. Imagen tipo *Street View* para el paso 4 de la Figura 2.10.

También se pueden agregar puntos intermedios para forzar a la ruta a cambiar su recorrido, el número de puntos intermedios pueden ser tantos como el usuario lo desee (Figura 2.12).

Figura 2.12. Ruta forzada a cambiar su trayecto.

2.1.5. MapQuest

MapQuest cuenta con una interfaz muy similar a la de *Google Maps,* haciendo que la forma en que el usuario hace una búsqueda para ir de un sitio a otro sea prácticamente la misma. Asimismo, *MapQuest* también ofrece al usuario la opción de elegir el medio de transporte en el cual desea realizar su recorrido; los resultados se muestran en el mapa de la Figura 2.13, tanto en forma de lista (siendo un paso distinto cada que se realiza un quiebre de calle o un movimiento y adjuntando una pequeña descripción), como en forma gráfica sobre un mapa de la zona. Así como *Google Maps*, *MapQuest* da la oportunidad de modificar el resultado de la ruta agregando puntos intermedios; esto se puede realizar haciendo *click* sobre la línea de la ruta y arrastrándola al punto deseado.

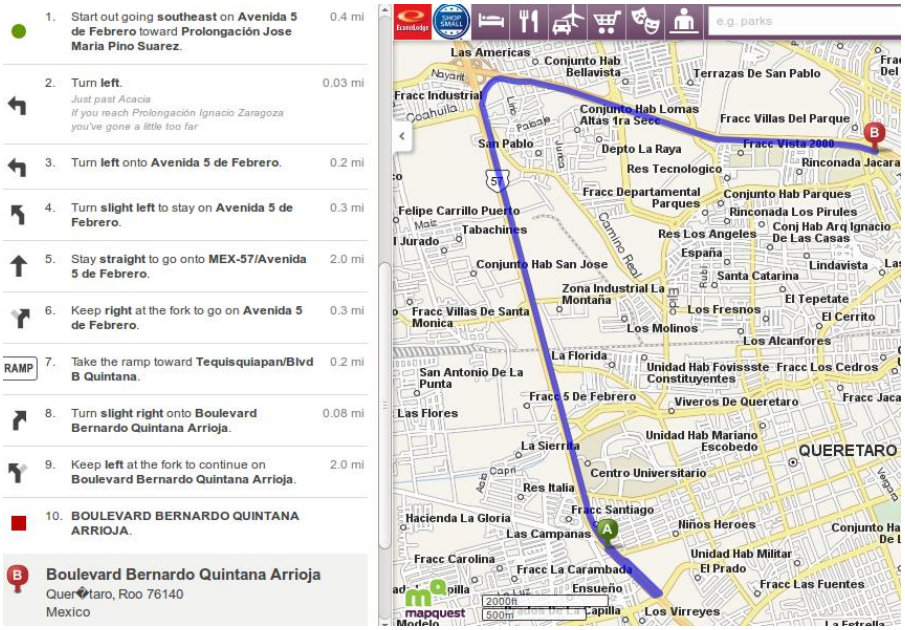


Figura 2.13. Resultado de la ruta en *MapQuest*.

2.1.6. ViaDF.org

El sistema *VíaDF*, como su nombre lo sugiere, brinda servicio a los usuarios del transporte público de la Ciudad de México. Los servicios que ofrece el sitio son muy similares a los que se buscan incluir en el prototipo que se plantea en este trabajo, aunque en el caso de *VíaDF* el sistema está diseñado

para proporcionar información acerca del STP de la Ciudad de México. En los puntos de origen y destino se introduce la delegación por medio de una lista que muestra las 16 delegaciones del Distrito Federal; en una siguiente lista se elige la colonia correspondiente a la delegación y, por último, en un cuadro de texto se escribe la calle y número. También se pueden indicar lugares de interés (Figura 2.14). Éste último campo no necesariamente debe ser completado por el usuario, puesto que aún sin haber escrito la calle, es posible realizar la búsqueda por colonias. La hora deseada de llegada al destino también se indica en un cuadro de texto. Por último, se eligen los tipos de transporte público que el usuario quiere utilizar en su recorrido; entre ellos están el metro, el metrobús, la red de transporte de pasajeros (RTP), el trolebús, el microbús y la EcoBici.

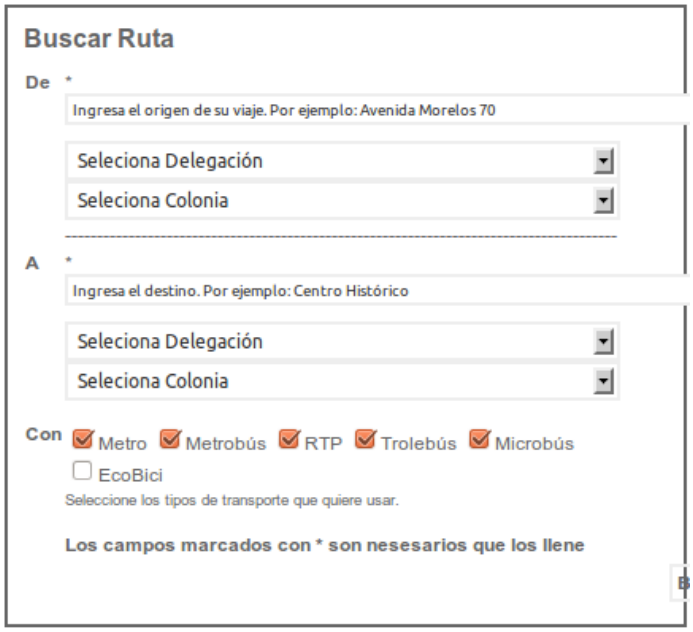


Figura 2.14. Formulario para generar la ruta en *VíaDF*.

Los resultados -así como en los demás sitios- se despliegan de forma similar: la ruta marcada en un mapa y los detalles de cada cruce o cambio de calle se muestran en una tabla o en forma de lista (Figura 2.15 y Figura 2.16). En la tabla, las columnas indican el lugar, la hora con el tiempo de duración aproximado, distancia, el medio de transporte y su costo (Figura 2.15). El mapa,

basado en *Google Maps*, resalta la ruta del punto de inicio al punto de destino (Figura 2.16).

Resultados - Michoacán 129, Condesa, Cuauhtémoc A Avena 430, Granjas México, Iztacalco

Lugar	Hora (aprox.)	Duración	Distancia	Como	Precio
Michoacán 129, Condesa, Cuauhtémoc Metro Patriotismo (Linea 9)	18:25	13min	0.56km	Caminado	
Metro Patriotismo hacia Pantitlán (Linea 9) Metro Velódromo (Linea 9)	18:38	14min	8.21km	Metro	$3
Metro Velódromo (Linea 9) Avena 430, Granjas México, Iztacalco	18:52	8min	0.7km	Caminado	
En Total	**19:00**	**35min**	**9.47km**		**$3**

Figura 2.15. Resultado detallado de la búsqueda en *VíaDF*.

Figura 2.16. Trazo del resultado de la búsqueda.

2.1.7. Buscaturuta.com

El sitio, teniendo a Guadalajara como ciudad principal entre nueve ciudades disponibles, provee el servicio de búsqueda de rutas usando el transporte público de la ciudad elegida. En la interfaz, tanto en el origen como en el destino, hay un menú para buscar por domicilio, y además dentro de éste enlista sitios de interés como terminales de autobuses, aeropuertos, museos, bares, restaurantes, hoteles, etc. La última parte del recuadro de búsqueda permite indicarle al sistema la hora aproximada de llegada a nuestro destino (Figura 2.17).

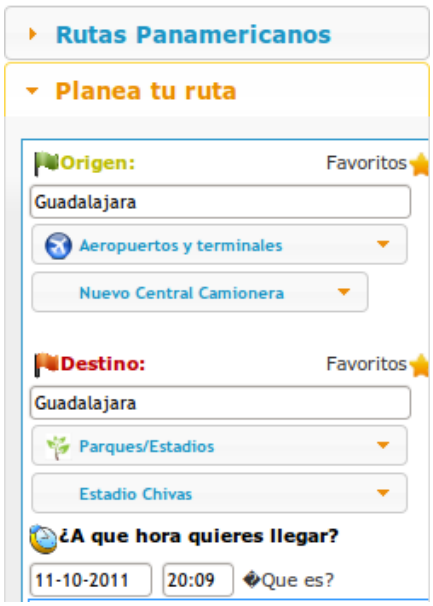


Figura 2.17. Formulario de búsqueda en *Buscaturuta*.

Los resultados (Figura 2.18) tienen cierto parecido con varios de los sitios mencionados anteriormente, especialmente con *Google Maps*. El sistema también sugiere más de una ruta si es posible. No todas son rutas necesariamente óptimas en términos de distancia, tiempo y costo del medio de transporte.

Figura 2.18. Resultado detallado de la búsqueda en *Buscaturuta*.

Al hacer *click* en las diferentes rutas sugeridas en los resultados, se despliega automáticamente en el mapa la ruta seleccionada. Se tiene también la vista de *Street View* de *Google Maps* en cada punto marcado en el mapa (Figura 2.19).

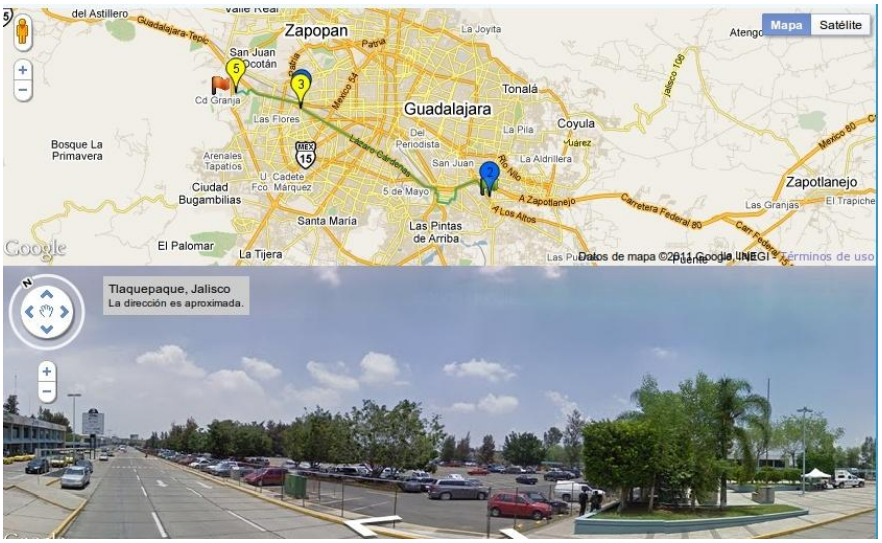


Figura 2.19. Traza del resultado de la búsqueda en *Buscaturuta*.

2.2. Otros sistemas de información de rutas para un STP.

Existen prototipos de planeación de itinerarios en sistemas de transporte público enfocados a lugares fuera de México, desarrollados sobre plataformas para teléfonos móviles (Kjeldskov, Andersen y Hedegaard, 2007) y también para *smartphones* con *GPS* integrado (Repenning y Ioannidou, 2006).

En países como Israel, se han venido desarrollando algoritmos que con base a los horarios y frecuencia de paso de autobuses, trenes o barcos, son capaces de calcular la ruta óptima minimizando el número de transbordos que el pasajero debe realizar; los cuales pueden adaptarse a cualquier STP (Association for Computing Machinery, 2011).

Algunos países desarrollados cuentan con sistemas más avanzados, los cuales arrojan resultados óptimos y rutas alternas basadas en el tiempo total, horarios de servicio de las unidades del STP, y costo, por ejemplo. Los sistemas desarrollados para las ciudades de nuestro país están enfocados a dar como resultado la ruta óptima en cuanto a distancia principalmente.

3. MODELO COMPUTACIONAL DEL SISTEMA DE TRANSPORTE PÚBLICO DE LA CIUDAD DE QUERÉTARO

"Es mejor hacer el problema correcto de la forma equivocada que el problema equivocado de la forma correcta"
-Richard Hamming (1915 - 1998)

El objetivo principal de este trabajo consiste en modelar computacionalmente un sector del sistema de transporte público de la Ciudad de Querétaro, así como aplicar algoritmos para determinar las rutas más cortas entre cualesquiera dos puntos de dicho modelo, como lo son el algoritmo de Dijkstra y el de Bellman-Ford.

Los algoritmos para la planeación de rutas sobre redes de transporte han sido sometidos recientemente a un rápido proceso de desarrollo, conduciendo a métodos que son hasta tres millones de veces más rápidos que el algoritmo de Dijkstra (Delling, Sanders, Schultes y Wagner, 2009). No es el objetivo del presente trabajo el desarrollar un nuevo algoritmo para calcular rutas más cortas, sino utilizar algoritmos existentes para ser implementados en una red de transporte público real.

Los modelos computacionales de redes de transportes reales difieren de las redes generadas aleatoriamente en lo siguiente: las redes reales no son muy uniformes, pueden observarse *clusters* en áreas como el centro de cada ciudad y pequeños *clusters* a lo largo de la ciudad en diversas colonias o áreas conurbadas. En redes generadas de manera aleatoria por computadora, los pesos

de sus aristas se elijen de forma aleatoria, siguiendo una distribución uniforme dentro de un intervalo dado. Por otro lado, en redes de transporte reales las aristas están conformadas por pesos que representan tanto distancias cortas como distancias más largas (Riko, Marathe y Kai, 1999).

Debido a la naturaleza de las aplicaciones, los investigadores que estudian los problemas de transporte necesitan procedimientos muy flexibles y eficientes para calcular las rutas más cortas, desde el punto de vista del tiempo de ejecución y también para los requerimientos de memoria. Como no existe el "mejor" algoritmo para cada tipo de problema de transporte, no hay un algoritmo existente el cual muestre siempre la misma conducta, independientemente de la estructura del grafo (Pallotino y Scutellá, 1997).

3.1. Sistema de Transporte Público en Querétaro

La instancia gubernamental que regula el transporte público del estado de Querétaro es la Secretaría de Seguridad Ciudadana (SSC). De acuerdo con el Coordinador de Operación del Transporte de esa instancia gubernamental, hasta principios del 2011 no se contaba con un Sistema de Información que dé cuenta del conglomerado de rutas del STP de la Ciudad de Querétaro, y por ende el trabajo de planeación se ve menoscabado.

En el sitio web de la SSC, explicado en la sección 2.1.2, existe únicamente un apartado dedicado al STP que contiene una interfaz basada en *Google Maps* y que despliega de manera gráfica los recorridos de 88 rutas del STP. A partir de esta interfaz no es posible recuperar las trazas de los recorridos de las rutas del STP. Sin embargo con dicha información visual del sitio se creó y trazó el prototipo del modelo computacional del STP propuesto en esta tesis.

Las más de 90 rutas del STP actual de la Ciudad de Querétaro tienen una cobertura de gran parte de la zona metropolitana de la Ciudad de Querétaro y

áreas conurbadas; sin embargo, varias rutas repiten sus recorridos o tramos parciales de recorridos, haciendo que en calles y avenidas donde ocurre este fenómeno se presente la superposición de rutas. Tal es el caso de la avenida Zaragoza, donde 87 rutas la atraviesan en alguno de sus tramos (Banda, 2012).

En el mes de marzo de 2012, fue promulgada la Ley de Movilidad para el Transporte del Estado de Querétaro. Dicha ley pretende asegurar una mejor eficiencia en el STP, modificando las rutas existentes e implementando un nuevo sistema de prepago y descuentos en las tarifas, así como la integración de tecnologías para el monitoreo de la red del STP (Gobierno del Estado de Querétaro, 2011). Sin embargo, es importante señalar que todo el trabajo realizado durante la tesis, se hizo teniendo el STP de la Ciudad de Querétaro que a esta fecha sigue en funcionamiento.

3.2. Levantamiento de información geográfica de trazas de la red del STP de la Ciudad de Querétaro

Al ser uno de los principales objetivos del proyecto de investigación el modelar la red del STP de la Ciudad de Querétaro, representándola por medio de un grafo dirigido conexo, es necesario llevar los datos de cada recorrido al modelo computacional.

Google Earth es un software que se puede descargar y utilizar de manera gratuita bajo la licencia *Freeware* (Google, Inc., 2011). En él es posible visualizar todo el planeta en imágenes de satélite; mostrar ciudades y superficies en 3D; nombres de calles y poblaciones, además de poder usar el motor de búsqueda de *Google*. Dadas éstas características se utilizó el software en su versión 6.0.1 para levantar el modelo geográfico. Otras ventajas que da el software para la creación del modelo son el permitir ver toda la ciudad a gran detalle en cuanto a calles, trazado de líneas, coordenadas geográficas, distancias entre dos puntos en el mapa, la distancia total de una ruta a mano alzada, posicionamiento de puntos a

partir de coordenadas, etc. Cabe mencionar que la *API* de este software se encuentra disponible para su consulta en línea (Google, Inc., 2011).

Una de las finalidades del proyecto es el tener los resultados en una interfaz de *Google Maps* en web; para ello se requiere pasar las coordenadas geográficas de varios puntos de cada ruta del STP, para así obtener una ruta unida por una sucesión de puntos coordenados. Cualquier ubicación en *Google Earth* tiene coordenadas en latitud y longitud, de acuerdo con la documentación del mismo software (Google, Inc., 2011).

A continuación se presenta una breve definición para los términos “proyección cartográfica” y “datum geodésico”. Una “proyección cartográfica” se utiliza para realizar mediciones de la Tierra en una superficie plana, mientras que un “datum geodésico” se utiliza para describir la forma de la Tierra en términos matemáticos. La forma de la Tierra no es esférica, sino un elipsoide. Un datum define la asociación de las coordenadas en latitud y longitud con puntos en la superficie terrestre; también funciona como base para las mediciones de elevación. Tal y como en las proyecciones, existe más de una interpretación matemática para la forma de la Tierra. *Google Earth* utiliza el datum WGS84 (Google, Inc., 2011).

Los archivos que guarda *Google Earth* pueden ser de dos tipos, los cuales tienen dos distintas extensiones distintas, *kmz* y *kml*. Por la facilidad de uso y amplia documentación existente, se optó por utilizar el tipo *kml*. KML (*Keyhole Markup Language*) es un estándar internacional mantenido por el *Open Geospatial Consortium, Inc*. (*OGC*) (Google, Inc., 2011). Este tipo de archivo es creado con cualquier editor de textos, siguiendo las etiquetas necesarias para darle la estructura adecuada. En la documentación que *Google* proporciona sobre el estándar KML, provee del árbol de clase para los elementos KML (Figura 3.1). Los elementos mayormente utilizados son *Geometry* y su rama *LineString*.

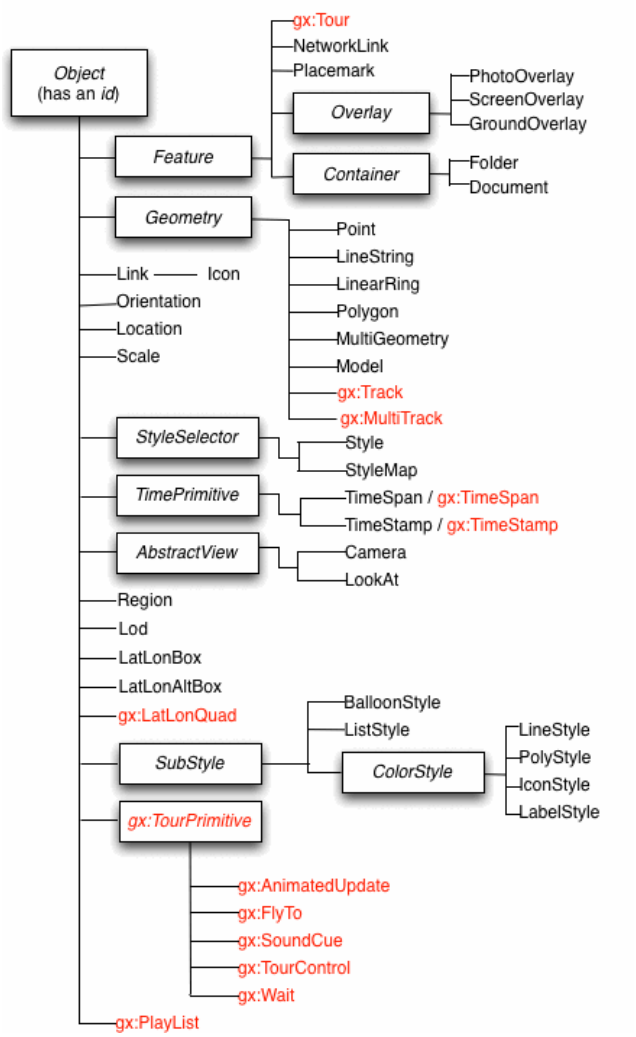


Figura 3.1. Árbol de clases para los elementos KML.

Los elementos en el árbol, a la derecha de una rama en particular, son extensiones de los elementos a su izquierda. Por ejemplo, *Point* es un tipo especial de *Geometry*. *Point* contiene todos los elementos que pertenecen a *Geometry*, agregando algunos elementos que son particulares a *Point*.

En la primera etapa del proyecto se llevó a cabo el levantamiento de las trazas de las rutas de interés. Para ello se eligieron aquéllas cuya longitud total está entre las mayores y atraviesan la ciudad casi de extremo a extremo, conforme a los distintos puntos cardinales, creando propiamente una estrella. Con ello se logra una mayor cobertura con un número mínimo de rutas (Figura 3.2).

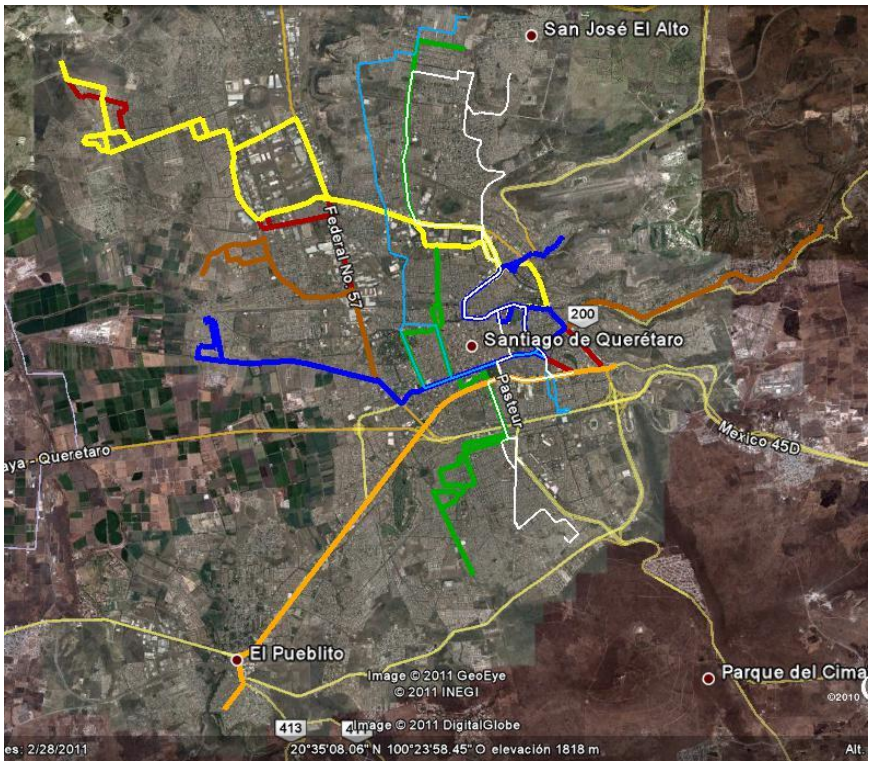


Figura 3.2. Interfaz de *Google Earth* mostrando la zona metropolitana de la ciudad con las rutas trazadas del STP.

El levantamiento de las trazas consiste en la adquisición de las coordenadas geográficas que constituyen la ruta. Para ello, se definen los puntos de interés como aquéllos que corresponden a puntos de inflexión, donde la unidad de transporte hace un quiebre para incorporarse a otra calle o avenida, y puntos de ajuste, los cuales resultan necesarios para el proceso de linealización de las trazas. Asimismo, cuando se agrega una nueva ruta al modelo computacional, en el proceso de construcción del grafo dirigido conexo del STP, eventualmente aparecen nuevos puntos en los sitios de cruce, a los que se les ha llamado puntos de intersección. Este proceso se llevó a cabo en el ambiente provisto por la interfaz de *Google Earth*, utilizando su herramienta llamada *Path.*

En la documentación que provee *Google* para su software *Google Earth* se explica cómo manipular los archivos con extensión *kml*. Los archivos respectivos de cada ruta con extensión *kml* son importados en el software *Mathematica* para procesar la información geográfica con los algoritmos

especializados que ese software ofrece. Siguiendo la estructura de clase para el archivo *kml*, es posible importar cada ruta y, de esa forma, crear las aristas que pertenecerán al grafo del STP.

3.3. Sistematización de la información

En la sección anterior se discutió de manera general el proceso de levantamiento de las trazas del STP. Sin embargo en esta parte se detallará el proceso completo para obtener el modelo matemático.

La interfaz de *Google Earth* provee del contexto computacional apropiado para el levantamiento de trazas del prototipo. La herramienta *Path*, como su nombre lo indica, crea una ruta sobre el mapa, consistente en una sucesión de puntos coordenados, los cuales son almacenados en un archivo con extensión *kml* que puede accesarse mediante algún editor de textos. Como previamente se había mencionado, el sitio de la SSC contiene los recorridos oficiales de la mayoría de las rutas del STP, siendo dicho sitio la fuente principal de información gráfica para obtener las rutas y así generar el modelo geográfico.

Antes de comenzar a levantar cada ruta, se definieron algunos criterios para el punteo de recorridos, los cuales se mencionan a continuación. Los puntos de inflexión naturales de la ruta son considerados vértices. Cuando una ruta atraviese el recorrido de otra ya existente, se marca un nuevo vértice como punto de intersección. En avenidas largas también se marcaron puntos intermedios, los cuales representan los sitios donde existe ya sea un ascenso o un descenso de usuarios, sin tener que llegar hasta el próximo punto de inflexión o intersección.

Como en toda ciudad, existen calles y avenidas que tienen tramos con formas curvas, y la ciudad de Querétaro no es la excepción. Los recorridos de las rutas del STP que pasan por esos tramos se deben adecuar a la forma de la calle. Por lo que al modelo computacional se refiere, y a efecto de que éste sea lo más

apegado a la realidad de las calles, se llevó a cabo un proceso de linealización de rutas, para de este modo obtener una mejor aproximación a la distancia real del recorrido de la ruta. Estos vértices punteados en las curvas son llamados puntos de ajuste (Figura 3.3).

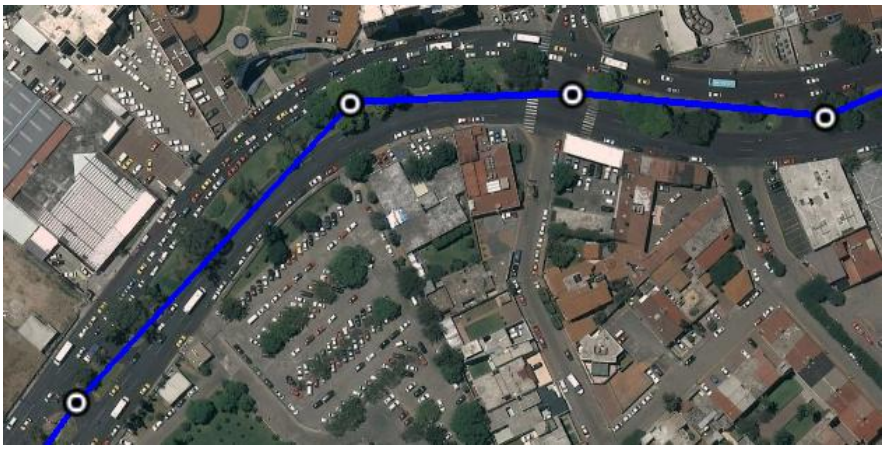


Figura 3.3. Puntos de ajuste de un recorrido en una calle curva.

En las avenidas principales de la ciudad, o en avenidas donde no existe algún cruce de calle en varios cientos de metros, se fijaron vértices nuevos que no necesariamente forman una intersección con otro recorrido, ni tampoco son puntos de ajuste; a estos vértices se les denominó puntos intermedios. Dichos puntos fueron creados con el objetivo de que, una vez estando en el modelo computacional, el itinerario resultante para el usuario no le indique hacer una parada hasta la siguiente intersección con otro recorrido, o hasta el siguiente punto de inflexión, en caso de que la avenida sea una recta (estos puntos intermedios pueden indicar de hecho un punto de ascenso y descenso de pasajeros).

Cada recorrido se levantó sobre el mapa de *Google Earth* con su herramienta *Path*, haciendo una comparación visual de la interfaz de *Google Maps* que provee el sitio web de la SSC y la interface de *Google Earth*. No tiene un gran nivel de complejidad, porque los recorridos están marcados sobre la zona metropolitana de la Ciudad de Querétaro. El proceso de ir marcando los

recorridos debe respetar las mismas calles que los recorridos que muestra el sitio de la dependencia de gobierno.

En la Tabla 3.1 se presentan las rutas elegidas para hacer el prototipo funcional, siguiendo las direcciones norte a sur, este a oeste, noroeste a sureste y noreste a suroeste, con el objetivo de cubrir una amplia zona de la ciudad.

Tabla 3.1. Rutas del STP levantadas para integrarse al prototipo.

Rutas							
2	7	9	11	38	46	69	X (110)

El trabajo de punteo debe ser extremadamente cuidadoso al momento de integrar una ruta nueva a las ya existentes. Con lo anterior se hace referencia a la siguiente situación en un crucero de avenidas (Figura 3.4).

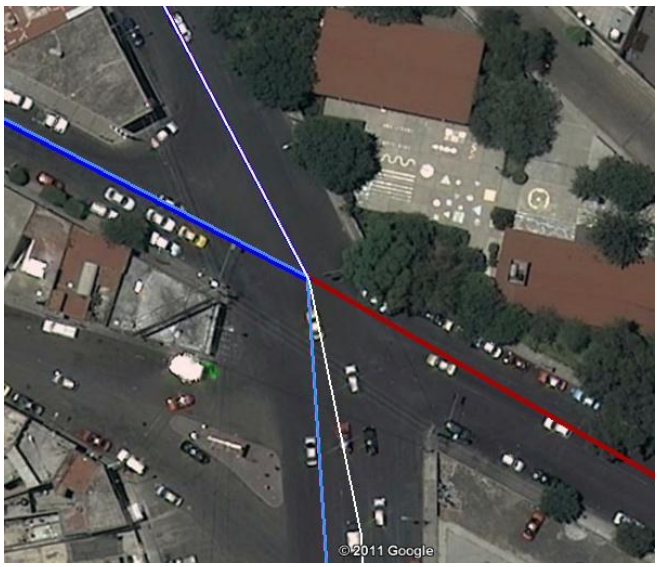


Figura 3.4. Intersección de rutas en dos avenidas.

Los recorridos conforman un subgrafo del STP, por lo tanto si se marcan nuevos vértices alrededor de los vértices existentes, de cierta forma se genera una desconexión. Con el término desconexión se hace referencia al siguiente problema: cuando un usuario realice una consulta sobre el prototipo, y el recorrido

del itinerario resultante pasa por un vértice desconectado de los demás, se dará el caso en el que no exista la posibilidad de transbordar de una ruta a otra.

Partiendo de los criterios de punteo definidos anteriormente, el método para el levantamiento de las trazas de las rutas del STP se describe a continuación: con la herramienta *Path* se procedió a marcar el recorrido de una primer ruta la cual fungió como referencia para las demás, en el caso de que aquéllas se intersectaran con ésta primer ruta.

A las aristas, conformadas por objetos *LineString* (creados por la herramienta *Path*) se les asignó un número de seguimiento en su propiedad *name* dado en orden ascendente, comenzando desde la primera arista hasta la última; llevando de este modo un cierto orden y control de aristas por ruta.

Se tiene en cuenta que el recorrido de cada ruta es un ciclo; esto quiere decir que el punto de origen es el mismo punto de destino, por lo que existe la posibilidad de que la ruta atraviese el mismo tramo de calle o avenida en uno o más sentidos durante su recorrido.

Cada que el recorrido atraviesa más de una vez el mismo tramo de calle o avenida en dos sentidos distintos, se revisó manualmente el archivo *kml* generado hasta ese momento y, ubicando por su número de seguimiento a la arista ya existente en ese tramo, se tomaron sus puntos contenidos en la etiqueta *coordinates* para así crear un nuevo objeto *LineString,* intercambiando el orden de dichos puntos coordenados. Ahora tenemos como resultado dos aristas en sentidos contrarios que atraviesan el mismo tramo de calle, usando exactamente los mismos puntos de origen y destino; así se evita una desconexión del grafo.

El proceso de incorporación de una nueva ruta se sigue bajo lo descrito anteriormente. La problemática que se presentó al agregar más rutas al modelo

fue al momento de marcar las intersecciones de un recorrido nuevo con uno ya existente, presentando desconexiones en el grafo.

Así como fueron creadas aristas sobre tramos ya levantados, fue tomado en cuenta el mismo método, sólo que ésta vez aplicado a las intersecciones. Si el nuevo recorrido (ruta A) que está siendo marcado, intersecta con un recorrido ya levantado previamente (ruta B) en un punto o vértice, se revisa el punto coordenado correspondiente en el archivo *kml* de B. El vértice deberá ser usado en ambas rutas, o mejor dicho, en los archivos *kml* de dos o más rutas que se estén intersectando, asegurándose que se haya copiado exactamente igual. Si el nuevo recorrido de A atraviesa B en una ubicación para la cual no exista un vértice, es decir en un punto medio de alguna arista de B, de igual manera se coloca este punto en el archivo *kml* de A. En B también se agrega dicho punto junto con dos aristas nuevas. Se realiza esta división porque A está “partiendo” la arista de B con un nuevo punto de intersección. Se debe tomar en cuenta que, si la arista recién dividida es de un solo sentido, serán dos aristas nuevas las que serán creadas en B; en caso de que el recorrido intersecte un tramo de calle donde se encuentren dos aristas en ambos sentidos, serán cuatro aristas las que deberán ser creadas en el archivo *kml*.

Esta misma metodología fue seguida para el levantamiento de las ocho rutas en Google Earth para nuestro modelo del STP.

Cada recorrido se guarda en un archivo por separado con extensión *kml*, para que sea importado al software *Mathematica*. Dentro del ambiente de este software se procesa la información para obtener el modelo matemático; el grafo dirigido ponderado. En la siguiente sección se hablará sobre el modelo computacional del STP.

3.4. Modelo computacional del sistema

El prototipo del STP de la Ciudad de Querétaro se construyó en el ambiente provisto por *Mathematica*, el cual ofrece un lenguaje de programación de alto nivel basado en el paradigma funcional. El prototipo modela computacionalmente el STP a partir del modelo matemático basado en un grafo dirigido ponderado conexo. *Mathematica* ofrece un conjunto de funciones especializadas que implementan diversos algoritmos de ruteo para calcular las distancias entre cada par de puntos o vértices del grafo. De estas funciones, la función *FindShortestPath* es la mayormente utilizada en este trabajo, la cual genera el itinerario del usuario del STP, basado en la ruta óptima, que le permite transitar desde un punto origen al punto destino a través de la red.

Los recorridos de cada una de las rutas punteadas previamente en *Google Earth* se guardan con la extensión *kml*, con el propósito de que el software *Mathematica* los importe de una forma simple. Haciendo uso del árbol de clases *kml* provisto por Google, una función de *Mathematica* se encarga de importar únicamente los datos que son importantes para el prototipo: la sección con el elemento *Geometry* de cada archivo es tratada por el software. El elemento *Geometry* contiene las coordenadas geográficas dadas en grados decimales del recorrido completo de la ruta. Inmediatamente al ser importado cada archivo, sus valores son asignados a una variable que contiene, dentro de una lista, todos los puntos coordenados en forma consecutiva.

Una característica de *Google Earth*, con respecto a las coordenadas geográficas, es que cuando las escribe en un archivo *kml* las guarda en el formato “longitud, latitud, elevación”. En general, las coordenadas geográficas normalmente se leen como “latitud, longitud”. Por lo mismo, el contenido de cada elemento de las listas -creadas al momento de importar los archivos *kml* de rutas- debe cambiar de posición. Además, el dato de la elevación es desechada al no ser requerido por el modelo computacional, porque al realizar el cálculo de

distancias no es necesario tomar en cuenta la elevación respecto al nivel del mar; posteriormente las coordenadas geográficas deberán ser sometidas a un proceso de normalización a coordenadas cartesianas. Dentro de las muchas funciones que integran el software *Mathematica*, existen algunas dedicadas a posicionamiento geográfico. *GeoPositionXYZ* es una de ellas. La salida de los parámetros que son pasados a la función, representa una posición en un sistema geocéntrico de coordenadas cartesianas. El objetivo de hacer la normalización es facilitar el manejo de la información de los vértices y aristas en el grafo, al utilizar coordenadas cartesianas en números positivos y no coordenadas geográficas, que son representadas por grados decimales. Cada variable que contiene una lista de coordenadas de una ruta es tratada con la función *GeoPositionXYZ*, para que todas las rutas levantadas tengan valores de coordenadas cartesianas.

Posteriormente las coordenadas normalizadas se utilizan para crear los vértices y aristas que forman el grafo del modelo computacional del STP de la Ciudad de Querétaro (Figura 3.5). Al ser los recorridos una sucesión consecutiva de puntos coordenados, el sentido de las aristas sigue el orden de los puntos formando un ciclo, iniciando en el punto de partida o "base" –en término coloquial del Transporte Público-, continuando con su recorrido hasta el punto más alejado del punto de inicio para luego regresar a la base.

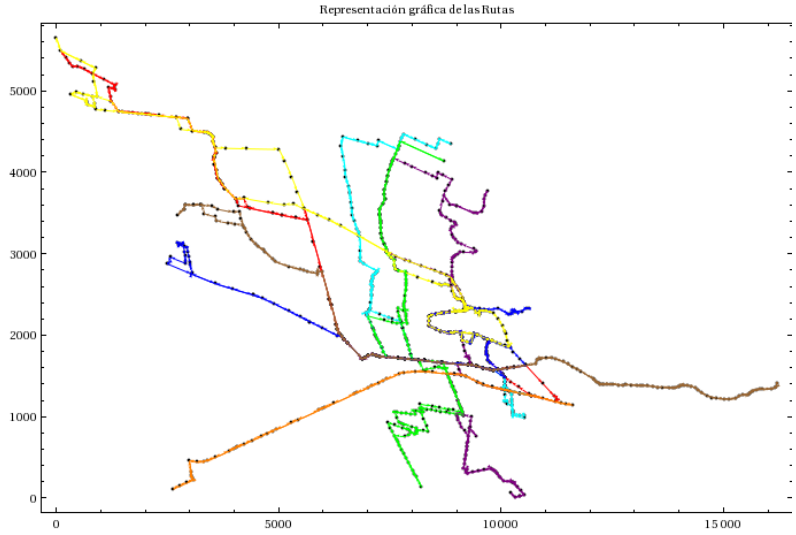


Figura 3.5. Representación del grafo del STP.

4. RUTAS ÓPTIMAS A TRAVÉS DE UNA RED MODELADA MEDIANTE UN GRAFO DIRIGIDO PONDERADO CONEXO

"La acechante sospecha de que algo podría ser simplificado es la fuente más rica del mundo de gratificantes retos."

-Edsger W. Dijkstra (1930 - 2002)

Muchas situaciones en el mundo real se pueden describir convenientemente en forma de un diagrama consistente en un conjunto de puntos y líneas que unen ciertos pares de estos puntos. Por ejemplo los puntos pueden representar personas con líneas que unen a los pares de amigos, o los puntos pudieran ser centros de comunicación y las líneas representan los enlaces de comunicación. Nótese que en ciertos diagramas, a veces estamos mayormente interesados en que dos puntos dados están unidos por una línea; la forma en la cual estos puntos están unidos, es intrascendente. Una abstracción matemática de situaciones de este tipo nos lleva al concepto de *grafo* (Bondy y Murty, 2008).

4.1. Teoría de Grafos

En el año de 1736, un artículo publicado por el matemático suizo Leonhard Euler marcó el inicio de la Teoría de Grafos. En dicho artículo, Euler dio solución al problema de "Los siete puentes de Königsberg" (Jungnickel, 2007). La ciudad de Königsberg (actual Kaliningrado, Rusia) estaba dividida por el río Pregel en cuatro zonas y estas zonas estaban comunicadas por siete puentes (Figura 4.1).

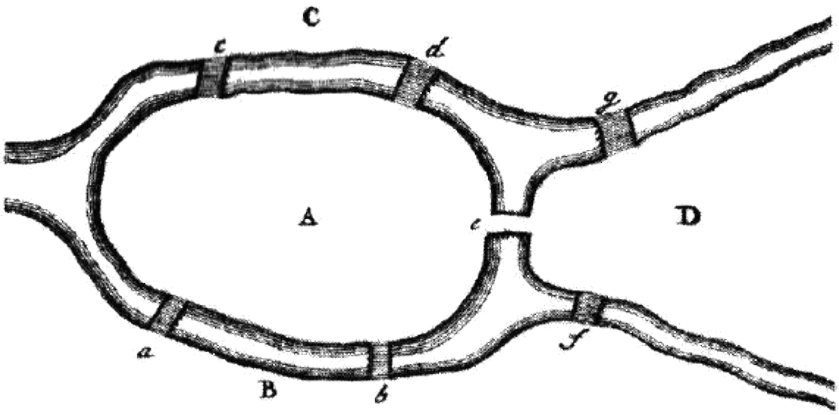


Figura 4.1. Los siete puentes de Königsberg.

El problema consistía en crear un circuito caminando a través de todos los puentes, pasando una sola vez por cada uno de ellos. Euler se dio cuenta que la forma de la isla y las cuatro zonas no eran importantes, sino la conectividad de las zonas. Cada zona la representó por medio de puntos (vértices) y los puentes por curvas que unían a cada punto (Figura 4.2).

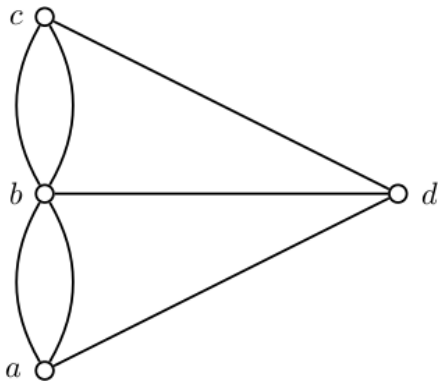


Figura 4.2. Grafo que representa los siete puentes y las cuatro zonas.

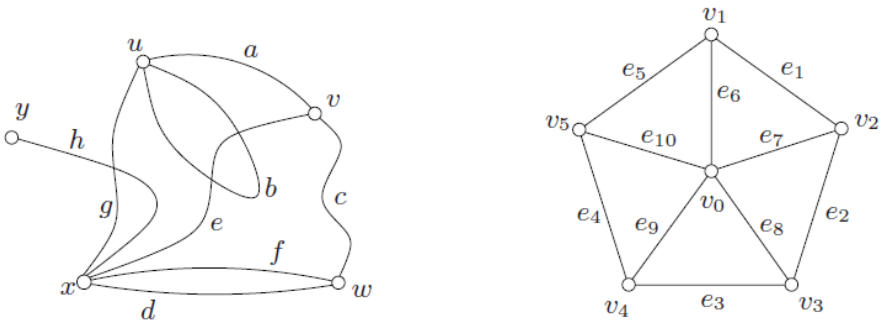
Si se quiere realizar el circuito caminando por todos los puentes, iniciaremos el tour por el vértice (punto) b, haciendo uso de dos de los cinco puentes que conectan con b, regresamos al mismo vértice por primera vez. Para el siguiente retorno a b ya se han utilizado cuatro puentes. Ahora dejamos el mismo vértice de nuevo, pero no hay posibilidad de regresar a b sin utilizar uno de los cinco puentes por segunda vez. Así se demuestra que el problema no tiene solución. Usando un argumento similar, se observa que también es imposible

encontrar cualquier circuito caminando el cual utilice cada puente exactamente una sola vez. Más aún, Euler demostró que existe una condición necesaria y suficiente para que un grafo arbitrario admita un recorrido circular: el número de aristas que inciden en cada uno de los vértices del grafo, debe ser par.

Matemáticamente, un grafo G se puede definir como un par (V, E), el cual consiste de un conjunto V ≠ ∅, y un conjunto E, conformado por subconjuntos de dos elementos de V. Los elementos de V son llamados *vértices*. Un elemento e = {a, b} de E es denominado *arista,* con los vértices a y b.

La mayoría de definiciones y conceptos en la Teoría de Grafos están representados gráficamente. Dicha representación nos ayuda a comprender muchas de sus propiedades. Cada vértice es indicado por un punto y cada arista por una línea que une dos puntos, los cuales representan sus extremos.

No hay una forma única y correcta de dibujar un grafo; las posiciones relativas de puntos que representan los vértices y las formas de las líneas que representan las aristas usualmente no tienen un significado. En la Figura 4.3 las aristas de G son curvas, mientras que las de H son segmentos rectos. El diagrama de un grafo solamente muestra la relación de incidencias que se mantiene entre sus vértices y aristas.



(G) (H)

Figura 4.3. Diagramas de los grafos G y H.

Los extremos de una arista se dice que son incidentes con el vértice y viceversa. Dos vértices que inciden con una arista en común son adyacentes, como lo son dos aristas que inciden en un mismo vértice; dos distintos vértices adyacentes son llamados vecinos. El conjunto de vecinos de un vértice υ en un grafo G es denotado por $N_G(\upsilon)$. En la Figura 4.4 se ilustran estos conceptos.

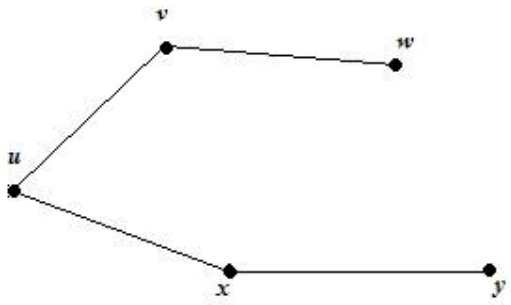


Figura 4.4. En el grafo anterior, *u* y *v* son vértices adyacentes. Las aristas *(u, v)* y *(v, w)* son adyacentes. Una sola arista incide en *y*.

Siguiendo con la representación gráfica, la dirección de una arista se indica al colocar una flecha dirigida sobre ella; de esta manera la estructura se denomina grafo dirigido o dígrafo (Figura 4.5). Cuando no importa la dirección de las aristas, la estructura G = (V, E) donde E es ahora un conjunto de pares no ordenados sobre V, es un grafo no dirigido (Grimaldi, 2004). En el presente trabajo, el grafo del STP de la ciudad es un grafo dirigido.

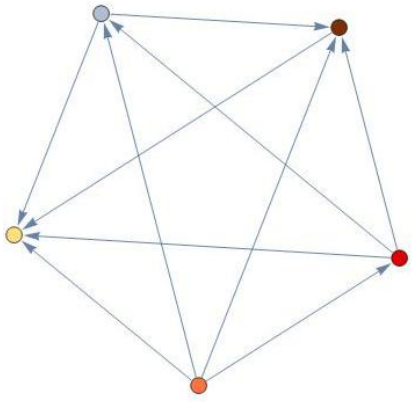
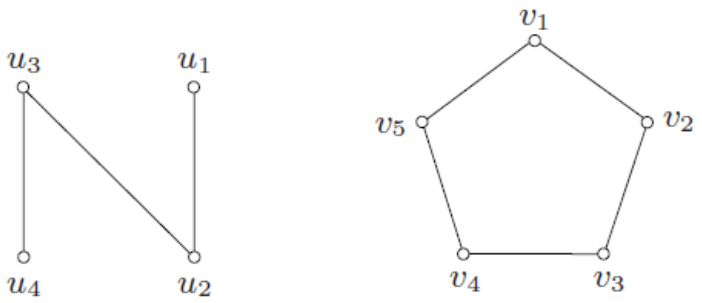


Figura 4.5. Ejemplo de un grafo dirigido.

Para muchas aplicaciones -especialmente aquéllas relacionadas a problemas de redes de tráfico vehicular y redes de transporte- es útil darle una dirección a las aristas del grafo, por ejemplo al indicar una calle de un solo sentido en el mapa de una ciudad. Tenemos que un grafo dirigido o dígrafo es el par G = (V, E) que consiste de un conjunto finito V y un conjunto E de pares ordenados (a, b), donde a ≠ b son elementos de V (Cormen, Leiserson, Ronald y Stein, 2001).

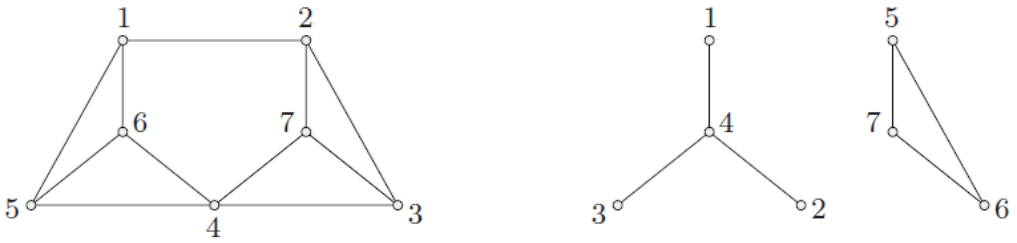
Un camino es un grafo simple cuyos vértices pueden ser ordenados en una secuencia lineal de tal forma que dos vértices son adyacentes si son consecutivos en dicha secuencia. Así mismo un ciclo en tres o más vértices es un grafo simple cuyos vértices pueden ser ordenados en una secuencia cíclica, de tal manera que dos vértices son adyacentes si éstos son consecutivos en la secuencia. La longitud de un camino o de un ciclo está dada por el número de sus aristas (Figura 4.6).



(a) (b)

Figura 4.6. (a) Camino de longitud tres y (b) ciclo de longitud cinco.

En la secuencia W := $v_0e_1v_1$ … $v_{\ell-1}e_\ell v_\ell$, cuyos términos son vértices y aristas en forma alternada de un grafo G (no necesariamente distintos), tales que v_{i-1} y v_i son los extremos de e_i, $1 \leq i \leq \ell$. Si $v_0 = x$ y $v_\ell = y$, decimos que W conecta a *x* con *y*, refiriéndonos a W como un camino xy. Los vértices *x, y* son los extremos del camino, siendo *x* el vértice inicial e *y* el vértice destino. Si para cualquier par de vértices a, b en un grafo G existe un camino, se dice que G es un grafo conexo (Figura 4.7) (Grimaldi, 2004).



(a) (b)

Figura 4.7. (a) Un grafo conexo y (b) un grafo disconexo.

El grado de un vértice υ en un grafo G, denotado por $d_G(\upsilon)$ es el número de aristas de G que inciden en υ, donde un lazo cuenta como dos aristas. En particular si G es un grafo simple, $d_G(\upsilon)$ es el número de vecinos de υ en G (Bondy y Murty, 2008).

Un árbol se define como un grafo conexo, no dirigido y sin ciclos. Así mismo un grafo no dirigido que no tiene ciclos pero posiblemente es disconexo es llamado bosque (Figura 4.8).

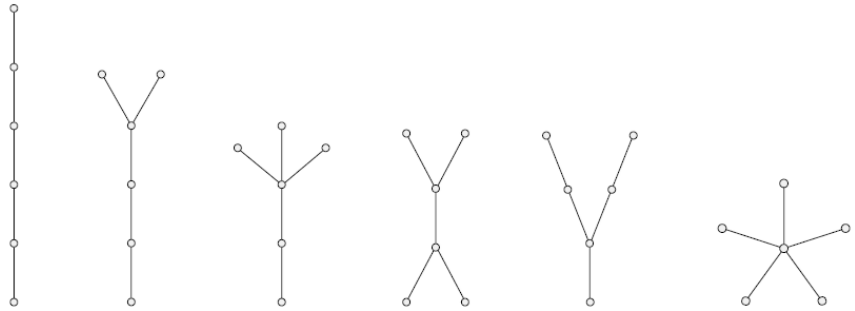


Figura 4.8. Árboles de seis vértices que forman un bosque.

4.2. Ruta más corta

Suponga que se tiene un grafo dirigido conexo sin lazos G = (V, E). A cada arista *e = {a, b}* de este grafo se le asigna un número real positivo al que se le denomina peso de *e*, denotado por *p(e)* o *p(a, b)*. Si $x, y \in V$ pero $\{x, y\} \notin E$, se define $p(x, y) = \infty$. Cuando un grafo G = (V, E) tiene las asignaciones de peso

anteriormente descritas se dice que el grafo es un grafo ponderado (Figura 4.9)(Grimaldi, 2004).

```
LayeredGraphPlot[{{1 -> 5, "7"}, {1 -> 6, "12"}, {2 -> 4,"6"}, {2
-> 6, "10"}, {3 -> 4, "5"}, {3 -> 5, "8"}, {4 -> 6,"20"}, {5 -> 6,
"4"}}, VertexLabeling -> True]
```

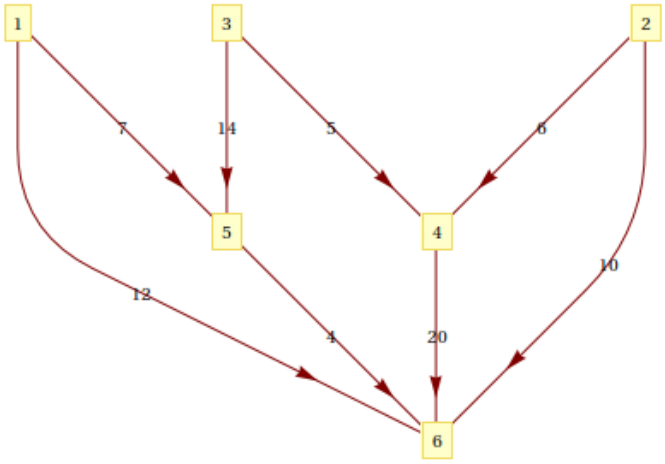


Figura 4.9. Grafo dirigido ponderado de 5 vértices y 8 aristas en *Mathematica*.

Calcular una ruta óptima en una red de transporte, desde un punto de origen a un punto destino en específico, es una de las aplicaciones de la Teoría de Grafos. Como se mencionó previamente, existen varias aplicaciones, como la planeación logística o la simulación de tráfico, que necesitan resolver un gran número de solicitudes para calcular rutas óptimas en un grafo que represente una red de transporte. A veces, inconscientemente hacemos uso de estas funcionalidades al planear viajes en automóvil o anticipar traslados en transporte público. El punto clave de este trabajo de investigación es calcular los costos de rutas óptimas entre pares arbitrarios de puntos de origen y destino.

Dado un sistema de carreteras de un solo sentido en una ciudad, por ejemplo, uno desearía determinar la ruta más corta entre dos diferentes lugares en específico dentro de una ciudad. Esto equivale a encontrar un camino dirigido con un peso mínimo que conecte a dos vértices específicos en el grafo dirigido

ponderado cuyos vértices son los cruces de calles, y cuyas aristas son las calles y avenidas que unen a estos cruces.

Nos referimos al peso de un recorrido en un grafo dirigido ponderado como su longitud. De la misma manera, por ruta *(x, y)* más corta se hace referencia a la distancia desde *x* a *y*, denotada por *d(x, y)*. Por ejemplo, la ruta indicada en el grafo de la Figura 4.10 es la ruta *(x, y)* más corta entre los puntos *x* e *y*, y *d(x, y) = 3 + 1 + 2 +1 + 2 + 1 + 2 + 4 = 16* es la longitud de ese camino.

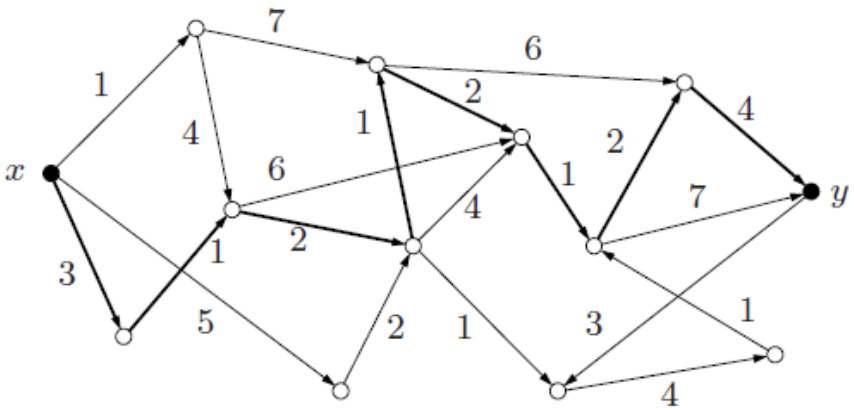


Figura 4.10. Ruta más corta (x, y) en un grafo dirigido ponderado.

Cuando todos los pesos son positivos, el problema para obtener la ruta más corta puede ser resuelto eficientemente por medio de una búsqueda de anchura. Una búsqueda de anchura comienza en un vértice inicial, el cual pasa a ser el vértice activo, se visitan los vecinos del vértice activo antes de pasar al siguiente. Una vez que hayan sido visitados todos los vecinos del vértice activo, se toma como nuevo vértice activo el primer vértice visitado después del vértice activo actual. De una forma similar funciona el algoritmo de Dijkstra (Jungnickel, 2007).

4.3. Algoritmo de Dijkstra

En el ámbito de las redes de transporte, el algoritmo de Dijkstra es el algoritmo clásico para la planeación de itinerarios, manteniendo un arreglo de

distancias tentativas desde el nodo origen al resto. El algoritmo fija los nodos de la red de transporte en el orden de su distancia al nodo de origen; de esta manera el algoritmo termina cuando el nodo destino es visitado; siendo el orden de complejidad $O(n^2)$ (Sanders y Schultes, 2007).

Definamos el problema de la siguiente forma: partiendo de que G = (V, E) es un grafo dirigido ponderado con pesos positivos, dado un vértice *s* como origen, obtener las rutas más cortas hacia el resto de los vértices de V.

El algoritmo de Dijkstra mantiene los siguientes conjuntos: un conjunto de vértices *S* el cual contiene los vértices para los que la distancia más corta desde el origen ya es conocida, comenzando con $S = \varnothing$; un conjunto de vértices $Q = V\text{-}S$ que mantiene, para cada vértice, la distancia más corta desde el origen pasando a través de los vértices que pertenecen a *S*. Para guardar las distancias temporales se usa un vector *D*, donde se almacena la distancia desde *s* al vértice *D[i]*.

Se comienza por extraer del conjunto *Q* el vértice *u* cuya distancia temporal *D[u]* sea la menor entre el vértice de origen *s* y *u*, por lo tanto ya no es posible encontrar un camino más corto desde *s* hasta *u* utilizando algún otro vértice del grafo. Se inserta *u* en *S* para el vértice donde se ha calculado la ruta más corta desde *s*. Posteriormente se actualizan las distancias temporales de los vértices de *Q* adyacentes a *u* cada que mejoren usando un nuevo camino. Estos pasos se repiten hasta que *Q* quede vacío, teniendo en *D*, para cada vértice, la distancia más corta desde el origen (Merrifield, 2010).

4.4. Características estructurales de la red

En el lenguaje *Mathematica* se programó una parte de la red del STP de la Ciudad de Querétaro representada por un grafo dirigido ponderado, constando de ocho rutas que atraviesan la ciudad en diferentes extremos (Figura 4.11). Tal como se mencionó en el Capitulo 2, las rutas programadas atraviesan la ciudad de norte a sur, de este oeste, de noreste a suroeste y de noroeste a sureste.

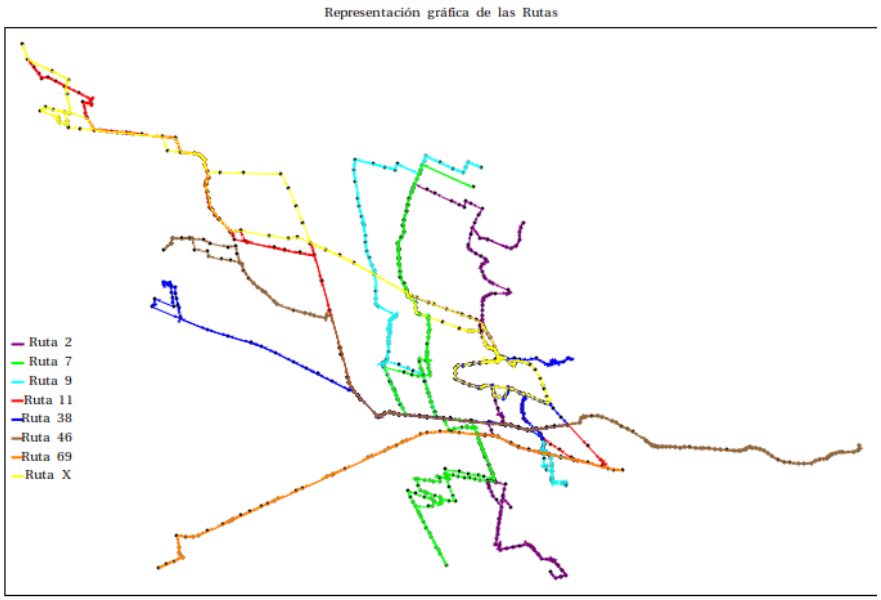


Figura 4.11. Representación de las rutas del prototipo de STP de la Ciudad de Querétaro en un grafo dirigido ponderado conexo.

El grafo en su totalidad consta de 1158 aristas con 682 vértices (Figura 4.12). Del total de aristas, la longitud mínima que se presentó fue de 20 metros, y la longitud máxima de 1018 metros. El promedio de longitud total por arista es de 180.26 metros. En la Figura 4.13 se muestra una gráfica comparativa de la longitud de todas las aristas que conforman el grafo parcial del STP.

Cabe mencionar que las aristas que tienen una longitud total muy pequeña, es debido a que la gran mayoría une puntos de ajuste o puntos de inflexión, siguiendo los criterios de punteo descritos en la sección 3.3. Como anteriormente se mencionó, los vértices de nuestro modelo no están colocados en puntos de ascenso y descenso necesariamente, puesto que las unidades de transporte no siempre respetan dichas paradas. De ser agregado este nuevo criterio de punteo se incrementaría notablemente la complejidad del modelo desde el momento del levantamiento geográfico de las trazas. Otro motivo por el cual no se realizó de ésta manera fue a la falta de información acerca de todas y cada una de las paradas, cuya ubicación geográfica exacta haya sido registrada previamente.

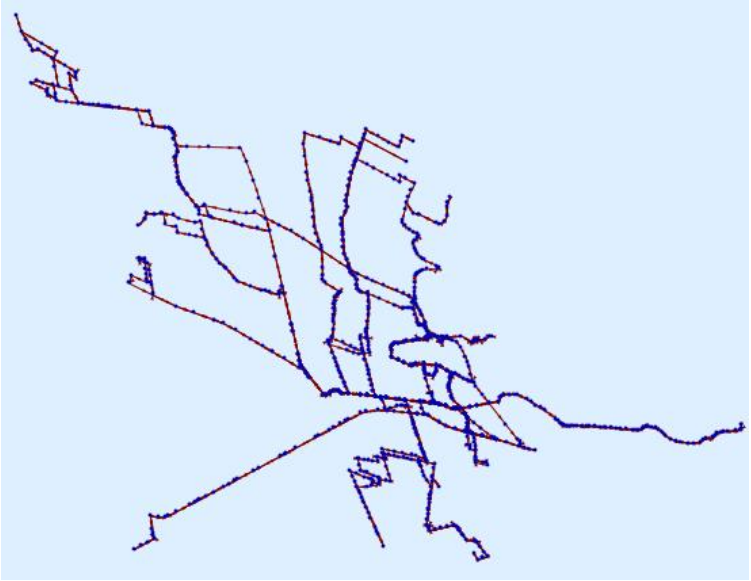


Figura 4.12. Grafo dirigido ponderado del STP.

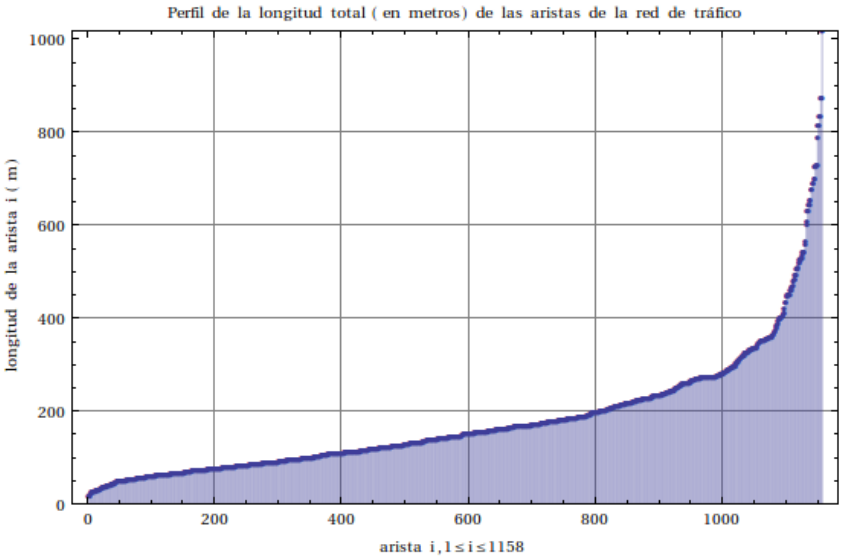


Figura 4.13. Longitudes totales de las aristas.

Se presume que para cada uno de los 682 vértices existe una ruta óptima hacia todos los demás. De esta manera, y haciendo el cálculo de 682^2-682, se tienen 464442 posibles rutas óptimas en el grafo. Para dicho número de rutas óptimas, la longitud promedio de cada ruta es de 8.82 Km.

En la Tabla 4.1 se exponen las longitudes totales de los recorridos completos, así como el número de vértices del grafo que pertenecen a cada ruta, para las ocho rutas incluidas en el prototipo del STP de la Ciudad de Querétaro. Siendo 273.6 kilómetros la distancia total de la red representada, dada por la suma total de cada una de las rutas sin analizar las aristas repetidas en los recorridos.

Tabla 4.1. Longitudes totales de cada ruta en el grafo del STP.

Ruta	Vértices	Longitud total (Km)
2	194	39.7
7	148	34.2
9	102	31.5
11	87	38.02
38	110	31.4
46	128	36.3
69	59	25.01
X (110)	114	37.4

La forma en cómo *Mathematica* calcula la ruta óptima es con la función integrada *FindShortestPath*, la cual encuentra la ruta más corta desde un vértice de inicio a un vértice destino en un grafo G. Si a la función se le indica como parámetro un grafo dirigido no ponderado, cada arista se interpretará con un peso de 1; por otra parte si se indica un grafo dirigido ponderado será tomado en cuenta el peso que se le dé a cada arista. La función tiene una opción para indicarle el método a utilizar. Este parámetro se refiere al algoritmo que se usará para realizar el cálculo, estando entre las opciones el algoritmo de Dijkstra y el algoritmo de Bellman-Ford (Figura 4.14). Para el prototipo fue utilizado el algoritmo de Dijkstra, al ser un algoritmo que soporta pesos positivos, considerando quer las aristas del grafo tienen pesos positivos.

```
(* Coordenadas de los vértices de origen y destino *)
vertIni = {3089.2470929038245`, 3591.95157314837`};
vertFin = {11455.068803029368`, 1149.9400465991348`};
```

```
(* Definición de shortPath usando FindShortestPath *)
shortPath[g_, vertIni_, vertFin_] :=
   FindShortestPath[
      g, vertIni, vertFin, Method -> "Dijkstra"]

(* Uso de shortPath, teniendo al grafo como primer
   parámetro y los vertices de inicio y destino como
   parámetros 2 y 3 respectivamente *)
shortPath[g1, vertIni, vertFin]
```

Figura 4.14. Uso de *FindShortestPath* para el cálculo de la ruta más corta entre dos vértices del grafo del STP.

Haciendo uso del código de la Figura 4.14 para el grafo del STP, se obtienen los resultados que se observan en la Figura 4.15. La ruta más corta está resaltada. Cada punto de color indica el vértice donde una ruta distinta intersecta a la ruta óptima, por lo que en dichos puntos un usuario del STP puede transbordar de una ruta a otra distinta.

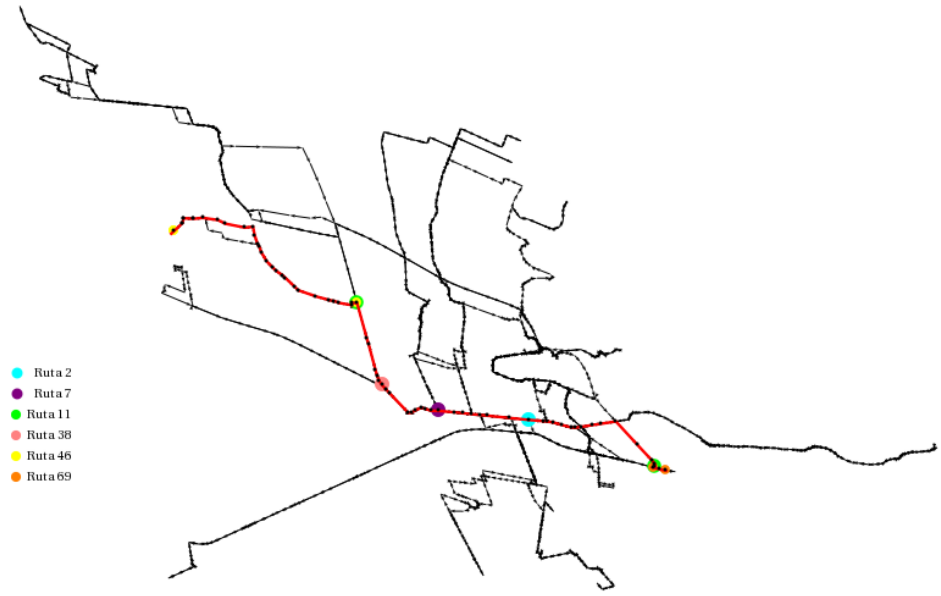


Figura 4.15. Ruta más corta generada con *FindShortestPath*.

5. PROTOTIPO DE UN SISTEMA DE INFORMACIÓN PARA CALCULAR ITINERARIOS ÓPTIMOS: UN CASO DE ESTUDIO APLICADO AL STP DE LA CIUDAD DE QUERÉTARO

"Sólo podemos ver un poco hacia adelante, pero lo suficiente para ver que hay un montón de cosas por hacer."

-Alan Turing (1912 – 1954)

En los últimos años se ha venido presentando un crecimiento importante en el uso de tecnologías y servicios basados en web, esto debido a las facilidades que se tienen hoy en día para acceder al uso de una computadora conectada a la Internet. No sólo el uso de las computadoras personales que tienen conexión a la Internet se ha incrementado, sino que también ha surgido una gran variedad de dispositivos que la sociedad está consumiendo; entre ellos, los más destacados son las *tablets* y los *smartphones*, todos con la posibilidad de estar conectados a la web vía las tecnologías para comunicaciones inalámbricas *WiFi* o *3G*.

El incremento de la disponibilidad de dispositivos portables de cómputo, además de la impresionante infraestructura inalámbrica actual, ha provocado el desarrollo y despliegue de una amplia gama de nuevas aplicaciones y servicios. Las aplicaciones orientadas a la localización son un ejemplo notable de esa nueva clase de aplicaciones (Koshimoto, Bromage, Petkov y Obraczka, 2009).

La demanda de la sociedad actual de equipos móviles y servicios basados en web ha desencadenado la creación de un sinfín de sitios y productos destinados para todo tipo de personas y dispositivos. Los servicios que últimamente han presentado un mayor impacto en los consumidores a nivel

mundial son aquéllos que utilizan mapas y geolocalización, haciendo uso de la Internet o de las redes de telefonía celular para que, con base en la ubicación física del usuario, puedan ofrecerle a éste algún tipo de información de su interés.

Entre los servicios que usan geolocalización existen los que están enfocados a proporcionar información geográfica, como son carreteras, ciudades, calles, puntos de interés, comercios, entre otros. La planeación de itinerarios también se ofrece en los servicios de estos sitios, tal y como se hace mención en el Capítulo 2. Las aplicaciones en Internet sobre mapas son una potente herramienta para todos los usuarios; sobre todo proporcionan una forma nueva y útil de utilizar, en la mayoría de los casos, un vasto material cartográfico. De esta manera, no es necesario tener mapas en papel, debido a que toda la información se encuentra “en línea”.

5.1. Conceptos y ambiente de desarrollo

La cantidad de usuarios que utilizan Internet cada día ha obligado a las empresas desarrollan servicios basados en la web a proveer sus herramientas a los programadores, para que puedan hacer uso de sus servicios libremente mediante *APIs*. Tal es el caso de *Google Inc.*, que ofrece acceso libre y gratuito a una gran variedad de sus productos y herramientas de desarrollo; entre ellos *Google Maps*, producto en el cual está basado parte del prototipo desarrollado en esta etapa del trabajo.

Debido al constante y acelerado desarrollo de la web, las tecnologías mismas que la hacen posible se encuentran también en continua evolución y desarrollo. Uno de los lenguajes mayormente utilizados en la web es el lenguaje de programación interpretado *JavaScript,* el cual es ejecutado en el lado del cliente o navegador web, donde este lenguaje esta implementado por defecto, permitiendo el manejo de elementos y datos en páginas dinámicas (también llamadas aplicaciones web). *JavaScript* no es un lenguaje fuertemente tipado, las

variables declaradas no se asocian directamente a uno de los seis tipos de datos: *number*, *Date*, *string*, *function*, *array* u *Object* (Figura 5.1); a su vez permite la manipulación de los elementos contenidos en el sitio web, a través del *DOM* (*Document Object Model*).

```
function factorial(n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

Figura 5.1. Función recursiva en *JavaScript* donde el parámetro no es un tipo de dato en específico.

Dentro de la evolución del lenguaje *JavaScript*, dos herramientas destacan entre las técnicas y métodos para el desarrollo de proyectos en base a este lenguaje, como lo son *AJAX (Asynchoronous JavaScript and XML)* y *JSON (JavaScript Object Notation)*, por mencionar algunos.

El término *AJAX* se utiliza para representar a un grupo de métodos y tecnologías web, las cuales son usadas para desarrollar una aplicación web generalmente del lado del cliente, entre los cuales se incorporan *HTML*, *CSS*, *JavaScript* y *XML*. Con *AJAX* las aplicaciones web son capaces de enviar y recuperar datos de un servidor en segundo plano de forma asíncrona; sin interferir con la vista o el comportamiento de la página. Esto permite incorporar contenido remoto (obtenido del servidor) sin necesidad de cargar la página completa con lo cual se aligera la carga de transacción entre cliente y servidor, y al mismo tiempo los usuarios se encuentran con interfaces web más rápidas y cómodas de usar.

AJAX utiliza diversos formatos para la transferencia de datos entre el servidor y la aplicación web, uno de estos es *JSON*. *JSON* es un formato ligero basado en texto para el intercambio de información en Internet, derivado del

lenguaje *JavaScript* para representar estructuras de datos y conjuntos de arreglos que forman un objeto. Es usado frecuentemente para la transferencia de datos entre un servidor y aplicaciones web. Se destaca por ser un estándar abierto, ligero, independiente de lenguajes, con una estructura de datos y sintaxis fácil de interpretar y escribir.

El formato *JSON* está construido en dos estructuras: la primera consta de una colección de pares nombre/valor; en diferentes lenguajes esto es conocido como un objeto, struct, diccionario o tabla hash. La segunda estructura está formada por una lista ordenada de valores, que en otros lenguajes son conocidos como un arreglo, vector, lista o secuencia (Figura 5.2).

```
{
     "nombre" : "John",
     "apellido" : "Doe",
     "edad" : 22,
     "direccion"  :
     {
         "calle" : "Zaragoza 25 nte",
         "ciudad" : "Querétaro",
         "estado" : "QRO",
         "codigoPostal" : "76000"
     },
     "numeroTelefono":
     [
         {
           "tipo"  : "casa",
           "numero": "442 256-7585"
         },
         {
           "tipo"  : "fax",
           "numero": "442 236-7519"
         }
     ]
 }
```

Figura 5.2. Objeto *JSON* que representa a una persona.

Como ya se mencionó anteriormente, el extenso uso y gran popularidad que presenta el lenguaje *JavaScript* ha impulsado a muchos programadores a crear librerías basadas en dicho lenguaje de programación, las cuales están

enfocadas a ahorrar tiempo en el desarrollo de sitios. Una de las librerías ampliamente utilizadas actualmente es *JQuery* debido a su licencia de libre uso y ser de código abierto, pero sobre todo a la facilidad de implementación.

Las principales funcionalidades que ofrece *JQuery* son el seleccionar los elementos de un sitio, crear animaciones, manejar una gran variedad de eventos (Figura 5.3); también incluye funciones para el desarrollo de aplicaciones web con *AJAX* y *JSON*.

```
<body>
   <p>Primer párrafo</p>
   <p>Segundo párrafo</p>
   <p>Tercer párrafo</p>

   <script>
      /* Manejador del evento 'click' */
      $("p").click(function(){
         /* Animación */
         $(this).slideUp();
      });

      /* Manejador del evento 'hover' */
      $("p").hover(function(){
         $(this).addClass("hilite");
      }, function(){
         $(this).removeClass("hilite");
      });
   </script>

</body>
```

Figura 5.3. Código en *JQuery* para el manejo de eventos y generar una animación.

Al principio de este capítulo se hizo mención a las aplicaciones web que trabajan con mapas y geolocalización; esto fue porque los mapas e imágenes por satélite actualmente facilitan la búsqueda de direcciones y localización y ubicación de establecimientos como restaurantes, museos, entre otros en un mapa.

Sin duda el referente de estas aplicaciones son los mapas y, como se hizo mención, las imágenes tomadas vía satélite. Una ventaja de los mapas es su

simplicidad de uso para un usuario común. Los usuarios pueden controlarlos simplemente arrastrando el cursor y desplazando el cursor para acercar o alejar la vista.

Hasta el día de hoy, la aplicación más conocida en cuanto a mapas es, con un arrasante índice de uso, *Google Maps*, un servidor de aplicaciones de mapas online. Sin embargo, la parte interesante de este servicio, y también la más atractiva para el presente trabajo, es su *API*. El cual consiste en un conjunto de herramientas para el programador que le permiten incluir, manipular y hacer operaciones con los mapas de *Google* en sus propias aplicaciones. Esto incluye el manejo del *DOM* generado por Google Maps, y los eventos que el usuario genera en la interfaz, tales como *clicks*, doble *clicks*, etc.

Como muchas otras aplicaciones de *Google*, el servicio de mapas utiliza *JavaScript* de manera extensa; cabe mencionar que la forma en cómo el servicio controla la transferencia de datos entre el servidor y los clientes es por medio de *JSON*.

Google provee una referencia o documentación para el uso de su *API* de mapas. En ella se describe el objeto *Map*, el cual es la representación abstracta de un mapa, así como los controles para manipular el comportamiento de *Map*, las capas que se sobreponen en el mapa (*overlays*), entre ellas una línea, un icono, una animación, etc. Las partes importantes y de interés para este trabajo son el manejo de eventos, aquellas que indican la manipulación del objeto *Map* y las capas para mostrar una línea, la cual servirá para indicar el recorrido del itinerario resultante que se pretende generar (Figura 5.4).

```
var map;
function initialize() {
   var myLatlng = new google.maps.LatLng(
      -25.363882,131.044922
   );
   var myOptions = {
      zoom: 4,
      center: myLatlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
   }
   map = new
      google.maps.Map(
         document.getElementById("map_canvas"), myOptions
   );

   var marker = new google.maps.Marker({
      position: myLatlng,
      map: map,
      title:"Click!"
   });
   google.maps.event.addListener(marker, 'click',
   function() {
      map.setZoom(8);
   });
}
```

(a)

(b)

Figura 5.4. (a) Código *JavaScript* invocando funciones de *Google Maps API*. (b) Mapa de *Google Maps* mostrando el resultado del código.

Las funciones, librerías y el lenguaje de programación *JavaScript* que hasta este punto han sido descritos se integran en el desarrollo del prototipo basado en web.

El servidor sobre el cual está montado el prototipo es el servidor *Apache Tomcat 6.0*, el cual es de código abierto y no requerir licencia alguna para su uso. *Apache Tomcat* es un servidor definido como un *Servlet container*. Este servidor almacena componentes desarrollados bajo la especificación web de Java, como son los *Java Servlets* y *Java Server Pages (JSP)*, proveyendo un ambiente apropiado para ejecutar código *Java*, un lenguaje de programación orientado a objetos.

El grafo del STP de la Ciudad de Querétaro modelado en *Mathematica* se maneja de forma bastante experimental con la librería *JGraphT*. Siendo una librería escrita en su totalidad en el lenguaje Java, permite ser ejecutada en el servidor *Apache Tomcat*. *JGraphT* contiene objetos y algoritmos matemáticos para la representación y manejo de grafos. Algunos de los tipos de grafos que *JGraphT* soporta son los grafos dirigidos y no dirigidos, ponderados, grafos “no modificables” para que sean de sólo lectura, subgrafos, etc. Así mismo, *JGraphT* también integra algoritmos para generar rutas más cortas, son el algoritmo de Bellman-Ford y el algoritmo de Dijkstra, siendo este último de gran importancia para nuestro prototipo.

Desde la interfaz gráfica del prototipo, el usuario debe hacer un par de *clicks* sobre el mapa, indicando el punto de inicio y el punto de destino para su recorrido. Ambos puntos coordenados son enviados al servidor para ser procesados sobre el grafo realizado con *JGraphT*. La librería recibe la solicitud y realiza el cálculo de la ruta más corta, así el resultado es enviado de regreso al cliente, para que el usuario obtenga el itinerario de forma gráfica sobre el mapa.

5.2. Implementación del prototipo basado en web

Para un usuario común, un sistema efectivo y de simple uso es aquél que no le es difícil de manejar, con una interpretación sencilla en los resultados de búsqueda y con una interfaz simple. Por lo tanto, siguiendo esta premisa, la interfaz principal del prototipo fue diseñada para mostrar al usuario los elementos mínimos y esenciales en los resultados de su petición de la ruta óptima.

Se tiene pensado en que la primera impresión que el usuario debe tener ante el sistema es el de una interfaz limpia y clara, dejando fuera múltiples botones y formas que otros sistemas incluyen en sus interfaces, haciéndolos confusos y molestos para un usuario que desea hacer una consulta rápida. La única instrucción que el sistema le indica al usuario es hacer un par de *clicks* sobre el mapa para calcular su itinerario. En esta primera versión del proyecto no se añadieron elementos como cajas de texto para obtener direcciones escritas por el usuario; se infiere que el usuario tiene una idea acerca de la ubicación de sus puntos de origen y destino, de tal forma que con sólo ver el mapa de la ciudad sea capaz de hacer una consulta. En una versión posterior pudiera integrarse dicha funcionalidad.

La pantalla principal se encuentra dividida en dos secciones (Figura 5.5). Al costado derecho se sitúa un mapa de *Google Maps*, el cual está visible todo el tiempo. Del lado izquierdo se encuentra una columna en donde se despliega el itinerario una vez que es generado. La única acción que el usuario da al sistema es un par de *clicks* para conocer los puntos de inicio y destino, no es necesario escribir direcciones, abrir otras secciones o navegar entre ventanas.

Cuando se indican ambos puntos, de inicio y destino, se obtienen las coordenadas geográficas de dichos puntos sobre el mapa. Las coordenadas se envían al servidor como parámetros, para así comenzar con el primer paso del cálculo de la ruta óptima.

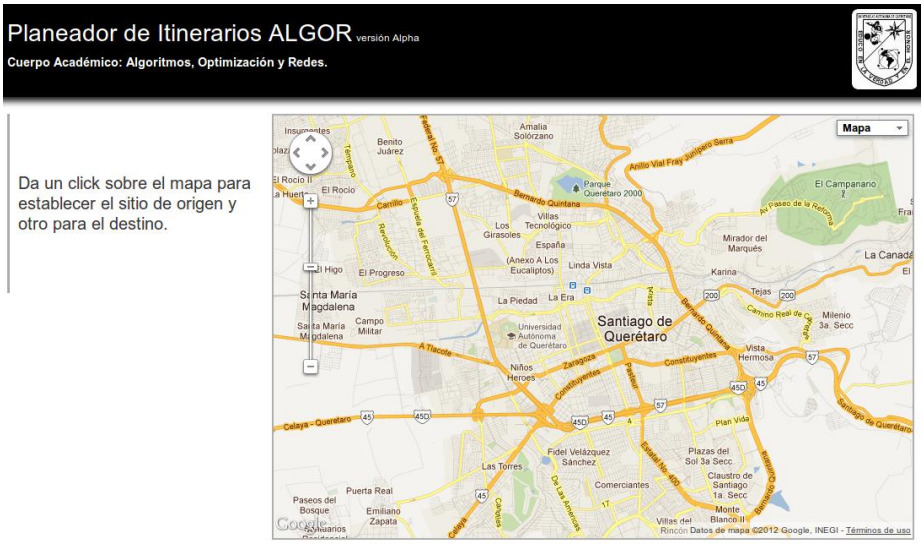


Figura 5.5. Interfaz principal del proyecto en web.

El punto exacto donde el usuario le indica al sistema el inicio o el destino puede coincidir o no con la ubicación de un vértice del grafo del STP. Para asegurar que el punto del usuario coincida con un vértice se utiliza la fórmula para calcular la distancia Euclidiana entre dos puntos. Las coordenadas del punto del usuario son evaluadas para cada punto del grafo, almacenando temporalmente en un arreglo las distancias resultantes. El vértice más cercano al *click* del usuario será aquél que tenga la distancia mínima. Es probable que llegue a darse un caso en el cual no se genere ningún resultado, dando aviso al usuario para que cambie sus ubicaciones sobre el mapa.

El proceso anterior se repite para los dos puntos del usuario. Conforme el grafo aumente su tamaño, será necesario implementar algún otro algoritmo más eficiente para calcular el vértice más cercano, de esa manera no se hará el cálculo para cada vértice del grafo.

Los vértices más cercanos que fueron obtenidos mediante el cálculo de la distancia Euclideana, junto con el grafo del STP, son pasados como parámetros al constructor de la clase *DijkstraShortestPath* de la librería *JGraphT*, la cual implementa el algoritmo de Dijkstra con estos tres objetos. La clase regresa la

ruta óptima entre ambos vértices, siendo este subgrafo sobre el cual se manipulan sus resultados para ser enviados de regreso al cliente.

Los resultados que se envían al cliente se dividen en dos objetos y un valor, todos de tipo *JSON*, contenidos en otro objeto del mismo tipo. El valor consta únicamente de la distancia total en kilómetros de la ruta más corta. El primer objeto es un arreglo de puntos coordenados, cada elemento se compone del par de valores latitud y longitud. La sucesión de puntos es manejada con funciones propias de la *Google Maps API* para mostrar en el mapa la ruta más corta a través de las calles, en forma de una línea coloreada. El segundo objeto también es un arreglo de puntos; a diferencia del anterior, este último contiene los puntos en los cuales existe un cambio de ruta. Es decir, en el conjunto de vértices de la ruta más corta se compara el atributo *route* de cada objeto vértice; cuando existe un cambio o dicho atributo es distinto al anterior, el vértice actual es agregado al nuevo arreglo de tipo *JSON*. El objeto contiene el arreglo con los vértices que indican al usuario donde transbordar o le informan de un conjunto de rutas posibles para continuar con su recorrido.

Para una mejor interpretación de los resultados por parte del usuario, el primer punto y el último, junto con cada uno de los puntos que indican donde transbordar o continuar, son desplegados con la dirección aproximada de la calle en donde se encuentran. Las direcciones se obtienen realizando peticiones a otro servicio de *Google*, llamado *Google Geocoding*, por lo que al momento de desplegar los resultados, el sistema pudiera tardar en indicar cada paso; esto debido al tiempo que se tarda el servidor del servicio en responder a nuestra aplicación.

Como primera etapa del prototipo, no existió un diseño exhaustivo siguiendo heurísticas de usabilidad o algún patrón de diseño. Pensando como usuario común que necesita percibir instrucciones simples y concisas, cada paso de la columna de resultados está compuesto por un icono, el texto que indica las

rutas y la dirección de la calle. Al mismo tiempo, los iconos de las instrucciones son dibujados en el mapa (Figura 5.6).

Figura 5.6. Resultados de una consulta en el prototipo.

En caso de que el usuario desee efectuar una nueva consulta, únicamente debe arrastrar el par de iconos que posicionó sobre el mapa al inicio de la consulta anterior, colocándolos en las nuevas ubicaciones donde necesita saber el itinerario de la ruta óptima. De esta forma la aplicación sabrá que es necesario calcular una nueva ruta óptima y arrojará los resultados correspondientes a esta nueva consulta. El resultado es calculado automáticamente, esto es logrado por la manipulación de eventos del *API* de *Google Maps*.

6. CONCLUSIONES

“La cosa más difícil es dormir por la noche sabiendo que hay tantas cosas urgentes que necesitan ser hechas. Existe una brecha enorme entre lo que sabemos que es posible con las máquinas de hoy en día y lo que hemos sido capaces de terminar.”

-Donald Knuth (1938 -)

Los algoritmos para calcular la ruta óptima son indispensables para resolver problemas de redes de transporte. Muchas técnicas de ruteo han tenido un avance tremendo en los últimos años; algunos de sus resultados pueden verse en sitios web para planear rutas, en aparatos que utilizan *GPS*, redes de computadoras, logística, etc.

En ciudades grandes donde el tráfico es a menudo más denso, trasladarse en automóvil puede resultar altamente costoso, por lo que se requiere una mejor planeación. La mayoría de las veces, se invierte mucho tiempo valioso en embotellamientos y en buscar un lugar dónde estacionarse. A pesar de esto, el uso de transporte público no es elegido, dado que en gran frecuencia es percibido como complejo, inflexible e ineficiente.

Los sistemas de comunicaciones y la tecnología -para nuestro caso el Internet- comienzan a tener un gran impacto en la eficiencia y la usabilidad del transporte público. Como bien lo puede ser la contabilización de automática de usuarios para ocasionar un aumento o retiro de unidades de transporte, monitoreo del tráfico vehicular, predicción de llegadas y salidas de acuerdo a la hora del día, información dinámica al pasajero y el “re-ruteo” dinámico -como lo hacen algunos

dispositivos *GPS*-, son sólo algunos ejemplos para procesos que requieren sistemas de comunicaciones y algoritmos.

Si bien existen algunos sistemas basados en web que permiten a una persona planificar su itinerario para desplazarse en la Ciudad de Querétaro, ninguno de ellos ofrece un itinerario basado en la red de trasporte público. Aunque este proyecto, que implementa un prototipo consistente en solamente ocho rutas de las más de 90 que dan servicio de transporte en la ciudad, lo que se busca es demostrar la factibilidad de contar con una herramienta computacional que sería de utilidad para un gran sector de la población en la Ciudad de Querétaro.

Durante este trabajo se ha tenido en cuenta que en diversos países desarrollados existen aplicaciones como las evaluadas en el segundo capítulo. Sin embargo, debido al alcance de esta tesis el prototipo desarrollado tiene limitantes cuyos símiles extranjeros no. Un ejemplo claro de estas limitantes es que en nuestro país es difícil encontrar un STP cuyos horarios y frecuencias de paso sean rigurosamente respetados. También, por mencionar otro punto, los lugares asignados para el ascenso y descenso de pasajeros.

Es evidente que quedan abiertas varias líneas de trabajo a futuro, como lo sería un estudio exhaustivo que demuestre si la red de transporte público de la Ciudad de Querétaro se encuentra obsoleta, al presentar una superposición excesiva de rutas en algunas calles y avenidas, dando como consecuencia embotellamientos, problemas ambientales, altos costos de traslados en cuanto a tiempo. Otros puntos importantes con posibilidad de ser mejorados son la implementación de un algoritmo para minimizar el número de veces que el usuario debe transbordar, así como los algoritmos mencionados en el Capítulo 4 para obtener los puntos más cercanos a los *clicks* indicados por el usuario, para evitar la necesidad de hacer un cálculo en todos y cada uno de los vértices del grafo.

Actualmente ha tomado fuerza otro concepto de interés para las redes de transporte; éste es el concepto de administración dinámica de los sistemas de transporte. Los avances que presenta el concepto han traído consigo un nuevo interés hacia el estudio de problemas de rutas más cortas. Sin embargo, también viene acompañado de otra característica, el peso de cada enlace entre dos puntos de la red de transporte depende de un cierto tiempo asignado (Chabini, 1997).

Al modelo computacional desarrollado en este trabajo de tesis se le podrían agregar diversas métricas para hacerlo más robusto, como es el factor del tiempo, agregando a cada ruta de transporte sus horarios y frecuencia de paso. También existe la posibilidad de crear un algoritmo a la medida para el caso particular de la Ciudad de Querétaro, agregando factores como el tráfico vehicular, tarifas, calificaciones por parte del usuario dependiendo de la compañía transportista, etc.

Finalmente, la nueva Ley de Movilidad para el Transporte del Estado de Querétaro establece que permitirá la ayuda de distintas tecnologías para el seguimiento de los recorridos, horarios, intervalos, respeto a las paradas e inclusive límites de velocidad. Esto puede dar paso al desarrollo de robustas herramientas computacionales y algoritmos enfocados al STP de la ciudad, al contar con el monitoreo de cada una de las distintas métricas que se mencionaban anteriormente. Otro punto que en esta Ley se menciona es que la ciudadanía tiene el derecho al acceso de toda información relativa al STP. Del mismo modo, el trabajo aquí presentado podría adecuarse a distintos puntos de la Ley y del STP reestructurado, para ofrecer una mayor eficiencia a la movilidad del usuario.

REFERENCIAS

Association for Computing Machinery. (2011). *ACM TechNews*. Recuperado en Febrero de 2011, de http://technews.acm.org/archives.cfm?fo=2011-02-feb/feb-18-2011.html#507629

Banda, L. (3 de Marzo de 2012). Arriban 95 vehículos diarios. *Diario de Querétaro* pág. 9A. México.

Bondy, A. y Murty, U. (2008). *Graph Theory.*Estados Unidos: Springer.

Buscaturuta.com. (2011). *Informacion de transporte publico Guadalajara*. Recuperado en Febrero de 2011, de http://www.buscaturuta.com/

Centro Queretano de Recursos Naturales. (2011). *Centro Queretano de Recursos Naturales*. Recuperado en Junio de 2011, de http://www.concyteq.edu.mx/cqrn2/

Chabini, I. (1997). Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexityand Algorithms with Optimal Run Time. Estados Unidos: Massachusetts Institute of Technology.

Cormen, T., Leiserson, C., Ronald, R.y Stein, C. (2001). *Introduction to Algorithms.* Estados Unidos: MIT Press.

Delling, D., Sanders, P., Schultes, D.y Wagner, D. (2009). Engineering Route Planning Algorithms. Alemania.

Gobierno del Estado de Querétaro. (2011). *La Sombra de Arteaga - Periódico Oficial del Gobierno del Estado de Querétaro.* Recuperado el 3 de Marzo de 2012, de http://www2.queretaro.gob.mx/disco2/servicios/LaSombradeArteaga/2012/20120313-01.pdf

Google, Inc. (2011). *A Note About Projections and Datums - Google Earth Help*. Recuperado en Junio de 2011, de http://support.google.com/earth/bin/answer.py?hl=en&answer=148110

Google, Inc. (2011). Google Earth (Versión 6.0.1 Beta) [Software]. Recuperado en Enero de 2011. Disponible en http://www.google.com/earth/index.html

Google, Inc. (2011). *Google Maps*. Recuperado en Febrero de 2011, de http://maps.google.com.mx/

Google, Inc. (2011). *Google Maps/Google Earth Terms and Conditions*. Recuperado en Marzo de 2012, de http://earth.google.com/intl/en/license.html

Google, Inc. (2011). *KML - Google Code*. Recuperado en Mayo de 2011, de https://code.google.com/apis/kml/

Grimaldi, R. P. (2004). *Discrete and Combinatorial Mathematics: An Applied Introduction.* Estados Unidos: Addison Wesley.

Jungnickel, D. (2007). *Graphs, Networks and Algorithms.* Alemania: Springer.

Kjeldskov, J., Andersen, E.y Hedegaard, L. (2007). Designing and Evaluating Buster – an Indexical Mobile Travel Planner for Public Transportation. Dinamarca: Aalborg University.

Koshimoto, J., Bromage, M., Petkov, V. y Obraczka, K. (2009). SlugTransit: A Location-Based Public Transportation Management System. Estados Unidos: University of California, Santa Cruz.

MapQuest, Inc. (2011). *Map of Mexico*. Recuperado en Febrero de 2011, de http://www.mapquest.com/maps?country=MX

Merrifield, T. (2010). Heuristic Route Search in Public Transportation Networks. *Tesis de maestría no publicada* . Estados Unidos: University of Illinois at Chicago.

Naveh, B. (2011). JGraphT (Versión 0.8.3) [Software]. Recuperado en Septiembre de 2011. Disponible en http://www.jgrapht.org/

Pallotino, S.y Scutellá, M. G. (1997). Shortest Path Algorithms in Transportation models: classical and innovate aspects. Italia: Università di Pisa.

Repenning, A.y Ioannidou, A. (2006). Mobility Agents: Guiding and Tracking Public Transportation Users. Estados Unidos: AgentSheets, Inc.

Riko, J., Marathe, M. V.y Kai, N. (1999). A Computational Study of Routing Algorithms for Realistic Transportation Networks. Estados Unidos: Los Alamos National Laboratory.

Rutero Online. (2011). *Rutero Online Querétaro* | *Mapa de Rutas de Camiones en Querétaro* | *Sitios Turísticos y Más*. Recuperado en Febrero de 2011, de http://www.ruteroonline.com

Sanders, P. y Schultes, D. (2007). Engineering Fast Route Planning Algorithms. Alemania: Universität Karlsruhe.

Secretaría de Comunicaciones y Transportes. (2011). *Rutas punto a punto*. Recuperado en Marzo de 2011, de http://aplicaciones4.sct.gob.mx/sibuac_internet/ControllerUI?action=cmdEscogeRuta

Secretaría de Seguridad Ciudadana. (2011). *Transporte / Principal - Secretaría de Seguridad Ciudadana*. Recuperado en Febrero de 2011, de http://seguridad.queretaro.gob.mx/ssc/ssc/transporte

ViaDF.com.mx. (2011). *ViaDF - ¿Cómo llegar en transporte público?* Recuperado en Marzo de 2011, de http://viadf.org/

Wolfram Research, Inc. (2011). Wolfram Mathematica (Versión 8.0) [Software]. Recuperado en Febrero de 2011. Disponible en http://www.wolfram.com/mathematica/

ANEXO A. PÓSTER PRESENTADO EN EL VERANO DE LA CIENCIA 2011 REGIÓN CENTRO

ALGOR

Algoritmos, Optimización y Redes.

Grupo de Investigación

UNIVERSIDAD POLITÉCNICA DE QUERÉTARO

MODELADO DE REDES DE TRANSPORTE PÚBLICO: UN CASO DE ESTUDIO APLICADO A LA CIUDAD DE QUERÉTARO

Ayala Elizarraraz Pedro[1], Vega Escobedo Edwin Paul[1], González Gutiérrez Arturo[1], González Gutiérrez Fidel[2,1], Rangel Mondragón Jaime[1], Díaz Delgado Guillermo[1].

[1]Universidad Autónoma de Querétaro
[2]Universidad Politécnica de Querétaro

Introducción

La ciudad de Querétaro ha venido experimentando un crecimiento poblacional de manera exponencial, especialmente a partir de 1985 (Figura 1). Este fenómeno ha propiciado una creciente demanda en materia de servicios públicos y por ende la aplicación de grandes inversiones en infraestructura, particularmente, en la construcción de vías rápidas de comunicación para el transporte público y privado. En lo que se refiere al servicio de transporte público (STP), se ha venido construyendo un sistema complejo consistente de más de 90 rutas que cubren la zona conurbada de Querétaro mayormente, y algunas de las poblaciones colindantes. Sin embargo, a pesar de dicha complejidad, hoy en día no se cuenta con un sistema que le permita al usuario del STP utilizarlo de manera informada y eficiente, de modo que tenga acceso a información básica acerca de los recorridos, horarios, puntos de ascenso y descenso de pasajeros, y todavía de mayor interés, poder conocer el itinerario óptimo que el usuario requiere para transportarse de un punto de origen en la ciudad a otro punto destino, a través del sistema de transporte.

El objetivo principal de este proyecto de investigación consiste en modelar redes de transporte público, representándolas por medio de la estructura de datos sofisticada consistente en un grafo dirigido conexo, para su utilización en procesos de simulación basados en algoritmos de ruteo, entre los cuales destacan los algoritmos de Dijkstra y Bellman-Ford (4). Particularmente, se presenta un prototipo de la red de transporte público de la ciudad de Querétaro conformada por 8 rutas representativas que cubren una amplia zona de la ciudad de Querétaro de acuerdo a su orientación cardinal desplazándose de Norte a Sur, Este a Oeste, Noreste a Suroeste, y Noroeste a Sureste.

Método

En la primera etapa del proyecto se llevó a cabo el levantamiento de las trazas de las rutas de interés. Para ello se eligieron aquellas cuya longitud total están entre las mayores y atraviesan la ciudad casi de extremo a extremo, conforme a los distintos puntos cardinales, creando propiamente una estrella. Con ello se logra una mayor cobertura con un número mínimo de rutas (Figura 2).

El levantamiento de las trazas consiste en la adquisición de las coordenadas geográficas que constituyen la ruta. Para ello, se definen los puntos de interés como aquellos que corresponden a puntos de inflexión, donde la unidad de transporte hace un quiebre para incorporarse a otra calle o avenida, y puntos de ajuste, los cuales resultan necesarios para el proceso de linealización de las trazas.

En el proceso de construcción del grafo dirigido conexo del STP, cuando se agrega una nueva ruta al modelo computacional, eventualmente aparecen nuevos puntos en los sitios de cruce, a los que se les ha llamado puntos de intersección. Si la traza de la nueva ruta se intersecta con otra existente, entonces se inserta el identificador de la nueva ruta en las propiedades de las aristas ya existentes que resultan de dicha intersección.

Este proceso se llevó a cabo en el ambiente provisto por la interfaz de Google Earth, utilizando su herramienta llamada Path.

Figura 1. Crecimiento urbano de la Ciudad de Querétaro.

Figura 2. Interfaz de *Google Earth* que muestra las ocho rutas trazadas del STP de la zona conurbada de la ciudad de Querétaro.

Resultados

El prototipo del STP de la ciudad de Querétaro se construyó en el ambiente provisto por Mathematica® (3), el cual ofrece un lenguaje de programación de alto nivel basado en el paradigma funcional. El prototipo modela computacionalmente el STP a partir del modelo matemático basado en un grafo dirigido ponderado conexo (Figura 3). Mathematica ofrece un conjunto de funciones especializadas que implementan diversos algoritmos de ruteo para calcular las distancias entre cada par de puntos o vértices del grafo. De estas funciones, la función FindShortestPath (Figura 5) es la mayormente utilizada, la cual genera el itinerario del usuario del servicio de transporte, basado en la ruta óptima, que le permite transitar desde un punto origen al punto destino a través de la red (Figura 4).

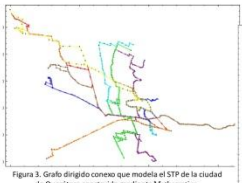


Figura 3. Grafo dirigido conexo que modela el STP de la ciudad de Querétaro construido mediante Mathematica.

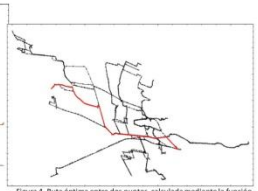


Figura 4. Ruta óptima entre dos puntos, calculada mediante la función FindShortestPath de Mathematica.

```
shortPath[g_, vertIni_, vertFin_] :=
 FindShortestPath[g, vertIni, vertFin, Method → "Dijkstra"]
shortPath[g1, vertIni, vertFin]
```

Figura 5. Código en Mathematica para el cálculo de rutas óptimas utilizando el algoritmo de Dijkstra.

Conclusiones

A lo largo de esta investigación se pudo constatar que si bien existen algunos sistemas basados en Web que permiten a una persona planificar su itinerario para desplazarse en la ciudad de Querétaro (2), ninguno de ellos ofrece un itinerario basado en la red de trasporte público. Así que este proyecto que produce un prototipo consistente en solamente 8 rutas de más de 90 en total que dan servicio de transporte en la ciudad, no pretende resolver el problema, pero sí demostrar la factibilidad de contar con una herramienta computacional que sería de utilidad para un gran sector de la población en la ciudad de Querétaro. De la presente investigación también se pueden derivar varios trabajos a futuro, como llevar a cabo estudios estructurales del sistema de trasporte para plantear métricas de calidad y propuestas para ganar eficiencia, ya que hasta el momento resulta evidente que causa de su ineficiencia es la superposición de las trazas de rutas ocasionando un incremento vehicular innecesario.

Referencias bibliográficas

1. Centro Queretano de Recursos naturales – Crecimiento Urbano Consultado en Junio 25, 2011; http://www.concyteq.edu.mx/cqrn2/
2. Gobierno del Estado de Querétaro. Secretaría de Seguridad Ciudadana – Rutas de Transporte. Consultado en Junio 20, 2011; http://seguridad.queretaro.gob.mx/ssc/ssc/transporte/rutas_de_transporte
3. Wolfram, Stephen. "The Mathematica Book". 3rd edition, Wolfram Media Inc. 2003.
4. Sahni, Sartaj. "Data Structures, Algorithms and Applications in C++", 2nd edition. Silicon Press. 2004.

Dirección Electrónica de Referencia: payala01@alumnos.uaq.mx

ANEXO B. ARTÍCULO PRESENTADO EN EL PRIMER CONGRESO INTERNACIONAL DE MOVILIDAD URBANA SUSTENTABLE

MODELADO DE REDES DE TRANSPORTE: UN CASO APLICADO A LA CIUDAD DE QUERÉTARO[1]

Arturo González Gutiérrez[2,1], Teófilo F. González[2], Fidel González Gutiérrez[1,3], Jaime Rangel Mondragón[1], Edwin Paul Vega Escobedo[1], Guillermo Díaz Delgado[1]
[1]Universidad Autónoma de Querétaro
[2]Universidad de California en Santa Barbara
[3]Universidad Politécnica de Querétaro

RESUMEN. En este artículo se presenta el modelo computacional de redes de tráfico vehicular para transporte privado y público en el contexto de la ciudad de Querétaro. Con base en este modelo, se simula el tráfico vehicular de acuerdo a un esquema basado en el cálculo de rutas óptimas entre cada par de puntos de la red, para responder a preguntas relacionadas con características estructurales de la red que determinan su eficiencia. Asimismo, se ofrece un prototipo del sistema de transporte público en la ciudad de Querétaro y zonas conurbadas, mediante el cual, se puede construir el itinerario a seguir por el usuario del sistema a través de un recorrido de longitud total mínima.

Palabras Clave. Redes de tráfico vehicular, Planeación de Itinerarios, Modelación Computacional, Transporte Público y Privado.

1. INTRODUCCIÓN

Las redes de tráfico vehicular pueden modelarse apropiadamente mediante un grafo. En particular, un grafo dirigido ponderado, consistente en un conjunto de vértices y aristas que representan intersecciones de calles y los tramos de las calles delimitados por aquellas intersecciones, respectivamente. Cada arista tiene asociada su longitud y el sentido o sentidos en que el flujo vehicular es permitido en el tramo de la calle representado por la arista. Para modelar una red de tráfico vehicular es necesario considerar los diferentes tipos de intersecciones de calles, tomando en cuenta variables como el número de calles que confluyen en un mismo punto de intersección, los sentidos permitidos en cada tramo de calle, así como los retornos y pasos de transferencia al transitar de una calle a otra u otras.

[1] Este material está basado en trabajo realizado gracias al financiamiento del Instituto México Estados Unidosde la Universidad de California (UC MEXUS) y el Consejo Nacional de Ciencia y Tecnología de México (CONACYT).
[2] Dirección de contacto: aglez@cs.ucsb.edu

Figura 1. Cruce de Avenidas Universidad y Corregidora.

Figura 2. Cruce de Avenida Zaragoza y calle Allende.

Por ejemplo, los grafos superpuestos sobre los mapas de las Figuras 1 y 2 representan el sentido vial al transitar por la ciudad de Querétaro. Particularmente el grafo de la Figura 2 modela el hecho de que se pueda dar vuelta a la derecha para tomar la calle de Allende cuando transita sobre la Avenida Zaragoza de Oriente a Poniente, mientras que no podría dar vuelta hacia la izquierda porque encontraría Allende en sentido contrario y ni siquiera podría retornar en ese punto de intersección. Formalmente, sea $G=(V,E)$ un digrafo que modela una red de tráfico vehicular consistente en el conjunto V de nodos o vértices, un conjunto E de aristas dirigidas, y una función de peso $w:E\rightarrow R^{+}$. Cada arista $e=(a, b)$ del grafo tiene asociado un número real positivo $w(e)$ que representa la longitud de la arista e. Mediante algoritmos eficientes, como el clásico algoritmo de Dijkstra, se pueden calcular rutas óptimas entre dos vértices en la red. El problema entonces consiste en dado un digrafo G y un vértice origen $s\in V$, encontrar la ruta óptima del vértice s a todos los demás. El tiempo de ejecución del algoritmo de Dijkstra, usando un arreglo lineal para la cola de prioridad Q, es $O(|V|^2+|E|)=O(|V|^2)$ o bien $O((|V|+|E|)\ log|V|)=O(|E|\ log|V|)$ si se implementa la cola de prioridad Q como una estructura de datos *heap* binaria. Más aún, si $|E|=O(|V|)$, entonces el algoritmo se ejecuta en tiempo $O(|V|\ log|V|)$, el cual es mejor que $O(|V|^2)$. El algoritmo de Dijkstra asume que todos los pesos de las aristas son números positivos (Sahni, 2004; Cormen, 2009) y mantiene un conjunto S de vértices, donde la ruta óptima y su longitud total desde el nodo origen s hasta el vértice $v\in S$ ya han sido determinados. Para todos los vértices $v\in S$, se tiene entonces que $d[v]=\delta(s,v)$, donde $\delta(s,v)=min\{w(p)|s\rightarrow v\}$ si existe una ruta p de s a v, con una longitud total $w(p)$, de otro modo $\delta(s,v)=\infty$.

Para resolver el problema de las rutas óptimas entre cada par de vértices en la red, se utiliza el algoritmo Floyd-Warshall (Sahni, 2004) basado en la Programación Dinámica y resuelve dicho problema en un tiempo de ejecución $O(|V|^3)$. Sin embargo, asumiendo que $|E|=O(|V|)$, otra manera más eficiente consiste en ejecutar el algoritmo de Dijkstra $|V|$ veces produciendo las rutas óptimas para cada par de vértices en un tiempo $O(|V||E|\ log|V|)=O(|V|^2 log|V|)$, lo cual es mejor que el tiempo $O(|V|^3)$ que consume el algoritmo basado en programación dinámica.

2. MODELO DE LA RED DE TRÁFICO VEHICULAR

Se presenta el modelo de la red de tráfico vehicular de un sector del centro de la ciudad de Querétaro, circunscrito por las Avenidas Universidad, Corregidora, Constituyentes y 5 de Febrero. El grafo que modela tal sector consiste en los conjuntos de 412 vértices y 736 aristas. Los vértices representan los puntos de intersección de calles o avenidas, además de otros puntos que resultan de la necesidad de linealizar algunos tramos de calles que tienen forma curveada. Estos últimos puntos son llamados “comodines” ya que no representan estrictamente cruces entre dos calles, sin embargo permiten que el

modelo sea más representativo y finalmente contribuye en la obtención de resultados más precisos en lo que se refiere a longitud total de aristas o rutas. Sin pérdida de generalidad, los puntos de intersección y comodines constituyen los vértices del grafo que representa la red de tráfico vehicular del centro de la ciudad de Querétaro (ver Figura 3). Los sentidos vehiculares son representados por las aristas dirigidas en el grafo (ver Figura 4). Mediante dicho modelo es posible estudiar las características estructurales de la red vehicular del centro de la ciudad de Querétaro.

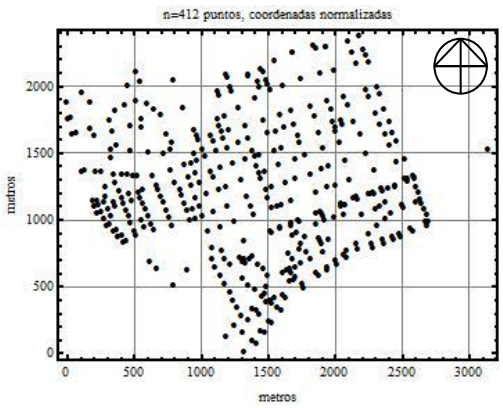


Figura 3. Puntos de intersección y comodines.

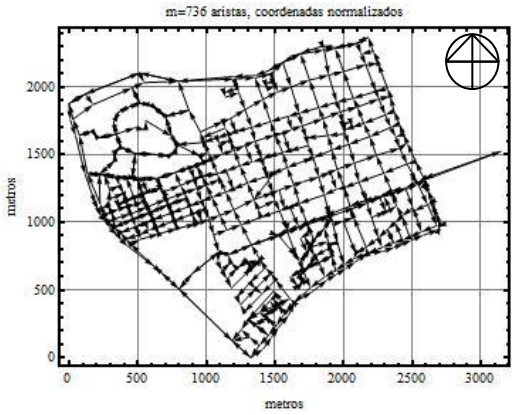
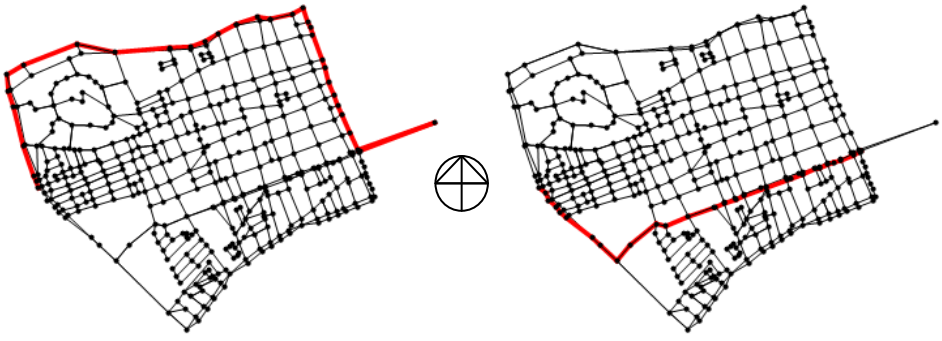


Figura 4. Dirección vehicular en calles.

El valor de la función $w:E \rightarrow R^{+}$ que se asigna a cada arista, corresponde a la distancia euclidiana entre los vértices adyacentes de la arista en cuestión. Se puede afirmar que, en la medida en que la longitud de los tramos de calles (aristas) tiende a ser más pequeña, la red presenta problemas de tráfico lento por la alta frecuencia de cruces y eventual presencia de semáforos. En particular, la red de tráfico vehicular de nuestro caso de estudio presenta dicho problema ya que la longitud de un poco más de 400 aristas (de 736 en total) se encuentra por debajo del promedio, que es de 103 metros.

Conforme a la red de tráfico vehicular se construye la estructura de datos subyacente y se implementan los algoritmos para el cálculo de rutas óptimas usando el lenguaje de programación de alto nivel llamado *Mathematica* (Trott, 2004; Gaylord, 1995; Skiena 2006). Con el prototipo resultante se calcula la ruta óptima y su longitud total para ir de un punto a otro en la ciudad. Más de la mitad de las rutas óptimas entre cada par de vértices tienen una longitud total por debajo de la longitud promedio de 1.6 km. En la Figura 5 se muestran las rutas óptimas de mayor longitud que unen dos vértices, de oriente a poniente y viceversa, en el área de estudio.



(a) (b)

Figura 5. Ruta óptima de mayor longitud (a) De Oriente a Poniente; (b) De Poniente a Oriente

Obsérvese que es deseable, a efecto de contar con una red de tráfico vehicular eficiente, que la diferencia entre las longitudes de la ruta *(i,j)* y la ruta *(j,i)* sea mínima, lo cual no sucede en el caso mostrado en la Figura 5.

Una red de tráfico vehicular podría ser rediseñada de modo que se abatan congestiones en términos de uso de tramos de calles por las rutas óptimas entre dos puntos, mitigando así el sobreuso de algunas calles, y motivando el uso de rutas alternas. Se puede, entonces, determinar las calles más utilizadas por rutas óptimas entre dos puntos del centro de Querétaro, a partir de un esquema de simulación que asume que por lo menos existe un usuario asociado a cada par de vértices *(i, j)* en la red que busca transportarse desde el nodo *i* hasta el nodo *j*. Cabe resaltar que la simulación de la red de tráfico vehicular se lleva a cabo sobre las 412^2-412 rutas posibles a transitar. Así, para el caso de estudio que nos ocupa, encontramos que el perfil del uso de las aristas por rutas óptimas tiene una distribución que crece a un ritmo bajo casi constante hasta antes de encontrar, al final, unas cuantas aristas que son las más utilizadas, las cuales denotan un problema de congestionamiento. Se puede afirmar, por tanto, que a la red prevalente de tráfico vehicular le subyace un error de diseño. Obsérvese en la Figura 6 cómo las últimas aristas se separan del resto de la gráfica denotando el uso excesivo de las mismas. Correspondientemente se puede observar en la Figura 7 que siete aristas están asociadas a tramos de calle que no solamente son usados el mayor número de veces (entre 15,000 y 20,000 veces) sino que se encuentran en la última cuarta parte de mayor incidencia de uso en las rutas óptimas. Es importante establecer que se esperaría, en un caso ideal, que el perfil de frecuencia de uso de aristas (de acuerdo al esquema de simulación establecido anteriormente) fuera modelado por una función lineal con baja pendiente (compárese con Figura 6).

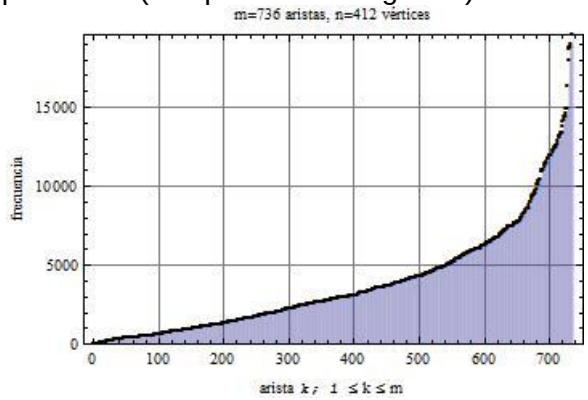


Figura 6. Frecuencia de uso de la arista *k* en rutas óptimas que unen un par de vértices.

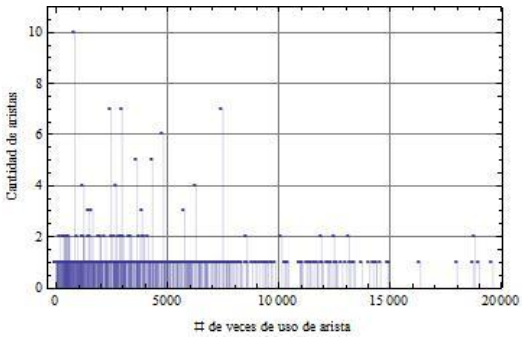


Figura 7. Cantidad de aristas que son usadas un determinado # de veces.

Los tramos de calle de mayor incidencia de uso en las rutas óptimas se determinan con base en un criterio de precisión que se establece previamente al trabajo de rediseño de la red de tráfico vehicular. Por ejemplo, para un criterio de precisión del 25%, se tiene que existen 7 tramos de calles, indicados en la Figura 8, los cuales se encuentran en la última cuarta parte de la distribución de uso de las aristas del grafo utilizado en nuestro caso de estudio.

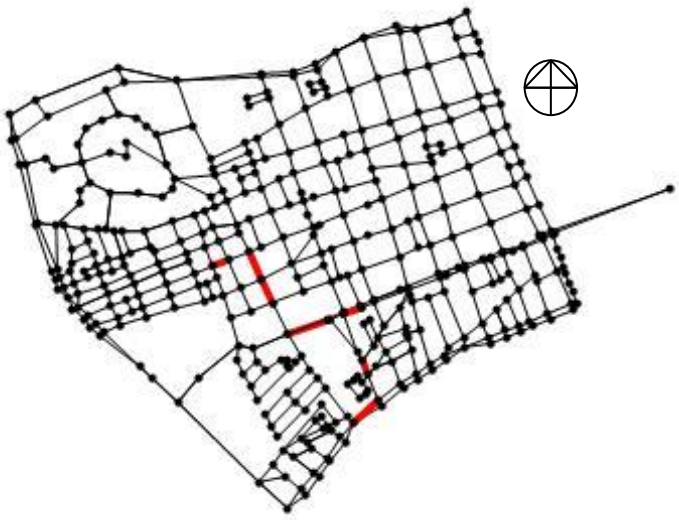


Figura 8. Los siete tramos de calles mayormente usados en rutas óptimas.

3. MODELADO DE LA RED DE TRANSPORTE PÚBLICO

Existen varios sistemas que responden a *queries* de usuarios respecto al cálculo de rutas óptimas que conectan dos puntos geográficos. Estos sistemas van de los de uso general a uso particular que considera únicamente el transporte público. Entre los sistemas de uso general se tienen:

a. Aquellos para los que, dados dos puntos por el usuario, calculan la ruta óptima a través del sistema de calles de una ciudad sin tomar en consideración los sentidos vehiculares. Aunque estos sistemas funcionan sobre varias opciones respecto a transportación pública y privada, no se tiene el mismo servicio para la ciudad de Querétaro. Encontramos aquí los más conocidos como el *Google Maps* (http://maps.google.com.mx/) y el *MapQuest* (http://www.mapquest.com/) por ejemplo.
b. Aquellos que están orientados a dar servicio a los usuarios de nueve ciudades importantes del país como son DF, Guadalajara, Monterrey, Aguascalientes, Cancún, Puerto Vallarta, Saltillo, Torreón, Villahermosa; así como las ciudades de Mazatlán y Puebla en versión beta hasta el momento. Estos sistemas ofrecen a los usuarios rutas eficientes haciendo uso de transportación motorizada pública y privada, así como los desplazamientos físicos que el usuario debe llevar a cabo. Algunos ejemplos son el *Buscaturuta* (http://www.buscaturuta.com/)y el *ViaDF.org* (http://viadf.org/). La Secretaría de Comunicaciones y Transportes también ha puesto en línea su sistema de *Rutas punto a punto* (http://www.sct.gob.mx/), mediante el cual ofrece a los usuarios el ruteo óptimo a través del sistema carretero nacional.

Entre los sistemas que particularmente atienden preguntas de ruteo de transporte público en la ciudad de Querétaro se encuentran:

a. El *Rutero Online Querétaro* (http://www.ruteroonline.com), donde dado un punto geográfico por el usuario, el sistema despliega la línea de transporte público más cercana.
b. Un sistema provisto por la *Secretaría de Seguridad Ciudadana* (SSC) (http://seguridad.queretaro.gob.mx/ssc/ssc/transporte), el cual despliega a petición del usuario la traza de una o varias rutas.

Sin embargo, todos los sistemas mencionados, incluyendo estos dos últimos sobre la ciudad de Querétaro, no responden a las necesidades del usuario de contar con un ruteo óptimo conformado por una secuencia de cruces de calles, así como los puntos de transbordos necesarios para que el usuario arribe al sitio de destino.

En el sitio web de la SSC existe únicamente un apartado dedicado al STP que contiene una interface basada en *Google Maps* y que despliega de manera gráfica los recorridos de 86 rutas del STP. A partir de esta interface no es posible recuperar las trazas de los recorridos de las rutas del STP que aquí se presenta. Sin embargo con dicha información visual del sitio se creó y trazó el prototipo del modelo computacional del STP. Las más de 90 rutas del Sistema de Transporte público actual tienen una cobertura en gran parte de la zona metropolitana de la Ciudad de Querétaro y áreas conurbadas, sin embargo algunas rutas se intersectan con otras, haciendo que en calles y avenidas donde ocurre este fenómeno se presente la *sobreposición* de rutas, con el resultado indeseado que contribuye al congestionamiento vial.

Respondiendo a las necesidades de transporte público, se ha venido construyendo un sistema complejo de transportación público, que hasta hace un año ha consistido de más de 90 rutas que cubren la zona metropolitana de Querétaro mayormente, y algunas de las poblaciones colindantes. Sin embargo, a pesar de dicha complejidad, hoy en día no se cuenta con un sistema que le permita al usuario del STP utilizarlo de manera informada y eficiente, de modo que tenga acceso a información básica acerca de los recorridos, horarios, puntos de ascenso y descenso de pasajeros, y todavía de mayor interés, poder conocer el itinerario óptimo que el usuario requiere para transportarse de un punto de origen en la ciudad a otro punto destino, a través del STP.

En la primera etapa del proyecto se llevó a cabo el levantamiento de las trazas de las rutas de interés. Para ello se eligieron aquellas cuya longitud total están entre las mayores y atraviesan la ciudad casi de extremo a extremo, conforme a los distintos puntos cardinales, creando propiamente una estrella. Con ello se logra una mayor cobertura con un número mínimo de rutas como se muestra en la Figura 9. Cada recorrido se construyó bajo el ambiente provisto por *Google Earth 6.0.1* obteniendo la traza del recorrido como una secuencia de puntos referidos a coordenadas geográficas. El sistema soportado por la SSC que muestra el recorrido de la ruta solicitada por el usuario se utilizo para validar nuestro modelo. Los recorridos están marcados sobre la zona metropolitana de la Ciudad de Querétaro.

La Tabla 1 muestra las rutas elegidas con un desplazamiento orientado en las direcciones norte a sur, este a oeste, noroeste a sureste y noreste a suroeste, cubriendo así una amplia zona de la ciudad.

El prototipo del STP de la ciudad de Querétaro se construyó en el ambiente provisto por *Wolfram Mathematica*, el cual ofrece un lenguaje de programación de alto nivel basado en el paradigma funcional. El prototipo modela computacionalmente el STP a partir del modelo matemático basado en un grafo dirigido ponderado conexo. *Mathematica* ofrece un conjunto de funciones especializadas que implementan diversos algoritmos de ruteo para calcular las distancias entre cada par de puntos o vértices del grafo. De estas funciones, la función *FindShortestPath* es la mayormente utilizada, la cual genera el itinerario del usuario del servicio de transporte, basado en la ruta óptima, que le permite transitar desde un punto origen al punto destino a través de la red.

En la Figura 10 se muestran los recorridos de las rutas como una sucesión de puntos coordenados, y en la Figura 11, el sentido de las aristas.

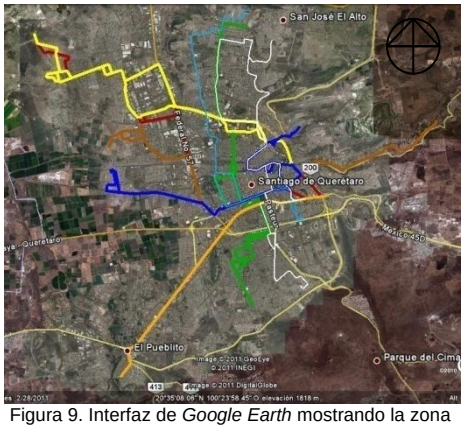


Figura 9. Interfaz de *Google Earth* mostrando la zona metropolitana de la ciudad con las rutas seleccionadas del STP

Rutas							
2	7	9	11	38	46	69	X (110)

Tabla 2. Rutas del STP levantadas que constituyen el prototipo

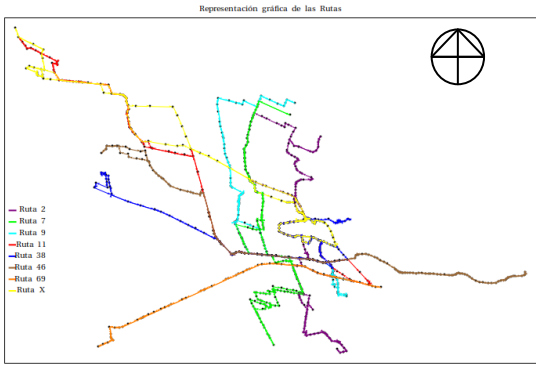


Figura 10. Sucesión de vértices del grafo representando las rutas del STP.

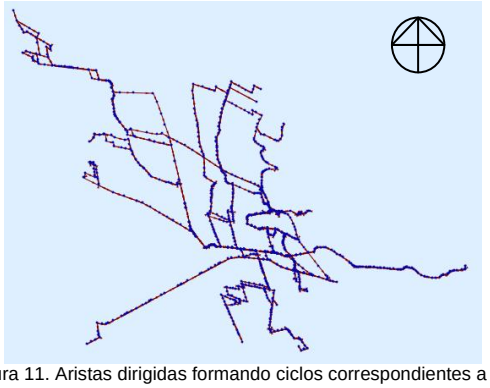


Figura 11. Aristas dirigidas formando ciclos correspondientes a los recorridos de rutas del STP.

El grafo consta de 1589 aristas y 1076 vértices. La longitud mínima, máxima y promedio de las aristas es de 20, 1018 y 171.88m. El grafo posee la propiedad de ser conexo, es decir, para cada par de vértices en el grafo existe un conjunto de aristas que los une. Para nuestro caso particular, existen 1´156,700 rutas diferentes posibles que unen pares de vértices. La longitud promedio de las rutas óptimas es de 17,527.2 metros. En la Tabla 2 se muestran las longitudes totales de los recorridos completos de las ocho rutas seleccionadas, así como el número de vértices del grafo que pertenecen a cada ruta.

La ruta óptima se calcula con la función *FindShortestPath* de *Mathematica* dados el vértice de inicio y el vértice destino en el grafo. La función tiene la opción de usar el algoritmo de Dijkstra o el algoritmo de Bellman-Ford. En la Figura 12 se presenta el recorrido de un usuario a través del STP desde un punto origen a otro destino; la línea roja representa la ruta óptima y cada punto (indicado con diferentes colores) representa un cambio de ruta del usuario.

Ruta	No. de vértices	Longitud total (km)
2	213	39.7
7	162	34.213
9	123	31.568
11	140	38.023
38	126	31.399
46	133	36.315
69	60	25.014
X (110)	134	37.449

Tabla 2. Longitudes totales de cada ruta en el grafo del STP.

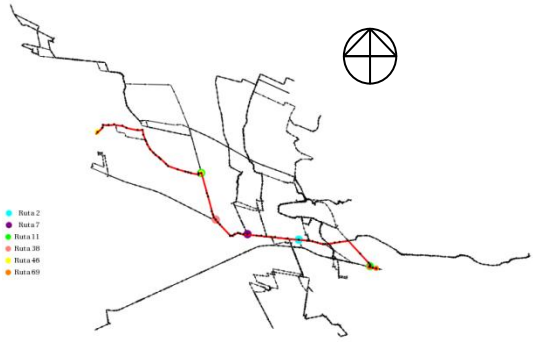


Figura 12. Ruta más corta generada con *FindShortestPath*

4. PLANEACIÓN DE ITINERARIOS USANDO EL TRANSPORTE PÚBLICO

Con los resultados obtenidos del prototipo se plantea el desarrollo de un prototipo de sistema basado en web, donde un usuario podría consultar el itinerario óptimo para poder transportarse en la ciudad de Querétaro a través del STP, partiendo de un punto origen y arribando a un punto destino. La Figura 13 muestra la interfaz del sistema consiste en dos secciones. Del lado derecho se tiene el mapa de la ciudad de Querétaro desplegado en el ambiente de *Google Maps*. Del lado izquierdo se despliega el itinerario a seguir por el usuario. El usuario marca el punto de origen y el punto de destino sobre el mapa como datos de entrada, produciendo el itinerario del recorrido que incluye la ruta que puede abordar así como los puntos en donde el usuario debe hacer cambio de ruta para llegar al punto de destino, siguiendo un recorrido óptimo en términos de la mínima longitud total.

Da un click sobre el mapa para establecer el sitio de origen y otro para el destino.

Figura 13. Interface principal del proyecto en web.

Los puntos de inicio y destino que selecciona el usuario sobre el mapa usualmente no coinciden con la ubicación de algún vértice del grafo que representa al STP. Las coordenadas del punto de origen y destino marcado por el usuario entonces son comparadas con cada punto del grafo, almacenando temporalmente en una estructura de datos la distancia. El vértice más cercano al punto origen/destino del usuario es el que se selecciona como el punto de arribo y descenso del usuario del STP. La Figura 14 muestra el resultado de una consulta al sistema.

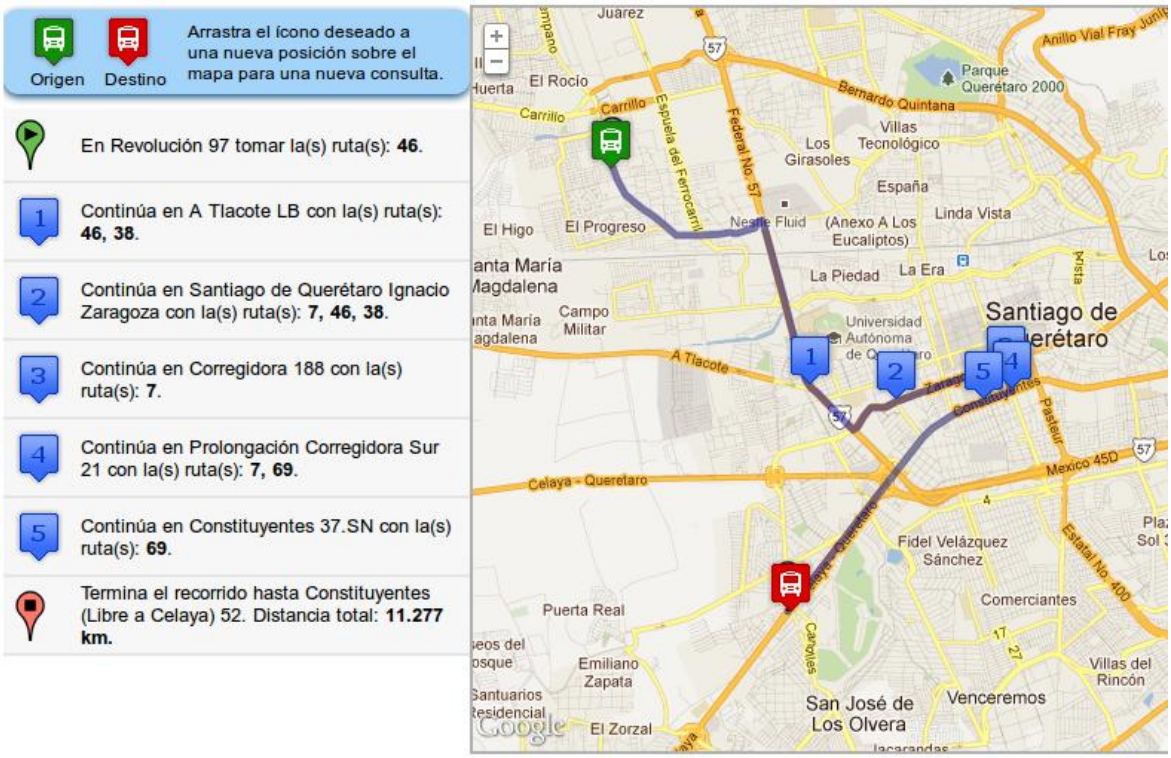


Figura 14. Resultados de una consulta en el prototipo.

5. CONCLUSIONES

Un concepto interesante a considerar en las redes de transporte es la administración dinámica de los sistemas de transporte, donde el peso de cada enlace entre dos puntos de la red de transporte depende del tiempo (Chabini, 1997). Al presente modelo computacional se le podrían agregar otras métricas para convertir el modelo en uno más robusto y apegado más a la realidad, como el tráfico vehicular, tarifas, calificaciones por parte del usuario dependiendo de la compañía transportista, entre otros.

6. REFERENCIAS

1. Cormen, Thomas H; Leiserson, Charles E. & Rivest, Ronald L, "Introduction to Algorithms", MIT Press-McGraw Hill, Massachusetts USA, **2009.**
2. Skiena, Steven & Pemmaraju Sriram, "Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica", Cambridge University Press, New York USA, **2006.**
3. Trott, Michael, "The Mathematica GuideBook for Programming", Springer Verlag, New York USA, **2004.**
4. Gaylord, Richard J. & Wellin, Paul R., "Computer Simulations with Mathematica. Explorations in Complex Physical and Biological Systems", Springer Verlag, New York USA, **1995**.
5. Chabini, I., "Discrete Dynamic Shortest Path Problems in Transportation Applications: Complexityand Algorithms with Optimal Run Time", MIT, Massachusetts USA, **1997**.
6. Sahni, Sartaj: "Data Structures, Algorithms, and Applications in C++", Silicon Press, **2004**.

ANEXO C. IMPLEMENTACIÓN DEL MODELADO EN MATHEMATICA

```
Needs["GraphUtilities`"]


(* Datos del modelo geográfico importados desde archivos kml *)
imR2 = Import["~/Desktop/proyecto/KML/edges_ruta2.kml", "Data"];
pointListR2 = "Geometry" /. Cases[imR2, Verbatim[Rule]["Geometry", _], Infinity];
pR2 = pointListR2 /. Line[stuff_] ->stuff;

imR38 = Import["~/Desktop/proyecto/KML/edges_ruta38.kml", "Data"];
pointListR38 = "Geometry" /. Cases[imR38, Verbatim[Rule]["Geometry", _],
 Infinity];
pR38 = pointListR38 /. Line[stuff_] ->stuff;

imR11 = Import["~/Desktop/proyecto/KML/edges_ruta11.kml", "Data"];
pointListR11 = "Geometry" /. Cases[imR11, Verbatim[Rule]["Geometry", _],
Infinity];
pR11 = pointListR11 /. Line[stuff_] ->stuff;

imR7 = Import["~/Desktop/proyecto/KML/edges_ruta7.kml", "Data"];
pointListR7 = "Geometry" /. Cases[imR7, Verbatim[Rule]["Geometry", _], Infinity];
pR7 = pointListR7 /. Line[stuff_] ->stuff;

imR69 = Import["~/Desktop/proyecto/KML/edges_ruta69.kml", "Data"];
pointListR69 = "Geometry" /. Cases[imR69, Verbatim[Rule]["Geometry", _],
Infinity];
pR69 = pointListR69 /. Line[stuff_] ->stuff;

imR46 = Import["~/Desktop/proyecto/KML/edges_ruta46.kml", "Data"];
pointListR46 = "Geometry" /. Cases[imR46, Verbatim[Rule]["Geometry", _],
Infinity];
pR46 = pointListR46 /. Line[stuff_] ->stuff;

imR110x = Import["~/Desktop/proyecto/KML/edges_ruta110x.kml", "Data"];
pointListR110x = "Geometry" /. Cases[imR110x, Verbatim[Rule]["Geometry", _],
Infinity];
pR110x = pointListR110x /. Line[stuff_] ->stuff;

imR9 = Import["~/Desktop/proyecto/KML/edges_ruta9.kml", "Data"];
pointListR9 = "Geometry" /. Cases[imR9, Verbatim[Rule]["Geometry", _], Infinity];
pR9 = pointListR9 /. Line[stuff_] ->stuff;


dropElev[path_] := Flatten[Map[{Drop[First[#],-1], Drop[Last[#], -1]} &, path],1]
swapLonLat[listPath_] := Map[Reverse[#] &, listPath]
coorFunc[listSwap_] := Map[{listSwap[[#]], listSwap[[# + 1]]} &, Range[1,
Length[listSwap] - 1, 2 ]]
edgesFunc[coord_] := Map[First[#] -> Last[#] &, coord]


ruta2 = edgesFunc[coorFunc[dropElev[pR2]]];
ruta7 = edgesFunc[coorFunc[dropElev[pR7]]];
ruta9 = edgesFunc[coorFunc[dropElev[pR9]]];
ruta11 = edgesFunc[coorFunc[dropElev[pR11]]];
ruta38 = edgesFunc[coorFunc[dropElev[pR38]]];
ruta46 = edgesFunc[coorFunc[dropElev[pR46]]];
ruta69 = edgesFunc[coorFunc[dropElev[pR69]]];
```

```
rutaX = edgesFunc[coorFunc[dropElev[pR110x]]];
grafo = Union[ruta2, ruta7, ruta9, ruta11, ruta38, ruta46, ruta69, rutaX];


(* Aristas y vértices de cada ruta *)
rangFunc[func_] := Union[Flatten[Map[{First[#], Last[#]} &, func], 1]] // Length;
edgeFunc[f1_, f2_] := f1 /. Map[Last[#] -> First[#] &, Thread[{Range[f2],
Union[Flatten[Map[{First[#], Last[#]} &, f1], 1]]}]];
vertFunc[data_] := Union[Flatten[Map[{First[#], Last[#]} &, data], 1]];


verticesTotales = Union[Flatten[Map[{First[#], Last[#]} &, grafo], 1]];

edges2 = edgeFunc[ruta2, rangFunc[ruta2]];
edges7 = edgeFunc[ruta7, rangFunc[ruta7]];
edges9 = edgeFunc[ruta9, rangFunc[ruta9]];
edges11 = edgeFunc[ruta11, rangFunc[ruta11]];
edges38 = edgeFunc[ruta38, rangFunc[ruta38]];
edges46 = edgeFunc[ruta46, rangFunc[ruta46]];
edges69 = edgeFunc[ruta69, rangFunc[ruta69]];
edgesX = edgeFunc[rutaX, rangFunc[rutaX]];

vertices2 = vertFunc[ruta2];
vertices7 = vertFunc[ruta7];
vertices9 = vertFunc[ruta9];
vertices11 = vertFunc[ruta11];
vertices38 = vertFunc[ruta38];
vertices46 = vertFunc[ruta46];
vertices69 = vertFunc[ruta69];
verticesX = vertFunc[rutaX];

edgesTotales = edgeFunc[grafo, rangFunc[grafo]];


(* Comprueba la conectividad del grafo *)
grafoModelo = Graph[grafo];
Select[Flatten[GraphDistanceMatrix[grafoModelo]], # == ∞\[Infinity] &] // Length


(*Peso de cada arista dado por la distancia geodésica entre dos puntos *)
sw2 = swapLonLat[dropElev[pR2]];
weight2 = Map[IntegerPart[GeoDistance[sw2[[#]], sw2[[# + 1]]]] &,
Range[1, Length[sw2] - 1, 2 ]];
c = 1;
edgeweightR2 = Map[{First[#], Last[#], weight2[[c++]]} &, ruta2];

sw7 = swapLonLat[dropElev[pR7]];
weight7 = Map[IntegerPart[GeoDistance[sw7[[#]], sw7[[# + 1]]]] &,
Range[1, Length[sw7] - 1, 2 ]];
c = 1;
edgeweightR7 = Map[{First[#], Last[#], weight7[[c++]]} &, ruta7];

sw9 = swapLonLat[dropElev[pR9]];
weight9 = Map[IntegerPart[GeoDistance[sw9[[#]], sw9[[# + 1]]]] &,
Range[1, Length[sw9] - 1, 2 ]];
c = 1;
edgeweightR9 = Map[{First[#], Last[#], weight9[[c++]]} &, ruta9];

sw11 = swapLonLat[dropElev[pR11]];
weight11 = Map[IntegerPart[GeoDistance[sw11[[#]], sw11[[# + 1]]]] &,
Range[1, Length[sw11] - 1, 2 ]];
c = 1;
edgeweightR11 = Map[{First[#], Last[#], weight11[[c++]]} &, ruta11];
```

```
sw38 = swapLonLat[dropElev[pR38]];
weight38 = Map[IntegerPart[GeoDistance[sw38[[#]], sw38[[# + 1]]]] &,
Range[1, Length[sw38] - 1, 2 ]];
c = 1;
edgeweightR38 = Map[{First[#], Last[#], weight38[[c++]]} &, ruta38];

sw46 = swapLonLat[dropElev[pR46]];
weight46 = Map[IntegerPart[GeoDistance[sw46[[#]], sw46[[# + 1]]]] &,
Range[1, Length[sw46] - 1, 2 ]];
c = 1;
edgeweightR46 = Map[{First[#], Last[#], weight46[[c++]]} &, ruta46];

sw69 = swapLonLat[dropElev[pR69]];
weight69 = Map[IntegerPart[GeoDistance[sw69[[#]], sw69[[# + 1]]]] &,
Range[1, Length[sw69] - 1, 2 ]];
c = 1;
edgeweightR69 = Map[{First[#], Last[#], weight69[[c++]]} &, ruta69];

sw110x = swapLonLat[dropElev[pR110x]];
weight110x = Map[IntegerPart[GeoDistance[sw110x[[#]], sw110x[[# + 1]]]] &,
Range[1, Length[sw110x] - 1, 2 ]];
c = 1;
edgeweightR110x = Map[{First[#], Last[#], weight110x[[c++]]} &, rutaX];

sw121 = swapLonLat[dropElev[pR121]];
weight121 = Map[IntegerPart[GeoDistance[sw121[[#]], sw121[[# + 1]]]] &,
Range[1, Length[sw121] - 1, 2 ]];
c = 1;
edgeweightR121 = Map[{First[#], Last[#], weight121[[c++]]} &, ruta121];

edgesWeight = Union[edgeweightR2, edgeweightR7, edgeweightR9, edgeweightR11,
edgeweightR38, edgeweightR46, edgeweightR69, edgeweightR110x];


list1 = DeleteDuplicates[Map[First[#] &, Sort[grafo]]];
list2 = Map[{Position[edgesWeight, list1[[#]]]} &, Range[Length[list1]]];
list3 = Map[Flatten[Flatten[list2[[#]], 1] ->#  , 1] &, Range[Length[list2]]];
vertex = Sort[Map[{First[#], Last[#]} &, grafo]];
list4 = ReplacePart[vertex, list3];
list5 = Map[#[[3]] &, edgesWeight];
k = 1;
edgesWeighted = Map[{#, list5[[k++]]} &, list4];


(* Etiquetado de vértices *)
l2 = Map[Append[{#}, "2"] &, vertices2];
l7 = Map[Append[{#}, "7"] &, vertices7];
l9 = Map[Append[{#}, "9"] &, vertices9];
l11 = Map[Append[{#}, "11"] &, vertices11];
l38 = Map[Append[{#}, "38"] &, vertices38];
l46 = Map[Append[{#}, "46"] &, vertices46];
l69 = Map[Append[{#}, "69"] &, vertices69];
lX = Map[Append[{#}, "X"] &, verticesX];
ltotal = Union[l2, l7, l9, l11, l38, l46, l69, lX];

vEtiquetados = Reverse[Union[Flatten[#, 1]]] & /@ (Cases[ltotal, {{#[[1]],
#[[2]]}, _}] & /@ Flatten[Union[{First[#]} & /@ ltotal], 1]);
vertexList = Map[Reverse, Map[First[#] -> Last[#] &,
Thread[List[Range[Length[vEtiquetados]], vEtiquetados]]]];


(* Número de aristas y vértices, (m, n) respectivamente *)
{m, n} = Map[Length, {edgesWeighted, verticesTotales}]
```

```
(* Desplegado del grafo *)
positions = Table[i -> verticesTotales[[i]], {i, 1, n}];
GraphPlot[matDis, VertexLabeling -> Tooltip,
 PlotRangePadding -> Automatic, AspectRatio -> 0.75,
 VertexCoordinateRules -> positions, FrameStyle -> Thick,
Background -> LightBlue,
PlotLabel ->
  "Dirección de la Red de Transporte; m=1158 aristas, n=682 vértices"]


(* Ruta óptima entre dos puntos del grafo *)
s = FindShortestPath[grafoModelo, {-100.4493793547862, 20.6070705126980},
      {-100.394292204587, 20.564027629560480}];


grafoShortestPath = Map[{Arrowheads[0.006], Black,
Arrow[{verticesTotales[[First[#]]],
verticesTotales[[Last[#]]]}]} &, edgesTotales];

Graphics[{grafoShortestPath, {Thickness[.003], Red ,
Line[{s}]}, {Black, PointSize[0.004], Point[sPath]}},
 AspectRatio -> 0.70, ImageSize -> 800, Frame ->True ,
 FrameTicks -> None,
FrameLabel -> {"", "",
   "Ruta optima del vértice inicial: \
{-100.4493793547862,20.607070512698} al vértice final: \
{-100.394292204587,20.564027629560480} "}]
```

ANEXO D. IMPLEMENTACIÓN DEL PROTOTIPO BASADO EN WEB

1 Clase para modelar un vértice.

```
package rutero.graph;

/**
 * Modelado de un vértice en el prototipo
* @author Edwin Vega
 */
public class RuteroVertex {

      private String id;
      private doublelatitude;
      private doublelongitude;
      private String route;

      public RuteroVertex(String id, double latitude, double longitude, String
          route) {
          this.id = id;
          this.latitude = latitude;
          this.longitude = longitude;
          this.route = route;
      }

      public String getId() {
          return id;
      }

      publicvoid setId(String id) {
          this.id = id;
      }

      public double getLatitude() {
          returnlatitude;
      }

      public void setLatitude(long latitude) {
          this.latitude = latitude;
      }

      public double getLongitude() {
          return longitude;
      }

      public void setLongitude(long longitude) {
          this.longitude = longitude;
      }

      public String getRoute() {
          return route;
      }

      public void setRoute(String route) {
          this.route = route;
      }
}
```

2 Clase para modelar una arista.

```
package rutero.graph;

import org.jgrapht.graph.DefaultWeightedEdge;

/**
 * Modelado de una arista en el prototipo a partir de DefaultWeightedEdge
* @author Edwin Vega
 */
public class RuteroEdge extends DefaultWeightedEdge {

      private String id;

      public String getId() {
          returnid;
      }

      public void setId(String id) {
          this.id = id;
      }
}
```

3 Clase para modelar un grafo.

```
package rutero.graph;

import java.util.HashMap;
import java.util.Map;

import org.jgrapht.graph.DirectedWeightedMultigraph;

/**
 * Modelado de un grafo a partir de DirectedWeightedMultigraph
* @author Edwin Vega
 */

public    class    RuteroGraph    extends    DirectedWeightedMultigraph<RuteroVertex,
      RuteroEdge>{

      private static final longserialVersionUID = 1L;

      private Map<String, RuteroVertex>mapping;

      public RuteroGraph(){
          super(RuteroEdge.class);
          mapping = new HashMap<String, RuteroVertex>();
      }

      public boolean addVertex(RuteroVertex v) {
          mapping.put(v.getId(), v);
          return super.addVertex(v);
      }

      public RuteroVertex getVertex(String id) {
          returnmapping.get(id);
      }
}
```

4 Clase contenedor de un objeto de tipo *RuteroGraph*.

```
package rutero.graph;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * Contenedor de un digrafo ponderado
 * @author Edwin Vega
*/
public class RuteroContainer {

      private RuteroGraph graph;

      public RuteroGraph getGraph() {
          returngraph;
      }

      public void setGraph(RuteroGraph graph) {
          this.graph = graph;
      }
}
```

5 Clase para cargar los datos del grafo del STP.

```
package rutero.loader;

import java.io.*;
import java.util.*;

import rutero.graph.*;

/**
 * Carga los archivos que contienen los datos de vértices y aristas
* @author Edwin Vega
 */
public class GraphLoader {

      private RuteroContainer graph;
      private String vertexFile = "/graph_files/vertices.txt";
      private String edgesFile = "/graph_files/edges.txt";

      public GraphLoader(RuteroContainer graph){
          this.graph = graph;
          this.load(vertexFile, edgesFile);
      }

      private void load(String vertexFile, String edgesFile){
          this.graph.setGraph(new RuteroGraph());
          this.loadVertex(vertexFile);
          this.loadEdges(edgesFile);
      }

      /* A partir del archivo de texto, se lee cada línea y se cargan los vértices
      con sus respectivos atributos */
      private void loadVertex(String vertexFile) {
          BufferedReader reader;
          String line;
```

```
    String[] tokens;
    Map<String, Double> latitudes = new HashMap<String, Double>();
    Map<String, Double> longitudes = new HashMap<String, Double>();
    Map<String, String> route = new HashMap<String, String>();
    List<RuteroVertex> vertex;

    try{
        reader = newBufferedReader(new FileReader(vertexFile));

        while ((line = reader.readLine()) != null&&
               line.trim().length() > 0) {
            tokens = line.split("\\s");
            latitudes.put(tokens[3], Double.valueOf(tokens[0]));
            longitudes.put(tokens[3], Double.valueOf(tokens[1]));
            route.put(tokens[3], tokens[2]);
        }

        vertex = defineVertex(latitudes, longitudes, route);

        for (RuteroVertex v : vertex) {
            this.graph.getGraph().addVertex(v);
        }
    } catch (IOException e){
        System.out.println(e.toString());
    }

    finally{
        try{
            reader.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
    }
}

/* Del archivo de texto de aristas, se leen las aristas y se cargan en el
objeto grafo */
private void loadEdges(String edgesFile) throws IOException{
    BufferedReader reader;
    String line;
    String[] tokens;
    List<RuteroEdge> edges = new ArrayList<RuteroEdge>();
    RuteroEdge edge;
    RuteroGraph graph = this.graph.getGraph();

    try{
        reader = newBufferedReader(new FileReader(edgesFile));
        while ((line = reader.readLine()) != null&&
            line.trim().length() > 0) {
            tokens = line.split("\\s");

            edge = graph.addEdge(graph.getVertex(tokens[0]),
               graph.getVertex(tokens[1]));
            this.graph.getGraph().setEdgeWeight(edge,
                   Double.parseDouble(tokens[2]));
            edges.add(edge);
        }
    } catch (IOException e){
        System.out.println(e.toString());
    }
    finally{
        try{
            reader.close();
```

```
            } catch (IOException e) {
                System.out.println(e.toString());
            }
        }

        /* Los vértices leidos desde el archivo de texto son cargados a una
        estructura List */
        public List<RuteroVertex> defineVertex(Map<String, Double> latitudes,
            Map<String, Double> longitudes, Map<String, String> route) {
            List<RuteroVertex> vertex = new ArrayList<RuteroVertex>();
            RuteroVertex v;

            for (String key : latitudes.keySet()) {
                v = newRuteroVertex(key, latitudes.get(key), longitudes.get(key),
                route.get(key));
                vertex.add(v);
            }

            return vertex;
        }
}
```

6 Clase para calcular el vértice más cercano al *click* del usuario sobre el mapa.

```
package rutero.dijkstra;
import rutero.graph.*;

import java.lang.Math;
import java.util.*;

/**
 * Clase para calcula el vértice más cercano al click del usuario sobre el mapa
* @author Edwin Vega
 */

public class FindClosestPoint {

      private doublelatPoint;
      private doublelonPoint;
      private Set<RuteroVertex>vertexSet;
      private Map<String, Double>sortedMap;

      public FindClosestPoint(double latPoint, double lonPoint,
              Set<RuteroVertex> vertexSet) {
          this.latPoint = latPoint;
          this.lonPoint = lonPoint;
          this.vertexSet = vertexSet;
          findPoint(latPoint, lonPoint, vertexSet);
      }

      /* Metodo para calcular la distancia euclideana del click del usuario a cada
      vértice del grafo */
      private void findPoint(double latCenter, double lonCenter, Set<RuteroVertex>
      vertexSet){

          Map<String, Double> distances = new HashMap<String, Double>();

          for(RuteroVertex v : vertexSet){
              double twoPointsDistance = Math.sqrt(
                     Math.pow((v.getLatitude() - latCenter), 2) +
```

```
                    Math.pow((v.getLongitude() - lonCenter), 2)
                );
                distances.put(v.getId(), twoPointsDistance);
            }

            Map<String, Double> sortedDistances = sortByComparator(distances);
            this.setSortedMap(sortedDistances);
        }


        /* Las distancias euclideanas resultants son pasadas a una estructura Map,
        para posteriormente ser ordenadas de menor a mayor */
        public static Map sortByComparator(Map unsortMap) {

List list = newLinkedList(unsortMap.entrySet());

 Collections.sort(list, new Comparator() {
 public int compare(Object o1, Object o2) {
            return ((Comparable) ((Map.Entry) (o1)).getValue())
                        .compareTo(((Map.Entry) (o2)).getValue());
                 }
            });

Map sortedMap = newLinkedHashMap();
           for (Iterator it = list.iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry)it.next();
            sortedMap.put(entry.getKey(), entry.getValue());
          }

return sortedMap;
        }

        public double getLatPoint() {
            returnlatPoint;
        }

        public void setLatPoint(double latPoint) {
            this.latPoint = latPoint;
        }

        public double getLonPoint() {
            returnlonPoint;
        }

        public void setLonPoint(double lonPoint) {
            this.lonPoint = lonPoint;
        }

        public Set<RuteroVertex> getVertexSet() {
            returnvertexSet;
        }

        public void setVertexSet(Set<RuteroVertex> vertexSet) {
            this.vertexSet = vertexSet;
        }

        public Map<String, Double> getSortedMap() {
            returnsortedMap;
        }

        public void setSortedMap(Map<String, Double> sortedMap) {
            this.sortedMap = sortedMap;
        }
```

```
}
```

7 Servlet para el manejo de datos en la aplicación web.

```
package rutero.servlets;

import rutero.graph.*;
import rutero.loader.*;
import rutero.dijkstra.*;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.jgrapht.alg.DijkstraShortestPath;
import com.google.gson.*;

/**
 * Servlet GetUserPoints. Servlet para manejar la peticion y la respuesta al
     cliente, dando la ruta más corta en un onjeto JSON
* @author Edwin Vega
 */
public class GetUserPoints extends HttpServlet {
     private static final longserialVersionUID = 1L;

public GetUserPoints() {
super();
}

     protected   void   doGet(HttpServletRequest   request,   HttpServletResponse
     response) throws ServletException, IOException {
         try {
             PrintWriter out = response.getWriter();
             response.setContentType("application/json");

             /* Lectura de archivos de texto para el objeto grafo */
             RuteroContainer container = newRuteroContainer();
             GraphLoader stp = newGraphLoader(container);
             List<RuteroEdge> edges = new ArrayList<RuteroEdge>();

             /* Coordenadas recuperadas de ambos clicks del usuario */
             double srcLat = Double.parseDouble(request.getParameter("src_lat"));
             double srcLon = Double.parseDouble(request.getParameter("src_lon"));
             double destLat =
                    Double.parseDouble(request.getParameter("dest_lat"));
             double destLon =
                    Double.parseDouble(request.getParameter("dest_lon"));

             /* Calculo de los vértices mas cercanos a los clicks del usuario */
             FindClosestPoint userSource = newFindClosestPoint(srcLat, srcLon,
                          container.getGraph().vertexSet());
             FindClosestPoint userTarget = newFindClosestPoint(destLat, destLon,
                          container.getGraph().vertexSet());

             Map<String, Double> sourceMap = userSource.getSortedMap();
```

```
        Object[] idSource = sourceMap.keySet().toArray();

        Map<String, Double> targetMap = userTarget.getSortedMap();
        Object[] idTarget = targetMap.keySet().toArray();

        System.out.println("Request from: " + request.getRemoteAddr());

        /* Vértices de origen y destino */
        RuteroVertex src =
                container.getGraph().getVertex(idSource[0].toString());
        RuteroVertex tg =
                container.getGraph().getVertex(idTarget[0].toString());

        /* Llamada al algoritmo de Dijkstra en JGraphT */
        DijkstraShortestPath<RuteroVertex, RuteroEdge> dijkstra;
        dijkstra = new DijkstraShortestPath<RuteroVertex,
                RuteroEdge>(container.getGraph(), src, tg);
        edges = dijkstra.getPathEdgeList();

        List<RuteroVertex> vertices = new ArrayList<RuteroVertex>();

        /* Respuesta en un objeto JSON */
        Gson gson = newGson();
        JsonObject json = newJsonObject();
        JsonArray shortestPath = newJsonArray();
        JsonArray transfers = newJsonArray();

        json.addProperty("distance", (dijkstra.getPathLength() / 1000));

        Iterator<RuteroEdge> it = edges.iterator();
    while (it.hasNext()){
        RuteroEdge edge = (RuteroEdge) it.next();
        RuteroVertex srcVertex =
                dijkstra.getPath().getGraph().getEdgeSource(edge);
        RuteroVertex tgtVertex =
                dijkstra.getPath().getGraph().getEdgeTarget(edge);

        vertices.add(srcVertex);
        vertices.add(tgtVertex);

        /* Puntos de la ruta optima a ser dibujados en el mapa */
        JsonObject srcPoint = newJsonObject();
        srcPoint.addProperty("lon", srcVertex.getLongitude());
        srcPoint.addProperty("lat", srcVertex.getLatitude());

        JsonObject destPoint = newJsonObject();
        destPoint.addProperty("lon", tgtVertex.getLongitude());
        destPoint.addProperty("lat", tgtVertex.getLatitude());

        shortestPath.add(srcPoint);
        shortestPath.add(destPoint);
    }
    json.add("Points", shortestPath);

    String string_change = "";
    List<RuteroVertex> route_changes = new ArrayList<RuteroVertex>();

    /* Puntos de transbordo del usuario */
    Iterator<RuteroVertex> vertices_it = vertices.iterator();
    while (vertices_it.hasNext()){
        RuteroVertex vertex = (RuteroVertex) vertices_it.next();
        if(!string_change.equals(vertex.getRoute())){
                string_change = vertex.getRoute();
```

```
                    route_changes.add(vertex);

                    String line = vertex.getRoute().replaceAll(":", ", ");
                    JsonObject transfer = newJsonObject();
                transfer.addProperty("lon", vertex.getLongitude());
                transfer.addProperty("lat", vertex.getLatitude());
                transfer.addProperty("route", line);

                transfers.add(transfer);
                }
            }
            json.add("Transfers", transfers);

            /* Envío del objeto JSON al navegador del usuario */
            out.println(gson.toJson(json));
            out.close();

            } catch (Exception e) {
                System.err.println(e.toString());
            }
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
            doGet(request, response);
        }
}
```

8 Archivo HTML del sitio principal.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta property="og:title" content="ALGOR Route Planner" />
      <meta property="og:description" content="Route Planner Prototype for the
         Queretaro Public Transportation Network" />
      <meta property="og:image"
         content="http://img688.imageshack.us/img688/112/ruterothumbnail.png" />
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <link href="css/rutero.css" type="text/css" rel="stylesheet" />
      <link rel="shortcut icon" href="/Rutero/img/favicon.ico"
         type="image/x-icon" />
      <script type="text/javascript"
         src="http://maps.google.com/maps/api/js?sensor=false&language=es">
      </script>
      <script type="text/javascript"
         src="js/jquery/jquery-1.6.2.min.js"></script>
      <script type="text/javascript"
         src="js/jquery/jquery-ui-1.8.16.custom.min.js"></script>
      <script type="text/javascript" src="js/load_map.min.js"></script>
      <title>Planeador de Itinerarios ALGOR</title>
   </head>

   <body>
      <div id="wrapper">
         <div id="header">
            <div id="page-head">
               <div id="logos"></div>
               <div id="titles">
```

```
                    <h1 id="h1-header">Planeador de Itinerarios ALGOR</h1>
                    <div id="version">versión Alpha</div>
                    <h2 id="h2-header">Cuerpo Académico: Algoritmos,
                       Optimización y Redes.</h2>
                 </div>
              </div>
           </div>
           <div id="content">
              <div id="left-column">
                 <div id="clicks"></div>
                 <table id="drag-message"></table>
                 <table class="instructions"></table>
                 <div id="loadBox">
                    <p><img alt="load" src="/Rutero/img/ajax-loader.gif"></p>
                    <p>Cargando resultados...</p>
                 </div>
                 <div id="error"></div>
              </div>

              <div id="map-canvas"></div>
           </div>

           <div id="footer"></div>
        </div>
     </body>
</html>
```

9 Hoja de estilo para el sitio principal.

```
/*
 * ALGOR Route Planner
 * @author Edwin Paul Vega Escobedo
*/

@CHARSET"UTF-8";

html, body{
      margin:0;
      height:100%;
      padding-bottom:20px;
      font-family:Arial, Helvetica, sans-serif;
      cursor:default;
}

#wrapper{
      height:auto !important;
      height:100%;
      min-height:100%;
}

#header{
      padding-bottom:17px;
      background-color:black;
      margin-bottom:20px;
      box-shadow:-1px 5px 7px #0f0f0f;
}

#page-head{
      margin:0 auto;
      width:980px;
```

```
      padding:5px 10px;
}

#titles{
      width:800px;
      padding-top:15px;
      color:#ffffff;
}

#logos{
      width:78px;
      height:95px;
      float:right;
      border:solid 3px black;
      border-radius:10px;
      background-image:url("/Rutero/img/uaq.jpg");
      background-repeat:no-repeat;
}

#h1-header{
      font-weight:normal;
      width:460px;
      margin:12px 5px 12px 0;
      font-size:175%;
      display:inline;
}

#h2-header{
      font-size:90%;
      width:550px;
      margin:10px 0;
}

#version{
      display:inline;
      font-size:75%;
}

#content{
      margin:auto;
      width:80%;
      height:100%;
}

#clicks{
      font-size:125%;
      color:#3a3a3a;
      margin:auto;
      padding:70px 10px;
      border-left:3px solid #aaa;
}

#loadBox{
      font-size:100%;
      color:#4d4d4d;
      background-color:#e8e8e8;
      padding:3px 0;
      visibility:hidden;
      border-radius:8px;
      margin:10px 0;
}

p{
```

```
      text-align:center;
      margin:5px 0;
}

#map-canvas{
      width:70%;
      height:500px;
      border:2px solid #afafaf;
}

#left-column{
      width:29%;
      height:500px;
      float:left;
      overflow:auto;
}

.locations{
      background:#f4f4f4;
      font-weight:bold;
      padding-left:20px;
}

#drag-message{
      background-color:#A5D4FA;
      font-size:75%;
      border-radius:10px;
      padding:0 15px;
      margin:3px 0 10px 0;
      box-shadow:0 2px 5px #004073;
}

table.instructions{
      border-collapse:collapse;
      font-size:80%;
      width:100%;
}

table.instructions td{
      border-bottom:2px solid #e6e6e6;
      background-color:#f0f0f0;
      padding:3px 4px 3px 8px;
}

table.instructions td:HOVER{
      border-bottom:1px solid #e6e6e6;
      background-color:#F9EDBE;
      cursor:pointer;
}

#error{
      border-radius:5px;
      background-color:#ff7777;
      font-size:70%;
      padding:2px 10px;
      margin-top:10px;
      visibility:hidden;
}

.imgLocation{
      padding:4px 0;
}
```

```
#results{
      list-style:none;
      padding:0;
      margin:0;
}

#footer{
      clear:both;
      width:400px;
      margin:25px auto;
      text-align:center;
      color:#3a3a3a;
      font-size:80%;
      padding:5px 10px;
}
```

10 Archivo JavaScript para el manejo dinámico del contenido en el sitio.

```
/**
 * ALGOR Route Plannner
 * Universidad Autónoma de Querétaro
 * Facultad de Informática
 * Grupo de Investigación: Algoritmos, Optimización y Redes (ALGOR).
 * Querétaro, México. 2012
 *
 * @author Edwin Paul Vega Escobedo
*/

var source_lat, source_lon, dest_lat, dest_lon, map, info = new
     google.maps.InfoWindow();
var source_marker, destination_marker, startMarker, endMarker, transferMarker;
var markersArray = [], err = 0, timeout = 100, addressRequests = [],
     stepsQueue = [], routeMarkers = [];
var geocoder = new google.maps.Geocoder();

$(function() {
   $(document).ready(function() {
      function initialize(){

         /* Inicialización del objeto map de Google Maps */
         var mapDiv = document.getElementById('map-canvas');

         map = new google.maps.Map(mapDiv, {
            center: new google.maps.LatLng(20.593131, -100.392314),
            zoom: 13,
            mapTypeId: google.maps.MapTypeId.ROADMAP,
            mapTypeControl: true,
            mapTypeControlOptions: {
               style: google.maps.MapTypeControlStyle.DROPDOWN_MENU
            },
            streetViewControl: false,
            navigationControl: true,
            navigationControlOptions: {
               style: google.maps.NavigationControlStyle.ZOOM_PAN
            }
         });

         /* Dibuja los marcadores una sola vez al momento de hacer click */
         google.maps.event.addListenerOnce(map, 'click', function(event) {
            source_marker = new google.maps.Marker({
```

```
            position: event.latLng,
            map: map,
            title: 'Origen',
            draggable: true,
            icon: '/Rutero/img/source_bus.png'
        });
        source_lat = source_marker.getPosition().lat();
        source_lon = source_marker.getPosition().lng();

        google.maps.event.addListener(source_marker, 'dragend', function() {
            source_lat = source_marker.getPosition().lat();
            source_lon = source_marker.getPosition().lng();
            sendPoints(source_lat, source_lon, dest_lat, dest_lon);
        });

        google.maps.event.addListenerOnce(map, 'click', function(event) {
            $('#clicks').hide();

            destination_marker = new google.maps.Marker({
                position: event.latLng,
                map: map,
                title: 'Destino',
                draggable: true,
                icon: '/Rutero/img/destination_bus.png'
            });

            /* Instrucción para una nueva consulta */
            $('#drag-message').append(
                '<tr><td><img class="img-drag" src="' +
                source_marker.getIcon() +
                '" alt="Inicio" />Origen<td><img class="img-drag" src="' +
                destination_marker.getIcon() + '" alt="Destino" />Destino' +
                '<td class="msg-drag" colspan="2">Arrastra el ícono
                deseado a una nueva posición sobre el mapa para
                una nueva consulta.</tr>'
            ).show('shake', 'slow');

            dest_lat = destination_marker.getPosition().lat();
            dest_lon = destination_marker.getPosition().lng();
            sendPoints(source_lat, source_lon, dest_lat, dest_lon);

            google.maps.event.addListener(destination_marker, 'dragend',
            function() {
                dest_lat = destination_marker.getPosition().lat();
                dest_lon = destination_marker.getPosition().lng();
                sendPoints(source_lat, source_lon, dest_lat, dest_lon);
            });
        });
    });

    /* Función para manejar los datos con AJAX*/
    function sendPoints(src_lat, src_lon, dest_lat, dest_lon) {
        /* Remove all markers, overlays and instructions column*/
        while(markersArray[0]){
            markersArray.pop().setMap(null);
        }
        for(var i = 0; i < addressRequests.length; i++) {
            clearTimeout(addressRequests[i]);
        }
        while(stepsQueue[0]){
            stepsQueue.pop();
        }
        while(routeMarkers[0]){
```

```
   routeMarkers.pop();
}

$('.instructions').empty();
$('#error').empty();

var ruta = new google.maps.Polyline({
   strokeColor: '#000099',
   strokeOpacity: 0.5,
   strokeWeight: 5
});
var puntos = ruta.getPath();

/* Función de JQuery para el manejo de AJAX */
$.ajax({
   data: {
      "src_lat":src_lat,
      "src_lon":src_lon,
      "dest_lat":dest_lat,
      "dest_lon":dest_lon
   },
   url: 'GetUserPoints',
   success: function(Response){
      map.controls[google.maps.ControlPosition.TOP_CENTER].clear();

      if((Response == null) || (Response.Points.length == 0)){
         var control = document.createElement('DIV');
         control.style.padding = '3px';
         control.style.border = '1px solid #000';
         control.style.marginTop = '5px';
         control.style.backgroundColor = '#dd0000';
         control.style.color = '#ffffff';
         control.innerHTML = 'Intenta otra ubicación';

         map.controls[google.maps.ControlPosition.TOP_CENTER].
            push(control);

      } else {
         /* Agrega los puntos de la ruta óptima a un objeto Google
         Maps polyline */
         for(var i = 0; i < Response.Points.length; i++) {
            puntos.push(new google.maps.LatLng(Response.Points[i].lat
               , Response.Points[i].lon));
         }
         ruta.setMap(map);
         markersArray.push(ruta);

         /* Dibuja el marcador del punto de origen */
         startMarker = new google.maps.Marker({
            position: new google.maps.LatLng(
               Response.Transfers[0].lat,
               Response.Transfers[0].lon),
            map: map,
            title: 'Inicio, ruta(s): ' + Response.Transfers[0].route,
            icon: 'http://maps.google.com/mapfiles/dd-start.png'
         });
         markersArray.push(startMarker);

         google.maps.event.addListener(startMarker, 'click',
         function(){
            info.setContent(startMarker.getTitle());
            info.open(map, this);
         });
```

```
getAddress(startMarker, 0, Response.Transfers[0].route,
function(address, routeOptions,map_marker, index_marker){
   $('#loadBox').css({visibility:"visible"}).show('scale',
      'fast');

   var instruction = '<tr id="step' + index_marker +
      '"><td><img class="imgLocation" src="' +
      map_marker.getIcon() + '" alt="Inicio" /><td>En ' +
      address + ' tomar la(s) ruta(s): <strong>' +
      Response.Transfers[0].route + '</strong>.</tr>';
   stepsQueue[index_marker] = instruction;
   routeMarkers[index_marker] = map_marker;
});

/* Dibuja los puntos de transbordo */
for(var i = 1; i < Response.Transfers.length; i++) {
   var transfer = Response.Transfers[i];
   var transfer_options = transfer.route;

   transferMarker = new google.maps.Marker({
      position: new google.maps.LatLng(
         transfer.lat, transfer.lon),
      map: map,
      title: 'Transbordar a ruta(s): ' + transfer.route,
      icon: '/Rutero/img/transfer-icons/number_' + i +
         '.png'
   });
   markersArray.push(transferMarker);

   getAddress(transferMarker, i, transfer_options,
   function(address, routeOptions, map_marker,
   index_marker){
      var instruction = '<tr id="step' + index_marker +
         '"><td><img class="imgLocation" src="' +
         map_marker.getIcon() + '" alt="Transfer" />
         <td>Continúa en ' +address + ' con la(s) ruta(s):
         <strong>'+ routeOptions + '</strong>.</tr>';
      stepsQueue[index_marker] = instruction;
      routeMarkers[index_marker] = map_marker;

      google.maps.event.addListener(map_marker, 'click',
      function(){
         info.setContent(map_marker.getTitle());
         info.open(map, this);
      });
   });
}

/* Dibuja el marcador para el punto destino */
endMarker = new google.maps.Marker({
   position: new google.maps.LatLng(
      Response.Points[(Response.Points.length - 1)].lat,
      Response.Points[(Response.Points.length - 1)].lon),
   map: map,
   title: 'Punto destino',
   icon: 'http://maps.google.com/mapfiles/dd-end.png'
});
markersArray.push(endMarker);

google.maps.event.addListener(endMarker, 'click',
function(){
   info.setContent(endMarker.getTitle());
```

```
                        info.open(map, this);
                    });

                    getAddress(endMarker, Response.Transfers.length, null,
                    function(address, routeOptions, map_marker, index_marker){
                        var instruction = '<tr id="step' + index_marker +
                            '"><td><img class="imgLocation" src="' +
                            endMarker.getIcon() + '" alt="Destino" />
                            <td>Termina el recorrido hasta ' + address +
                            '. Distancia total: <strong>' + Response.distance +
                            'km.</strong></tr>';
                        stepsQueue[index_marker] = instruction;
                        routeMarkers[index_marker] = endMarker;

                        addToDom(stepsQueue, routeMarkers, map);

                        $('#loadBox').hide('fold', 'slow');
                    });
                }
            },
            /* Muestra un mensaje al usuario en caso de existir un error al
            enviar o recibir los datos */
            error: function(){
                $('#error').append('<tr><td class="error" colspan="2">Algún
                error sucedió ' + 'con el servidor. Inténtalo más tarde.</tr>'
                ).css({visibility:"visible"});err = 0;
            },
        });
    }
}
google.maps.event.addDomListener(window, 'load', initialize);

$('#clicks').append('Da un click sobre el mapa para establecer el sitio de
 origen y otro para el destino.');

var d = new Date();
$('#footer').append('© ' + d.getFullYear() + ', Facultad de
Informática<br>Campus Juriquilla, Universidad Autónoma de Querétaro');
});
});

/* Función para obtener las direcciones de los marcadores usando Google reverse
    Geocoding*/
function getAddress(marker, index, transferRoutes, callback) {
   geocoder.geocode(
      {'latLng': marker.getPosition()},
      function(results, status) {
         if(status == google.maps.GeocoderStatus.OK) {
            if(results[0]){
               var routeOptions = transferRoutes;
               var map_marker = marker;
               var index_marker = index;
               var address = results[0].address_components[1].short_name + ' ' +
                  results[0].address_components[0].short_name;

               callback(address, routeOptions, map_marker, index_marker);
            }
         } else if(status == google.maps.GeocoderStatus.OVER_QUERY_LIMIT) {
            addressRequests[addressRequests.length] = setTimeout(function() {
                  getAddress(marker, index, transferRoutes, callback);
               }, (timeout * 2)
            );
         } else {
```

```
            err++;
         }
      }
   );
}

/* Agrega los pasos en la table izquierda de la interfaz junto con un listener
para el evento click, en caso de que el usuario haga click sobre cada paso */
function addToDom(instructionsArray, transferMarkersArray, map) {
   var row_id = null, marker = null;

   for(var i = 0; i < instructionsArray.length; i++) {
      $('.instructions').append(instructionsArray[i]);
      marker = transferMarkersArray[i];
      row_id = '#step' + i.toString();

      var rowClick = (function(marker) {
         return function() {
            map.panTo(marker.getPosition());
            map.setZoom(17);
         };
      })(marker);
      $('.instructions').delegate(row_id, 'click', rowClick);
   }
}
```

ANEXO E. INSTALACIÓN DEL PROTOTIPO BASADO EN WEB

En los siguientes pasos se describe el proceso de instalación del software utilizado para montar el prototipo basado en web. El sistema operativo donde se llevó a cabo el prototipo fue en Ubuntu 11.10 x64, por lo que el software al que se hará referencia en las instrucciones posteriores, está dedicado a dicho sistema operativo.

1. Descargar *Eclipse IDE for Java EE Developers* desde el sitio: http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/indigo/SR2/eclipse-jee-indigo-SR2-linux-gtk-x86_64.tar.gz

2. Descargar el servidor *Apache Tomcat 6* desde el sitio: http://apache.webxcreen.org/tomcat/tomcat-6/v6.0.35/bin/apache-tomcat-6.0.35.tar.gz

3. Descomprimir los archivos del paso 1 y 2 en cualquier ubicación, de preferencia en la carpeta *home* del usuario.

4. Entrar en la carpeta creada al descomprimir el archivo del paso 1. Abrir el archivo ejecutable.

5. Estando dentro de *Eclipse*, asignar la ubicación del *workspace*. Ir al menú *File/New* y seleccionar *Dynamic Web Project*. En la ventana del nuevo proyecto se debe determinar un nombre para el proyecto, después, ir al apartado "*Target Runtime*" y dar click en "*New Runtime…*". Abrir los subniveles de la carpeta "*Apache*" y seleccionar "*Apache Tomcat v6.0*", dar click en "*Next*" y con el botón "*Browse…*" seleccionar la carpeta donde se descomprimió el archivo del paso 2. Por último, dar click en el botón "*Finish*" de la ventana actual, del mismo modo en el botón "*Finish*" de la ventana para el nuevo proyecto.

6. Descargar la librería *JGraphT* del siguiente enlace: http://prdownloads.sourceforge.net/jgrapht/jgrapht-0.8.3.tar.gz?download y extraer el archivo .jar en la siguiente ubicación del proyecto: */WebContent/WEB-INF/lib*.

7. Descargar la librería *Google Gson* del siguiente enlace: http://google-gson.googlecode.com/files/google-gson-2.1-release.zip y extraer el archivo *gson-2.1.jar* en la ubicación del proyecto: */WebContent/WEB-INF/lib*.

8. Descargar la librería *JQuery* desde: http://jqueryui.com/download, extraer de la carpeta*js* ambos archivos de extensión*.js*, para colocarlos en la ubicación del proyecto */WebContent/js/jquery/*.

9. Agregar los primeros siete archivos de código *Java* del Anexo C en la ubicación del proyecto: */Java Resources/src*.

10. El octavo archivo del Anexo C, agregarlo en */WebContent/index.html*. El noveno archivo del Anexo C deberá ser agregado en la ubicación */WebContent/ccs/rutero.css*. El último archivo del mismo Anexo C será agregado en /*WebContent/js/load_map.js*

11. En *Eclipse*, ubicar el *Project Explorer* en la columna del lado izquierdo. Dar click derecho sobre la carpeta principal del proyecto, ir a *Run As/Run on Server*. De esta manera se ejecuta el proyecto.