VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

HÍPER RUTAS MÁS CORTAS CON RESPECTO DEL TIEMPO ENTRE LAS INSTALACIONES DENTRO DE CIUDAD UNIVERSITARIA: UN MODELO DE TRANSPORTE MULTIMODAL

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

DE SISTEMAS EN TRANSPORTE

P R E S E N T A :

DAVID LÓPEZ FLORES

TUTORA:

DRA. ANGÉLICA DEL ROCÍO LOZANO CUEVAS

2010

JURADO ASIGNADO:

Presidente:	DR. RICARDO ACEVES GARCÍA
Secretario:	DRA. MAYRA ELIZONDO CORTÉS
Vocal:	DRA. ANGÉLICA DEL ROCÍO LOZANO CUEVAS
1er. Suplente:	M. I. FRANCISCO GRANADOS VILLAFUERTE
2do. Suplente:	M. I. LUIS ALEJANDRO GUZMÁN CASTRO

Lugar donde se realizó la tesis:

FACULTAD DE INGENIERÍA, UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

TUTORA DE TESIS:

DRA. ANGÉLICA DEL ROCÍO LOZANO CUEVAS

FIRMA

A Rosy, Flor, Pablo, Ale, Amaya, Bety y Luis. A todos mis tios, tias, primos y primas (afortunadamente son muchos). Y por supuesto también a todos mis carnalitos del alma y amigos. Una parte de todos ustedes está en este trabajo.

Agradecimientos

Agradezco a la Dra. Angélica del Rocío Lozano Cuevas por ser mi guía en la realización de este trabajo, al M. en G. Pablo López Ramírez por su ayuda en la programación del algoritmo. Y a los sinodales; Dr. Ricardo Aceves García, Dra. Mayra Elizondo Cortés, M.I. Francisco Granados Villafuerte y M.I. Luis Alejandro Guzmán Castro por sus aportes al trabajo.

David López Flores

Introducción

Conforme el parque vehicular crece y por lo tanto la contaminación aumenta, cada vez se hace más importante invitar a la población a cambiar el uso del automóvil por el transporte público. Para que esta transición sea viable es necesario contar con un transporte eficiente en donde el usuario tenga suficiente información sobre los tiempos de viaje y las rutas que debe seguir para llegar más rápido a su destino. Por lo general, en países desarrollados el transporte público responde a un itinerario que se cumple día a día, hecho que facilita a los usuarios planear mejor sus recorridos para trasladarse en las ciudades. Sin embargo, en los países en desarrollo el usuario tiene cierta incertidumbre de las rutas y los tiempos de recorrido desde el momento en que llega a la parada hasta el arribo a su destino.

La incertidumbre en los tiempos de llegada y de recorrido puede ser debido al tráfico o a que se desconocen los horarios de arribo a las estaciones de las diferentes líneas. En el caso de Ciudad Universitaria (CU), el tráfico no es un factor que afecte significativamente el tiempo de recorrido de los autobuses del sistema de transporte público, ya que la mayoría de las líneas de autobús tienen carriles exclusivos de circulación, pero, no existe un itinerario para el transporte público universitario. Entonces, lo mejor que se puede hacer para estimar los tiempos de espera en las paradas es utilizar las frecuencias de los autobuses, por lo tanto el uso de modelos que tomen en cuenta la incertidumbre de los tiempos de espera es de vital importancia para que los sistemas de transporte sin itinerario, puedan mejorar de una forma económica, en donde el usuario esté más informado acerca del sistema de transporte.

En CU, como en otras partes del mundo, es posible trasladarse entre un origen y un destino usando diferentes modos de transporte público. En particular en CU el usuario puede moverse en bicicleta, en autobús o caminado; cada persona tiene su propia opinión acerca de las ventajas y desventajas de cada modo sobre los demás. Habrá gente que para distancias cortas prefiera el uso de la bicicleta sobre el uso del autobús, ya que el traslado en bicicleta es más rápido debido a los tiempos de espera en las paradas de autobús. Algunos usuarios preferirán un uso combinado de bicicleta y autobús pues reúnen comodidad con rapidez. Pero, ¿cuál combinación de modos es la más rápida? ¿cuáles son las diferentes opciones del usuario para trasladarse en CU de forma eficiente?

El **objetivo** de la tesis es encontrar las rutas más rápidas dentro de CU si se considera que es posible trasladarse caminado, en autobús o en bicicleta, en donde además, los tiempos de espera

en las paradas de autobús son inciertos y el usuario decide cuántas veces está dispuesto a cambiar de modo. Para encontrar las híper rutas más cortas se adaptó y programó el algoritmo *Shortest Viable Hyperpath* (SVH) de Lozano & Storchi (2002).

Para cumplir el objetivo, la tesis se organizó de la siguiente forma. En el primer capítulo se explica la teoría de híper grafos que sienta las bases para la elaborar el modelo de híper rutas propuesto. Después se estudian algunos de los problemas de rutas más cortas (RMC) en donde las variables pueden ser aleatorias o deterministas, esto con el propósito de adentrar al lector a los problemas de RMC en general. Al final, se define el problema de rutas más cortas en sistemas de transporte multimodal y se estudia el estado del arte de este tipo de problemas.

En el segundo capítulo, se describe la zona de estudio, es decir, su área, sus instalaciones, la población flotante y la distribución de las personas en los edificios entre otros. La descripción brinda un panorama amplio de que el problema aquí tratado no es sencillo y que los resultados de la tesis resultan útiles para la comunidad universitaria. En el tercer capítulo, se muestran los datos obtenidos en trabajo de campo que son necesarios para definir la híper red de transporte multimodal.

En el cuarto capítulo, se explica el proceso que se siguió para elaborar la híper red de transporte multimodal de CU. La elaboración de la híper red permite visualizar los resultados y ayuda a crear una base de datos en donde el algoritmo pueda consultar la información necesaria para su ejecución. En el quinto capítulo se enuncia el algoritmo SVH y se estudian los antecedentes de éste. Además se describe a detalle cómo se adaptó el SVH para las condiciones del sistema de transporte público de CU y se explican las diferentes clases del algoritmo programado en *Java*.

En el sexto y último capítulo se presenta una serie de resultados calculados con el algoritmo que se comparan con resultados obtenidos en trabajo de campo. Por último, se dan las conclusiones de la tesis y en diferentes anexos se muestra el código *Java* del SVH.

Índice general

CAPÍTULO 1

Híper grafos e híper rutas en sistemas de transporte multimodales

El capítulo se divide en tres secciones, en la primera se definen los conceptos de híper grafos e híper rutas. En la segunda sección se comentan los problemas de rutas más cortas (RMC) y diferentes métodos que se han utilizado para resolverlos. Por último, en la tercera sección, se integra la teoría de híper grafos y los problemas RMC, es decir, las híper rutas más cortas (HRMC), y por último se describe cómo han tratado este tipo de problemas diversos autores.

1.1. Híper grafos dirigidos

Un híper grafo dirigido es una generalización de un grafo dirigido, así que para comprender mejor esta teoría, se explica qué son los grafos dirigidos.

1.1.1. Grafos dirigidos

Los arcos de un grafo pueden o no tener una dirección. En particular, si se desea modelar vialidades con el uso de grafos, es necesario que la dirección del grafo esté representada para poder reproducir el sentido de las calles. Por lo tanto, es necesario conocer acerca de los grafos cuyas arcos poseen una dirección, es decir, los grafos dirigidos o digrafos.

Definición 1.1.1 *Un digrafo o grafo dirigido es una pareja $D = (V, F)$, donde V es un conjunto de puntos llamados nodos, finito y no vacío, y F es un conjunto de parejas ordenas de elementos de V, llamados arcos, donde a cada pareja ordenada le corresponde a lo más dos arcos y éstas constituyen una relación no reflexiva. Al conjunto de nodos y arcos de D se le denota por $V(D)$ y $F(D)$ respectivamente.*

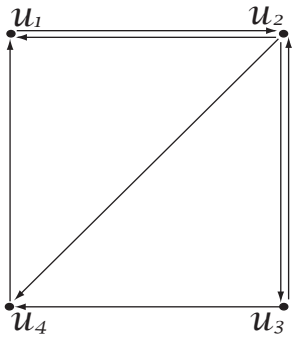


Figura 1.1. Diagrama de D

Ya que la relación F del digrafo D no necesariamente es simétrica, entonces si (u, v) es una arco de D no siempre (v, u) es una arco de D. El diagrama que se muestra en la Figura 1.1 es el generado por el conjunto de nodos $V = \{u_1, u_2, u_3, u_4\}$ y el conjunto de arcos $F = \{(u_1, u_2), (u_2, u_1), (u_3, u_4),$-$(u_2, u_3)\}$. A excepción del digrafo de la Figura 1.1, los demás digrafos que aparezcan en el texto que tengan arcos dobles, es decir, arcos que vayan de un nodo a otro y de regreso, estarán representados en el diagrama como un arco que une a dos nodos con una punta de flecha en cada extremo.

Definición 1.1.2 *La cardinalidad del conjunto de nodos y de arcos de D se denotan por $|V(D)|$ y por $|F(D)|$, respectivamente.*

Definición 1.1.3 *Sea v un nodo de D. El ingrado de v es el número de arcos de D que inciden en v y se denota por $\delta_D^-(v)$.*

Definición 1.1.4 *Sea v un nodo de D. El exgrado es el número de arcos que salen de v y se denota por $\delta_D^+(v)$.*

Definición 1.1.5 *El grado de v es la suma del ingrado más el exgrado y se denota por $\delta_D(v)$.*

Definición 1.1.6 *Sea v un nodo de D. La vecindad del nodo i en D, se define como el conjunto $N(v) = \{w \in V(D)$ tal que $(v, w) \in F(D)\}$, es decir, $N(v)$ es el conjunto de nodos adyacentes a v.*

Definición 1.1.7 *Sea v un nodo de D. Se define al conjunto de arcos $B(v) = \{(v, w) \in F(D)$, -$(u, v) \in F(D)$ tal que $u, w \in F(D)\}$, es decir, $B(v)$ es el conjunto de arcos que inciden y salen de v.*

El subíndice de las tres definiciones anteriores indica que el nodo v pertenece al digrafo D, sin embargo, si es claro que v es un nodo de D se puede denotar al ingrado, el exgrado o el grado de v, simplemente por $\delta^-(v)$, $\delta^+(v)$ o $\delta(v)$ respectivamente. En la Figura 1.1 $\delta^-(u_2) = 1$, $\delta^+(v) = 2$, $\delta(v) = 3$, $N(u_1) = \{u_2\}$ y $B(u_1) = \{(u_4, u_1), (u_2, u_1), (u_1, u_2)\}$.

Las nociones de camino, paseo y ruta, son necesarias para modelar segmentos de la red de trasporte, entonces, si un usuario desea trasladarse de un punto A a un punto B, una ruta, camino

o paseo modela las indicaciones que el usuario debe seguir para llegar al destino.

Definición 1.1.8 *Sean u y v nodos de un digrafo D. Un uv-camino dirigido, es una sucesión de nodos $\overrightarrow{C}= (u = u_1, u_2, u_3, \ldots, u_n = v)$ tal que $(u_i, u_{i+1}) \in F(D)$ para toda $i \in \{1, 2, \ldots, n-1\}$.*

Definición 1.1.9 *Sean u y v nodos de un digrafo D. Un uv-paseo dirigido $\overrightarrow{P}$ es un uv-camino dirigido que no repite arcos.*

Definición 1.1.10 *Sean u y v nodos de un digrafo D. Una uv-ruta dirigida $\overrightarrow{T}$, es un uv-camino dirigido que no repite arcos.*

Definición 1.1.11 *Un ciclo dirigido $\overrightarrow{C}$, es una uv-ruta dirigida donde los únicos nodos que se repiten son el primero y el último.*

Los árboles y árboles de expansión dirigidos son una clase de digrafos en la cuál se basan varios algoritmos para encontrar rutas más cortas, (ver Ahuja, Magnanti & Orlin 1993). Para comprender bien el concepto de árbol es necesario conocer el concepto de conexidad en los digrafos, que se enuncia a continuación.

Definición 1.1.12 *Sea D un digrafo.*

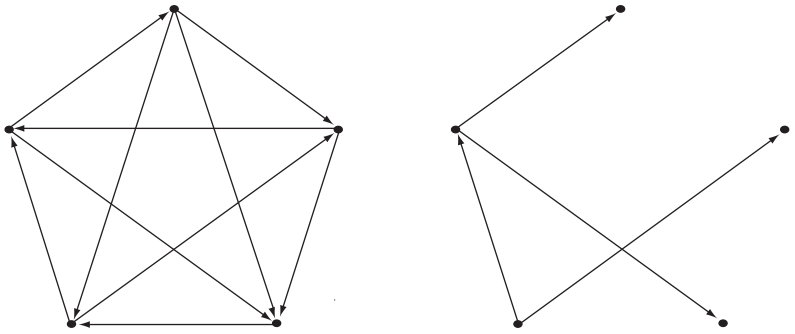
- *D es conexo si para todo $u, v \in V(D)$ existe un uv-camino contenido en D*
- *D es unilateralmente conexo si para todo $u, v \in V(D)$ existe un uv-camino dirigido contenido en D.*
- *D es fuertemente conexo si para todo $u, v \in V(D)$ existe un uv-camino dirigido y también un vu-camino dirigido contenidos en D.*

Se observa que para redes viales, los digrafos que las representan son fuertemente conexos, puesto que siempre es posible ir de un lugar a otro y de regreso. Por ejemplo, si se está parado en una esquina de Tlalpan y se desea ir al Zócalo, siempre es posible ir y venir a través de la red vial, entre este par de lugares.

Definición 1.1.13 *Un digrafo D es un árbol dirigido, si D es conexo y sin ciclos.*

Definición 1.1.14 *Sea D un digrafo conexo. El árbol generador de D, es un sub digrafo de D que incluye a todos los nodos de D y además es un árbol.*

La Figura 1.2 muestra uno de los árboles generadores del grafo D.



D Árbol generador de D

Figura 1.2. D y uno de su árboles generadores.

1.1.2. Híper grafos dirigidos

El estudio de los híper grafos es de gran ayuda para la modelación de redes de transporte público pues éstos permiten reproducir el fenómeno de esperar un autobús en determinada estación. A diferencia de los digrafos cuyas flechas sólo representan un costo (distancia, tiempo, comodidad, etcétera) los híper grafos también pueden modelar, con la ayuda de híper arcos, una distribución de probabilidad que en el caso de los sistemas de transporte público se entiende como la frecuencia de algún modo de transporte. El uso de híper grafos es de especial ayuda en donde el transporte público carece de itinerarios, pero en donde es posible aproximar las frecuencias de las diferentes rutas en cada estación. A continuación, se definen conceptos similares a los de digrafos, en los híper grafos.

Definición 1.1.15 *Un híper grafo dirigido o h-grafo es un pareja $H = (V, E)$, en donde $V = (v_1, v_2, \dots v_n)$ es un conjunto de nodos y $E = (e_1, e_2, \dots e_m)$ es un conjunto de híper arcos. Un h-arco $e \in E$ es la pareja $e = (t(e), h(e))$ en donde $t(e) \subset V$ denota el conjunto de los nodos cola y $h(e) \subset V$, denota el conjunto de los nodos cabeza. Es decir, el h-arco conecta subconjuntos de V.*

Los híper grafos pueden ser representados de dos formas, la Figura 1.3 muestra dichas representaciones.

Definición 1.1.16 *La cardinalidad de un h-arco es el número de nodos contenidos en él y se denota por $|e|$. El tamaño de H es la suma de las cardinalidades de los h-arcos, es decir,*

$$tamaño(H) = \sum_{e \in E} |e|$$

Definición 1.1.17 *La estrella saliente de $u \in V(H)$ es el conjunto de arcos que salen de u y se define por $FS(u) = \{(u, y) \in E \text{ tal que } y \in V\}$. Por otro lado, la estrella entrante $u \in V(H)$ es el conjunto de arcos que entran a u y se define por $BS(u) = \{(y, u) \in E \text{ tal que } y \in V\}$*

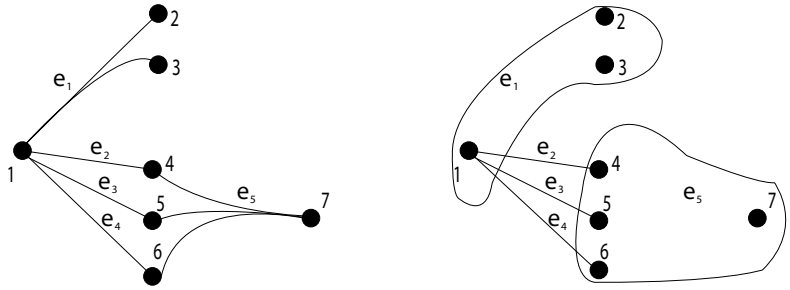


Figura 1.3. Dos forma de representar híper grafos

Definición 1.1.18 *Sea H un h-grafo. Una ruta, r_{od}, que conecta un origen o y un destino d es una secuencia de nodos y h-arcos, $r_{od} = (o = t(e_1), e_1, t(e_2), e_2, \dots e_m, d)$, en donde $t(e_{i+1}) \in h(e_i)$ para $i = 1, 2, \dots m-1$ y $d \in h(e_m)$.*

Definición 1.1.19 *Una híper ruta r_{od} es el conjunto mínimo sin ciclos de rutas r_{od}, tal que el destino, d, está conectado a cualquier nodo que pertenezca a r_{od}.*

Definición 1.1.20 *Un híper grafo H es un h-árbol, si H es conexo y sin ciclos.*

Existe una gran cantidad de textos y resultados acerca de los híper grafos, en esta tesis sólo se mencionan algunos de los avances que se han encontrado sobre el problema *HRMC*. Para un análisis más extenso de la teoría de híper grafos consultar *Introduction to Graph and Hypergraph Theory* de Voloshin (2009) o *Hypergraphs: Combinatorics of Finite Sets* de Berge (1989).

1.2. Rutas más cortas

A grandes rasgos el problema de la ruta más corta (RCM) consiste en encontrar la ruta entre dos nodos tal que la suma de los pesos de los arcos que la conforman sea minimizado. En esta tesis se considera que los pesos asociados a cada arco son una variable que representa el tiempo de viaje entre cada par de nodos. Para el caso determinista existen varios algoritmos que resuelven el problema, como el de *Dijkstra* o el de *Bellman-Ford-Moore* (ver Bang-Jensen & Gutin 2007).

Hay dos formas de ver el problema cuando el tiempo de recorrido asociado a las arcos, es una variable que depende de la hora de llegada a una arco en particular. Una posibilidad es que se considere que el peso de los arcos cambia con el tiempo, es decir, que para cierta hora del día el peso del arco es una constante. Si lo que se busca es optimizar el tiempo de viaje, es posible encontrar una solución única o un conjunto de soluciones igualmente buenas. Más adelante se muestran dos ejemplos de cómo se trataron este tipo de problemas.

Si en cambio, el peso de los arcos está dado por una función de probabilidad que depende de la hora del día, la dificultad del problema se incrementa, ya que pueden existir varias soluciones con una cierta probabilidad de ser las mejores (Miller-Hooks 1997). Aunque en esta tesis no se trabaja

con variables aleatorias, es importante tener en cuenta este enfoque para tener un panorama más amplio del problema.

Ya que el peso de los arcos es una variable aleatoria que depende de la hora del día, entonces una misma ruta puede tener diferentes pesos con cierta probabilidad de realización, por lo que es inmediato concluir que el conjunto de rutas que van del origen al destino tienen asociado un peso con cierta probabilidad de realización. Por lo tanto, el conjunto de soluciones en un problema de rutas más cortas con variables aleatorias es un conjunto de rutas r_i con peso t_i y probabilidad p_i tal que $1 \leq i \leq s$.

Entonces, para encontrar una solución óptima al problema RMC con variables aleatorias es necesario encontrar un conjunto de soluciones *Pareto-optimal*, esto es, determinar el sub-conjunto del conjunto de soluciones, tales que, mejoren en tiempo y probabilidad a todo el conjunto de soluciones, pero es imposible hacer una comparación entre las soluciones del sub-conjunto. Por ejemplo, si el conjunto de soluciones son las rutas; r_1 con costo 20 y probabilidad 0.30, r_2 con costo 25 y probabilidad 0.35 y r_3 con costo 26 y probabilidad 0.28, se observa que r_1 y r_2 mejoran en costo y probabilidad a r_3, sin embargo, aunque r_1 mejora el costo de r_2, r_1 no mejora la probabilidad de realización de r_2. Por lo tanto r_1 y r_2 forman parte del conjunto *Pareto-optimal*.

Miller-Hooks (1997) describe tres algoritmos para encontrar el conjunto *Pareto-optimal*. Los primero dos algoritmos utilizan la *dominancia determinista* para encontrar el conjunto *Pareto-optimal*. La *dominancia determinista* mantiene que para una hora de salida fija, si el peor tiempo de viaje de una ruta r_1 es menor que el mejor tiempo de viaje de una ruta r_2, entonces la r_2 tiene probabilidad cero de dominar a r_1. Por lo tanto, para una hora de salida dada, se tiene la certeza de que r_1 mejorará en tiempo a r_2. Al conjunto de rutas que no son dominadas por ninguna otra con respecto a la *dominancia determinista* se les conoce como el conjunto *Pareto-optimal determinista*.

El tercer algoritmo utiliza en concepto de *dominancia estocástica de primer orden*. Esta dominancia plantea que; para una hora de salida dada, la ruta r_1 domina a la ruta r_2, si para todos los tiempos de viaje posibles, la probabilidad de que el tiempo de viaje de r_1 sea menor o igual que un tiempo t, siempre es mayor que la probabilidad de que el tiempo de viaje de r_2 sea menor o igual que t, entonces, se dice que la r_1 domina a r_2. Al conjunto de rutas que no son dominadas bajo este criterio se les conoce como el conjunto *Pareto-optimal estocástico de primer orden*. Se observa que en la *dominancia estocástica de primer orden*, existe la posibilidad que una ruta dominante no lo sea, pues es probable que alguna otra ruta lo domine. Para revisar más a fondo los algoritmos que se presentan a continuación consultar Miller-Hooks (1997).

- *Stochastic,Time Dependant Least Time(Determinstic Domoniance)* o *STDLT(DD)*: Algoritmo basado en dominancia determinista, que genera todas la rutas de menor tiempo y sus correspondientes funciones de distribución cumulativas.

- *Range*: Conociendo el menor tiempo de viaje y el mayor de cada ruta, el algoritmo genera el conjunto *Pareto-optimal determista* y los rangos asociados a cada ruta.

- *Stochastic,Time Dependant Least Time(Stochastic Domoniance)* o *STDLT(SD)*: Similar al

algoritmo *STDLT(DD)*, sólo que para determinar el conjunto *Pareto-optimal* utiliza el enfoque estocástico.

En párrafos anteriores se explicó el caso donde el peso de los arcos cambia con el tiempo. Enseguida se muestran dos procedimientos para resolver este tipo de problemas.

Nie & Wu (2009) proponen una forma de transformar una red con variables aleatorias en una red con variables estáticas. Ellos definen el problema de las RMCVA en digrafos de la siguiente forma; sea $D(N, A, P)$ un digrafo fuertemente conexo, en donde, N es el conjunto de nodos, A el conjunto de arcos y P es la función de probabilidad que asigna el tiempo de recorrido a los arcos. Si se supone que el tiempo es una variable aleatoria independiente y cada variable está definida por una distribución aleatoria con función de densidad p_{ij}. Entonces, dependiendo de la hora de llegada al nodo origen, es posible descomponer el problema de forma tal que las variables aleatorias sean estáticas, y por lo tanto, encontrar una solución a cada sub-problema no presente mayores dificultades.

En Pallottino & Scutellà (2003) el tiempo está dado por una función que cambia el peso a un subconjunto de arcos, este cambio mejora o empeora el tiempo de recorrido en el subconjunto. Por medio de programación lineal los autores definen el problema de la siguiente forma.

Sea $D(N, A)$ un digrafo conexo, sin arcos dobles, donde N es el conjunto de nodos, con cardinalidad n y A el conjunto de arcos con cardinalidad m. Se define la función $c : A \to \mathbb{R}$ que la asigna un costo c_{ij} a cada arco (i, j). El problema consiste en encontrar el árbol generador del digrafo con nodo raíz r tal que minimice el costo de la ruta de r a i, para todo $i \in V(D)$ y $r \in V(D)$. Si se desea encontrar un solución finita al problema, es necesario que en el digrafo no existan ciclos dirigidos con costos negativos Bang-Jensen & Gutin (2007). El problema se formula de la siguiente manera.

$$min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$S.a:$$

$$\sum_{(j,i)\in BS(i)} x_{ij} - \sum_{(i,j)\in FS(i)} x_{ij} = \begin{cases} -n+1 \text{ si } i = r \\ 1 \text{ si } i \in N \setminus \{r\} \end{cases}$$

$$x_{ij} \geq 0 \quad \forall \quad (i, j) \in A,$$

donde los conjuntos de arcos $FS(i)$ y $BS(i)$, también conocidos como estrella saliente y entrante, respectivamente, se definen de la siguiente forma,

$$FS(i) = \{(u, v) \in A \quad \text{tal que} \quad u = i\}$$
$$BS(i) = \{(u, v) \in A \quad \text{tal que} \quad v = i.\}$$

A grandes rasgos, para resolver el problema primero es necesario encontrar un árbol de expansión T^* con costo mínimo y a partir de éste encontrar el árbol de expansión óptimo cuando una cierta cantidad de pesos en los arcos de T^* cambia. Usado reoptimización y algoritmos del tipo

Dijkstra Pallottino & Scutellà proponen una metodología para resolver el problema.

El enfoque de Pallottino & Scutellà (2003) es similar al enfoque de Nie & Wu (2009) pues el último supone que el peso de los arcos varía con el tiempo, es decir, para cierta hora del día a un subconjunto de los arcos se le altera el peso.

1.3. Rutas más cortas en sistemas de trasporte multimodal

En ciudades grandes, donde existen dos o más modos de trasporte y que además dichos modos conducen al mismo lugar por diferentes rutas, con distintos tiempos y diferentes precios, resulta importante conocer cómo moverse de un lugar a otro haciendo el menor tiempo posible, el menor número de trasferencias o el menor costo. Este tipo de situaciones se pueden modelar con el uso de híper grafos. A continuación se enuncia una definición general para los híper grafos que modelan redes de transporte multimodal.

1.3.1. Híper grafos multimodales

La híper red que modela varios modos de transporte o híper grafo multimodal, se define de la siguiente manera.

Definición 1.3.1 *Una híper red multimodal es la tripleta (N,E,M), donde N es el conjunto de nodos, E es el conjunto de los h-arcos y M son los modos de transporte asociados a los h-arcos.*

Lozano & Storchi (2002) consideran seis subconjuntos de modos de transporte, es decir, existe $M_i \subset M$, para toda $1 \leq i \leq 6$. Sin embargo, se observa que dependiendo de las necesidades del modelo, es posible utilizar menos o más subconjuntos de los aquí propuestos. Para este caso se consideran los subconjuntos,

- M_1: Trasporte sobre rieles (metro y tren).
- M_2: Trasporte público en superficie (camión, taxi, etcétera).
- M_3: Trasporte privado con necesidades de estacionamiento (automóviles y motocicletas).
- M_4: Trasporte privado sin necesidades de estacionamiento (bicicletas).
- M_5: Trasporte a pie.
- M_6: Transferencias modales. Por ejemplo, abandonar M_2 y continuar por algún M_i tal que $i \neq 2$.

En secciones anteriores, se estudió que un híper grafo conecta conjuntos de nodos entre sí a través de híper arcos. Algunos de estos nodos representan las estaciones de cada modo de trasporte y el resto simplemente son entronques de calles. Por otra parte, los *arcos* y *h-arcos* representan una de las siguientes opciones.

Definición 1.3.2 *Sea H una híper red multimodal. Los arcos y h-arcos de H se definen de la siguiente forma:*

- *Un h-arco de abordaje representa la acción de esperar o tomar algún modo M_2.*
- *Un arco de viaje representa la acción de viajar por un modo M_i, $i \in \{1,2,3,4,5\}$. O la de viajar por alguna línea de un modo M_i, $i \in \{1,2\}$, y después abandonarla.*
- *Un arco de transferencia modal representa la acción de abandonar el modo transporte actual y aproximarse a un nuevo modo de trasporte, o la acción abandonar el modo de trasporte actual, aproximarse al nuevo modo de trasporte y abordar dicho modo.*

Se observa que a cada tipo de arco se le deben de asignar diferentes pesos. Los *arcos de viaje* y los *arcos de trasferencia modal* se les asocia un tiempo de recorrido. Ahora, los *h-arcos de abordaje* tienen asociados la frecuencia del modo de transporte que será abordado.

Ya que en los sistemas de transporte público es común que existan diferentes líneas L del algún modo de transporte M_2, entonces, en algún punto dado existe un subconjunto de líneas L que pasan por dicho punto. Sea $L_i \subset L$, el subconjunto de líneas que pasa por el nodo $i \in N$.

Cuando un usuario arriba a la estación i, existe la posibilidad de que más de una línea de transporte lo lleve a su destino. El conjunto de líneas de trasporte que pasan por i y que llegan al destino seleccionado por el usuario recibe el nombre de *conjunto atractivo*. Es decir, el *conjunto atractivo* $L'_i \subseteq L_i$ es el conjunto de líneas que van de i a d, tal que en el nodo i el usuario está dispuesto a abordar la primera corrida que arribe a la estación i. Además, por construcción L'_i tiene asociado un *h-arco de abordaje* $e'_i = (i, h(e'))$, donde $h(e')$ es el conjunto de las paradas de cada línea en i.

La siguiente notación ayuda a definir el tiempo esperado de viaje del *conjunto atractivo* L'_i. Sea:

- ϕ_j la frecuencia de la línea $l_j \in L_i$, es decir, la frecuencia de alguna línea l_j que pasa por la estación i.
- $\Phi(L'_i)$ la frecuencia combinada del conjunto de líneas L'_i. Esto es, la frecuencia combinada del *conjunto atractivo*
- $\pi_j(L'_i)$ la probabilidad de que la línea l_j arribe a la estación i antes que cualquier otra línea de L'_i.
- t_j tiempo de viaje esperado entre la parada i y el destino si la línea l_j es utilizada. Este tiempo no incluye el tiempo de espera en i.
- $w(L'_i)$ es el tiempo promedio de espera del conjunto atractivo en la parada i.

Ahora, si se supone; que el usuario toma el primer autobús de su conjunto atractivo, que los pasajero arriban a la estación aleatoriamente y que los vehículos de las diferentes líneas llegan a las paradas con una distribución exponencial, esto es, el tiempo de espera promedio es igual al inverso de la frecuencias (Lozano & Storchi 2002), entonces, se cumple que:

- $\Phi(L'_i) = \sum_{l_j \in L'_i} \phi_j$
- $w(L'_i) = \dfrac{1}{\Phi(L'_i)}$
- $\pi_j(L'_i) = \dfrac{\phi_j}{\Phi(L'_i)}$

En general, los arcos de una híper red multimodal tienen asociado un tiempo de viaje $c(i,j)$ y un coeficiente $\pi_j(L'_i) = 1$. Se observa que $\pi_j(L'_i) = 1$, pues no existen otras opciones diferentes en la parada i. Por otro lado, si consideramos un *h-arco de abordaje*, como existen diferentes opciones en la parada i, el tiempo de espera depende de la probabilidad de abordar cierto *h-arco de abordaje* en el *conjunto atractivo*, esto es, el tiempo de espera es $C(e'_i) = w(e'_i)$ más tantos coeficientes como el número de nodos existan en $h(e'_i)$, en donde estos coeficientes son $\pi(e'_i, j)$ para toda $j \in h(e'_j)$ y cumplen que

$$\sum_{j \in h(e'_j)} \pi(e'_i, j) = 1,$$

la sumatoria es igual a uno, pues se supuso que siempre se aborda cualquiera de los arcos del *conjunto atractivo*.

Entonces el tiempo esperado de viaje $V_p(i)$ de ir del nodo i al destino d, asociado con cada híper ruta p_{id}, está dado por

$$V_p(i) = \begin{cases} c(i,j) + V_p(j) & \text{si } e = (i,j) \\ w(e'_i) + \sum_{j \in h(e'_j)} \pi(e'_i, j) V_p(j) & \text{si } e' \text{ es un } \textit{h-arco de abordaje.} \end{cases}$$

Las características descritas en los párrafos anteriores es la información mínima que necesita una híper red para que modele un sistema de transporte multimodal.

En la siguiente sección se enuncian diferentes métodos para encontrar las rutas más cortas en sistemas de transporte multimodal. Es importante mostrar que existe más de una forma de resolver el mismo problema y que cada método que se propone tiene ventajas o desventajas sobre los demás, por lo que no todos los métodos que se estudian en la siguiente sección sirven de guía para el modelo que esta tesis presenta.

1.3.2. Soluciones al problema de rutas más cortas en sistemas de trasporte público

A lo largo de la sección se explican cuatro diferentes métodos para resolver el problema de rutas más cortas en sistemas de transporte multimodal.

Una forma de encontrar las híper rutas más cortas (HRMC) la propusieron Lozano & Storchi en 2002 haciendo uso de *híper rutas viables mutimodales*.

Definición 1.3.3 *Una híper ruta multimodal es viable si las rutas que componen la híper ruta no incluyen más de una sub ruta formada por un mismo modo de transporte.*

El objetivo del problema HRMC en una red de transporte multimodal consiste en encontrar las híper rutas viables con menor tiempo esperado que no requieran más de k transferencias entre modos[1]. Lozano & Storchi (2002) dan una solución al problema utilizando un algoritmo que encuentra el conjunto de híper rutas con mínimo tiempo esperado de viaje y una cota superior para el número de transferencias modales. Ya que en dicho algoritmo se basa el trabajo de esta tesis, en el capítulo 5 se ampliará la discusión de éste.

En un trabajo realizado en 2001, Lozano & Storchi, modelan los sistemas de transporte multimodales, pero ésta vez utilizando grafos. En este caso se consideran tres modos de transporte, privado, metro y otros (autobuses o peatonal). Se observa que el uso de híper grafos no es necesario en este modelo pues no son necesarias las frecuencias de la diferentes líneas de transporte público sólo el tiempo promedio de espera en cada estación. Para desplazarse entre el origen y destino se consideran aquellas rutas que no requieran más de k transferencias modales y que sean *viables*, es decir, que no incluyan más de una sub ruta formada por un mismo modo de trasporte.

Los autores encuentran las rutas más cortas viables usando un modificación del algoritmo *Chrono-SPT*. El algoritmo primero encuentra, si es posible, la ruta más corta sin transferencias modales, después busca la ruta viable que mejore en tiempo a la anterior con una transferencia modal. El procedimiento se repite hasta encontrar el máximo número de transferencias modales permitido, es decir, k. Al final el algoritmo arroja un conjunto *Pareto-optimal* de rutas viables, en donde el usuario decide la ruta que mejor se adapte a sus preferencias.

Por otra parte, Pretolani (2000) propone un modelo donde el tiempo de viaje entre cada par de nodos es una variable aleatoria discreta y los tiempos de espera son enteros. Es decir, los tiempos de espera se discretizan en intervalos de longitud δ, por ejemplo, en una estación el vehículo arriba en un conjunto de instantes $\{0, \delta, 2\delta, t_{max}\delta\}$. En su artículo el autor muestra la forma de expandir la red R que modela el problema en una híper red H. La forma de hacer ésta transformación es asignando a cada tiempo de espera un nodo, esto es, por cada intervalo de tiempo $i\delta$ con $0 \geq i \geq t_{max}$ existe un nodo en H. Los arcos de R son *h-arcos* en H, con un tiempo de viaje que depende del momento en el que se llega al nodo cola. Y dos nodos u y v no son adyacentes si el tiempo de viaje que hay de u a v es inferior a la hora de arribo al nodo v.

El autor demuestra que para *h-redes* modeladas de esta forma, el problema de encontrar la ruta mínima en tiempo esperado se puede resolver en un tiempo $O(k)$, y por lo tanto, su solución no requiere un gran esfuerzo computacional.

Nielsen, Andersen & Pretolani (2003) intentan dar una solución al problema bi-criterio de híper rutas más cortas en donde las variables son aleatorias. Al añadir un criterio más, la complejidad computacional aumenta, y por lo tanto, resulta casi imposible para grafos suficientemente grandes, encontrar soluciones exactas. Nielsen et al. (2003) proponen una heurística para encontrar una

[1] El problema no considera las trasferencias entre las diferentes líneas del mismo modo. Por ejemplo, no se considera como transferencia la acción abandonar la línea 2 del metro y después abordar la línea 3.

solución aproximada en donde además incluyen experiencia computacional de su heurística.

La heurística que se trata en el artículo mencionado consta de dos fases, en la primera se averigua una aproximación a la frontera del conjunto de soluciones y sus correspondientes híper rutas. En la segunda fase, usando la frontera que se encontró y por medio de métodos especiales, se buscan todos los puntos *no-dominados*, para con esto *limpiar* la primera búsqueda. El resultado final es una aproximación buena del conjunto *Pareto-optimal*. Para comprobar esta afirmación los autores probaron el algoritmo utilizando el generador de híper grafos, *TEGP* (*time-expanded generato with peaks*) que emula la condiciones de las redes de tránsito como lo son el nivel de tráfico o los tiempos de espera.

Un acercamiento interesante al problema de RMC en medios de transporte multimodal lo propusieron Modestia & Sciomachen (1998), aunque la forma de modelar el problema con redes es diferente a las mencionadas anteriormente.

Los autores propusieron un modelo donde se consideran tres modos de trasporte estos son:

- Modo privado, esto es, usuarios de la red que se transportan en automóvil propio.
- Modo público, son los usuarios de la red que se transportan utilizando metro, autobús, tren, etcétera.
- Modo peatonal, usuarios de la red que se transportan a pie.

Para modelar la red de transporte privado, público y pedestre definen un digrafo $D = D_d \cup D_p \cup D_w$, en donde los digrafos $D_d(V_d, A_d)$, $D_p(V_p, A_p)$ y $D_w(V_w, A_w)$, modelan las redes de transporte privado, público y pedestre, respectivamente. Tal que, V_a y A_a son los conjuntos de nodos y arcos, respectivamente, para todo $a \in \{d, p, w\}$.

Además, existen subconjuntos de nodos $I_1 = V_d \cap V_w \neq \emptyset$ e $I_2 = V_p \cap V_w \neq \emptyset$. Es decir, hay nodos donde es posible bajarse del vehículo particular y continuar a pie o, bajarse del transporte público y continuar a pie. Sin embargo, $V_d \cap V_p = \emptyset$, ya que es imposible cambiar entre transporte público y privado o viceversa sin antes hacer un recorrido a pie. Cada uno de los arcos de D tiene un costo y un tiempo de viaje asociado dependiendo del modo de transporte al que pertenezcan.

Usando D se crea un grafo $D' = (V', E')$ que modela un sistema de transporte 3-*modal* (con tres modos de transporte distintos). D' se define de la siguiente manera.

Cada nodo $v \in V(D)$, se divide en tres nodos v_d, v_p y v_w, uno para cada modo de transporte, tal que todos los arcos en $B(v_d)$ pertenecen a A_d, todos los arcos $B(v_p)$ pertenecen a A_p y todos los arcos en $B(v_w)$ pertenecen a A_w. Y los nodos v_d, v_p y v_w son tales que $v_d \in V'_d$, $v_p \in V'_p$ y $v_w \in V'_w$, en donde V'_d, V'_p y V'_w son los conjuntos de nodos pertenecientes a la red privada, pública y peatonal, respectivamente en G'. Además, por construcción de V' las siguientes operaciones boleanas se mantienen $V'_d \cap V'_p = \emptyset$, $V'_d \cap V'_w = \emptyset$ y $V'_p \cap V'_w = \emptyset$ y además $V' = V'_p \cup V'_d \cup V'_w$.

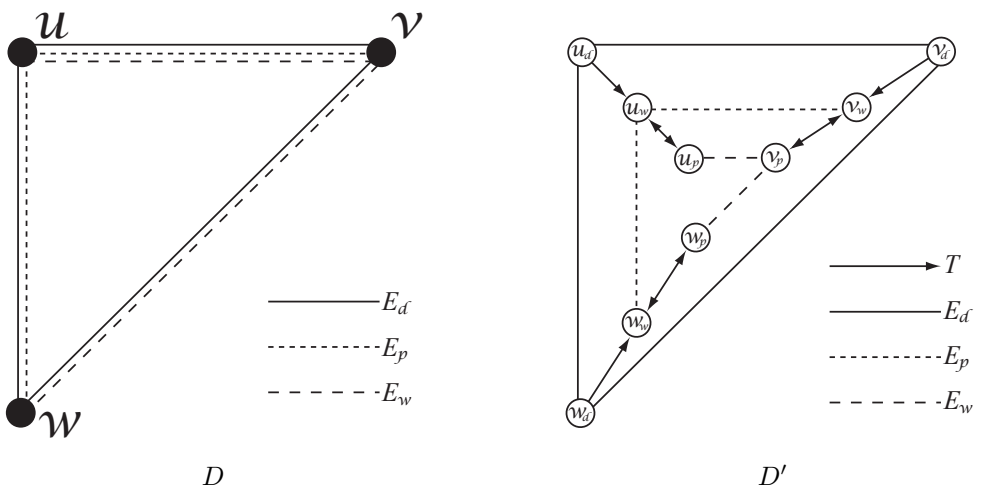


Figura 1.4. Cómo crear D' a partir de D. Ejemplo tomado de Modestia & Sciomachen (1998).

Ahora, se define la conmutación de la red, es decir, el cambio entre nodos de diferentes modos, de la siguiente manera. Sea T un conjunto de arcos tal que $T = T_1 \cup T_2 \cup T_3$, en donde $T_1 = \{(v_d, v_w)$ tal que $v_d \in V'_d$ y $v_w \in V'_w\}$, $T_2 = \{(v_p, v_w)$ tal que $v_p \in V'_d$ y $v_w \in V'_w\}$ y $T_1 = \{(v_w, v_p)$ tal que $v_w \in V'_w$ y $v_p \in V'_p\}$. Se observa que no existen arcos $(v_w, v_d) \in A(D')$, ya que se supone que una vez que desciendes de tu automóvil, no es necesario subir a éste otra vez para llegar a tu destino. La Figura 1.4, muestra un ejemplo de cómo construir D' a partir de D.

Haciendo uso de este red, los autores encuentran las rutas más cortas en donde se minimice, el costo total, el tiempo y la comodidad, encontrando el conjunto *Pareto-Optimal* mediante una heurística. Aunque el modelo de Modestia & Sciomachen (1998) no se basa en el uso de híper grafos y no toma en cuenta las frecuencias de autobuses, ni el número de transferencias, lo que resulta útil para este trabajo es la forma en cómo se creó la red a partir de una simplificación de ésta. Una aplicación similar a la construcción de la red propuesta por Modestia & Sciomachen (1998) se utiliza en esta tesis como se verá más adelante, en el capítulo 4.

CAPÍTULO 2

Ciudad Universitaria

Los antecedentes históricos de la Universidad Nacional Autónoma de México se remontan a 1551. Esta fue la primera universidad de América pero en sus inicios se le conocía como la Real Universidad de México fundada bajo cédulas reales emitidas por el Rey Felipe II de España.

No fue hasta el 22 de septiembre de 1910 que el entonces ministro de Instrucción Pública, Justo Sierra, inauguró la Universidad Nacional de México. Diecinueve años después, en 1929, la Universidad Nacional de México logró a través de un decreto presidencial su autonomía, quedando su nombre como hoy se le conoce: Universidad Nacional Autónoma de México (UNAM).

La inauguración oficial del campus Ciudad Universitaria (CU) fue el 20 de noviembre de 1952, aunque el inicio de las actividades en las facultades no comenzó sino hasta marzo de 1954. El 2 de julio del 2007, el campus central de CU formado por; la zona del estadio, la zona de facultades y la zona deportiva, indicadas en la Figura 2.1 con los polígonos naranja, azul y verde, respectivamente, fue inscrito dentro Patrimonio Cultual de la Humanidad por la Organización Nacional de Naciones Unidas para la Educación la Ciencia y la Cultura (UNESCO). Este gran honor se logró gracias a que se cumplieron los siguientes criterios de selección (UNESCO 2009):

> Criterio I: El campus central de CU de la UNAM, constituye un ejemplo único en el siglo XX donde más de 60 profesionales trabajaron en conjunto en la estructuración de un plan maestro, para crear un conjunto arquitectónico urbano que es un testimonio cultural y social de los valores universales significativos.
>
> Criterio II: Las más importantes tendencias de pensamiento arquitectónico del siglo XX convergen en el campus central de CU de la UNAM: arquitectura moderna, regionalismos histórico e integración plástica; las últimas dos siendo de origen Mexicano.
>
> Criterio IV: El campus central de CU la UNAM es uno de los pocos modelos

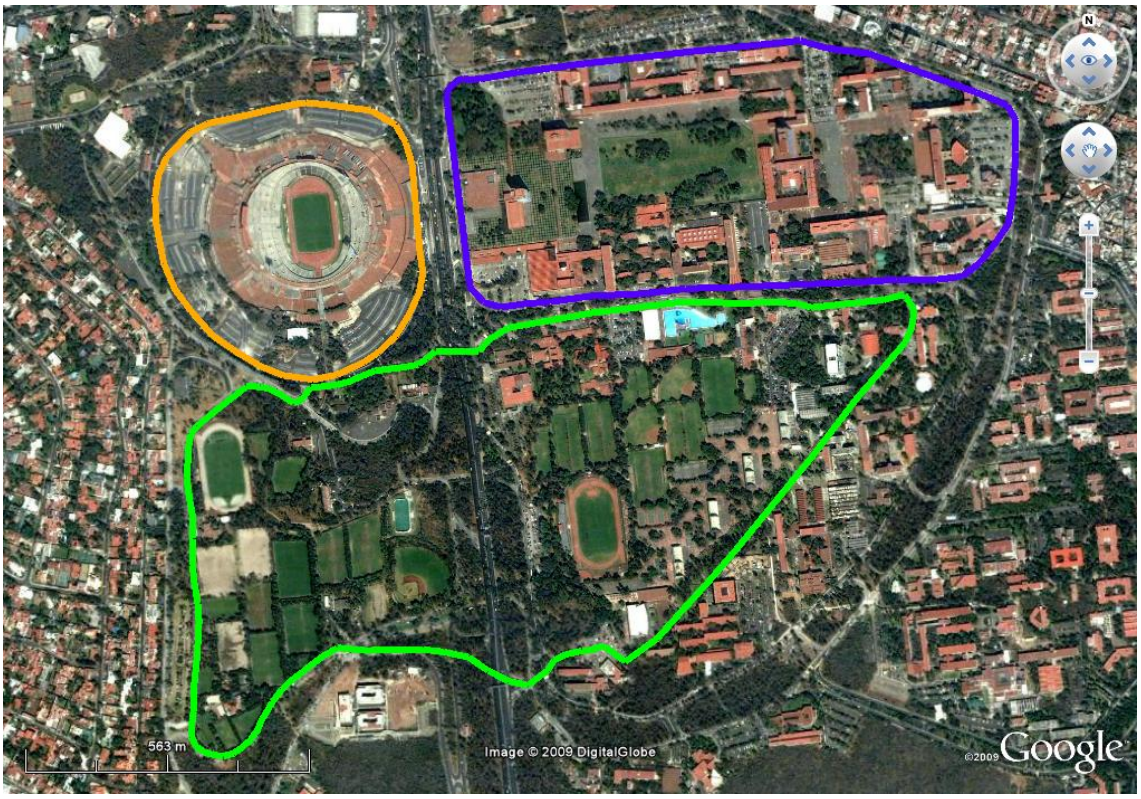


Figura 2.1. Tres zonas son patrimonio cultural de la humanidad.

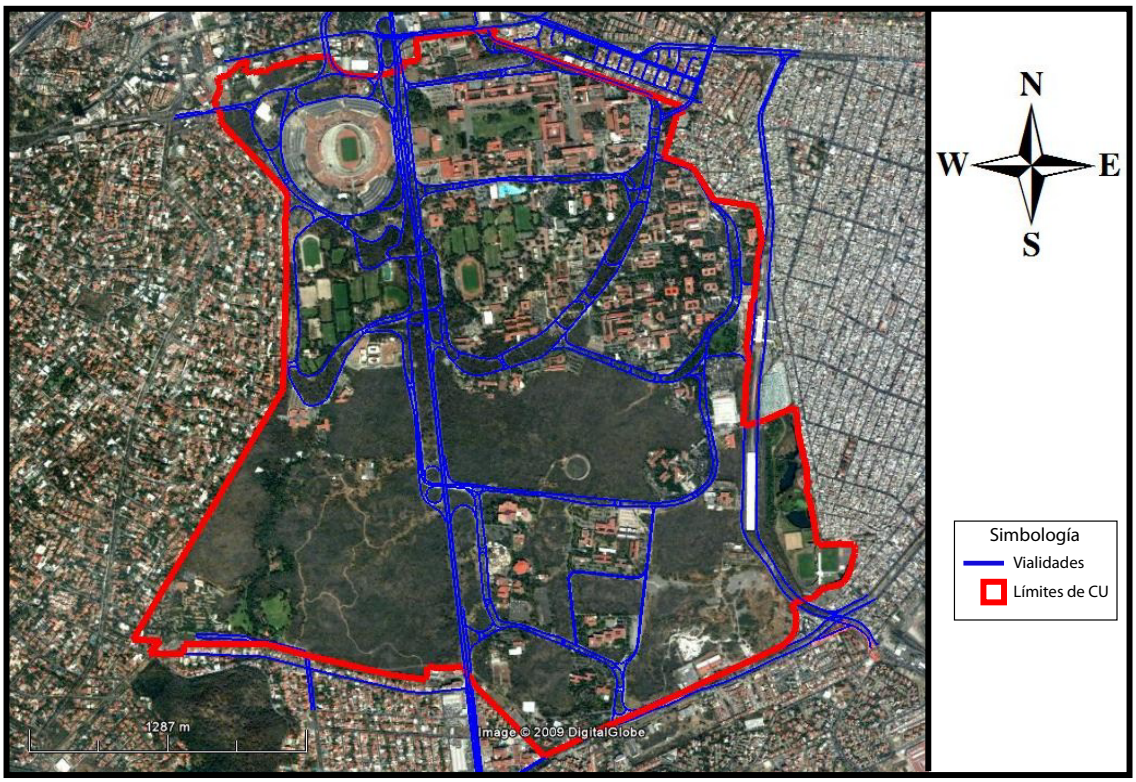


Figura 2.2. Imagen satelital de CU.

> alrededor del mundo donde los principios propuestos por la arquitectura moderna y el urbanismo son totalmente aplicados; el último propósito de tal es ofrecer al humano una mejora en la calidad de vida.

Ciudad Universitaria cuenta con un área construida de 1,139,000 (UNAM 2009*a*) metros cuadrados y cubre una superficie de 7,000,000 metros cuadrados. Su sistema vial se estructura mediante circuitos basado en el sistema inglés denominado Herrey (UNAM 2009*b*). La Figura 2.2 muestra una vista aérea del terreno que ocupa CU y sus vialidades.

Hasta el 2009 CU contaba con 85,000 alumnos de licenciatura, 10,000 alumnos en diversas especializaciones, 12,000 alumnos de maestría y doctorado, 20,000 académicos e investigadores y 25,000 personas destinadas a asuntos administrativos (UNAM 2009*a*). Es decir, existen aproximadamente 161,000 personas dentro de CU. En la Figura 2.3, se puede observar la cantidad de personas por edificio y en la Figura 2.4 su distribución de acuerdo a su ocupación, es decir, estudiantes, académicos o administrativos.

Se observa en la Figura 2.3 que la mayoría de la población universitaria se encuentra en la

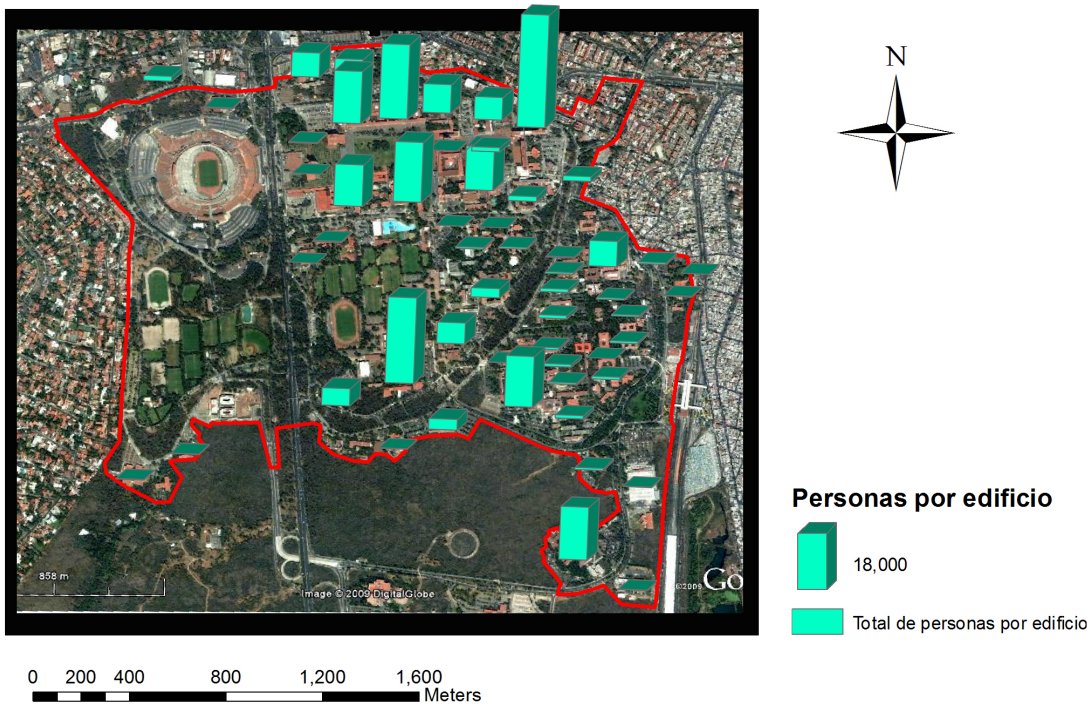


Figura 2.3. Personas por edificio

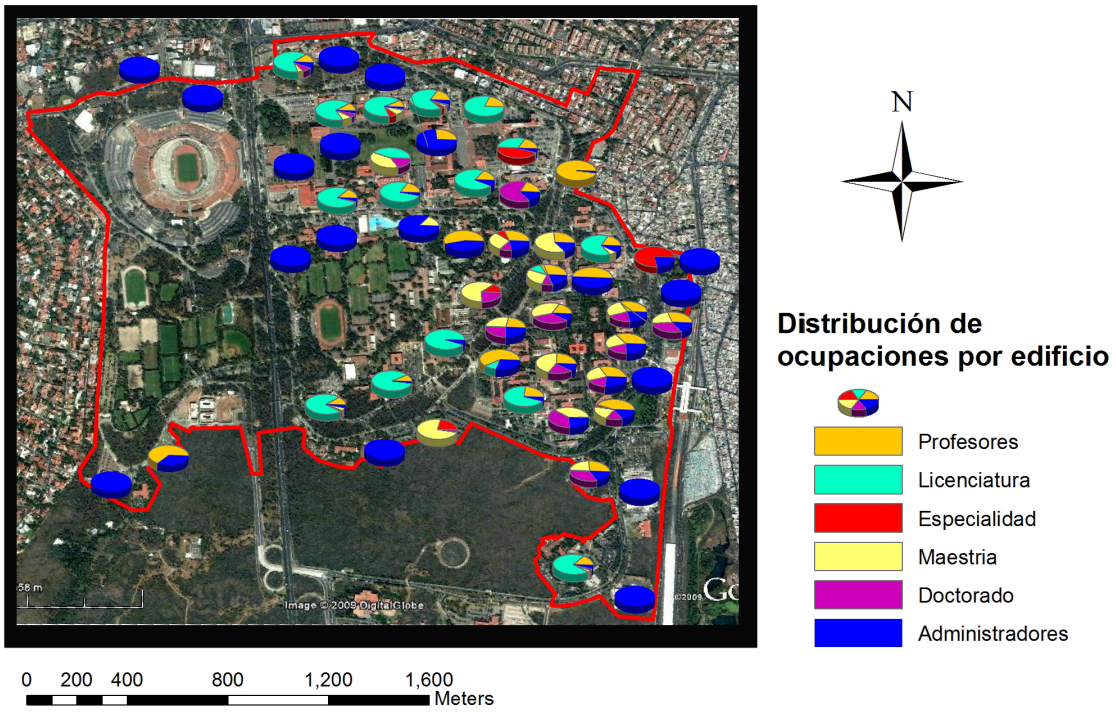


Figura 2.4. Distribución del personal académico, administrativo y estudiantil

zona de facultades, al norte de CU, por lo que es ahí donde transitan gran parte de las rutas de autobuses. Debido a la superficie que cubre CU y la cantidad de gente que labora y estudia ahí, es de gran importancia que el sistema de transporte público, entre otras cosas, sea eficiente y de primer nivel.

Como el territorio y la población de CU son inmensos se decidió delimitar la zona de estudio, sin embargo, el modelo que aquí se propone se puede extender fácilmente si se desea aplicar para CU en su totalidad. La Figura 2.5 muestra la zona de estudio. Se observa que la única parte que no se toma en cuenta es la Zona Cultural, ubicada al sur de CU, ya que esta zona no es considerada de alta actividad estudiantil y académica, el nivel de conexión con la red de autobuses es pobre, y no cuenta con una ciclopista oficial por lo que las alternativas de transbordo multimodales son pocas.

Figura 2.5. Zona de estudio

CAPÍTULO 3

Sistema de transporte público de Ciudad Universitaria

Diariamente en CU circulan automóviles particulares, autobuses de transporte público (también conocidos como *Pumabús*), taxis y bicicletas (las bicicletas de préstamo dentro de CU se les conoce como *Bicipuma*), por lo que la actividad vehicular dentro del campus suele ser intensa en las horas pico.

Antes del 2007 era permitido estacionarse al lado de la acera en casi cualquier lugar dentro de CU, esto ocasionaba que se redujera el espacio de circulación hasta un carril provocando terribles embotellamientos y aumentando la contaminación. Debido a estos problemas se tomó la decisión de prohibir el estacionamiento en banquetas y crear un carril exclusivo de circulación para el *Pumabús*. Para satisfacer la demanda de estacionamientos fue necesario crear nuevos espacios destinados a este propósito, por lo que se abrió al público universitario el estacionamiento del Estadio Olímpico Universitario y se crearon las rutas 6 y 7 de *Pumabús*, con el objetivo de transportar a estudiantes, académicos y trabajadores del estadio a las diferentes facultades y edificios de CU. Posteriormente se crearon las rutas 9, 10 y 11 y se amplió la ruta 8, para satisfacer la demanda generada por la estación de metrobús Ciudad Universitaria. A partir de agosto del 2010 se puso en marcha la actividad de la ruta 12[2] que comunica el área de institutos con el metro y metrobús CU. Hoy en día los únicos lugares donde es permitido estacionarse en la banqueta es en el circuito exterior y en el circuito de la investigación científica, sin embargo, debido al aumento del parque vehicular, esta zona es difícil de transitar, por lo que sería necesario reubicar a estos automóviles. La Figura 3.1 muestra los lugares donde está permitido y donde está prohibido el estacionamiento.

A continuación se describen los procedimientos para digitializar la red de *Pumabús*, *Bicipuma* y la red peatonal. Mas adelante se detalla la información correspondiente de cada una de estas redes.

La red del *Pumabús* y *Bicipuma* se obtuvo utilizando el mapa de rutas, tanto para autobuses

[2] A pesar de que la ruta 12 se encuentra dentro de la zona de estudio, no se tomó en cuenta ya que al momento de concluir la tesis no se encontraba en operación

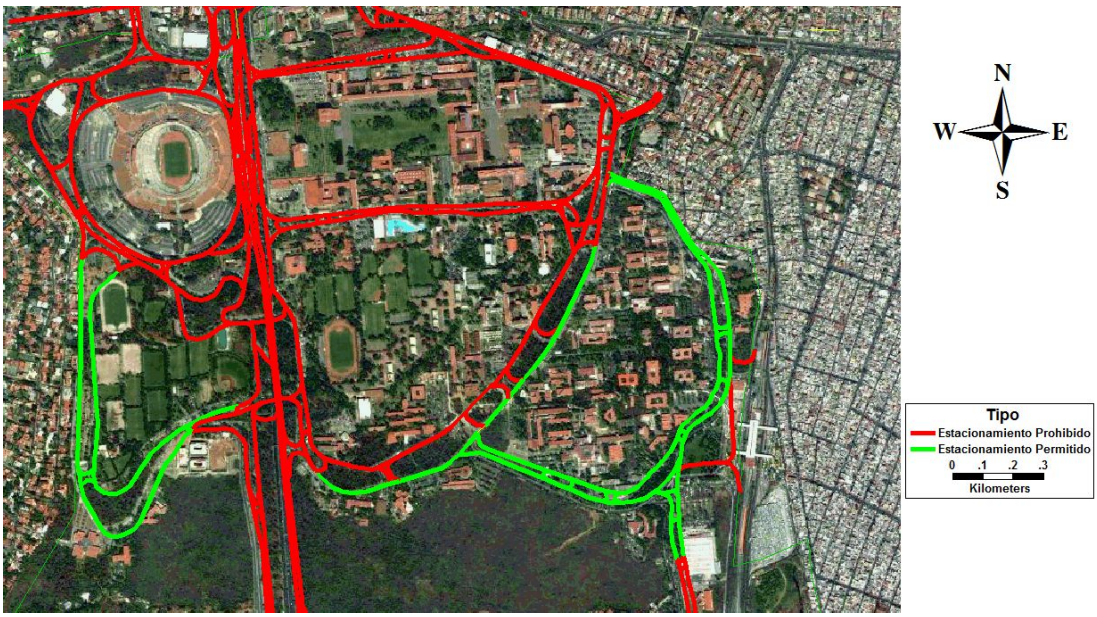


Figura 3.1. Zonas de estacionamiento libre y prohibido en CU

como para bicicletas, que se encuentran en las páginas; http://www.pumabus.unam.mx/ (UNAM 2009*d*) y http://www.tucomunidad.unam.mx/Bicipuma/ (UNAM 2009*c*), respectivamente. Ambas redes se digitalizaron utilizando *Google Earth* y después se exportaron a *TransCAD*.

Las paradas del *Pumabús* y módulos de *Bicipuma* también se digitalizaron en *Google Earth* y se exportaron a *TransCAD*. Para asegurar que la posición en el espacio de la paradas o módulos fuese lo más exacta posible, se visitaron aquellas cuya visualización en *Google Earth* no fuese clara. Por otra parte, las paradas de *Pumabús* localizadas en lados opuestos de la calle, por lo general tienen el mismo nombre, entonces, para evitar confusiones en el modelo, a cada una de las paradas, se les asignó la letra A y B, respectivamente. Por ejemplo, frente a la Facultad de Odontología hay una parada junto a la facultad y otra que se encuentra cruzando la calle por lo que se les nombró Odontología A y Odontología B, respectivamente. Un procedimiento similar se hizo para los paraderos de Ciencias, Derecho, Economía, Filosofía y Letras, Ingeniería, Medicina, Metrobús CU, Metro Universidad, Pista de Calentamiento y E-8.

Además, se consideraron tres horas pico para la obtención de datos del *Pumabús* y *Bicipuma*, estas fueron aproximadamente; de 9 a 10 de la mañana, de 2 a 3 de la tarde y de 7 a 8[3] por la noche (CalyMayor 2009).

Las frecuencias de los autobuses se capturaron en cinco puntos diferentes para medir la frecuen-

[3]Debido a que en el horario nocturno muchas de las paradas se encuentran solas y comienza a obscurecer, por seguridad la recolección de datos se realizó 30 minutos antes, es decir, 6:30 a 7:30

Figura 3.2. Puntos de captura y rutas capturadas.

cia de cada ruta en dos lugares distintos. La Figura 3.2 muestra los puntos donde se capturaron las frecuencias, así como las rutas que transitan por esos puntos. Durante el trabajo de campo se observó que aunque los autobuses están pintados del color de su ruta, no siempre recorren la ruta para la cual están diseñados. Por ejemplo, la ruta 1, que le corresponde el color verde, no siempre es recorrida por autobuses de este color. Este hecho nos muestra una mala planeación del sistema *Pumabús* que debiera ser corregida.

Por otro lado, también se observó que al parecer no existe un plan de corridas de los autobuses, es decir, conforme la demanda crece en cada ruta, crece el número de autobuses que mandan a recorrer dicha ruta. Esto puede ocasionar que otras rutas queden sin autobuses, pues se ocupan para abastecer la demanda de la ruta congestionada, y además, ocasiona que los intervalos entre autobuses no sean homogéneos, más adelante con base en las frecuencias obtenidas se ahondará en el tema

A continuación, se presenta una descripción de las redes de *Pumabús*, *Bicipuma* y para finalizar se comenta acerca de cómo se capturó la red peatonal.

Ruta 1 (16 de octubre 2009)					
9:20 a 10:20		14:20 a 15:20		18:30 a 19:30	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:22	9:24	14:23	14:25	18:39	18:40
9:37	9:39	14:26	14:28	18:49	18:53
9:39	9:42	14:30	14:32	19:08	19:11
9:45	9:47	14:44	14:47	19:16	19:19
9:49	9:51	14:56	15:00		
10:03	10:07	15:02	15:04		
10:11	10:13	15:08	15:11		
10:19	10:22	15:13	15:17		

Tabla 3.1. Frecuencias de la ruta 1 capturadas en metro Universidad

3.1. *Pumabús*

El *Pumabús* cuenta con 12 rutas que comunican a toda CU, el transporte es gratuito para cualquier persona y existen facilidades para discapacitados. Las rutas dentro de la zona de estudio son la $1, 2, 3, 4, 5, 6, 7, 8, 9$ y la 11. Hay tres terminales ubicadas en el Estadio Olímpico Universitario, en el metro Universidad y en el metrobús CU. Para cada terminal se consideraron diferentes tiempos de abordaje ya que cada una posee características diferentes, en cambio en el resto de las paradas se homogeneizó el tiempo de abordaje a 25 segundos y la velocidad aproximada de los autobuses es de $25km/h$. A continuación, se describen los atributos de las diferentes rutas en la zona de estudio.

3.1.1. Ruta 1: Metro Universidad - Circuito Interior

La Ruta 1, con terminal en metro Universidad, recorre aproximadamente 7.2 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. y sábados de 6:00 a 15:00 hrs. Las 16 paradas que incluye la ruta, en orden de dirección, son: Metro Universidad A, CENDI, Psiquiatría y Salud Mental, Química, CELE, Ingeniería A, Arquitectura, Rectoría, Psicología, Filosofía y Letras B, Derecho B, Economía B, Odontología B, Medicina B, Veterinaria, Instituto de Geofísica y Química conjunto D y E. La Figura 3.3 muestra la ruta y sus paradas.

Las Tablas 3.1 y 3.2 muestran la frecuencia de los autobuses de la ruta 1, capturadas en el metro Universidad y en la Facultad de Odontología, respectivamente. El tiempo de abordaje en terminal es de 2.7 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 8.5 autobuses cada hora, esto es, un autobús cada 7 minutos.
- De 2 a 3 de la tarde hay 7.5 autobuses cada hora, esto es, un autobús cada 8 minutos.
- De 7 a 8 de la noche hay 5 autobuses cada hora, esto es, un autobús cada 12 minutos.

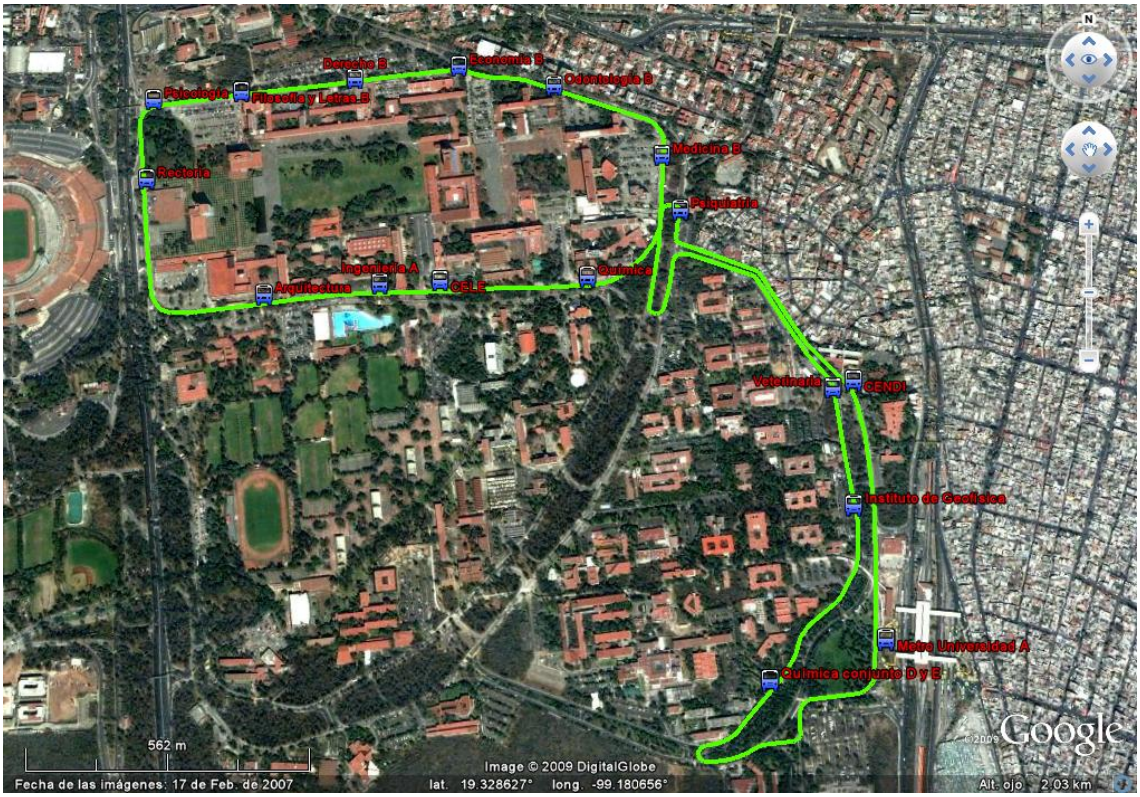


Figura 3.3. Ruta 1: Metro Universidad - Circuito Interior

Ruta 1 (6 de noviembre 2009)		
9:20 a 10:20	14:06 a 15:06	18:40 a 19:40
9:29	14:15	18:42
9:31	14:16	18:43
9:41	14:21	19:03
9:42	14:34	19:08
9:44	14:37	19:26
9:54	14:42	19:28
10:02	15:02	
10:10		
10:14		

Tabla 3.2. Frecuencias de la ruta 1 capturadas en la Facultad de Odontología

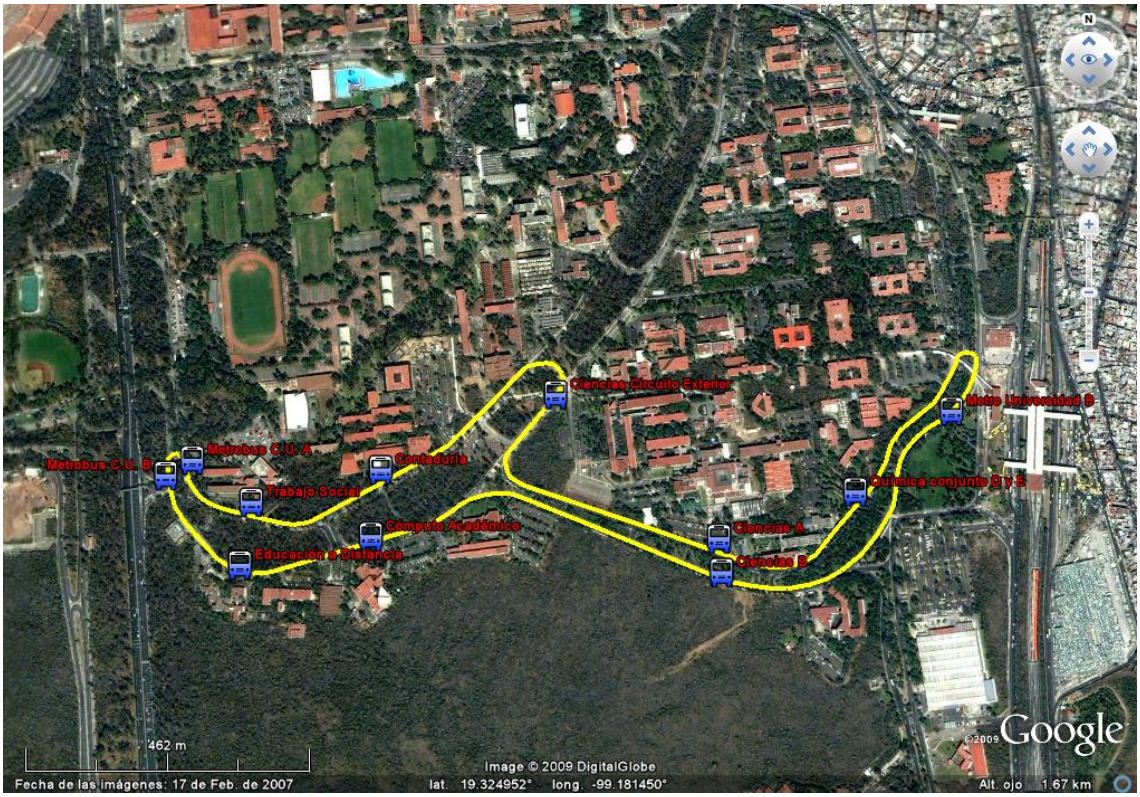


Figura 3.4. Ruta 2: Metro Universidad - Circuito Exterior

3.1.2. Ruta 2: Metro Universidad - Circuito Exterior

La Ruta 2, con terminal en metro Universidad, recorre aproximadamente 4.2 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. y sábados de 6:00 a 15:00 hrs. Las 10 paradas que incluye la ruta, en orden de dirección, son: Metro Universidad A, Química conjunto D y E, Ciencias A, Ciencias Circuito Exterior, Contaduría, Trabajo Social, Metrobús A, Metrobús B, Educación a Distancia, Cómputo Académico y Ciencias B. La Figura 3.4 muestra la ruta y sus paradas.

Las Tablas 3.3 y 3.4 muestran la frecuencia de los autobuses de la ruta 2, capturadas en metro Universidad y en el metrobús CU, respectivamente. El tiempo de abordaje en terminal es de 2.1 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 9.5 autobuses cada hora, esto es, un autobús cada 6 minutos.
- De 2 a 3 de la tarde hay 7 autobuses cada hora, esto es, un autobús cada 8.5 minutos.
- De 7 a 8 de la noche hay 5 autobuses cada hora, esto es, un autobús cada 12 minutos.

Ruta 2 (16 de octubre 2009)					
9:20 a 10:20		14:20 a 15:20		18:30 a 19:30	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:26	9:29	14:24	14:27	18:42	18:44
9:32	9:33	14:30	14:32	18:48	18:50
9:40	9:42	14:43	14:46	19:04	19:06
9:46	9:49	14:49	14:52	19:06	19:08
9:53	9:55	14:54	14:56	19:22	19:25
10:02	10:04	15:05	15:07		
10:09	10:11	15:13	15:15		
10:12	10:15				

Tabla 3.3. Frecuencias de la ruta 2 capturadas en metro Universidad

Ruta 2 (30 de octubre 2009)		
9:12 a 10:12	14:22 a 15:22	18:39 a 19:39
9:12	14:22	18:47
9:18	14:33	18:55
9:20	14:44	19:14
9:30	14:58	19:24
9:32	14:58	19:22
9:43	15:08	
9:46	15:15	
9:47		
9:50		
10:02		
10:05		

Tabla 3.4. Frecuencias de la ruta 2 capturadas en el metrobús CU

Ruta 3 (16 de octubre 2009)					
9:20 a 10:20		14:20 a 15:20		18:30 a 19:30	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:24	9:28	14:20	14:23	18:32	18:33
9:31	9:34	14:29	14:32	18:49	18:53
9:35	9:38	14:35	14:38	18:55	18:57
9:51	9:55	14:39	14:42	19:16	19:19
9:59	10:02	14:43	14:46	19:22	19:24
10:03	10:06	14:52	14:55		
10:18	10:22	14:56	15:00		
		15:06	15:09		
		15:12	15:16		

Tabla 3.5. Frecuencias de la ruta 3 capturadas en metro Universidad

3.1.3. Ruta 3: Metro Universidad - Zona Cultural

La Ruta 3, con terminal en metro Universidad, recorre aproximadamente 8.5 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs., sábados de 6:00 a 23:00 hrs. y los domingos de 6:00 a 23:00 hrs. Las 20 paradas que incluye la ruta, en orden de dirección, son: Metro Universidad B, Química conjunto D y E, Tienda UNAM, Ciencias Políticas, Investigaciones Jurídicas A, Biblioteca Nacional A, Centro Cultural, Revalidación de Estudios, Personal Académico, Archivo General, Avenida del Imán, Investigaciones Filosóficas I, Investigaciones Filosóficas II, Coordinación de Humanidades, Universum, Teatro y Danza, MUAC, Biblioteca Nacional B, Investigaciones Jurídicas B y TV UNAM . La Figura 3.5 muestra la ruta y sus paradas.

Las Tablas 3.5 y 3.6 muestran la frecuencia de los autobuses de la ruta 3, capturadas en metro Universidad y en la Facultad de Ciencias Políticas y Sociales, respectivamente. El tiempo de abordaje en terminal es de 3 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 7.5 autobuses cada hora, esto es, un autobús cada 8 minutos.
- De 2 a 3 de la tarde hay 8 autobuses cada hora, esto es, un autobús cada 7.5 minutos.
- De 7 a 8 de la noche hay 6 autobuses cada hora, esto es, un autobús cada 10 minutos.

Figura 3.5. Ruta 3: Metro Universidad - Zona Cultural

Ruta 3 (13 de noviembre 2009)		
9:18 a 10:18	14:00 a 15:00	18:30 a 19:30
9:24	14:05	18:46
9:26	14:10	18:50
9:30	14:34	18:57
9:41	14:39	19:03
9:52	14:45	19:08
9:55	14:50	19:17
10:01	14:59	19:28
10:10		

Tabla 3.6. Frecuencias de la ruta 3 capturadas en la Facultad de Ciencias Políticas y Sociales

Ruta 4 (16 de octubre 2009)					
9:20 a 10:20		14:20 a 15:20		18:30 a 19:30	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:30	9:33	14:27	14:29	18:31	18:32
9:41	9:44	14:35	14:38	18:34	18:35
9:35	9:38	15:05	15:08	18:55	18:57
9:49	9:52	15:14	15:16	18:58	18:59
9:54	9:57				
10:04	10:07				
10:15	10:17				

Tabla 3.7. Frecuencias de la ruta 4 capturadas en metro Universidad

3.1.4. Ruta 4: Metro Universidad - Jardín Botánico

La Ruta 4, con terminal en metro Universidad, recorre aproximadamente 9.7 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. y sábados de 6:00 a 15:00 hrs. Las 14 paradas que incluye la ruta, en orden de dirección, son: Metro Universidad B, Química conjunto D y E, Ciencias A, Ciencias Circuito Exterior, Contaduría, Trabajo Social, Metrobús C.U., Estadio de Prácticas, Campos Fútbol I, Jardín Botánico, Campos Fútbol II, Metrobús C.U. B, Educación a Distancia, Cómputo Académico y Ciencias B. La Figura 3.6 muestra la ruta y sus paradas.

Las Tablas 3.7 y 3.8 muestran la frecuencia de los autobuses de la ruta 4, capturadas en metro Universidad y metrobús CU, respectivamente. El tiempo de abordaje en terminal es de 2.3 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 6.5 autobuses cada hora, esto es, un autobús cada 9 minutos.
- De 2 a 3 de la tarde hay 4.5 autobuses cada hora, esto es, un autobús cada 13 minutos.
- De 7 a 8 de la noche hay 3.5 autobuses cada hora, esto es, un autobús cada 17 minutos.

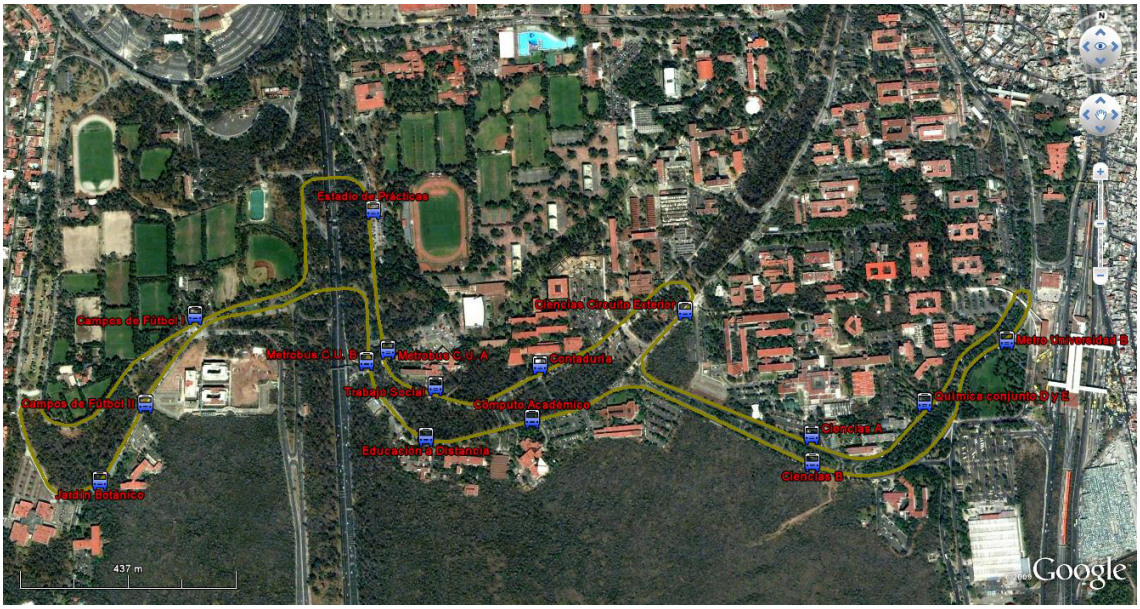


Figura 3.6. Ruta 4: Metro Universidad - Jardín Botánico

Ruta 4 (30 de octubre 2009)		
9:12 a 10:12	14:22 a 15:22	18:39 a 19:39
9:27	14:22	18:45
9:29	14:40	19:03
9:50	14:48	19:39
9:56	15:08	
9:58	15:16	
10:04		

Tabla 3.8. Frecuencias de la ruta 4 capturadas en el metrobús CU

Figura 3.7. Ruta 5: Metro Universidad - Barda Perimetral Norte

3.1.5. Ruta 5: Metro Universidad - Barda Perimetral Norte

La Ruta 5, con terminal en metro Universidad, recorre aproximadamente 8.3 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. y sábados de 6:00 a 15:00 hrs. Las 16 paradas que incluye la ruta, en orden de dirección, son: Metro Universidad A, CENDI, Psiquiatría, Medicina A, Odontología A, Economía A, Avenida Universidad, Derecho A, Filosofía y Letras A, Psicología, Filosofía y Letras B, Derecho B, Economía B, Odontología B, Medicina B, Veterinaria, Instituto de Geofísica y Conjunto de Química D y E. La Figura 3.7 muestra la ruta y sus paradas.

Las Tablas 3.9 y 3.10 muestran la frecuencia de los autobuses de la ruta 5, capturadas en metro Universidad y en la Facultad de Odontología. El tiempo de abordaje en terminal es de 2.5 minutos y las frecuencias de autobuses son:

Ruta 5 (16 de octubre 2009)					
9:20 a 10:20		14:20 a 15:20		18:30 a 19:30	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:26	9:28	14:28	14:31	18:27	18:30
9:37	9:39	14:32	14:34	18:46	18:48
9:43	9:44	14:43	14:46	18:56	18:58
9:45	9:50	14:48	14:50	19:12	19:14
9:56	9:59	15:09	15:12		
10:06	10:09	15:14	15:17		
10:11	10:13				
10:18	10:21				

Tabla 3.9. Frecuencias de la ruta 5 capturadas en metro Universidad

- De 9 a 10 de la mañana hay 8.5 autobuses cada hora, esto es, un autobús cada 7 minutos.
- De 2 a 3 de la tarde hay 6 autobuses cada hora, esto es, un autobús cada 10 minutos.
- De 7 a 8 de la noche hay 3.5 autobuses cada hora, esto es, un autobús cada 17 minutos.

Ruta 5 (6 de noviembre 2009)		
9:20 a 10:20	14:06 a 15:06	18:40 a 19:40
9:25	14:06	19:11
9:34	14:15	19:27
9:41	14:22	19:35
9:44	14:34	
9:49	14:40	
9:53	14:53	
10:05		
10:15		
10:17		

Tabla 3.10. Frecuencias de la ruta 5 capturadas en la Facultad de Odontología

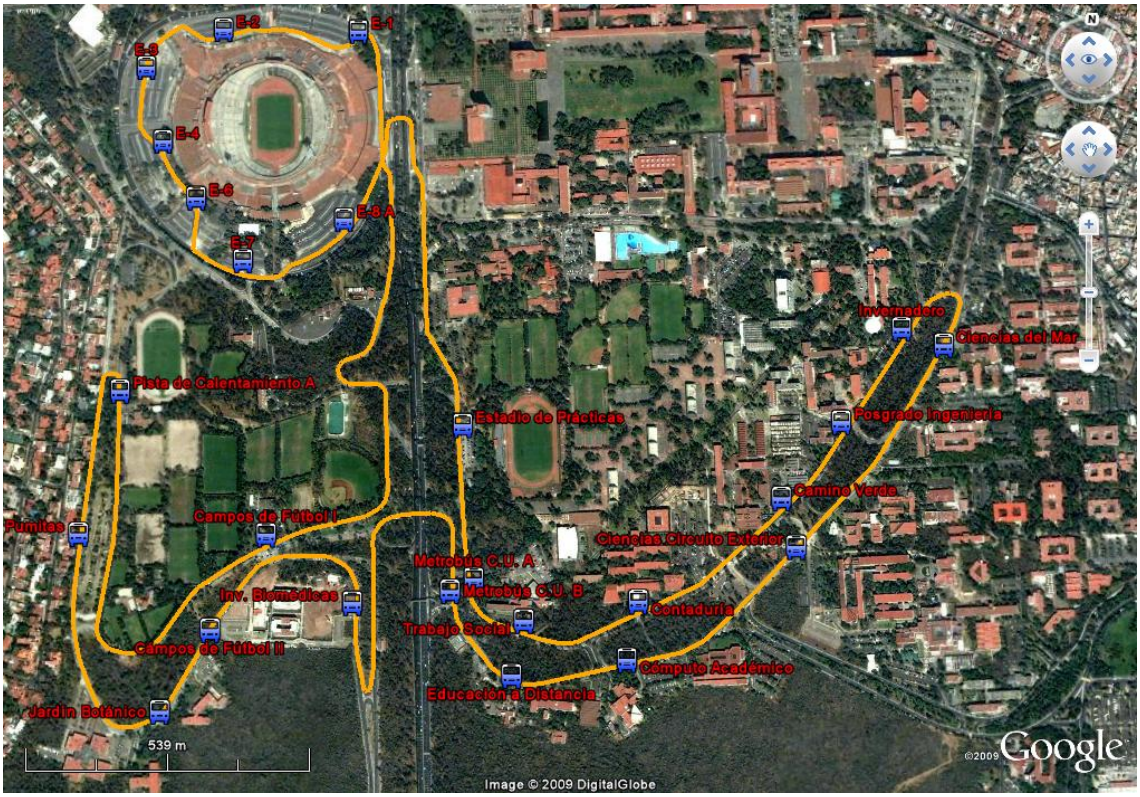


Figura 3.8. Ruta 6: Estadio Olímpico

3.1.6. Ruta 6: Estadio Olímpico

La Ruta 6, con terminal el Estadio Olímpico, recorre aproximadamente 10.2 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. Las 24 paradas que incluye la ruta, en orden de dirección, son: E-1, Campos de Fútbol 1, Jardín Botánico, Campos de Fútbol II, Investigaciones Biomédicas, Metrobús C.U. B, Educación a Distancia, Computo Académico, Ciencias Circuito Exterior. Ciencias del Mar y Limnología, Invernadero, Posgrado Ingeniería, Camino Verde, Contaduría, Trabajo Social, Metrobús C.U. A, Estadio de Prácticas, MUCA, E-8 A, E-7, E-6, E-4, E-3 y E-2. La Figura 3.8 muestra la ruta y sus paradas.

Las Tablas 3.11 y 3.12 muestran la frecuencia de los autobuses de la ruta 6, capturadas en el Estadio Olímpico Universitario y en el metrobús CU. El tiempo de abordaje en terminal es de 1.5 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 6 autobuses cada hora, esto es, un autobús cada 10 minutos.
- De 2 a 3 de la tarde hay 4.5 autobuses cada hora, esto es, un autobús cada 13 minutos.
- De 7 a 8 de la noche hay 3.5 autobuses cada hora, esto es, un autobús cada 17 minutos.

Ruta 6 (23 de octubre 2009)					
9:20 a 10:20		14:03 a 15:03		18:36 a 19:36	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:36	9:37	14:07	14:09	18:40	18:41
9:39	9:41	14:30	14:31	19:08	19:09
9:44	9:46	14:32	14:36	19:18	19:19
10:07	10:08	14:42	14:43		
10:18	10:19				

Tabla 3.11. Frecuencias de la ruta 6 capturadas en el Estadio Olímpico Universitario

Ruta 6 (30 de octubre 2009)		
9:12 a 10:12	14:22 a 15:22	18:39 a 19:39
9:17	14:26	18:42
9:26	14:42	18:49
9:29	14:49	19:07
9:44	14:52	19:22
9:53	14:20	
10:02		
10:10		

Tabla 3.12. Frecuencias de la ruta 6 capturadas en el metrobús CU

3.1.7. Ruta 7: Estadio Olímpico - Circuito Interior

La Ruta 7, con terminal el Estadio Olímpico, recorre aproximadamente 4.8 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. Las 16 paradas que incluye la ruta, en orden de dirección, son: E-1, Psicología, Filosofía y Letras B, Derecho B, Economía B, Odontología B, Medicina B, Química, CELE, Ingeniería A, Arquitectura , E-8 A, E-7, E-6, E-4, E-3 y E-2. La Figura 3.9 muestra la ruta y sus paradas.

Las Tablas 3.13 y 3.14 muestra la frecuencia de los autobuses de la ruta 7, capturadas en el estadio de Olímpico Universitario y en la Facultad de Odontología, respectivamente. El tiempo de abordaje en terminal es de 1.4 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 8.5 autobuses cada hora, esto es, un autobús cada 7 minutos.
- De 2 a 3 de la tarde hay 7.5 autobuses cada hora, esto es, un autobús cada 8 minutos.
- De 7 a 8 de la noche hay 11.5 autobuses cada hora, esto es, un autobús cada 5 minutos.

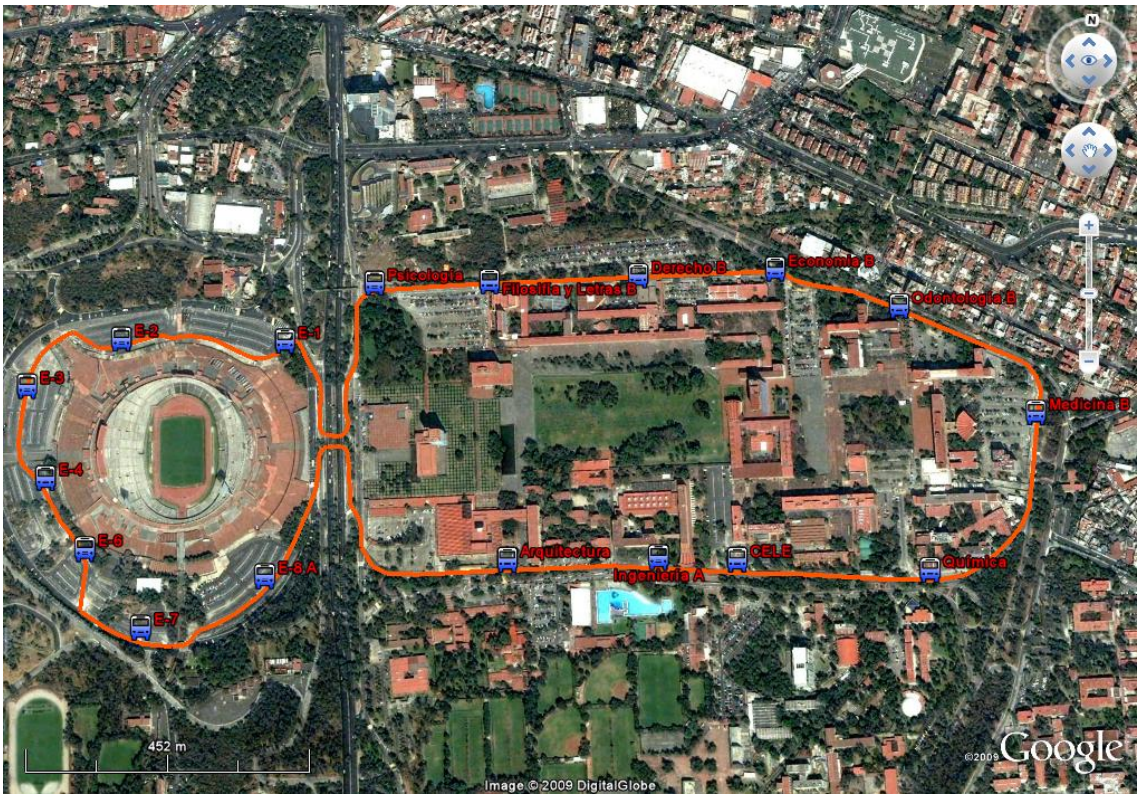


Figura 3.9. Ruta 7: Estadio Olímpico - Circuito Interior

Ruta 7 (23 de octubre 2009)					
9:20 a 10:20		14:03 a 15:03		18:36 a 19:36	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:24	9:25	14:03	14:06	18:36	18:38
9:34	9:35	14:17	14:18	18:40	18:41
9:39	9:41	14:18	14:19	18:49	18:51
9:42	9:44	14:28	14:29	18:54	18:55
9:45	9:48	14:40	14:41	18:56	18:57
9:54	9:56	14:47	14:48	18:58	19:00
9:59	10:02	14:50	14:51	19:03	19:04
10:05	10:06			19:13	19:15
10:10	10:11			19:18	19:19
				19:20	19:21
				19:22	19:23
				19:24	19:26

Tabla 3.13. Frecuencias de la ruta 7 capturadas en el Estadio Olímpico Universitario

Ruta 7 (6 de noviembre 2009)		
9:20 a 10:20	14:06 a 15:06	18:40 a 19:40
9:24	14:09	18:42
9:26	14:23	18:49
9:35	14:25	19:55
9:41	14:27	19:00
9:46	14:34	19:08
9:55	14:35	19:13
9:59	14:42	19:16
10:09	14:57	19:21
		19:31
		19:35
		19:38

Tabla 3.14. Frecuencias de la ruta 7 capturadas en la Facultad de Odontología

3.1.8. Ruta 8: Estadio Olímpico - Circuito Exterior

La Ruta 8, con terminal el Estadio Olímpico y metrobús CU, recorre aproximadamente 5.8 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. Las 18 paradas que incluye la ruta, en orden de dirección, son: E-1, Servicios Médicos, Alberca Olímpica, Frontones, IIMAS, Invernadero, Posgrado Ingeniería, Camino Verde, Contaduría, Trabajo Social, Metrobús C.U. A, Estadio de Prácticas, MUCA, E-8 A, E-7, E-6, E-4, E-3 y E-2. La Figura 3.10 muestra la ruta y sus paradas.

Las Tablas 3.15 y 3.16 muestra la frecuencia de los autobuses de la ruta 8, capturadas en el estadio de Olímpico Universitario y en el metrobús CU, respectivamente. El tiempo de abordaje en la terminal del Estadio Olímpico Universitario es de 1.3 minutos y en la del metrobús CU es de 1.4 minutos. Las frecuencias de los autobuses son:

- De 9 a 10 de la mañana hay 9 autobuses cada hora, esto es, un autobús cada 6.5 minutos.
- De 2 a 3 de la tarde hay 9 autobuses cada hora, esto es, un autobús cada 6.5 minutos.
- De 7 a 8 de la noche hay 7.5 autobuses cada hora, esto es, un autobús cada 8 minutos.

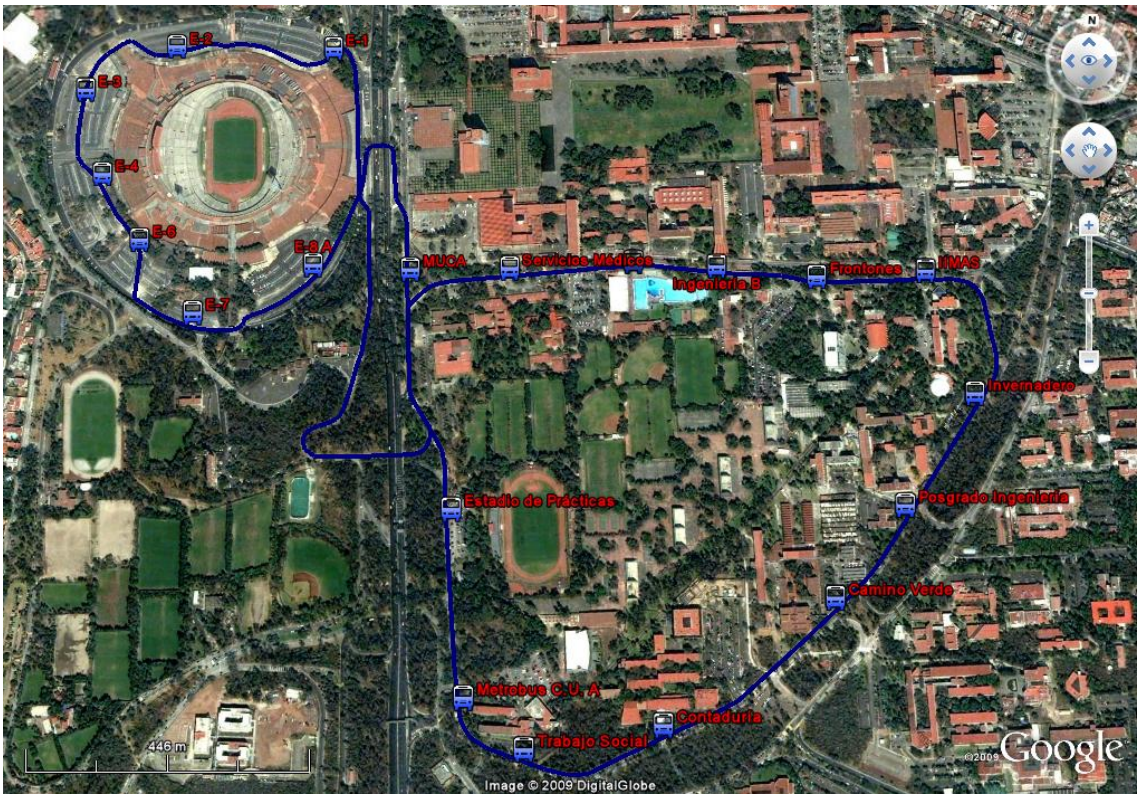


Figura 3.10. Ruta 8: Estadio Olímpico - Circuito Exterior

Ruta 8 (23 de octubre 2009)					
9:20 a 10:20		14:03 a 15:03		18:36 a 19:36	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:20	9:23	14:06	14:07	18:45	18:46
9:31	9:32	14:16	14:17	18:54	18:55
9:37	9:39	14:18	14:20	19:00	19:01
9:42	9:44	14:25	14:26	19:06	19:07
9:44	9:46	14:30	14:31	19:09	19:10
9:54	9:55	14:42	14:44	19:21	19:22
10:00	10:03	14:48	14:49	19:23	19:24
10:08	10:09	14:50	14:51	19:30	19:31
10:10	10:11	14:52	14:54	19:34	19:35
10:19	10:21				

Tabla 3.15. Frecuencias de la ruta 8 capturadas en el Estadio Olímpico Universitario

Ruta 8 (30 de octubre 2009)					
9:12 a 10:12		14:22 a 15:22		18:39 a 19:39	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:15	9:18	14:23	14:24	18:39	18:41
9:29	9:30	14:26	14:27	18:51	18:52
9:33	9:34	14:30	14:32	19:02	19:03
9:39	9:40	14:47	14:48	19:04	19:05
9:43	9:44	14:49	14:50	19:18	19:20
9:54	9:55	14:51	14:52	19:29	19:30
10:57	10:58	14:58	15:00		
10:06	10:09	15:16	15:17		
		15:18	15:19		

Tabla 3.16. Frecuencias de la ruta 8 capturadas en el metrobús CU

3.1.9. Ruta 9: Facultades

La Ruta 9, con terminal el metrobús CU, recorre aproximadamente 4.1 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs y sábados de 6:00 a 15:00 hrs. Las 13 paradas que incluye la ruta, en orden de dirección, son: Metrobús C.U. A, Estadio de Prácticas, MUCA, Rectoría, Psicología, Filosofía y Letras B, Derecho B, Economía B, Odontología B, Medicina B, Invernadero, Posgrado Ingeniería, Camino Verde, Contaduría y Trabajo Social. La Figura 3.11 muestra la ruta y sus paradas.

Las Tablas 3.17 y 3.18 muestra la frecuencia de los autobuses de la ruta 9, capturadas en el metrobús CU y en la Facultad de Odontología, respectivamente. El tiempo de abordaje en terminal es de 1.8 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 9 autobuses cada hora, esto es, un autobús cada 6.5 minutos.
- De 2 a 3 de la tarde hay 7 autobuses cada hora, esto es, un autobús cada 8.5 minutos.
- De 7 a 8 de la noche hay 7 autobuses cada hora, esto es, un autobús cada 8.5 minutos.

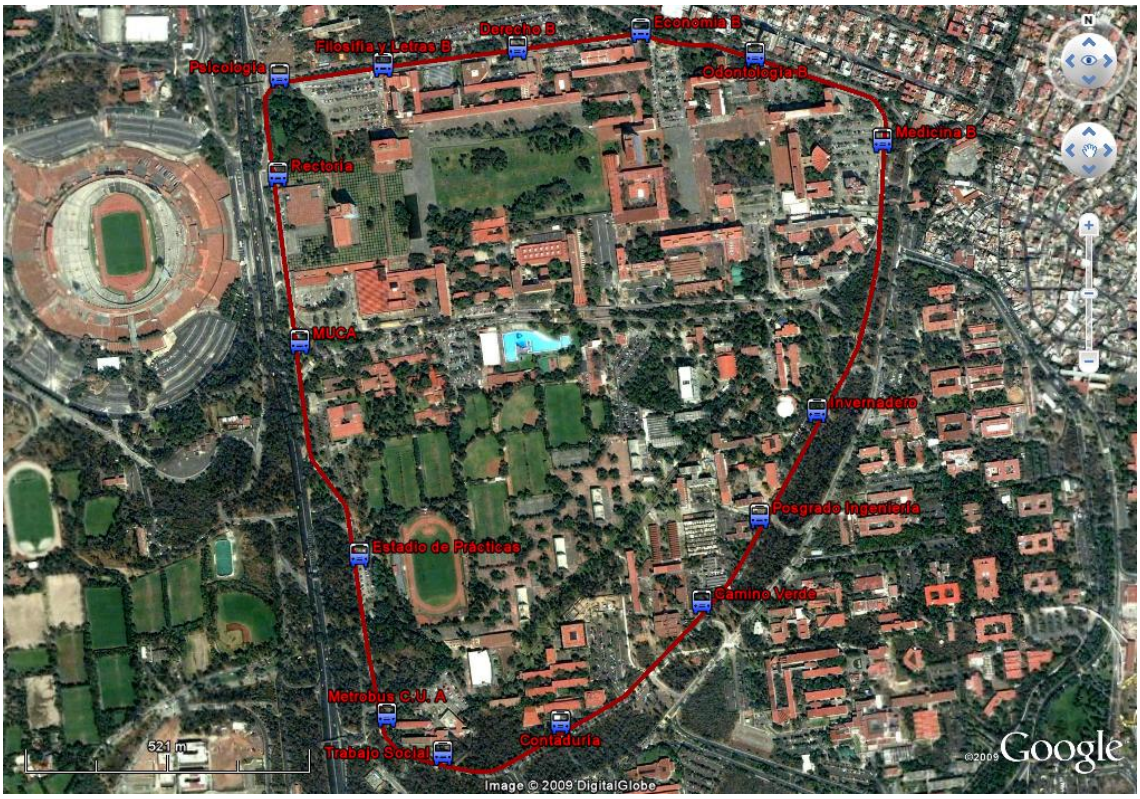


Figura 3.11. Ruta 9: Facultades

Ruta 9 (30 de octubre 2009)					
9:12 a 10:12		14:22 a 15:22		18:39 a 19:39	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:18	9:20	14:28	14:30	18:39	18:41
9:21	9:24	14:41	14:43	18:47	18:48
9:26	9:29	14:42	14:45	18:48	18:50
9:40	9:41	14:53	14:55	19:04	19:05
9:41	9:44	15:04	15:05	19:10	19:12
9:46	9:48	15:05	15:06	19:12	19:13
9:54	9:55	15:17	15:19	19:26	19:28
9:58	9:59			19:34	19:35
10:06	10:07				
10:09	10:17				

Tabla 3.17. Frecuencias de la ruta 9 capturadas en el metrobús CU

Ruta 9 (6 de noviembre 2009)		
9:20 a 10:20	14:06 a 15:06	18:40 a 19:40
9:21	14:10	18:42
9:27	14:19	18:43
9:39	14:20	19:03
9:47	14:24	19:08
9:58	14:32	19:26
10:06	14:40	19:28
10:10	14:52	
10:17		

Tabla 3.18. Frecuencias de la ruta 9 capturadas en la Facultad de Odontologia

Ruta 11 (30 de octubre 2009)					
9:12 a 10:12		14:22 a 15:22		18:39 a 19:39	
Entrada	Salida	Entrada	Salida	Entrada	Salida
9:12	9:14	14:29	14:33	18:56	18:59
9:16	9:19	14:49	14:52		
9:33	9:34	15:09	14:11		
9:35	9:38				
9:42	9:44				
9:54	9:57				

Tabla 3.19. Frecuencias de la ruta 11 capturadas en el metrobús CU

3.1.10. Ruta 11: Metrobús C.U. - Campos Deportivos

La Ruta 11, con terminal en metrobús CU, recorre aproximadamente 5.8 kilómetros, sus horarios de atención son de lunes a viernes de 6:00 a 22:00 hrs. Las 14 paradas que incluye la ruta, en orden de dirección, son: Metrobús C.U. A, Estadio de Prácticas, MUCA, E-8 B, E-7, Relaciones Laborales, Obras y Conservación, AAPAUNAM, UDUAL, Pista de Calentamiento B, Pumitas, Jardín Botánico, Campos de Fútbol II y Metrobús C.U. B. La Figura 3.12 muestra la ruta y sus paradas.

Las Tablas 3.19 y 3.20 muestra la frecuencia de los autobuses de la ruta 11, capturadas en el metrobús CU y en el estadio de Olímpico Universitario, respectivamente. El tiempo de abordaje en terminal es de 2.2 minutos y las frecuencias de autobuses son:

- De 9 a 10 de la mañana hay 5.5 autobuses cada hora, esto es, un autobús cada 11 minutos.
- De 2 a 3 de la tarde hay 3 autobuses cada hora, esto es, un autobús cada 20 minutos.
- De 7 a 8 de la noche hay un autobús cada hora.

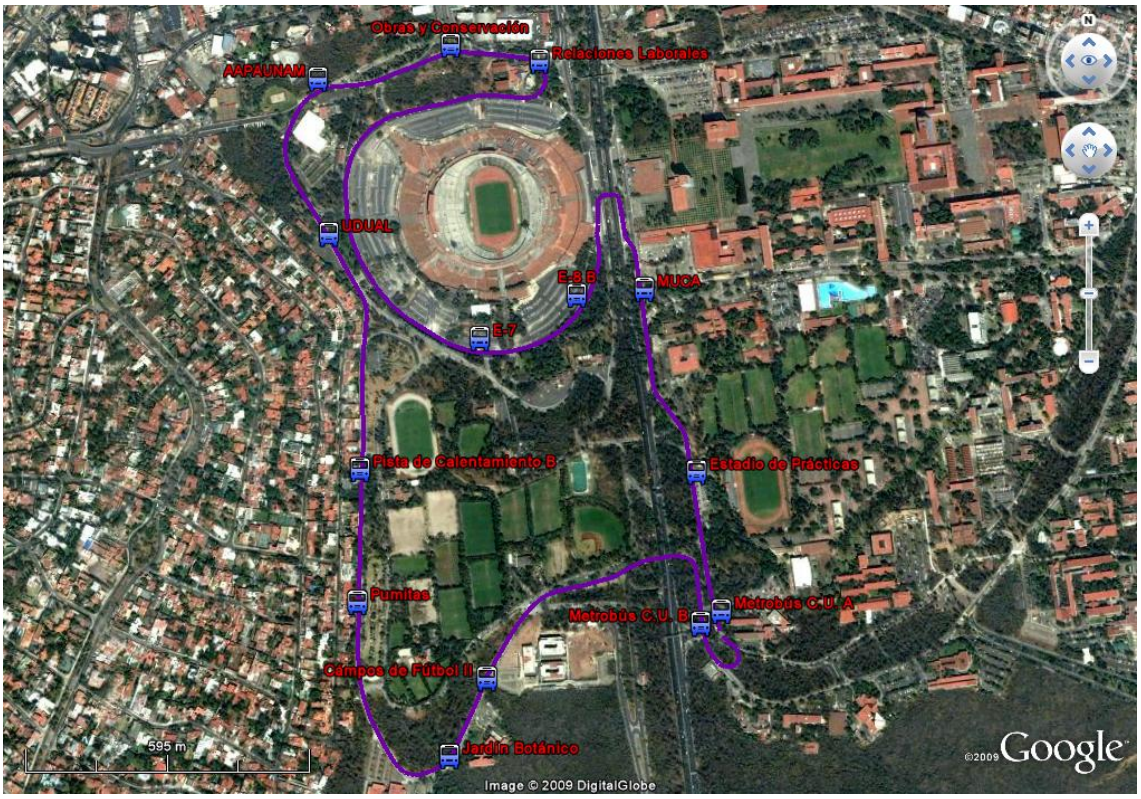


Figura 3.12. Ruta 11: Metrobús C.U. - Campos Deportivos

Ruta 11 (23 de octubre 2009)		
9:20 a 10:20	14:03 a 15:03	18:36 a 19:36
9:25	14:03	19:08
9:47	14:20	
9:55	14:47	
10:07		
10:19		

Tabla 3.20. Frecuencias de la ruta 11 capturadas en el Estadio Olímpico Universitario

En las tablas de esta sección se observa que el paso de los autobuses en las diferentes paradas no es homogéneo, es decir, es posible que en un intervalo de dos minutos pasen dos autobuses de la misma ruta por alguna parada y que diez minutos después no transite ninguno por esa parada, lo que provoca que el segundo autobús que pasa en el intervalo de dos minutos vaya vacío. Una posible razón de este hecho se observó durante el trabajo de campo en las terminales de metro CU y Estadio Olímpico Universitario. En estos puntos existe una persona o un grupo de personas que observa si la demanda crece o disminuye en dichas estaciones y con base en esas observaciones decide si manda más o menos autobuses a recorrer dicha ruta. Si no hay autobuses en la terminal de alguna ruta con alta demanda, se mandan autobuses de otra ruta para satisfacer la demanda, esto crea confusiones en el usuario que aún no está familiarizado con el sistema de transporte público universitario.

También existe otro problema en la forma de controlar el flujo de autobuses ya que la persona encargada de controlarlo no conoce las condiciones del resto del sistema, es decir, aunque la demanda sea poca en las terminales esto no quiere decir que los sea en el resto de las paradas. Entonces si hay pocas corridas en la terminal con baja demanda ocasiona que se formen filas de usuarios en otros puntos de alta demanda. Es posible evitar estos comportamientos si se elabora un estudio de demanda, pues se pueden planear las corridas con base en los estudios y no con base en la observación parcial. Un sistema de transporte organizado, le permite al usuario tener más información y por lo tanto hacer un uso más eficiente de éste.

3.2. *Bicipuma*

La red del *Bicipuma* cuenta con 5.74 kilómetros de ciclopistas y 12 módulos para préstamo de bicicletas. Cualquier persona con credencial de la UNAM puede tener acceso a las bicicletas y éstas se prestan por un periodo no mayor a veinte minutos. Cabe mencionar que también es posible el préstamo de bicicletas para personas que realicen estadías dentro de la UNAM. Los horarios de atención son de lunes a jueves de 6:50 a 11:45 hrs. y de 14:30 a 16:15, y viernes de 6:50 a 11:45 hrs. y de 14:30 a 15:15 hrs.

Los módulos del *Bicipuma* se encuentran en el Estadio Olímpico Universitario y en las Facultades de: Arquitectura, Filosofía y Letras, Derecho, Medicina, Ingeniería, Química, Ciencias, en el Anexo de Ingeniería, Ciencias Políticas y Sociales, así como, en el estadio Tapatío Méndez y en el metro Universidad. La Figura 3.13 muestra la red de ciclopistas y la localización de los módulos para préstamo de bicicletas.

Se observa en la Figura 3.13, que la ciclopista casi no cruza la red vial, por lo que es seguro transitar en ella, sólo existen dos puntos donde lo anterior no se cumple. Uno de los cruces de la red vial con la ciclopista está junto al módulo de la Facultad de Ciencias Políticas, y el otro se da al momento de atravesar el circuito exterior (al este del módulo de Ciencias), sin embargo, ambos puntos son seguros pues en el primero hay un tope, mientras que el segundo es un cruce semaforizado.

La velocidad promedio de un ciclista en instalaciones donde se cuente con carriles exclusivos es

Figura 3.13. Ciclopistas C.U.

aproximadamente $25km/hr$ (Wachs, Samuels & Skinner (2000)). Fue necesario conocer el tiempo aproximado de entrega y recolección de bicicletas en cada módulo, ya que ambas acciones afectan el tiempo de recorrido. Para la colección de datos se tomaron dos muestras. La primera se realizó en el módulo del Anexo de Ingeniería y la segunda en el módulo de Derecho. Ambas muestras se obtuvieron aproximadamente de 9 a 10 de la mañana y de 2 a 3 de la tarde, el horario nocturno se omitió pues no se prestan bicicletas a esas horas. A continuación se presentan las tablas con los resultados.

La Tabla 3.22 y 3.21, muestra los tiempos de entrega y recolección en el módulo del Anexo de Ingeniería. Y la Tabla 3.23 y 3.24, muestra los tiempos de entrega y recolección en el módulo del la Facultad de Derecho.

De las tablas mencionadas se calcula un tiempo promedio de entrega de 34, con desviación estándar de 21 segundos y un tiempo promedio de recolección de 28 segundos, con desviación estándar de 21 segundos. La desviación estándar es considerablemente grande gracias al proceso que sigue el usuario para recolectar o entregar una bicicleta, a continuación se describe este proceso.

La dispersión en los tiempos de recolección y de entrega, no es porque el usuario espere a ser atendido o por que no existan bicicletas disponibles, más aún, se observó en trabajo de campo que las colas de usuarios son raras y siempre hay bicicletas de sobra en todos los módulos. La razón de la diferencia en los tiempos se debe a la forma en que cada usuario realiza la entrega o recolección de bicicletas. En general, si se desea recolectar una bicicleta lo primero que se debe de hacer es tomarla y llevarla con el administrador del módulo, esta persona registra el código de barras de la credencial del usuario y el código de barras de la bicicleta, una vez terminado este procedimiento el usuario es libre de usar la bicicleta por un lapso no mayor a 20 minutos y dentro de las instalaciones de CU. El procedimiento de entrega es a la inversa, primero se registra la bicicleta con el administrador y después se coloca en el módulo. La disparidad entre los tiempos de recolección o los de entrega se da cuando el usuario busca su credencial, algunos arriban al módulo con la credencial preparada lo que hace más eficiente el préstamo o la entrega, mientras que otros, buscan su credencial al momento de registrarla con el administrador. Entonces, los usuarios que tardan en encontrar su credencial al momento de registrar la bicicleta demoran el proceso de entrega o recolección. Por lo que si se desea pedir prestada una bicicleta es recomendable llegar al módulo con la credencial preparada para no ocasionar demoras en el sistema.

Actualmente, en varias ciudades del mundo existe a disposición de la población renta de bicicletas, en donde los módulos de renta están totalmente automatizados. Si el usuario desea rentar un bicicleta; primero debe registrar su credencial de acceso en el módulo, donde se le indicará qué bicicleta puede tomar. El tiempo de préstamo y la zona de circulación varía en cada ciudad. Al momento de entregar la bicicleta, simplemente la estaciona en cualquier módulo y el sistema automáticamente libera el préstamo. También es posible reportar bicicletas defectuosas o la carencia de bicicletas, en el módulo de préstamo. Un sistema similar puede ser adoptado en CU, lo que permitiría que el préstamo de bicicletas estuviera abierto por lo menos durante toda la mañana y la tarde, para impulsar cada vez más el uso de la bicicleta como medio de transporte dentro de las instalaciones de CU.

Entrega en Anexo de Ingeniería (15 de octubre 2009)					
9:04 a 10:04			14:30 a 15:30		
Entrada	Salida	Total Segundos	Entrada	Salida	Total Segundos
09:10:19	09:11:47	31	14:30:13	14:30:40	27
09:11:21	09:11:50	26	14:31:28	14:31:53	25
09:12:25	09:12:52	49	14:31:28	14:32:00	32
09:13:15	09:13:46	61	14:32:52	14:33:31	39
09:17:21	09:18:54	93	14:42:24	14:42:44	20
09:18:25	09:19:09	82	14:45:09	14:46:31	82
09:21:06	09:21:23	97	14:46:44	14:47:06	22
09:21:10	09:21:32	36	14:46:54	14:47:20	26
09:21:19	09:21:53	74	14:46:57	14:47:57	60
09:22:32	09:22:58	25	14:47:57	14:48:18	21
09:22:09	09:24:10	17	14:49:04	14:49:24	20
09:24:21	09:25:00	17	14:50:27	14:51:02	35
09:26:30	09:27:05	19	14:50:58	14:51:17	19
09:28:19	09:28:46	20	14:52:54	14:53:15	21
09:30:22	09:30:46	17	14:54:47	14:55:03	16
09:35:49	09:36:24	95	14:55:33	14:55:48	15
09:36:48	09:37:14	17	14:57:36	14:57:50	14
09:37:04	09:37:28	79	14:58:32	14:58:52	20
09:37:35	09:37:48	20	15:00:38	15:01:02	24
09:38:24	09:38:36	23	15:02:15	15:02:43	28
09:41:40	09:42:10	26	15:07:18	15:08:08	50
09:47:56	09:48:05	50	15:09:27	15:09:46	19
09:49:25	09:49:35	15	15:09:55	15:10:23	28
09:51:07	09:51:53	13	15:10:53	15:11:20	27
09:51:20	09:52:12	105	15:13:00	15:13:15	15
			15:14:18	15:14:30	12
			15:15:25	15:15:45	20
			15:16:06	15:16:30	24
			15:16:47	15:16:55	8
			15:19:04	15:19:11	7
			15:19:16	15:19:30	14
			15:20:29	15:20:47	18

Tabla 3.21. Tiempos de entrega en el módulo del Anexo de Ingeniería

Recolección en Anexo de Ingeniería (15 de octubre 2009)					
9:04 a 10:04			14:30 a 15:30		
Entrada	Salida	Total Segundos	Entrada	Salida	Total Segundos
09:04:15	09:04:46	88	14:30:32	14:30:52	20
09:04:24	09:04:50	29	14:31:28	14:31:37	9
09:04:55	09:05:44	27	14:31:32	14:31:45	13
09:07:03	09:08:04	31	14:31:52	14:32:13	21
09:07:12	09:08:45	93	14:32:21	14:32:54	33
09:13:15	09:14:37	44	14:33:41	14:33:59	18
09:13:28	09:15:05	17	14:33:55	14:34:04	9
09:15:19	09:15:55	22	14:38:02	14:38:24	22
09:15:40	09:16:54	34	14:39:16	14:39:30	14
09:17:40	09:18:05	26	14:39:16	14:39:34	18
09:35:58	09:36:15	121	14:39:49	14:40:03	14
09:37:10	09:37:27	39	14:39:53	14:40:11	18
09:41:05	09:41:24	35	14:42:30	14:42:48	18
09:41:09	09:41:29	27	14:42:38	14:42:57	19
09:41:20	09:41:37	24	14:42:47	14:43:07	20
09:41:42	09:43:17	35	14:43:18	14:43:26	8
09:42:57	09:43:14	26	14:43:44	14:45:17	93
09:43:31	09:44:50	24	14:47:36	14:47:43	7
09:45:30	09:45:50	13	14:49:01	14:49:11	10
09:50:09	09:50:32	12	14:49:11	14:49:26	15
09:50:10	09:50:36	30	14:49:26	14:49:33	7
09:54:50	09:55:40	9	14:54:32	14:54:57	25
09:56:10	09:56:25	10	15:07:11	15:07:28	17
09:56:40	09:56:53	46	15:07:16	15:07:59	43
09:56:05	09:57:50	52	15:08:39	15:08:52	13
			15:10:01	15:10:27	26
			15:10:01	15:10:27	26
			15:12:14	15:12:34	20
			15:13:00	15:13:27	27
			15:13:07	15:13:40	33
			15:14:38	15:15:49	71
			15:21:25	15:21:55	30
			15:27:53	15:28:02	9
			15:27:54	15:28:28	34

Tabla 3.22. Tiempos de recolección en el módulo del Anexo de Ingeniería

Entrega en Derecho (8 de octubre 2009)					
9:10 a 10:10			14:30 a 15:30		
Entrada	Salida	Total Segundos	Entrada	Salida	Total Segundos
09:11:30	09:11:50	20	14:30:50	14:31:08	18
09:17:00	09:17:40	40	14:30:52	14:31:25	33
09:23:20	09:23:50	30	14:34:43	14:35:13	30
09:24:03	09:24:29	26	14:45:59	14:46:42	43
09:27:08	09:27:48	40	14:46:34	14:47:05	31
09:27:19	09:27:56	37	14:47:25	14:48:16	51
09:38:20	09:38:46	26	14:47:38	14:48:24	46
09:38:38	09:39:05	27	14:57:29	14:58:01	32
09:41:33	09:42:03	30	14:57:54	14:58:25	31
09:44:17	09:44:44	27	15:02:28	15:03:37	69
09:44:35	09:44:55	20	15:10:03	15:10:37	34
09:50:16	09:50:36	20	15:10:31	15:11:19	48
09:52:25	09:52:36	11	15:12:39	15:13:19	40
09:52:50	09:53:40	50	15:15:31	15:16:12	41
09:59:58	10:00:26	28	15:19:54	15:20:20	26
			15:20:43	15:20:58	15
			15:21:26	15:21:56	30
			15:22:11	15:22:49	38
			15:28:38	15:29:37	59

Tabla 3.23. Tiempos de entrega en el módulo de Derecho

Recolección en Derecho (8 de octubre 2009)					
9:10 a 10:10			14:30 a 15:30		
Entrada	Salida	Total Segundos	Entrada	Salida	Total Segundos
09:13:02	09:14:03	61	14:31:54	14:32:09	15
09:15:00	09:15:18	18	14:35:50	14:36:15	25
09:20:09	09:20:21	12	14:51:14	14:51:28	14
09:23:02	09:23:16	14	14:58:03	14:58:43	40
09:23:05	09:24:24	79	15:00:00	15:00:32	32
09:45:00	09:45:29	29	15:01:50	15:03:50	120
09:46:50	09:47:06	16	15:02:52	15:03:17	25
09:52:02	09:52:18	16	15:04:12	15:04:36	24
10:00:22	10:00:39	17	15:13:51	15:14:40	49
10:01:02	10:01:19	17	15:15:12	15:15:40	28
			15:17:08	15:17:39	31
			15:17:45	15:18:13	28
			15:20:36	15:21:14	38
			15:29:37	15:29:48	11

Tabla 3.24. Tiempos de recolección en el módulo de Derecho

3.3. Red peatonal

Se construyó una red peatonal que abarcara la mayor cantidad de territorio y al mismo tiempo que fuera lo más pequeña posible. Es por esto que sólo se consideraron aquellos caminos que comunicaran las instalaciones de la universidad, sin tomar en cuenta atajos, pasillos dentro de los edificios, ni aquellos espacios en donde el paso peatonal esté restringido por rejas, bardas o por falta de un camino pavimentado. Además, para las rutas peatonales que se encuentran junto a una vialidad con camellón, se consideró que la red es recorrida sobre éste, es decir, para evitar que el número de arcos y nodos creciera, los arcos que recorren ambos lados del camellón se fusionaron en uno sólo. Aunque no es posible caminar sobre la mayoría de los camellones, la red se trazó de esta forma, pues es necesario crear una red con la menor cantidad de nodos y arcos posibles, para que la cantidad de información que procese el algoritmo se minimice. La Figura 3.14 muestra la red peatonal, las entradas y las salidas de CU.

Para determinar la velocidad peatonal se procedió de la siguiente manera. Se realizaron diversos recorridos a pie sobre las rutas que se muestran en la Figura 3.15. Se consideraron estas rutas ya que, tienen un número considerable de pendientes que afectan la velocidad. El color verde es un camino sin pendientes y alcanzó una velocidad promedio de $5.50 km/h.$, el color rojo tiene una pendiente hacia abajo y se alcanzó una velocidad promedio de $6.06 km/h$, y por último, el color morado tiene una pendiente hacia arriba, donde se alcanzó una velocidad promedio de $5.07 km/h$.

Según el *Higway Capacity Manual* (HCM) del año 2000, la velocidad peatonal, donde a lo más

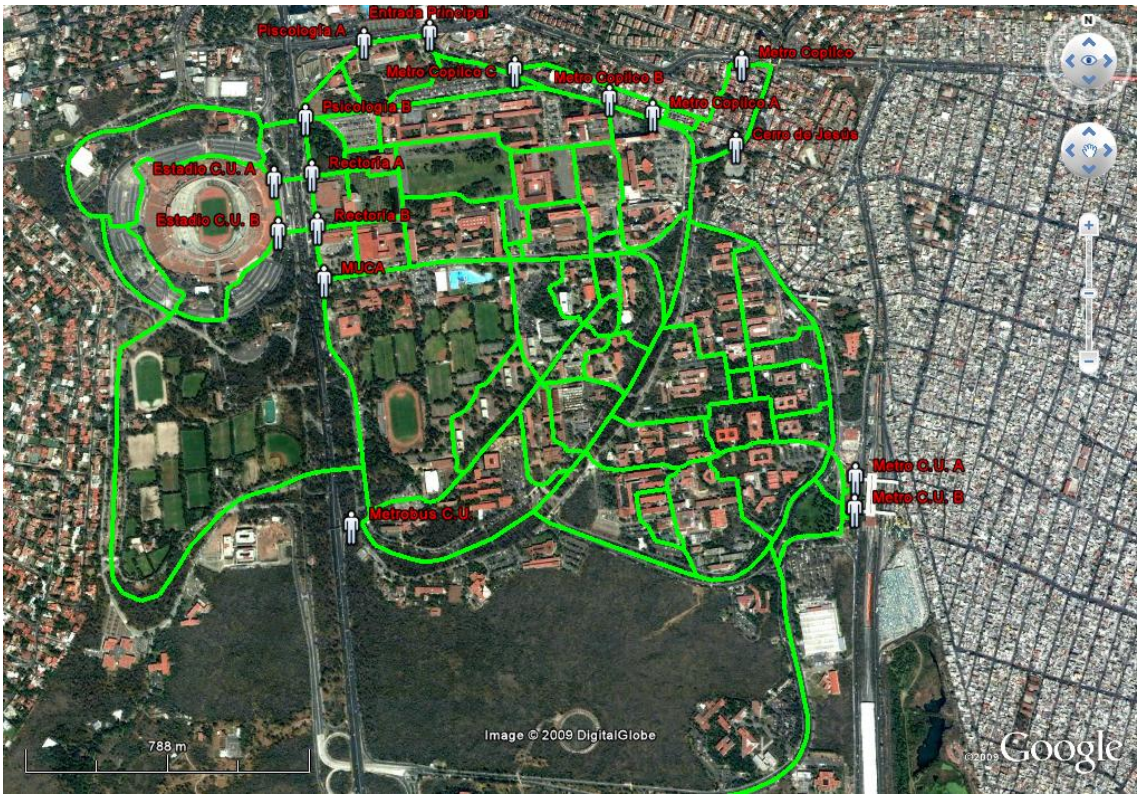


Figura 3.14. Red Peatonal, entradas y salidas de C.U.

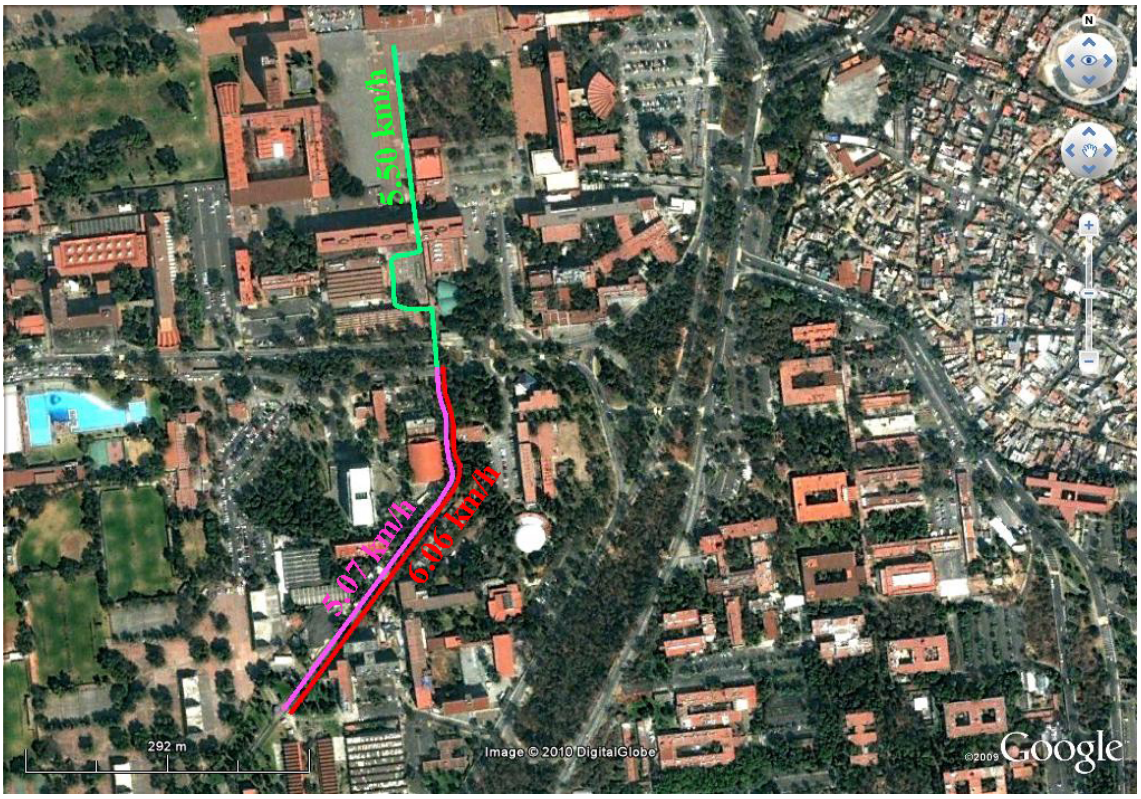


Figura 3.15. Diferentes recorridos a pie, para calcular la velocidad

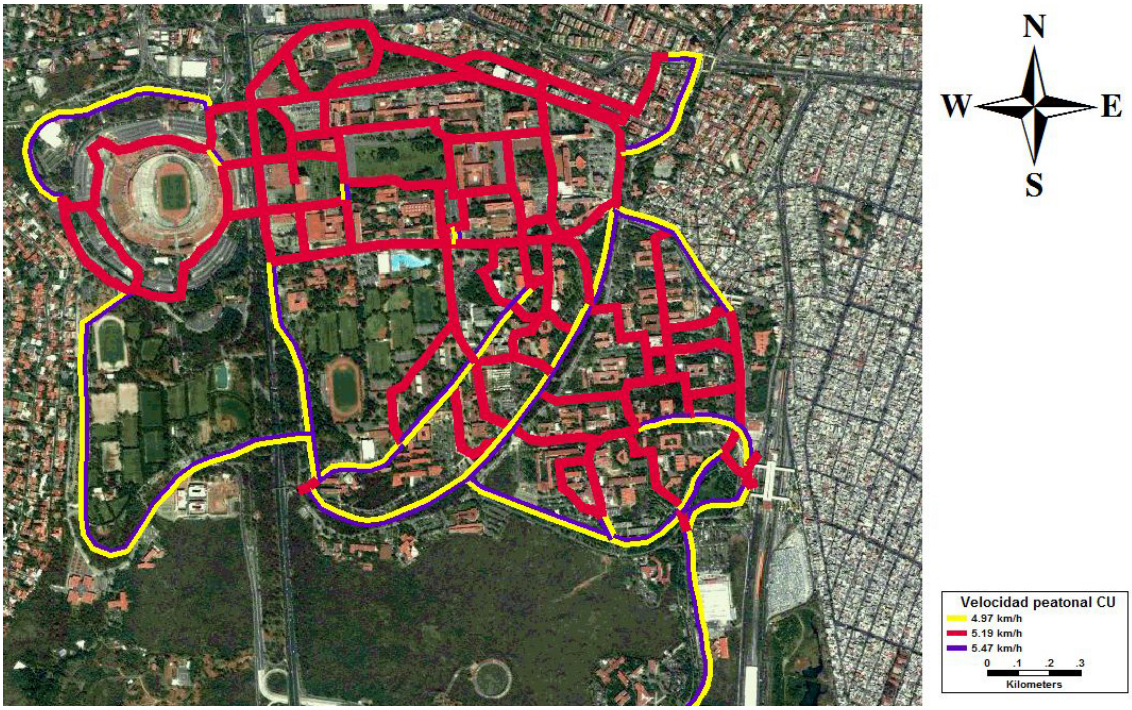


Figura 3.16. Velocidades de la red

20 % de la población es mayor a 65 años es de $4.32km/h$. Se observa que esta velocidad es inferior a la obtenida en trabajo de campo. Es valido considerar la velocidad del HCM, ya que en CU menos del 20 % de la población tiene más de 65 años (UNAM 2009*a*). Para intentar abarcar una velocidad más general se toma en cuenta que la velocidad peatonal dentro de CU es igual al promedio de las velocidades obtenidas en trabajo de campo y la del HCM, es decir, $5.19km/h$ para caminos sin pendiente, $5.47km/h$ para caminos con pendiente hacia arriba y $4.97km/h$ para caminos con pendiente hacia abajo. La Figura 3.16 muestra las diferentes velocidades para la red de peatones.

CAPÍTULO 4

Creación de la híper red

Una vez que se finalizó el trabajo de campo, fue necesario crear la híper red que modela el sistema de transporte público universitario. La construcción de la híper red consiste de tres pasos. Primero se digitaliza cada una de las redes mediante *Google Earth*, después se crean los archivos nativos de *TransCAD* y se da formato a la base de datos. Por último, se unen las diferentes redes para formar una híper red. A continuación se describen a detalle cada uno de los pasos.

4.1. Digitalización en *Google Earth*

Se digitalizaron por separado cada una de las rutas de *Pumabús*, así como la red de *Bicipuma* y la peatonal.

Con la herramienta *Añadir ruta* de *Google Earth* se trazaron las diferentes rutas del *Pumabús*. El icono de *Añadir ruta* se encuentra en la barra de herramientas que se localiza en la parte superior del programa o en el menú,

$$\text{Añadir} \rightarrow \text{ruta}.$$

También es posible acceder a esta herramienta utilizando el código $Crtl + T$. Es importante trazar las rutas en el sentido en que éstas son recorridas ya que al momento de utilizar *TransCAD* es más fácil indicar el sentido de los arcos que representan cada ruta.

La localización de las paradas se hizo con la herramienta *Añadir marca de posición*, cuyo icono se encuentra en la barra de herramientas que se mencionó en el párrafo anterior. También se puede acceder a ésta en el menú,

$$\text{Añadir} \rightarrow \text{Marca de posición}$$

o con le siguiente código: $Ctrl + P$. Para cada ruta se digitalizan sus respectivas paradas aunque estas se repitan en diferentes corridas.

Para saber el recorrido y las paradas de cada ruta se puede consultar el mapa del *Pumabús* que se encuentra en http://www.pumabus.unam.mx/img/planonvo.pdf. Cuando se digitalizan varias rutas o lugares es recomendable que se organicen los diferentes archivos en carpetas. Para el caso que se describe se creó una carpeta con el nombre de *Pumabús* y sub-carpetas para cada una de las rutas de la zona de estudio.

Una vez trazadas todas la rutas se guardan los archivos con la extensión *.kml*, usando el menú:

Archivo → guardar → guardar lugar como.

Posteriormente fue necesario transformar el archivo nativo de *Google Earth* en formato *shape* (*.shp*), para poder utilizar algún SIG o SIG-T, como *TransCAD*. Existen varios programas que transforman archivos *.kml* a *.shp*, en particular se recomienda la extensión *KML to SHP* del programa *Arcmap* de *ArcGIS*. Esta extensión así como las instrucciones para instalarla se encuentran en http://arcscripts.esri.com/details.asp?dbid=14569. Aunque si no se cuenta con el programa también está disponible una aplicación para convertir los archivos en línea en la página http://www.zonums.com/kml2shp.html.

Es sumamente importante convertir los archivos *.kml* a *.shp* indicando la proyección y elipsoide de referencia. *Google Earth* utiliza la proyección UTM zona 14 Norte y el elipsoide de referencia es WGS 1984 para los mapas del Distrito Federal.

Para la captura de las rutas de bicicleta se usó de referencia el mapa que se encuentra en http://www.tucomunidad.unam.mx/Bicipuma/rutas.jpg, mientras que para la red peatonal se realizó una selección de las rutas que abarcaran la mayor cantidad de territorio y que a su vez fuesen las menos posibles.

4.2. Creación de redes en *TransCAD*

A continuación se describen los procesos para crear la red del *Pumabús* en *TransCAD*. Este procedimiento se realiza para cada una de las diferentes rutas, esto es, se hace una red diferente por ruta.

Para abrir los archivos *.shp* en *TransCAD* se sigue el siguiente menú,

File → open.

Antes de que el programa cargue en la vista el *shape* que se desea abrir, se despliega una ventana donde es posible indicar el sistema de coordenadas y elipsoide de referencia. Si aún no se ha realizado esta operación es necesario hacerlo en este instante utilizando la proyección y el elipsoide de referencia que se indicó en la Sección 4.1.

Como *TransCAD* no permite modificar archivos *.shp* es necesario exportarlos en un formato nativo del programa. Entonces, se accede al menú

Tools → export.

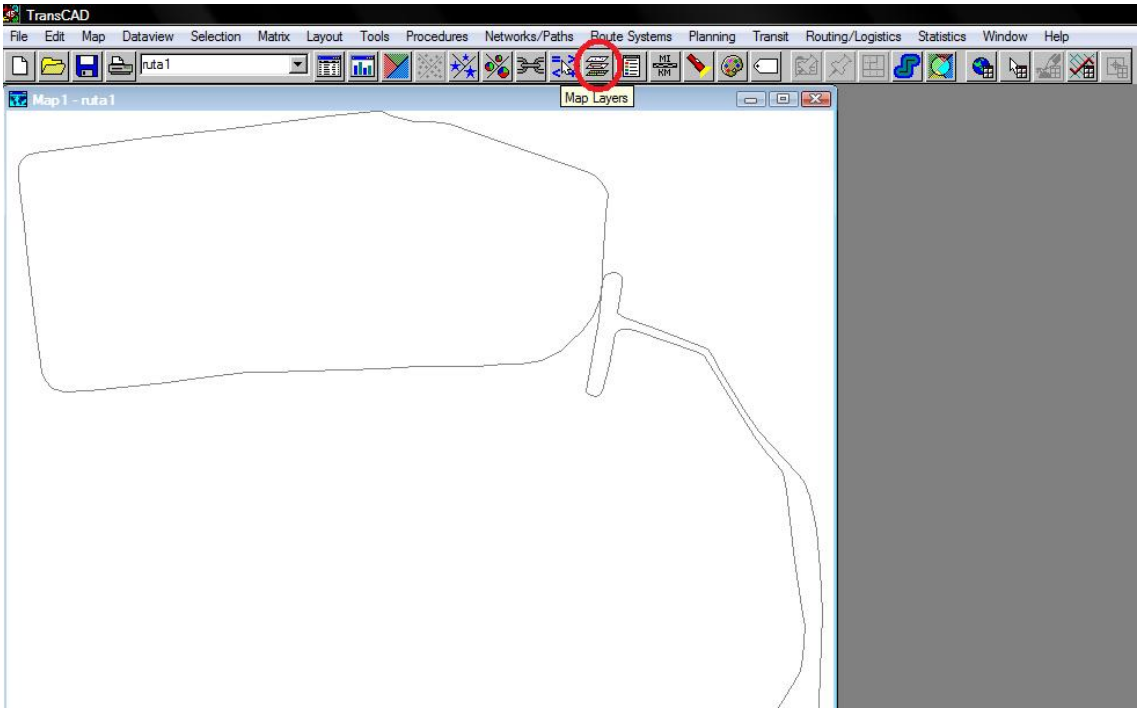


Figura 4.1. Hacer clic en el icono marcado

En la ventana de *Export* se selecciona *All Features*, en la opción *To* se selecciona *Estandard Geographic File*, en *Data Field* se selecciona *None* y se selecciona la casilla *Include Built-in Data* y por último se guarda el archivo. Este procedimiento convierte el archivo *.shp* a un *.dbd*. Ahora, se carga el archivo que se creó en una vista nueva.

A continuación, se añade el *shape* de estaciones de la ruta que se cargó previamente. El icono para añadir el *.shp* a la vista actual se localiza en la barra de herramientas de la parte superior del programa como se muestra el la Figura 4.1. Se desplegará una ventana en donde se elige la opción *Add Layer*.

Ya que se tiene en la vista el *.dbd* de la ruta y el *.shp* de estaciones, es necesario añadir nodos en la ruta para que éstos representen estaciones. Con la vista seleccionada de la ruta se accede a la herramienta *Map editing toolbox* que se encuentra en el menú

Tools → Map editing → Toolbox.

Y se agregan los nodos correspondientes a las estaciones en la ruta. La herramienta para agregar nodos se encuentra en el *Toolboox* que se abrió previamente. Antes de agregar los nodos se debe configurar la forma en cómo el programa va a modificar la red, para esto se oprime en el botón *Settings* (el botón muestra un desarmador y una llave de tuercas cruzados), después se oprime

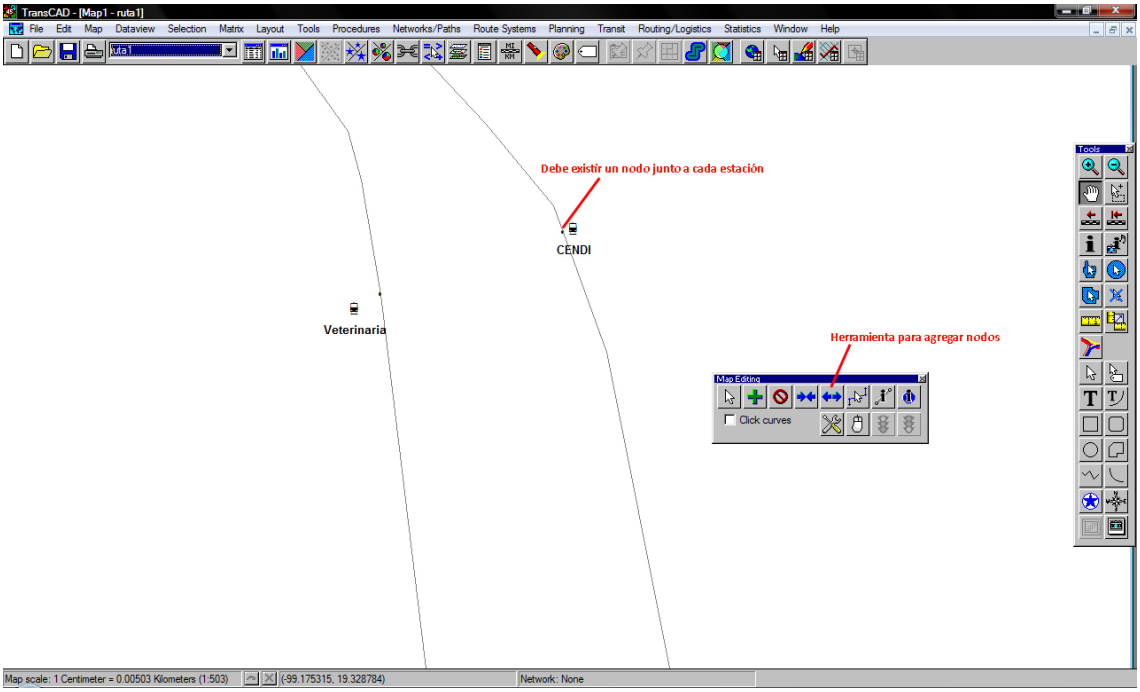


Figura 4.2. Existe un nodo por cada estación

Update, a continuación se elige el campo *Name* y se selecciona la opción *copy* que se encuentra en el recuadro de abajo. Esta configuración permite que al momento de dividir un arco por un nodo, los arcos resultantes contengan la información del arco original.

Una vez localizados todos los nodos en la red se guardan los cambios oprimiendo el botón del *Toolboox* que muestra un semáforo en verde. La Figura 4.2 muestra el botón que se utiliza para agregar nodos y un ejemplo de como se debe ver la red.

Para dar formato a la red de bicicletas se añade el *shape* que contiene los diferentes bicicentros y se crea el *.dbd* de la red de bicicletas. Esta red debe tener nodos en la intersección de dos o más arcos, entonces se selecciona la vista de la red de bicicletas y se abre la herramienta *Map editing toolbox*. Se oprime *Modify line*, se seleccionan los arcos y se arrastran el extremo de uno de ellos al lugar donde se quiera realizar la intersección, como se muestra en la Figura 4.3. Para guardar los cambios se oprime en el icono en el *Map editing toolbox* que muestra un semáforo en verde.

Los procedimientos para añadir nodos a la red que corresponden a los bicicientros son similares a los de añadir estaciones a las rutas del *Pumabús*. Y a su vez, los procedimientos para dar formato de la red peatonal son similares a los de la red de bicicletas.

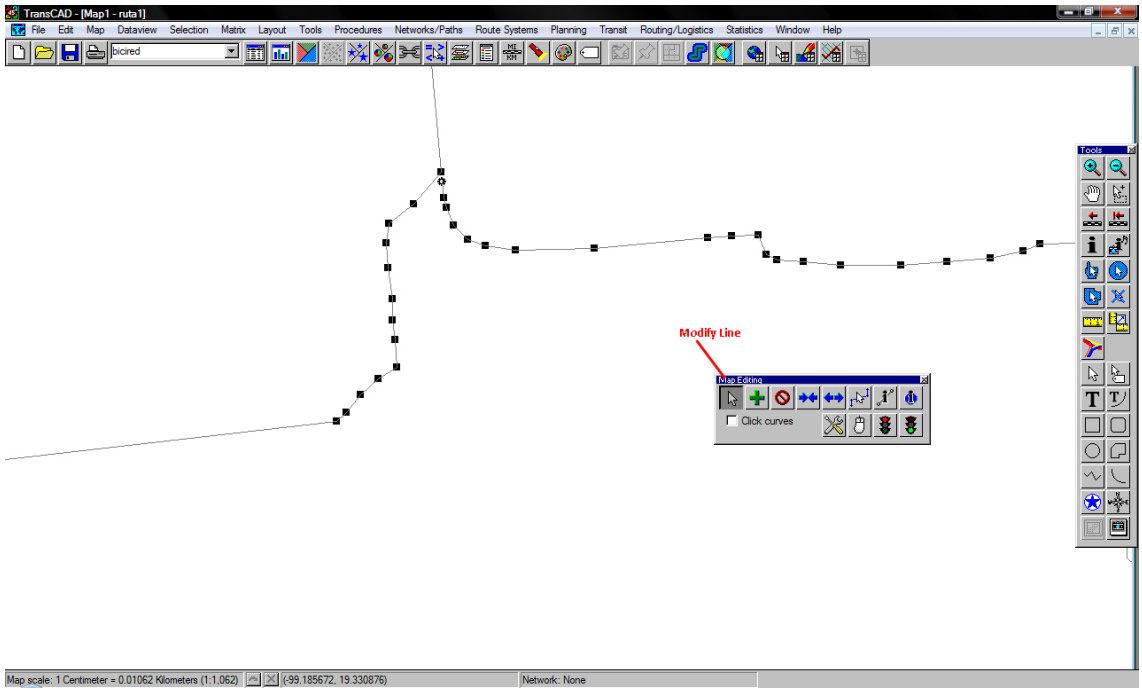


Figura 4.3. Selección de las líneas con *Modify line*

4.3. Creación de la híper red en *TransCAD*

La construcción de la híper red consta de dos pasos. Primero se cargan todas las redes de las diferentes líneas de *Pumabús* en un único archivo. Después, mediante arcos de transferencia modal se une la red peatonal y la de bicicletas con la híper red del *Pumabús*.

4.3.1. Creando la híper red de *Pumabús*

Por la construcción de la red de autobuses existe un archivo *.dbd* por cada línea de *Pumabús*. Ahora, es necesario cargar todas estas redes en un solo archivo. No es recomendable cargar las redes sin antes haberlas editado como se describe en la Sección 4.2 ya que es probable que se presenten problemas al momento del formato.

A menos que se disponga de una computadora muy potente, la forma mas sencilla de cargar redes es por pares. Para esto se abren dos redes en la misma vista y a continuación se accede al menú

$$\text{Tools} \rightarrow \text{Geographic Utilities} \rightarrow \text{Merge Geography},$$

se oprime el botón de aceptar y se guarda el nuevo archivo. Entonces se añade en una vista nueva la red que se creo junto con alguna otra. Y se repite el procedimiento de hasta cargar todas las redes en un solo archivo *.dbd*. La Figura 4.4 muestra ejemplo del procedimiento.

En seguida se modifica la tabla de atributos asociada a la red. Los atributos que por defecto vienen asociados a cualquier red son los siguientes,

- *ID*, es el identificador único de cada arco.
- *Lenght*, es la longitud del arco cuya unidad de medida son kilómetros.
- *Dir*, es la dirección de flujo del arco. Puede adquirir los valores de 0, 1 y -1.
- *Name*, es el nombre que se le asigna al arco.

Es importante que el campo *Dir* de la tabla esté capturado de forma correcta ya que éste nos dará el sentido que representa cada arco. Para conocer el sentido en que se trazaron las rutas en *Google Earth*, se accede a *Map Layers* y después se oprime el botón *Style*, que se encuentra en la parte inferior del recuadro. Entonces en la sección de *Arrowheads*, se selecciona la opción *Topology*. La opción *Direction Flow* muestra la dirección de flujo que tiene asignado cada arco en el campo *Dir*. Para capturar el campo *Dir* se consideran las siguientes reglas:

- *Dir*$=-1$, si la dirección topológica es contraria a la dirección de flujo que tiene la red.
- *Dir*$=0$, si la dirección de flujo es en ambos sentidos.
- *Dir*$=1$, si la dirección topológica es igual a la dirección de flujo que tiene la red.

Mas adelante se expondrá cómo se capturó el campo *Dir* para cada una de las redes.

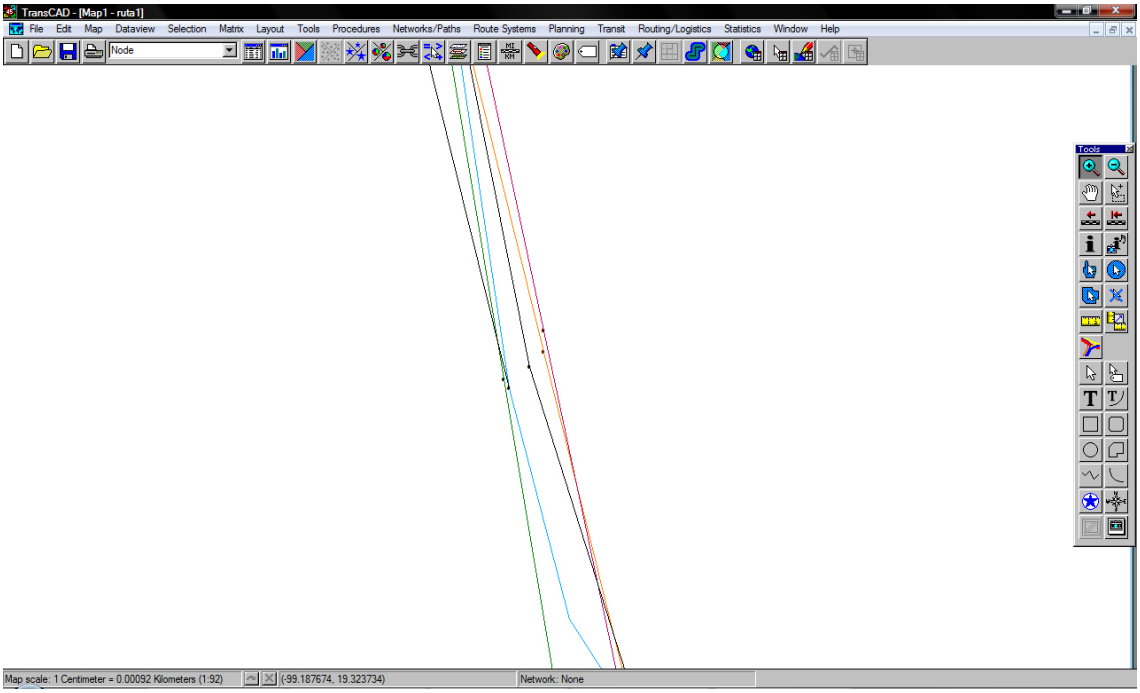


Figura 4.4. Vista de las diferentes líneas del *Pumabús*

Es necesario agregar atributos a los arcos de la red de manera que el algoritmo tenga información suficiente para calcular las híper rutas más cortas. Entonces, se modifica la tabla de atributos de la red de autobuses, seleccionando el menú,

Dataview → Modifiy table,

y continuación, se oprime el botón *Add Field*. Los nuevos campos de la tabla de atributos son:

- Arco_A *Integrer* (1 *byte*). Campo reservado para los arcos de la red de autobuses. Con valor 1 si el arco pertenece a los arcos de la red y 0 en cualquier otro caso.
- H_Arco *integrer* (1 *byte*). Campo reservado para los híper arcos de la red de autobuses. Con valor 1 si el arco pertenece a los híper arcos de la red y 0 en cualquier otro caso.
- Arco_P *integrer* (1 *byte*). Campo reservado para los arcos de la red peatonal. Con valor 1 si el arco pertenece a la red peatonal y 0 en cualquier otro caso.
- Arco_B *integrer* (1 *byte*). Campo reservado para los arcos de la red de bicicletas. Con valor 1 si el arco pertenece a la red de bicicletas y 0 en cualquier otro caso.
- Arco_Modal *integrer* (1 *byte*). Campo reservado para los arcos que representan un cambio de modo. Con valor 1 si el arco representa un cambio de modo y 0 en cualquier otro caso.
- Vel_AB *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota la velocidad en la dirección de flujo; su unidad de medida son kilómetros por hora. Este campo admite valores de todas los modos de transporte.
- Vel_BA *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota la velocidad en contra dirección de flujo; su unidad de medida son kilómetros por hora. Este campo admite valores de todas los modos de transporte.
- Tiem_reco_AB *real* (8 *bytes*, *width*=10, *decimals*=3). Atributo que denota el tiempo de recorrido en la dirección de flujo; su unidad de medida es en minutos. Este campo admite valores de todas los modos de transporte.
- Tiem_reco_BA *real* (8 *bytes*, *width*=10, *decimals*=3). Atributo que denota el tiempo de recorrido en contra dirección de flujo; su unidad de medida es en minutos. Este campo admite valores de todas los modos de transporte.
- Tiem_abord *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota el tiempo de abordaje en cada estación; su unidad de medida es en minutos. Este campo solo admite valores de los arcos de la red de autobuses.
- Frec_matu *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota la frecuencia del *Pumabús* en cada estación en el turno matutino. Su unidad de medida es autobuses/hora. Este campo solo admite valores de los híper arcos de la red de autobuses.
- Frec_vesp *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota la frecuencia del *Pumabús* en cada estación en el turno vespertino. Su unidad de medida es autobuses/hora. Este campo solo admite valores de los híper arcos de la red de autobuses.

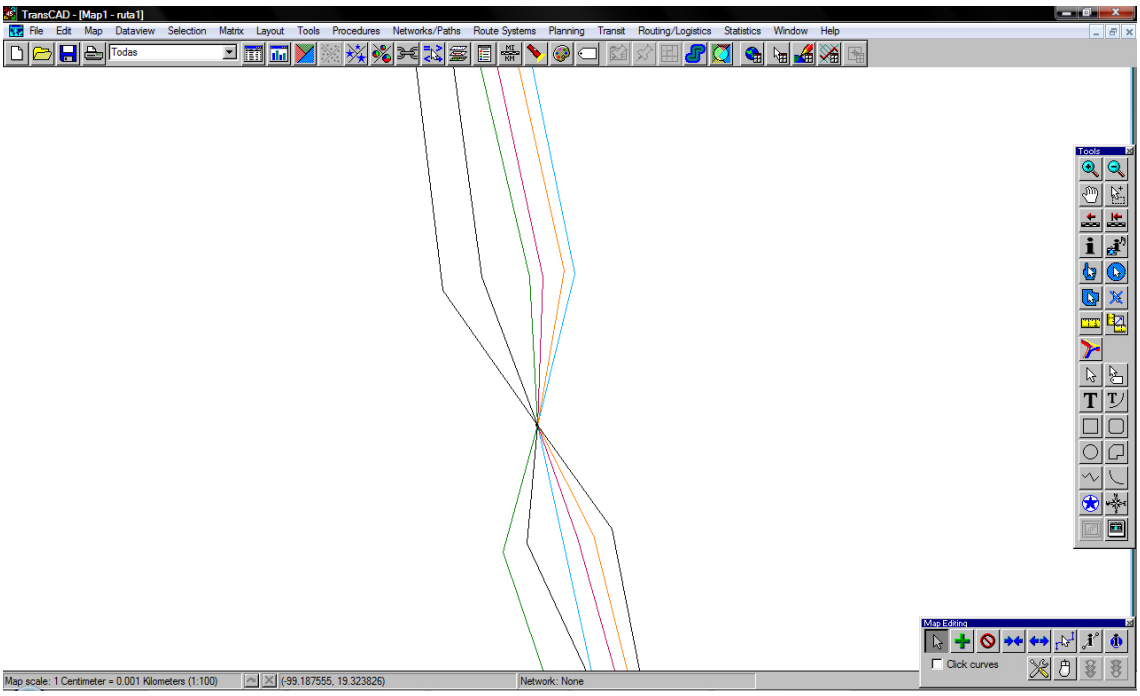


Figura 4.5. De un solo nodo salen los diferentes arcos

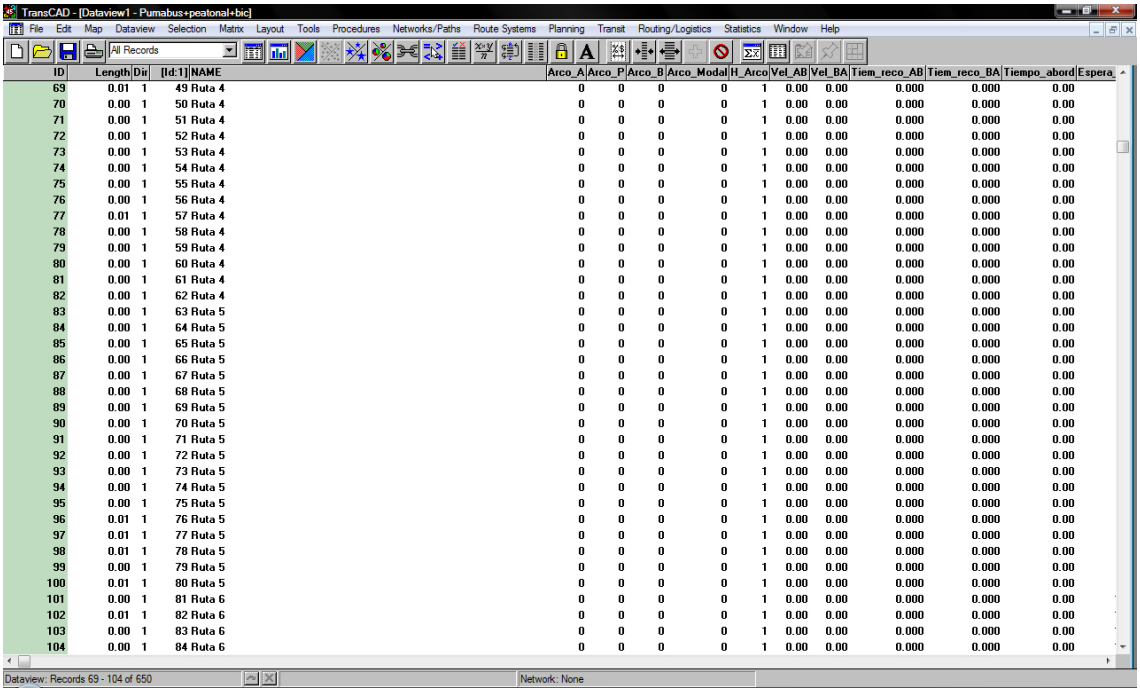
- Frec_noc *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota la frecuencia del *Pumabús* en cada estación en el turno nocturno. Su unidad de medida es autobuses/hora. Este campo solo admite valores de los híper arcos de la red de autobuses.

- Tiempo_recoleccion *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota el tiempo promedio de recolección de bicicletas en los módulos de préstamo; su unidad de medida es en minutos. Este campo solo admite valores de los arcos salientes de los nodos que representan módulos de prestamos de bicicletas.

- Tiempo_entrega *real* (8 *bytes*, *width*=10, *decimals*=2). Atributo que denota el tiempo promedio de entrega de bicicletas en los módulos de préstamo; su unidad de medida es en minutos. Este campo solo admite valores de los arcos salientes de los nodos que representan módulos de prestamos de bicicletas.

Ya que en la red de autobuses existe más de un nodo que representa una única estación, es necesario eliminar las repeticiones. Para esto, se carga la herramienta *Modify Line* del *Map editing toolbox* y se selecciona algún nodo para arrastrarlo hacia los demás. El procedimiento se repite hasta unificar todos los nodos en un único nodo. La Figura 4.5 muestra un ejemplo de como se debe ver el resultado de la operación descrita.

Una vez que se realizó el procedimiento para cada nodo de la red, se asigna el valor H_Arco= 1



ID	Length	Dir	[Id:1]	NAME	Arco_A	Arco_P	Arco_B	Arco_Modal	H_Arco	Vel_AB	Vel_BA	Tiem_reco_AB	Tiem_reco_BA	Tiempo_abord	Espera_
69	0.01	1	49	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
70	0.00	1	50	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
71	0.00	1	51	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
72	0.00	1	52	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
73	0.00	1	53	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
74	0.00	1	54	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
75	0.00	1	55	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
76	0.00	1	56	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
77	0.01	1	57	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
78	0.00	1	58	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
79	0.00	1	59	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
80	0.00	1	60	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
81	0.00	1	61	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
82	0.00	1	62	Ruta 4	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
83	0.00	1	63	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
84	0.00	1	64	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
85	0.00	1	65	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
86	0.00	1	66	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
87	0.00	1	67	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
88	0.00	1	68	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
89	0.00	1	69	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
90	0.00	1	70	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
91	0.00	1	71	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
92	0.00	1	72	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
93	0.00	1	73	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
94	0.00	1	74	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
95	0.00	1	75	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
96	0.01	1	76	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
97	0.01	1	77	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
98	0.01	1	78	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
99	0.00	1	79	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
100	0.01	1	80	Ruta 5	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
101	0.00	1	81	Ruta 6	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
102	0.01	1	82	Ruta 6	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
103	0.00	1	83	Ruta 6	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	
104	0.00	1	84	Ruta 6	0	0	0	0	1	0.00	0.00	0.000	0.000	0.00	

Figura 4.6. Tabla de atributos de la red

a cada uno de los arcos con el botón *New Dataview* del panel de iconos que se encuentra en la parte superior del programa. El botón despliega la tabla de atributos de la red, como se muestra en la Figura 4.6. A continuación, se ubica el cursor sobre el campo H_Arco, se oprime el botón derecho del ratón y se selecciona la opción *Fill*. En la ventana desplegada se selecciona la opción *Single Value* igual a 1. Se repite el procedimiento para llenara los campos Arco_A, Arco_P, Arco_B, Arco_Modal, Vel_AB, Vel_BA, Tiempo_abord, Tiempo_recoleccion y Tiempo_entrega con valor único de 0. Además, se captura en las columnas Frec_matu, Frec_vesp y Frec_noc la frecuencia de los autobuses matutina, vespertina y nocturna, respectivamente que se obtuvo en trabajo de campo.

Para crear los híper arcos de la red se procede de la siguiente forma. En la vista de la red de autobuses se abre el *Map editing toolbox* y se elige la herramienta *Split Line* (el icono con dos flechas apuntando hacia afuera) y se oprime con el ratón cada arco en algún punto cercano al nodo en donde convergen. De acuerdo a la dirección de flujo, se debe tener cuidado de crear los nuevos nodos después de donde convergen todas las rutas. La Figura 4.7 muestra la dirección de flujo y los nodos que se crearon.

Se hace una selección por atributos para completar los datos de los campos H_Arco y Arco_A. Para hacer una selección por atributos se selecciona la ruta

Selecction → Select by condition.

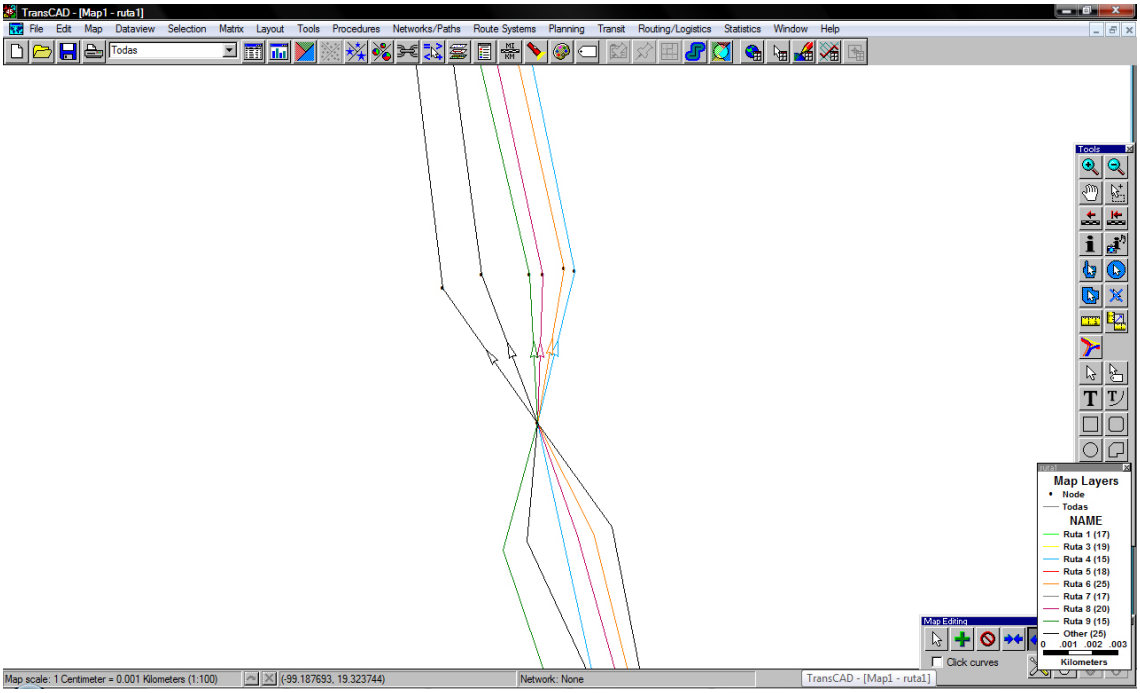


Figura 4.7. Híper arcos

En el campo *Enter a condition*, se escribe la sentencia de búsqueda

```
Arco_A=null,
```

es decir, se buscan todos los campos sin valor en la columna Arco_A. Automáticamente se despliegan en el *Dataview* todos los valores de la selección. Se selecciona la columna Arco_A y se asigna con valor único igual a 1. De manera similar se le asigna el valor 0 a las columnas H_Arco, Arco_P, Arco_B, Arco_Modal, Frec_matu, Frec_vesp, Frec_noc, Tiempo_recoleccion y Tiempo_entrega. Y se capturan las columnas Vel_AB, Vel_BA y Tiempo_abord de la tabla de atributos los datos correspondientes que se obtuvieron en trabajo de campo.

Para obtener los tiempo de recorrido en dirección de flujo, es decir Tiem_reco_AB, se utiliza el menú

$$\text{Fill} \rightarrow \text{Formula}$$

y en el recuadro que aparece se captura la fórmula,

$$(length/Vel\_AB) \times 60,$$

El multiplicador 60 se añade para transformar la unidades de medida a minutos. Análogamente, se capturan los datos para el tiempo de recorrido en contra de flujo, es decir Tiem_reco_BA. Por último, al campo *Dir* se le asigna el valor único 1, ya que los arcos fueron trazados en la dirección de flujo de los autobuses, en caso de que no se digitalize la red en el sentido de la dirección de flujo, será necesario indicar con un -1, los arcos que no cumplan esta regla.

Por último, falta agregar arcos a la red que representen la acción de no decender del *Pumabús*, para esto se traza un arco del nodo cabeza de cada híper arco al nodo cabeza del híper arco consecutivo, teniendo cuidado en que dichos híper arcos correspondan a autobuses de la misma línea. De esta forma el nuevo arco pasa junto al nodo que representa una estación, sin embargo, este nuevo arco, al no incidir en el nodo estación indica que no hace parada en éste. Los atributos de los nuevos arcos se capturan de forma similar a la de los arcos descritos anteriormente.

4.3.2. Añadiendo la red peatonal

En una vista nueva, se abre la red de peatonal y se agregan los campos en la tabla de atributos descritos en la página 64 de la sección 4.3. Se asigna el valor 1 a la columna Arco_P y en las columnas Vel_AB, Vel_BA, se capturan los datos correspondientes obtenidos trabajo de campo. En el caso que los arcos de la red representen un camino inclinado, la velocidad en sentido de flujo y en contra del flujo serán diferentes. Entonces, es necesario conocer que arcos de la red tienen una inclinación y a éstos asignarles la velocidad peatonal que le corresponda para lo cuál, se seleccionan manualmente los arcos con una inclinación usando la herramienta *Select by Pointing*, que se encuentra en el cuadro de herramientas del panel lateral derecho. Para seleccionar un conjunto de arcos es necesario mantener pulsada la tecla *Shift* y oprimir con el botón izquierdo del ratón los arcos deseados. La Figura 4.8 muestra donde se localiza el botón de *Select by Pointing* y un conjunto de arcos seleccionados.

Para capturar las velocidades del conjunto previamente seleccionado se consideran los siguientes criterios:

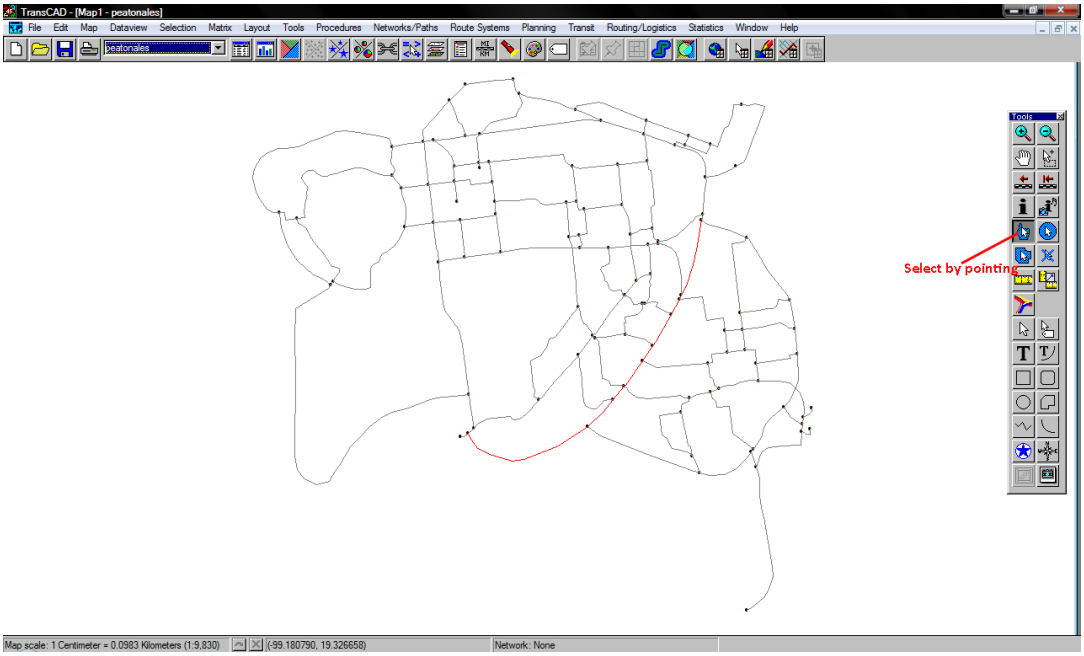


Figura 4.8. Localización de la herramienta *Select by Pointig*

- Si la pendiente inclinada hacia arriba va en la dirección de flujo o la pendiente inclinada hacia abajo va en contra de la dirección del flujo, entonces

$$\text{Vel\_AB}= 4.97km/h \text{ y } \text{Vel\_BA}= 5.7km/h$$

- Si la pendiente inclinada hacia arriba va en contra de la dirección de flujo o la pendiente inclinada hacia abajo va en la dirección del flujo, entonces

$$\text{Vel\_AB}=5.47km/h \text{ y } \text{Vel\_BA}=4.97km/h$$

Para los arcos que representan rutas planas se considera que Vel_AB=Vel_BA=$5.19km/h$. La Figura 4.9 muestra la red peatonal y sus diferentes velocidades.

Para calcular Tiem_reco_AB y Tiem_reco_BA se procede de forma similar al cálculo de tiempo en los autobuses. A los atributos H_Arco, Arco_P, Arco_A, Arco_Modal, Tiempo_abord, Frec_matu, Frec_vesp, Frec_noc, Tiempo_recoleccion y Tiempo_entrega se les asigna el valor cero.

Una vez configurada la tabla de atributos de la red, se añade la capa de la híper red de autobuses y se unen ambas accediendo al menú,

Tools → Geographic Utilities → Merge Geography.

Figura 4.9. Velcidades peatonales en las diferentes rutas de la red

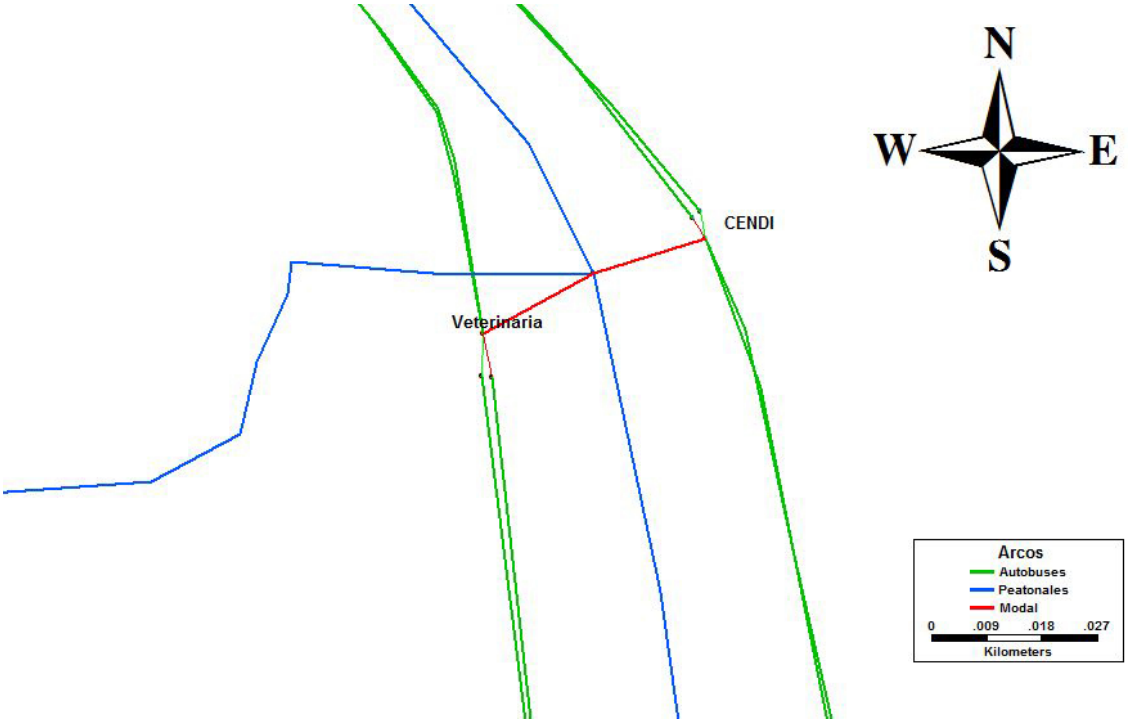


Figura 4.10. Arcos modales

Se guarda el nuevo archivo y se carga en una vista nueva. En ese momento la tabla de atributos de la red contendrá la información de la híper red de autobuses y la red peatonal.

Es necesario unir mediante arcos de transferencia modal (o arcos modales), la híper red de autobuses con la red peatonal para que sea posible transferirse entre los nodos de la híper red con los nodos de la red peatonal. Como el acceso a todas las paradas del *Pumabús* puede ser a pie, entonces, se trazan arcos que unan ambas redes con la herramienta *Add Line* (icono con una cruz verde), del *Map Editing Toolbox*. Los arcos que se tracen deben salir del nodo que representa una estación de autobuses al arco más cercano de la red peatonal, como se muestra en la Figura 4.10.

Una vez trazados todos los arcos que unen las redes, se captura la información de los arcos modales. Primero, se accede al menú

Selection → Selection by condition

y en el recuadro de *Enter a condition* se escribe la consulta

```
Arco_Modal=null
```

En el *Dataview* de la selección se asignan los siguientes valores a los campos de la tabla:

- Dir=0
- Arco_Modal=1
- Vel_AB=Vel_BA=5.19
- H_Arco, Arco_P, Arco_A, Arco_Modal, Tiempo_abord, Frec_matu, Frec_vesp, Frec_noc, Tiempo_recoleccion y Tiempo_entrega, se les asigna el valor 0.
- Tiem_reco_AB y Tiem_reco_BA, se calculan como previamente se explicó.

4.3.3. Añandiendo la red de bicicletas

Por último, falta integrar la red de bicicletas a la híper red que se creó en la sección anterior. En un vista nueva se carga la red de bicicletas que se configura de manera similar a la red peatonal, sólo que en este caso el campo de la tabla Arco_P es igual a 0 y Arco_B es igual a 1. En los campos de Vel_AB y Vel_BA es necesario capturar las velocidades correspondientes. Además, en las columnas Tiempo_recoleccion y Tiempo_entrega se capturan los tiempos de recolección y entrega de bicicletas que se obtuvieron en trabajo de campo.

Ya que no existen módulos de bicicleta que se encuentren en la inmediaciones de las estaciones de *Pumabús*, se decidió crear arcos de transferencia modal entre éste par de redes, si una o más estaciones de *Pumabús* se hayan en un radio de 150 metros alrededor de algún módulo de préstamo. Entonces, para conocer cuáles estaciones de *Pumabús* se encuentran dentro del radio descrito fue necesario crear *bandas* (*buffers*) sobre la red de peatonal. Se observa que las *bandas* se crean sobre la red peatonal, ya que sobre ésta se traslada el usuario para hacer la transferencia entre la red de bicicletas y la de autobuses.

Para crear las *bandas* se abre en una vista nueva la red peatonal y se compila el archivo de red (.net) accediendo al menú

Networks/Paths → Create.

Ahora se agrega el *shape* de los módulos de bicicletas a la vista actual. En la capa de nodos de la red peatonal se seleccionan aquéllos que se encuentren junto a un módulo de bicicletas[4].

Una vez seleccionados los nodos que representan módulos de préstamo se carga la aplicación *Network bands* en el menú

Networks/Paths → Network bands

En la ventana desplegada se captura la siguiente información; en el recuadro *Max. Impedance* se escribe 0.15, se selecciona la opción *Manual* y se da el valor de 0.15 en *Fixed Interval*, por último se oprime el botón de *OK* y se guarda el archivo. La Figura 4.11 muestra los bicicentros, la red peatonal y los *buffers* a 150 metros sobre la red.

[4]Si no existe un nodo cercano junto al módulo, es posible crear uno con la herramienta *Split line,* del *Map editing toolbox*

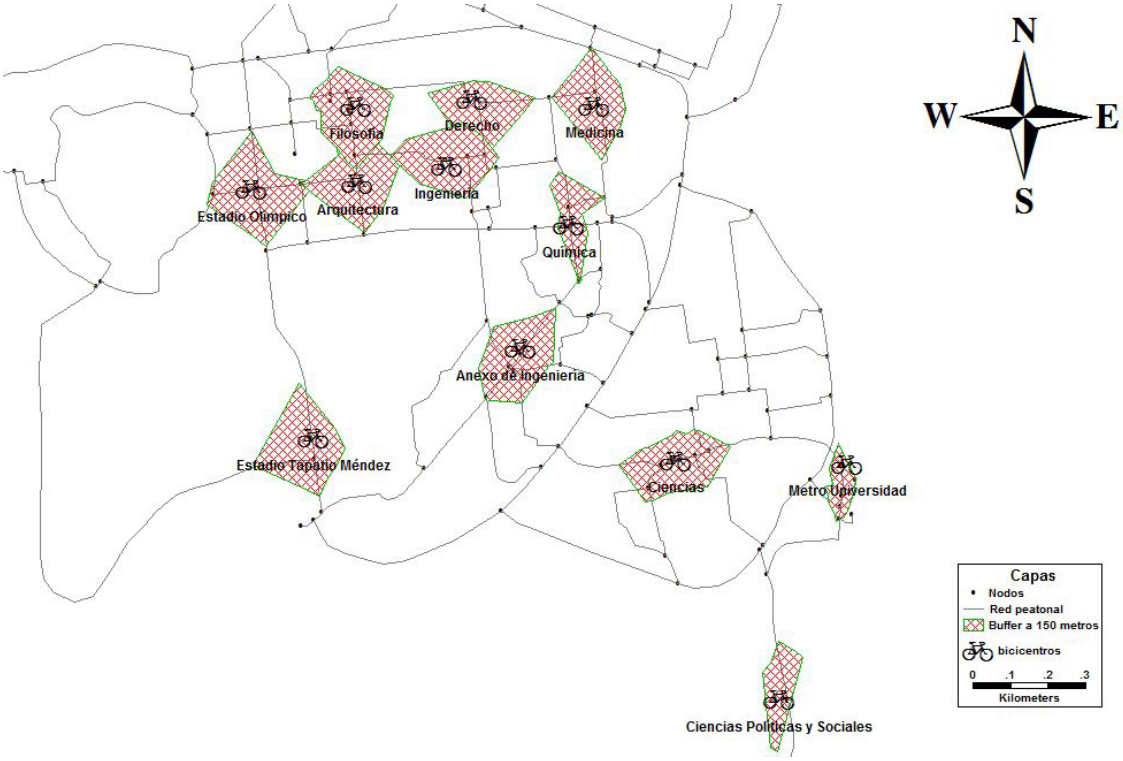


Figura 4.11. Buffers de red a 150 metros de los módulos de préstamo

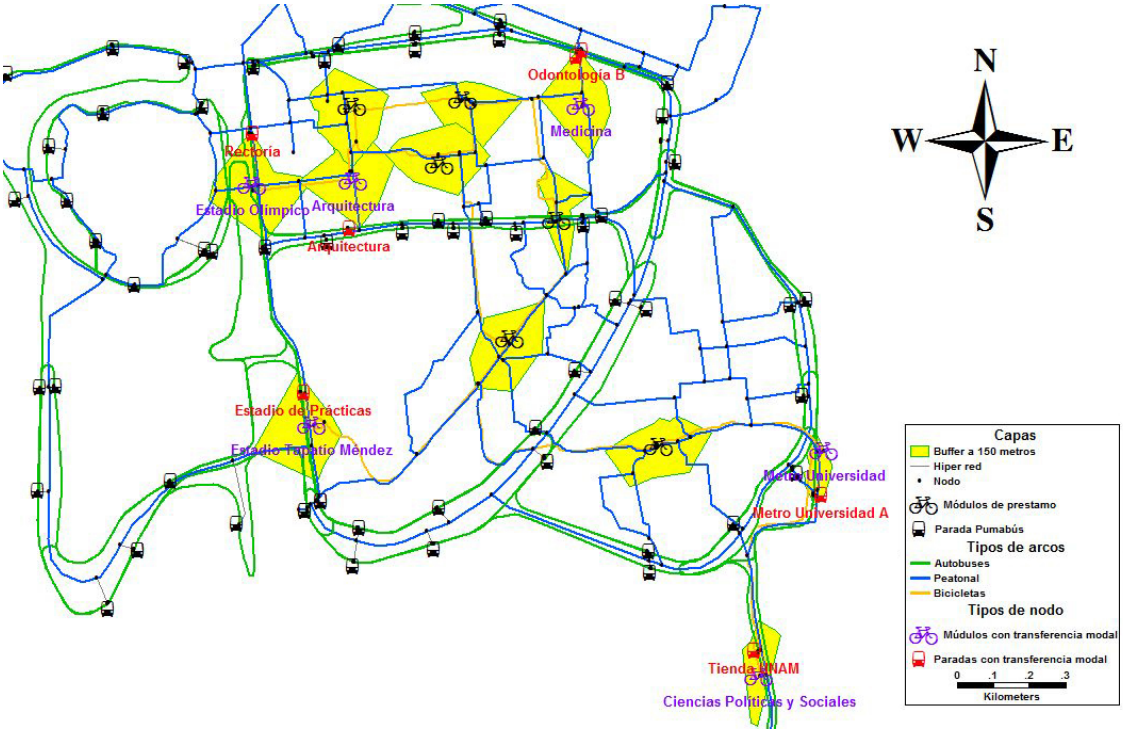


Figura 4.12. Transferencias modales entre módulos de bicicletas y paradas de autobuses

Ahora, en una vista nueva se cargan la red de bicicletas con la híper red que fue creada en la sección anterior y se unen ambas redes utilizando la herramienta *Merge Geography*. Una vez más, se abre en una nueva vista la nueva híper red y se añade la capa de *bandas* que se elaboró en párrafos anteriores. La Figura 4.12 muestra la capa de *bandas* y la híper red que contiene todos los modos de transporte. En esta figura se observa que existen cuatro módulos de bicicleta con al menos una estación de *Pumabús* en una radio de 150 metros. Los módulos y la estaciones con transferencia modal son los siguientes:

- Módulo Estadio Olímpico Universitario, tiene transferencia modal con la parada de autobús Rectoría.
- Módulo Arquitectura, tiene transferencia modal con la parada de autobús Arquitectura. Aunque esta parada no está dentro del *buffer* se encuentra en el límite de éste.
- Módulo Medicina, tiene transferencia modal con las paradas de autobús Odontología A y Odontología B.
- Módulo Metro Universidad, tiene transferencia modal con la parada de autobús Metro Universidad A.

- Módulo Ciencias Políticas y Sociales, tiene transferencia modal con la parada de autobús Tienda UNAM.
- Módulo Estadio Tapatío Méndez, tiene transferencia modal con la parada de autobús Estadio de Prácticas.

Para terminar, se une la red de bicicletas, a la peatonal y a la de autobuses mediante arcos de transferencia modal. Primero, se trazan los arcos de la red peatonal a la red de bicicletas de forma similar a la descrita en la Página 71. Se observa que la forma de trazar los arcos de la red de bicicletas a la híper red de autobuses deber ser respetando los caminos peatonales, es decir, los arcos modales se trazan lo más cercano posible a los arcos peatonales.

La información que se captura en los nuevos arcos modales, es la siguiente.

- Dir=0
- Arco_Modal=1
- Vel_AB=Vel_BA=5.19
- H_Arco, Arco_P, Arco_A, Arco_Modal, Tiempo_abord, Frec_matu, Frec_vesp, Frec_noc, Tiempo_recoleccion y Tiempo_entrega, se les asigna el valor 0.
- Tiem_reco_AB y Tiem_reco_BA, se calculan como previamente se explicó.

La Figura 4.13 muestra la híper red del trasporte público universitario resultante de todo el proceso.

4.4. Creación de la base de datos

Para que el algoritmo pueda consultar la información de la tabla de atributos de la híper red, se crea una base de datos geográfica. Ya que *TransCAD* no es propiamente un manejador de bases de datos (*Database Management System (DBMS)*) es necesario utilizar otro programa para que el algoritmo consulte fácilmente los datos creados en *TransCAD*. Actualmente existe una gran variedad de *DBSM*, en este caso se usa el programa de código abierto (*open-source*) *PostgreSQL*. Además se instala la herramienta *pgAdmin III* que administra la base de datos. Las instrucciones para instalar *PostgreSQL* y *pgAdmin III* se encuentran en http://www.postgresql.org/ y http://www.pgadmin.org/, respectivamente. Para facilitar la exportación de *shp* a *PostgreSQL* existe la extensión *PostGIS* que exporta fácilmente archivos *shp* en una base de datos geográfica. La extensión *PostGIS* y la instrucción de instalación se encuentran disponibles en http://postgis.refractions.net/.

Antes de exportar las tablas de atributos de la capa de híper arcos y de nodos, se crea en *TransCAD* la referencia de los nodos cola y cabeza de cada uno de los híper arcos. En el menú

Dataview → Formula Fields,

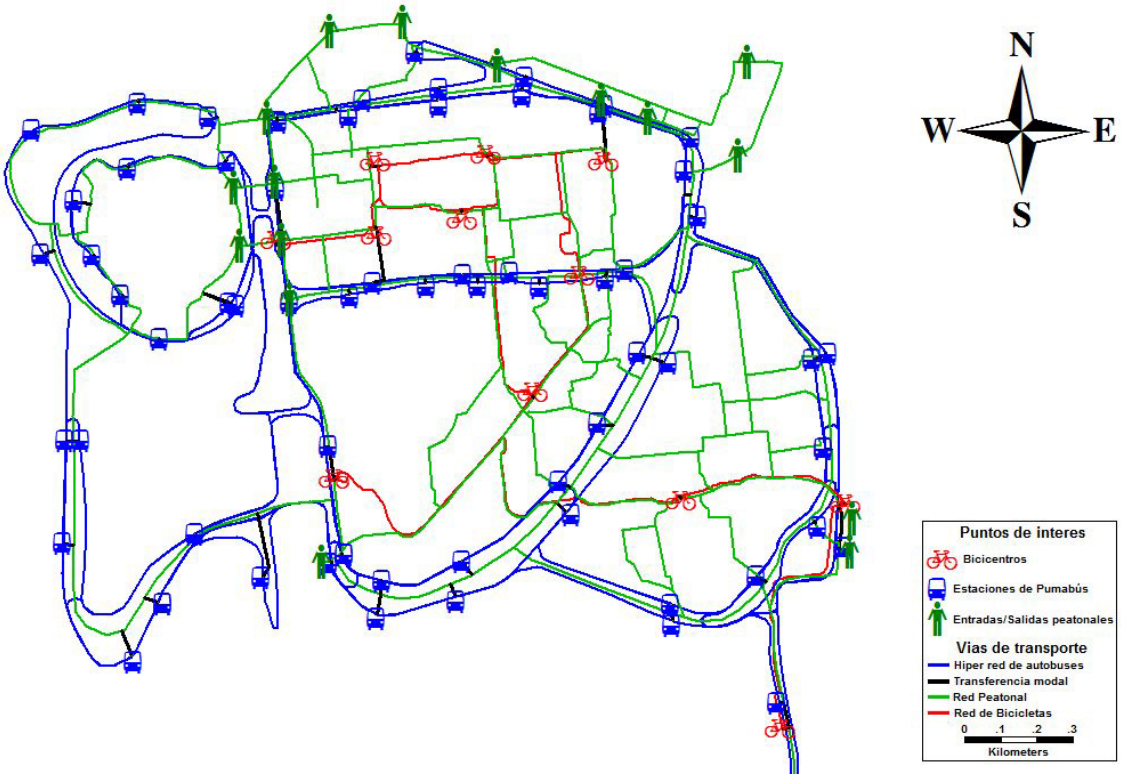


Figura 4.13. Híper red del sistema de transporte público universtario

se elige la opción *Node Field* y se selecciona el campo *ID*, con la opción marcada en *Both*. Esto crea en la tabla de atributos de los híper arcos, los campos *FromID* y *ToID*. Estos campos tienen asociado el identificador de nodo que corresponde al nodo cola y cabeza. Entonces se importa la capa de nodos e híper arcos a un *shp*. En el menú

$$\text{Tools} \rightarrow \text{Fxport},$$

se elige la opción *ESRI shp* y se guardan los archivos.

En *pgAdmin III* se crea una base de datos[5]. En el menú *plugins* se elige la opción *PostGIS shape file and DBF loader*. En la ventana que aparece; se selecciona la ruta donde están guardados los archivos *shape* de híper arcos y nodos, se da el valor 32614[6] al *SRID* y por último se presiona el botón de *Import*. Un ejemplo de cómo se configura la ventana con todos lo valores se muestra en la Figura 4.14.

Una vez creada la base de datos, se edita para que ésta contenga toda la información necesaria que requiere el algoritmo. Primero se agrega el campo *modos* a la tabla de arcos. Este campo toma los valores 1, 2, 3 y 4 que indican si se trata de un arco o híper arco de autobús, un arco peatonal, de bicicleta o de transferencias modal, respectivamente. Las siguientes sentencias de *SQL* crean el campo *modos* y dan los valores necesarios.

```
ALTER TABLE arcos ADD COLUMN modos integer;

UPDATE arcos
   SET modos = CASE WHEN arco_a = 1
                     THEN 1
                    WHEN arco_p = 1
                     THEN 2
                    WHEN arco_b = 1
                     THEN 3
                    WHEN arco_modal = 1
                     THEN 4
                    WHEN h_arco = 1
                     THEN 1
                    END
```

Con la siguiente sentencia *SQL* se crea una nueva columna en la tabla de arcos que indique a que línea corresponde cada ruta de *Pumabús*, es decir, esta columna se llena con los números 1,2,3,4,5,6,7,8,9 y 11 que son las líneas disponibles.

```
ALTER TABLE arcos ADD COLUMN ruta integer;

UPDATE arcos
```

[5]Existe una gran cantidad de manuales disponibles en la red para crear bases de datos con *pdAdmin III* por lo que se omitirá esta parte.

[6]Este valor corresponde al proyección espacial WGS 1984 UTM zona 14 Norte

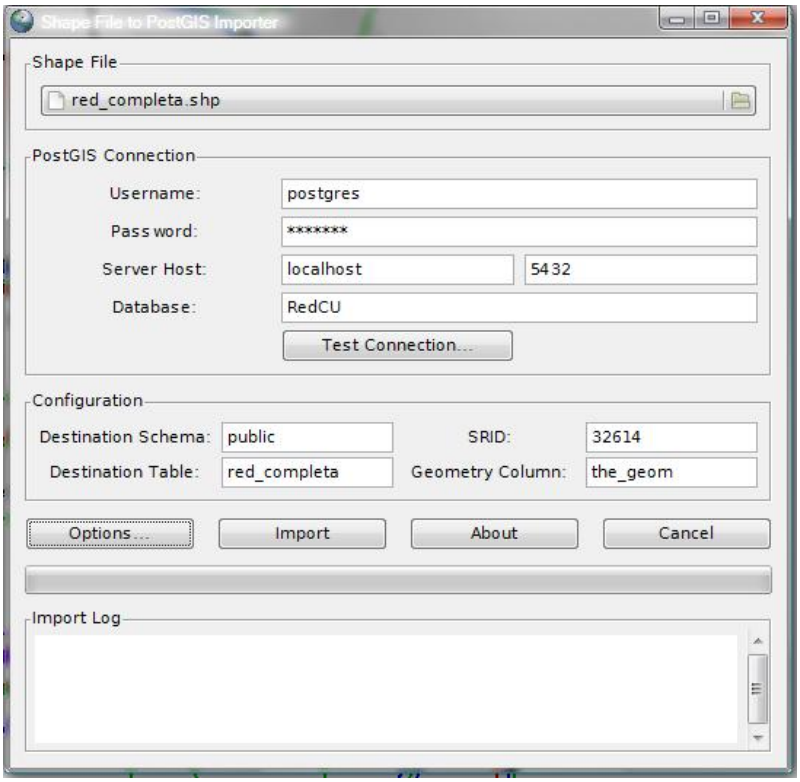


Figura 4.14. Exportar un *shape* a la base de datos

```
SET ruta = CASE WHEN name = 'Ruta 1'
                 THEN 1
                WHEN name = 'Ruta 2'
                 THEN 2
                WHEN name = 'Ruta 3'
                 THEN 3
                WHEN name = 'Ruta 4'
                 THEN 4
                WHEN name = 'Ruta 5'
                 THEN 5
                WHEN name = 'Ruta 6'
                 THEN 6
                WHEN name = 'Ruta 7'
                 THEN 7
                WHEN name = 'Ruta 8'
                 THEN 8
                WHEN name = 'Ruta 9'
                 THEN 9
                WHEN name = 'Ruta 11'
                 THEN 11
                END
```

Ya que el algoritmo toma un identificador de nodo para consultar en la base de datos los arcos entrantes, es necesario crear arcos en las dos direcciones siempre y cuando sea posible recorrer las vialidades que representan en ambos sentidos. Los arcos que representan rutas peatonales y de bicicleta siempre es posible recorrerlos en ambas direcciones, mientras que en los arcos de autobuses ésto no es posible. Entonces se agregan nuevos arcos a la base de datos en la tabla de arcos. Como el comportamiento de "ambas direcciones" se definió en *TransCAD*, resulta casi inmediato imitar con sentencias *SQL* dicho comportamiento. Los siguiente pasos crean los arcos de regreso en la base de datos .

1. Se crea una tabla los arcos que pertenezcan la red de bicicletas, peatonal y de transferencia modal. Estos son todos los arcos cuyo campo *Dir* es igual a cero.

   ```
   CREATE TABLE arcos_regreso AS
     SELECT * FROM arcos WHERE dir = 0;
   ```

2. Se crea una nueva tabla basada en la tabla del paso 1 con las columnas invertidas vel_ab, vel_ba, tiem_ab, tiem_ba, *fromID* y *toId*.

   ```
   CREATE TABLE arcos_regreso_invertido AS
      SELECT gid, id, length, dir, name, arco_a, arco_p, arco_b, arco_modal, h_arco,
          vel_ba, vel_ab, tiem_ba, tiem_ab, tiempo_abo, frec_matu, frec_vesp,
          frec_noc, tiempo_rec, tiempo_ent, to_id, from_id, the_geom, modos, ruta
          FROM arcos_regreso;
   ```

3. Para no repetir identificadores, se actualizan los identificadores de la tabla que se creó en el paso anterior

```
UPDATE arcos_regreso_invertido
   SET id = id + 824;
```

4. Por último, se inserta la tabla en la tabla de arcos

```
INSERT INTO arcos
SELECT * FROM arcos_regreso_invertido;
```

Todo lo expuesto en este capítulo es lo mínimo necesario para crear una híper red funcional con su base de datos correspondiente. Es importante recalcar que el *DBMS* es *software* libre, lo cual reduce costos si se desea implementar esta herramienta en otras áreas como en el gobierno. Y aunque la red se trazó en *TransCAD* también es posible trazar ésta en SIG's de distribución libre como *gvSIG*, con lo que se reducirían aún más los costos de implementar este tipo de sistemas. Por último, se decidió utilizar *TrasCAD* pues el manejo del programa se enseña durante los primeros semestres de maestría, por lo que la guía que presenta esta tesis para el trazado de la red, da un valor agregado para las futuras generaciones de Sistemas del Transporte.

CAPÍTULO 5

Algoritmos en híper rutas más cortas

En este capítulo se describen los algoritmos *Shortest Hyperpath Tree* (*sht*) de Pallottino & Schettino (1999) y *Shortest Viable Hyperpath Problem* (*SVHP*) de Lozano & Storchi (2002). Y por último, se explica a detalle el procedimiento que se usó para encontrar las híper rutas más cortas.

5.1. Algoritmo *Shortest Hyperpath Tree*

El algoritmo encuentra el híper árbol que contiene las híper rutas más cortas entre parejas de nodos, sin embargo, no considera la posibilidad de transbordos multimodales. Aunque, el procedimiento no sirve para los propósitos de la tesis, es en éste en el cual que se basa una buena parte de los algoritmos de híper rutas más cortas, de ahí su importancia y la razón de incluirlo en este trabajo. El algoritmo consta de un procedimiento principal llamado *Shortest Hyperpath Tree* y un sub procedimento llamado *Conjunto Atractivo*.

Las variables utilizadas en el procedimiento principal y el sub procedimiento son:

- L_u^* := Conjunto atractivo en la parada u.
- $L(u)$:= Conjunto de líneas que sirven la parada u.
- c_u := Costo de la híper ruta p_{ud}.
- θ := Parámetro de conversión en unidades de costo generalizado y el coeficiente de regularidad del servicio.
- ϕ_l := Es la frecuencia de la línea l.
- Φ_u^* := La frecuencia combinada del conjunto atractivo de líneas que pasan por u.
- $s(u)$:= El nodo sucesor de u en la híper ruta.

- $c(u) :=$ El costo del arco que incide en u, y está definido por

$$c_u = \begin{cases} \bar{c}_{uv} + c_v & \text{si } (u,v) \text{ no es un híper arco} \\ \dfrac{\theta + \displaystyle\sum_{(u,v)\in FS_p(u)} c_v \phi_v}{\displaystyle\sum_{(u,v)\in FS_p(u)} \phi_v} & \text{si } FS_p(u) \text{ pertenece al conjunto atractivo en la parada } u \end{cases}$$

  En donde $\bar{c}_{uv}$ es el costo del arco (u,v) y $FS_p(u)$ es la *estrella saliente* de u que pasa por la híper ruta p.

- $B(j) :=$ El conjunto de híper arcos que entran al nodo j.

El procedimiento *Conjunto Atractivo* encuentra el conjunto de líneas que van de u a d, tal que en el nodo u el usuario está dispuesto a abordar la primera corrida que arribe a la estación u y que lo lleve a su destino d. Este procedimiento funciona de la siguiente forma. Primero se encuentra la línea l con menor costo c y después se examinan cada una de las siguientes líneas que tengan parada en u. Si el costo c más el tiempo esperado mínimo de la otra línea es menor que la primera que se examinó, se agrega al conjunto atractivo, en caso contrario se desecha la línea. El proceso se repite hasta examinar todas las líneas que pasan por u. A continuación se enuncia el procedimiento.

Procedimiento Conjunto atractivo $(u, L(u), c, \phi, \theta, L_u^*, \Phi_u^*, val)$
begin
$Sort(L(u)); k := 2; L_u^* := \{v_1\}, \Phi_u^* = \phi_{v_1}, val = \theta/\Phi_u^* + c_{v_1}$
while $k = |L(u)|$ **and** $c_{v_k} < val$ **do**
begin
$\Phi_u^* := \Phi_u^* + \phi_{v_k}$
$val := val - (val - c_{v_k})\phi_{v_k}/\Phi_u^*$
$L_u^* := L_u^* \cup \{v_k\}$
$k = k + 1$
end
end

Antes de describir el algoritmo *Shortest Hyperpath Tree*, se enuncia la condición necesaria para determinar híper rutas más cortas. La condición generalizada de *Bellman* indica cuando es posible agregar un nodo u a la híper ruta. Esta condición establece que:

Condición 5.1.1 *Una híper ruta es óptima si y sólo si cumple que para toda* $u \in V$

$$c_u \leq \begin{cases} \bar{c}_{uv} + c_v & \textit{para todo arco } (u,v) \in FS(u) \\ \dfrac{\theta + \displaystyle\sum_{(u,v)\in FS_p(u)} c_v \phi_v}{\displaystyle\sum_{(u,v)\in FS_p(u)} \phi_v} & \textit{para todo híper arco } L \subseteq L(u) \end{cases}$$

Esta condición restringe que el costo de un nodo en la híper ruta, siempre debe de ser menor que el costo de sus nodos consecutivos. Si por alguna razón la condición no se cumple, es decir, el costo de un nodo u es mayor que el de alguno de sus nodos consecutivos, necesariamente la híper ruta que pase por u no es óptima, ya que, es posible acceder a los otros nodos por alguna híper ruta diferente y con menor costo.

La exploración del procedimiento *Shortest Hyperpath Tree* comienza en el destino d del la híper ruta p_{od} que se desea encontrar. En cada iteración se explora la estrella entrante de u, $BS(u)$, y para cada arco (v, u) se verifica si se respeta la condición de *Bellman*. En el caso de que u sea una parada, antes de explorar $BS(u)$, se llama al procedimiento *Conjunto atractivo* para determinar que sub conjunto de híper arcos se van a considerar en los siguientes pasos, una vez determinado el conjunto atractivo se verifica si éste cumple con la condición generalizada de *Bellman*.

Ahora, si el costo del conjunto atractivo que se determinó es menor que el costo de algún otro arco, lo datos del conjunto atractivo se almacenan y se borran los sucesores $s(u)$. En la exploración de estrella entrante en u, $BS(u)$, si $(v, u) \in As$ (es decir, es un híper arco) no se efectúan actualizaciones para c_v. Se incluye v en Q cuando el conjunto atractivo puede ser mejorado (es decir, si $c_v < c_u$). En el caso de un arco de otro tipo el procedimiento realiza las operaciones de un algoritmo de ruta mínima. En los párrafos siguientes se enuncia el pseudo código del algoritmo *Shortest Hyperpath Tree*.

Procedimiento Shortest Hyperpath Tree $(j, L, c, \overline{c}, \phi, \theta, s, L^*, \Phi^*)\}$
begin
for each $u \in N$ **do begin** $s(u) := 0$; $c_u := \bullet$ **end**;
$s(j) = \emptyset$; $c_j := 0$; $Q := \{j\}\backslash$;
repeat
$Select\&Remove(u, Q)$
if $u \in F$ **then**
begin
$Conjunto\ atractivo(u, L(u), c, \phi, \theta, set, freq, val)$
if $val < c_u$ **then**
begin $c_u := val, L^*_u := set, \Phi^*_u = freq, s(u) := 0$ **end**
end;
for each $(v, u) \in BS(u)$ **do**
if $(v, u) \in A_s$
then if $v \notin Q$ **and** $c_v > c_u$ **then** $Insert(v, Q)$
else if $c_v > c_u + \overline{c}_{vu}$ **then**
begin
$c_v := c_u + \overline{c}_{vu}$; $s(v) := u$;
if $v \in F$ **then** $\Phi^*_u = 0$;
if $v \notin Q$ **then** $Insert(v, Q)$
end

until $Q = \emptyset$
end

La operación $Select\&Remove(u, Q)$ funciona en forma de lista. En la secuencia de nodos almacenados, el primero de ellos se denomina *cabeza* y al último *cola*. Entonces el procedimiento inserta nodos en la *cola* con la operación $Insert(v, Q)$ y los extrae de la *cabeza* con la operación $Select\&Remove(u, Q)$.

5.2. Algortimo *Shortest Viable Hyperpath Problem*

El algoritmo *SVHP* encuentra las híper rutas viables que no contengan más de k transferencias modales. Este algoritmo consta de un procedimiento principal llamado *Shortest Viable Hyperpath Problem* y tres sub procedimientos llamados *Arc-Concatenation*, *h-Arc-Concatenation* y *Determine-s*. A continuación se explican cada uno de éstos.

El algoritmo usa las siguientes etiquetas en el procedimiento principal y los sub procedimientos.

- $V_s^*(i)$:= tiempo esperado de viaje en la actual híper ruta más corta $[i, s]$.
- $w_s(i)$:= cota superior del número de transferencias modales en la actual híper ruta más corta $[i, s]$.
- $SA_s(i)$:= híper arco sucesivo del nodo i, en la actual híper ruta más corta $[i, s]$.
- $ST_s(i)$:= conjunto de estados de las híper rutas más cortas que están actualmente en las cabezas del híper arco $SA_s(i) =$.
- $lastlabel_s(i)$:= tiempo esperado de viaje de la híper ruta más corta $[i, s]$ con cota superior de transferencias modales menor que $w_s(i)$.

Para cada híper arco que cumpla que $|h(e)| > 1$, es decir, aquellos híper arcos que representan más de dos rutas diferentes de transporte público, la siguiente información se conserva.

- $h(e)$:= el conjunto de nodos que componen la cabeza de e.
- $\Phi(e)$:= la frecuencia de la líneas que pueden ser tomadas a través de e.
- $C^*(e)$:= el tiempo esperado de viaje mínimo que se obtiene utilizando e.

En el procedimiento *Shortest Viable Hyperpath Problem* se etiqueta una pareja $[i, s]$ cuando el tiempo esperado de viaje de la híper ruta p_{id} en el *estado-s*[7] se mejora utilizado el arco (i, j). Después, si el arco (i, j) no es de transferencia modal, entonces $[i, s]$ se inserta en $Qnow$, de lo contrario se inserta en $Qnext$.

En cada iteración sólo se examinan los elementos de $Qnow$ y se van eliminando hasta que $Qnow = \emptyset$. Después se toman los elementos de $Qnext$ hasta que este conjunto está vacío. La dominancia entre rutas se controla de la siguiente forma, si $V_{s_j}^* < lastlable_{s_x}(j)$ entonces se actualiza

[7] Los *estados s* representan las diferentes combinaciones permitidas entre nodos. Para conocer más sobre los *estados-s* consultar Lozano & Storchi (2002).

la etiqueta $lastlable_{s_x}(j) = V^*_{s_j}$, de lo contrario la híper ruta p_{jd} de *estado-*s_x no se considera más. El procedimiento *SHVP* es el siguiente:

Procedimiento Shortest Hyperpath Tree
{
for each $j \in N$ **do** $V^*_{s_x} = \infty \forall s$;
for each $e \in E$ **do**
$\{C^*(e) = \infty; h^*(e) = \emptyset; \Phi^*(e) = 0\}$
$Qnow = \{[d, 0]\}; V^*_0(d) = 0\backslash;$
repeat{
while $Qnow \notin \emptyset$ {
select $[j, s_x]$ **from** $Qnow$
$Qnow = Qnow \setminus \{[j, s_x]\}$
if $(V^*_{s_j} < lastlable_{s_x}(j))$ {
$lastlable_{s_x}(j) = V^*_{s_j}$
for each $e \in B(j)$ **do** {
$i = t(e)$
if $(|h(e)|) = 1$
call Procedimiento Arc−Concatenation(e, s_x, V^*, c, w)
else
call Procedimiento h−Arc−Concatenation$(e, s_x, V^*, C^*, w, h^*, \phi)$
}
}
}
$Qnow = Qnext; Qnext = \emptyset; h = h + 1$
}**until** $h > k$ **or** $Qnow = \emptyset$
}

Con los procedimientos procedimientos *Arc-Concatenation* y *h-Arc-Concatenation* se van agregando arcos o híper arcos a la ruta p_{id}, según sea el caso. *Arc-Concatenation* analiza los arcos $e = (i, j)$ de la híper red y determina si existe una nueva híper ruta viable p con *estado-s* y que no supere al límite de transferencias modales. Para comprobar si es posible crear p con *estado-s* se compara con la híper ruta actual y con las demás híper rutas con *estados-*s_y, en donde s_y es el conjunto de estados preferidos a s. Si p mejora en costo a las otras, entonces se concatena (i, j) con p , en caso contrario, no se genera la híper ruta. Este procedimiento se describe a continuación.

Procedimiento Arc−Concatenation(e, s_x, V^*, c, w)
{
$wt = 0; s = 0; con = 0\backslash;$
switch $(mode_e)$
case $(\in M1:)$
if $s_x \neq 5, 12, 14, 20$ **call** $State-M1(s_x, s)$

break
case ($\in M3$:)
if $s_x \neq 8, 14, 18, 20$ **call** $State{-}M3(s_x, s)$
break
case ($\in M4$:)
if $s_x \neq 11, 16, 18, 20$ **call** $State{-}M4(s_x, s)$
break
case ($\in M6$:)
if $(w_{s_x}(j) < k$ **and** $s_x \neq 0)$ $wt = 1$; **call** $State{-}M4(s_x, s)$
break
default :
if $(s_x = 0)$ $s = 1$
else $s = s_x$
break
endswitch
if $(s \neq 0$ y $V^*_{s_x}(j) + c(e) < V^*_s(i))$ **call** $States(s, sM)$
else $SM = \emptyset$
while $(EOF(SM) = 0$ **and** $con \neq 1)\{$
select s_y **from** SM
if $(V^*_{s_x}(j) + c(e) < V^*_{s_y}(i))\{$
$V^*_s(i) = V^*_{s_x}(j) + c(e)$
$w_s(i) = w_{s_x}(j) + wt$
$SA_s(i) = e$
$ST_s(i) = s_x$
if $(wt = 0$ **and** $[i, s] \notin Qnow)$ $Qnow = Qnow \cup \{[i, s]\}$
if $(wt = 1$ y $[i, s] \notin Qnext)$ $Qnext = Qnext \cup \{[i, s]\}$
$con = 1$
}
}
}

h-Arc-Concatenation examina aquellos híper arcos e, tales que $|h(e)| > 1$, en donde la híper ruta p_{jd} que se concatena cumple que $j \in h(e)$. En el procedimiento primero se determina el *estado-s* de la híper ruta usando el sub procedimiento *Determine-s*, después se actualizan las etiquetas de los nodos cabeza de e, de la frecuencia combinada de e, el conjunto de híper rutas viable por e y el tiempo de viaje esperado mínimo. Dicha actualización se efectúa siempre y cuando $V^*_{s_x}(j) < V^*_s(i)$, es decir, siempre que el tiempo esperado de viaje de la híper ruta p_{jd} mejora al tiempo esperado de viaje de la híper ruta p_{id}, en caso contrario, la concatenación no se realiza y por lo tanto tampoco se actualizan las etiquetas. En seguida se describe el procedimiento.

Procedimiento h−Arc−Concatenation$(e, s_x, V^*, C^*, w, h^*, \phi)$

{
  $con = 0\backslash$;
  if $(\Phi^*(e) \neq 0)$ **call** *Determine−s* $(s, s_x, s_z(e))$
  else $s = s_x$
  if $(V^*_{s_x}(j) < V^*_s(i))$ {
    $\Phi^*(e) = \Phi^*(e) + \phi_j$
    $h^*(e) = h^*(e) \cup \{j\}$
    $state(e) = state(e) \cup \{s_x\}$
    $s_z(e) = s$
    if $(\Phi^*(e) = \phi_j)$ $C^*(e) = 1/\phi_j + V^*_{s_x}(j)$
    else $C^*(e) = C^*(e) - (C^*(e) - V^*_{s_x}(j))\phi_j/\Phi^*(e)$
    if $(C^*(e) < V^*_s(i))$ **call** $states(s; PS)$
    else $PS = \emptyset$
    while $(EOF(PS) = 0$ **and** $con \neq 1)$ {
      select s_y **from** SM
      if $(C^*(e) < V^*_{s_y}(i))$ {
        $V^*_s(i) = C^*(e)$
        $w_s(i) = w_{s_x}(j)$
        $SA_s(i) = e$
        $ST_s(i) = state(e)$
        if $([i, s] \notin Qnow)$ $Qnow = Qnow \cup \{[i, s]\}\}$
        $con = 1$
      }
    }
}

Por último, el procedimiento *Determine-s*, determina el estado que resulta de las transiciones entre el estado de la híper ruta actual y la nueva híper ruta. Este procedimiento está basado en un grafo de estados que depende del número de modos que se deseen modelar. Como el número de modos en CU es menor que el número de modos del procedimiento de Lozano & Storchi (2002), es necesario hace ciertas adaptaciones a este procedimiento para que funcione en CU. Dichas modificaciones junto con algunas adaptaciones de los demás procedimientos se describirán en la Sección 5.3.

Procedimiento Determine−s$(s, s_x, s_z(e))$
{
  $sum = s_z(e) + s_x$
  if $(s_z(e) = 1)$ $s = s_x$
  else if $(s_x = 1$ **or** $s_z(e) = s)$ $s = s_z(e)$
    else if $(sum = 19)$ {
      if $(s_z(e) = 14$ **or** $s_x = 14)$ $s = 14$

else $s = 18$
}
else if $(sum = 26$ **or** $29) s = 18$
else if $(sum = 16, 21$ **or** $27)$ $s = 16$
else if $(sum = 13$ **or** $22)$ $s = 14$
else$s = 20$
}

El procedimiento que se utiliza para resolver las híper rutas más cortas en CU es una adecuación del algoritmo *Shortest Viable Hyperpath Problem*. El algoritmo *SVHP* considera un sistema de transporte con seis modos de transporte y 20 estados posibles para las transiciones entre híper arcos, para el modelo de esta tesis solo se necesitan cuatro modos de transporte y por las características de la red sólo existen cuatro estados posibles para las transiciones entre híper arcos. En la siguiente sección se explica a detalle el funcionamiento del algoritmo propuesto esta tesis.

5.3. Concatenación de modos del algoritmo SVHP para el caso de CU

El híper grafo que modela el sistema de transporte público universitario consta de tres modos de transporte y un modo de transferencia modal para indicar en qué puntos es posible hacer un cambio de modo. Entonces sea H un *híper grafo multimodal* en donde los modos de trasporte asociados a esta híper grafo son;

- $M1$:= El subconjunto de híper arcos que representan las líneas del *Pumabús*.
- $M2$:= El subconjunto de arcos que representan las rutas peatonales.
- $M3$:= El subconjunto de arcos que representan las rutas del *Bicipuma*.
- $M4$:= El subconjunto de arcos que representan las transferencias modales.

Sea el híper grafo *modo-r* aquel cuyos híper arcos sólo pertenecen al subconjunto de híper arcos Mr, entonces un híper grafo *modo-r* es restringido si y sólo si existe una única sub híper ruta *modo-r* en la híper ruta que lo contiene. En el caso del modelo aquí presentado, el único modo restringido es el modo $M3$, entonces, cuando se desee encontrar la híper ruta más corta, p_{od}, ésta sólo podrá incluir una única sub híper ruta *modo-3*. El resto de los modos no tienen restricciones por lo que podrán incluir más de una sub híper ruta *modo-r* en p_{od}, tal que $r \in \{1, 2, 4\}$. Por lo tanto, una híper ruta es *viable* si y sólo si no contiene más de una sub híper ruta *modo-3* maximal.

Ya que existen restricciones en los modos de transporte y en el número de transferencias modales, es necesario tener un registro de los modos utilizados en p_{od}. Por lo que se necesita conocer el *estado-s* de la híper ruta que se está transitando.

Un *estado-s* es una clave que sirve para indicar las composiciones admisibles de modos en un híper ruta viable. Como la híper ruta viable está compuesta de sub híper rutas viables con

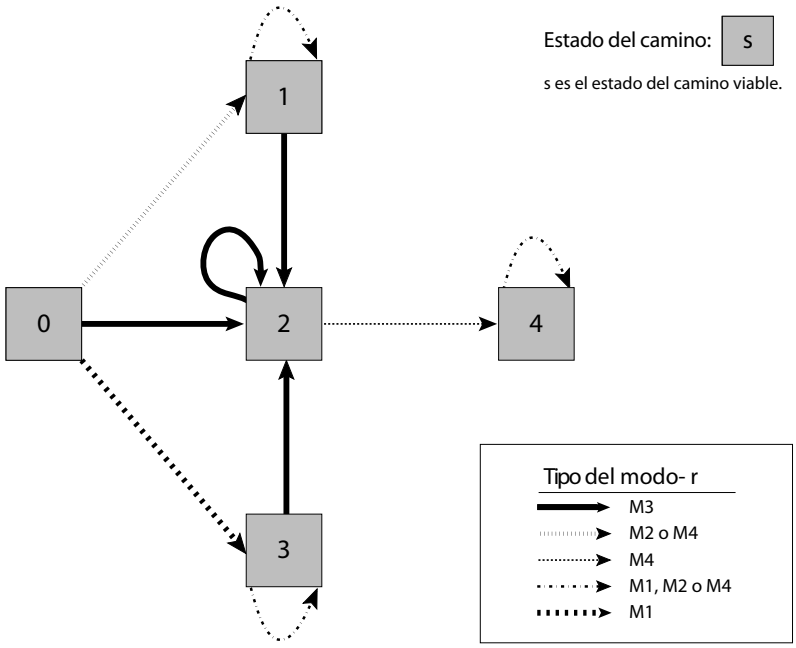


Figura 5.1. Transiciones posibles de los caminos viables. Adaptación del grafo de estados que se encuentra en Lozano & Storchi (2002)

algún *estado*, entonces, es posible asociar un *estado* a la híper ruta viable. Dicho *estado* servirá para indicar la secuencia de modos utilizados en todas las sub híper rutas que componen la híper ruta.

Ahora, como se mencionó en párrafos anteriores, es posible utilizar tantas veces como se desee los modos Mr tal que $r \in \{1, 2, 4\}$ y sólo se puede utilizar una vez el modo $M3$ por cada híper ruta. Entonces existen 4 *estados* posibles relacionados con las secuencias de modos que se utilicen en los híper caminos viables. La Figura 5.1 muestra la transiciones posibles entre los estados.

Sea $G_{estados}$ el grafo que se muestra en la Figura 5.1. Los nodos representan el *estado-s* asociado a algún camino viable, por otra parte los arcos representan un viaje a través del modo Mr. Se observa que un viaje siempre comienza en el *estado* 0 y a partir de este *estado* hay tres posibilidades para comenzar el recorrido, éstas son: en bicicleta, a pie o en autobús.

Ya que las rutas peatonales y los arcos de transferencia no están restringidos es posible asociar a ambos el *estado* 1. Por lo tanto, existe un arco del modo $M2$ o $M4$ en $G_{estados}$ que une al *estado* 0 con el *estado* 1. Mientras se continúe el viaje a pie, en autobús o se realice una transferencia se permanece en el estado 1, es decir, el estado 1 tiene un *lazo*[8]. De manera similar, se une el *estado* 0 con el *estado* 3 mediante un arco de modo $M1$ si se comienza el viaje en *Pumabús*.

[8]Un *lazo* es un arco que cuyo nodo cola y cabeza es él mismo

Por otro lado, es posible llegar a *estado* 2, si el viaje empieza en bicicleta, esto es, pasar del *estado* 0 al 2 por un arco de modo $M3$. Mientras se continúe el viaje en bicicleta se permanecerá en el *estado* 2. Si antes de abordar una bicicleta se proviene de algún otro modo de transporte, es necesario realizar una transferencia modal, es decir, se pasa del estado 1 al 2 o del 3 al 2. Si se desea salir del estado 2 la única forma de hacerlo, es también, por un arco de transferencia modal $M4$, y entonces se alcanza el *estado* 4. A partir del *estado* 4 la única forma de continuar el viaje para que sea viable es caminando, en autobús o haciendo una transferencia modal, por lo que en este estado existe un lazo de modo $M1$, $M2$ o $M4$.

Dependiendo en qué *estado* se encuentre la híper ruta viable, puede o no contener viajes en autobús, es decir, puede o no haber sub híper rutas viables contenidas en la híper ruta viable. Resulta inmediato verificar que los únicos *estados* que es seguro que contengan híper rutas viables son los *estados* 1 y 3. Ahora, si la híper ruta viable, p_{id} es el resultado de la concatenación del híper arco e y también se asume que, p_{id} contiene al menos dos híper rutas viables, $p_{j_1 d}$ con *estado* s_1 y $p_{j_2 d}$ con *estado* s_2 tal que $j_1, j_2 \in h(e)$, entonces es necesario determinar el *estado* de p_{id} a partir de los *estados* de $p_{j_1 d}$ y $p_{j_2 d}$. Para que se cumpla la restricción de modos el *estado-s* de p_{id} debe indicar la secuencia de modos utilizados en los estados s_1 y s_2.

Ya que los únicos *estados* con híper rutas son el 1 y 3, sólo existen tres posibilidades para el estado de p_{id} que son:

1. $p_{j_1 d}$ tiene *estado* 1 y $p_{j_2 d}$ tiene *estado* 1, como ambos son iguales, ambos tienen la misma secuencia de modos utilizados, entonces, el estado de p_{id} es 1.

2. $p_{j_1 d}$ tiene *estado* 3 y $p_{j_2 d}$ tiene *estado* 3, como ambos son iguales, ambos tienen la misma secuencia de modos utilizados, entonces, el estado de p_{id} es 3.

3. $p_{j_1 d}$ tiene *estado* 1 y $p_{j_2 d}$ tiene *estado* 4. Como la ruta $p_{j_2 d}$ contiene una sub ruta con un modo restringido, entonces, el estado de p_{id} es 4, lo que asegura que no se utilice más el modo restringido.

Como se mencionó anteriormente el número de transferencias modales está restringido, esto es, no se pueden realizar mas de k transferencias modales en una híper ruta p_{od}. El usuario del algoritmo establecerá k de acuerdo a sus necesidades. Ahora, la forma de contar transferencias modales dependerá del tipo de arco que se esté transitando. Suponga, que se concatena un arco (i, j) del tipo $M4$ (transferencia modal) a la híper ruta p_{jd} con h transferencias modales, entonces, la híper ruta p_{id} tendrá $h + 1$ transferencias modales. Dicha concatenación es posible si y sólo si $h + 1 \leq k$.

Por otro lado, si se concatena un híper arco $e' = (i, h(e'))$ al conjunto de híper rutas viables p_{jd} para $j \in h(e')$ tal que $h(e') \subseteq h(e)$, es posible que el número de transferencias de las sub híper rutas de p_{jd} sean diferentes. Por ejemplo, sea e' un híper arco, tal que $h(e') = \{j_1, j_2, j_3\}$, en donde, el número de transferencias modales para $p_{j_1 d}$, $p_{j_2 d}$ y $p_{j_3 d}$ es a, b y c respectivamente. Entonces el número de transferencias modales de la híper ruta viable p_{id}, es decir, la híper ruta viable que resulte de la concatenación de e' con p_{jd}, será igual al máximo de los valores entre a, b y c. Dicha concatenación es posible si y sólo sí $max\{a, b, c\} + h \leq k$. Por lo tanto, el número de transferencias

modales de p_{id} es $w(i) = max\{w(j) \text{ tal que } j \in h(e')\}$.

A continuación se presenta el Algoritmo SVHP para el caso de CU en pseudo lenguaje. El *Procedimiento Shortest Viable Hyperpath Problem*, se copia directamente para la aplicación del Algoritmo SVHP para el caso de CU pues en este punto no es necesario hacer modificaciones. Debido a que el número de modos de transporte en CU es menor que el número de modos considerados en el *Shortest Viable Hyperpath Problem* y además en el caso de CU se consideran tres turnos con diferentes frecuencias de *Pumabús* los procedimientos *Arc-Concatenation*, *h-arc-Concatenation* y *Determine-s*, son de la siguiente forma:

Procedimiento Arc−Concatenation CU(e, s_x, V^*, c, w)
{
wt=0;s=0;con=0
if $(turno = 1$ **or** $turno = 2)$ {
switch $(mode_e)$
case$(\in M1:)$
if $(s_x \neq 2)$ **call** $State{-}M1(s_x, s)$
break
case$(\in M3:)$
if $(s_x \neq 4)$ **call** $State{-}M3(s_x, s)$
break
case$(\in M6:)$
if$(w_{s_x}(j) < k)$ $wt = 1$; **call** $State{-}M4(s_x, s)$
break
default
if $(s_x = 0)s = 1$
else$\{s = s_x\}$
break
endswitch
}
else{
switch $(mode_e)$
case$(\in M1:)$
if $(s_x \neq 2)$ **call** $State{-}M1(s_x, s)$
break
case$(\in M3:)$
$s = 0$
break
case$(\in M6:)$
if$(w_{s_x}(j) < k)$ $wt = 1$; **call** $State{-}M4(s_x, s)$

```
      break
      default
        if (s_x = 0)s = 1
        else{s = s_x}
      break
      endswitch
  }
  if (s ≠ 0 and V*_{s_x}(j) + c(e) < V*_s(i)) call States(s, sM)
  else SM = ∅
  while (EOF(SM) = 0 y con ≠ 1) {
    select s_y from SM
    if (V*_{s_x}(j) + c(e) < V*_{s_y}(i)) {
      V*_s(i) = V*_{s_x}(j) + c(e)
      w_s(i) = w_{s_x}(j) + wt
      SA_s(i) = e
      ST_s(i) = s_x
      if (wt = 0 and [i, s] ∉ Qnow) Qnow = Qnow ∪ {[i, s]}
      if (wt = 1 and [i, s] ∉ Qnext) Qnext = Qnext ∪ {[i, s]}
      con = 1
    }
    }
}
```

Se observa que en el procedimiento *Arc-Concatenation* para CU se verifica el turno que se va a examinar. Cada uno de los turnos se representan con los números 1,2 y 3 dependiendo si se trata del turno matutino, vespertino o nocturno, respectivamente. En el turno 3 (el primer *else*) para el caso en $M3$ (modo bicicleta) el estado final es cero para asegurar que no pase el siguiente *if*, y de esta forma no se concatenen los arcos de bicicleta. Los procedimientos *State-*$M1(s_x, s)$, *State-*$M3(s_x, s)$ y *State-*$M4(s_x, s)$ se encargan de determinar el estado final a partir del estado inicial. El pseudo código de cada uno de estos estados es el siguiente:

```
Procedimiento State−M1(s_x, s)
{
  if (s_x = 0 or s_x = 3) s = 3
    else if (s_x = 1) s = 1
      else if (s_x = 4) s = 4
}

Procedimiento State−M3(s_x, s)
{
  if (s_x = 2) s = 2
```

else $s = 4$
}

Procedimiento $State{-}M3(s_x, s)$
{
switch s_x
case 0
$s = 1$
break
case 1
$s = 1$
break
case 2
$s = 4$
break
case 3
$s = 3$
break
case 4
$s = 4$
break
default
print Error
break
endswitch
}

Por último con el procedimiento *Determine-s CU* se determina cuál es el estado del híper camino resultante cuando dos híper caminos convergen en un mismo nodo.

Procedimiento Determine−s CU$(s, s_x, s_z(e))\}$
{
$sum = s_z(e) + s_x$
if $(s_z(e) = 1)$ $s = s_x$
else if $(s_x = 1)$ $s = s_z(e)$
else if $(sum = 6)$ $s = 3$
else if $(sum = 7$ **or** $8)$ 4
else $s = 4$
}

5.4. Código Java

La razones principales de programar el algoritmo en *Java* y no en un lenguaje de alto nivel como *C* fueron las siguientes: se tienen conocimientos previos sobre el lenguaje, lo que facilitó en gran medida su programación y por lo tanto la conclusión del trabajo en el tiempo previsto, y además no es necesario que el algoritmo arroje resultados inmediatos pues esta tesis no abarca la implementación del algoritmo para el público en general.

El código Java del algoritmo se divide en las clases de *HiperRutas*, *HiperCamino*, *Arco*, *Nodo* y *ParNodoEstado*. El código se escribió y compiló con *Eclipse* disponible en http://www.eclipse.org y se probó en una computadora con procesador *Intel Core 2 duo* con 2 GB de RAM y un sistema operativo *Windows Vista*. Dependiendo de la distancia entre los nodos y el número de transferencias modales, las híper rutas más cortas se calculan en máximo un minuto.

En seguida se da una breve explicación de cómo funcionan cada una de las clase y al final de documento en diferentes anexos se muestra el código *Java* del algoritmo.

La clase *HiperRutas* es la clase principal del algoritmo y es similar al procedimiento principal del *SVHP*. En esta clase se comparan los tiempos esperados de los híper caminos con su *last lable* y además se invocan las operaciones de concatenación de arco e híper arco. La información de los híper caminos que son candidatos al conjunto *Pareto Optimal* se guarda en forma de lista en la variable *losCaminosBuenos*. El código Java de esta clase se presenta en el Anexo A.

La clase *HiperCamino* construye híper caminos con las siguientes variables: tiempo esperado, *last lable*, número de transferencias máximas, arcos que la componen, entre otros. Además, en esta clase se decide cuándo es posible concatenar arcos e híper arcos y se determina el estado para algún híper camino que se esté probando. La clase contiene los procedimientos *Arc-Concatenation*, *h-Arc-Concatenation* y *Determine-s* del *SVHP*. El código Java de esta clase se presenta en el Anexo B.

La clase *Arco* construye arcos e híper arcos con las siguientes variables: frecuencia (matutina, vespertina y nocturna), tiempo de entrega, modo, *from Id* y *To id*, entre otras. El código Java de esta clase se presenta en el Anexo C.

Por otro lado, la clase *Nodo* construye nodos con un identificador[9], con una lista de todos los híper caminos que terminan en el nodo, con los arcos incidentes y con los estados asociados. Además esta clase se encarga de conectarse a la base de datos para obtener los identificadores de los arcos que inciden en el nodo. Estos arcos después se construirán en la clase *Arco*. También la clase *Nodo* verifica si los nodos pertenecen a *Qnow* o *Qnext*. El código Java de esta clase se presenta en el Anexo D.

Por último en la clase *ParNodoEstado* se crean parejas nodo-estado que son añadidas a *Qnow* o *Qnext* según sea el caso. El código Java de esta clase se presenta en el Anexo E.

[9]El algoritmo toma el identificador de la base de datos para construir un nodo. Este identificador corresponde al campo *FromId* o *ToId* de la tabla de *Arcos*

CAPÍTULO 6

Resultados

A continuación se presentan una serie de híper rutas más cortas multimodales que se obtuvieron con el algoritmo. El tiempo calculado por el algoritmo de las híper rutas se compara con el tiempo recolectado en trabajo de campo, para poder comprobar que los resultados que arroja el algoritmo son válidos y que la herramienta aquí propuesta es una buena guía para que el usuario.

La híper ruta de la Figura 6.1 va del Anexo de Ingeniería al estacionamiento E-3 del Estadio Olímpico Universitario y se calculó para el turno matutino. Esta híper ruta tiene un tiempo esperado de 14.95 minutos. Las posibilidades de recorrer la ruta son abordando la línea 6, 8 o 9. Si se aborda cualquiera de las primeras dos líneas éstas conducirán directamente al destino, en el caso de la línea 9 es necesario hacer transbordo en la estación Metrobús CU y ahí abordar las líneas 6, 8 o 11. En el caso de abordar la línea 11 es necesario hacer un transbordo en la estación E-7, en donde se aborda cualquiera de las líneas 6, 7 u 8 para llegar al destino. En trabajo de campo se recorrió la ruta abordando el *Pumabús* ruta 8 y se arribó al destino E-3 en 13.4 minutos, mientras que por el *Pumabús* ruta 6 se alcanzó un tiempo de 13.48 minutos.

Por otra parte, la ruta en forma inversa (ver Figura 6.2) no resulta tan eficiente en tiempo, ya que el *Pumabús* que conduce al Anexo de Ingeniería tiene que rodear primero la zona deportiva y la otra ruta resultante del conjunto *Pareto Optimal* es una combinación de bicicleta y recorrido a pie. Aunque el tiempo de ambas rutas es similar, 19.64 en *Pumabús* y 18.94 en bicicleta y a píe resulta interesante que es más rápido hacer el recorrido a pie y bicicleta que en *Pumabús*. Es una buena opción dar a conocer a los usuarios los tiempos aproximados tanto en *Pumabús* como en bicicleta para que el usuario decida si prefiere hacer una recorrido cómodo pero, tal vez, más lento a uno rápido pero que requiera de un esfuerzo extra. El tiempo en trabajo de campo de la ruta de *Pumabús* fue de 20 minutos y el tiempo de la ruta en bicicleta fue de 18.03 minutos.

La Figura 6.3 muestra un ejemplo en donde la bicicleta es la forma más rápida de trasladarse en distancias largas, más aún el conjunto *Pareto Optimal* no contiene resultados que incluyan el uso del *Pumabús*. Esta figura muestra una híper ruta del turno matutino, que se origina en la entrada

principal de Ciudad Universitaria y termina en la Facultad de Ciencias Políticas y Sociales. La primera ruta del conjunto *Pareto Optimal* tiene un tiempo esperado en bicicleta y a pie de 20.13 minutos, mientras que la segunda ruta tiene un tiempo esperado a pie de 34.5 minutos. Se observa que aunque existe un resultado del conjunto de soluciones en *Pumabús* de 25.14 minutos, éste no pertenece al conjunto *Pareto Optimal* ya que tarda más que el de bicicleta y tiene el mismo número de transferencias modales. En trabajo de campo se realizó un tiempo en bicicleta de 20.59 minutos, mientras que el recorrido a pie fue de 34.2 minutos.

Cuando el servicio de *Bicipuma* está cerrado, es decir, por la noche, la única forma de trasladarse es a pie o en *Pumabús*, lo que incrementa los tiempo de recorrido y por si esto fuera poco las frecuencias de *Pumabús* son más largas en el servicio nocturno, lo que puede llevar a casi duplicar los tiempos de recorrido con respecto de la mañana. Aunque en comparación la población universitaria disminuye en la noche, por experiencia personal se ha observado que ha aumentado la población vespertina de CU, por lo que cada vez se hace más necesario que el transporte sea eficiente a todas horas del día.

La Figura 6.4 muestra el mismo recorrido que la Figura 6.3, sólo que en el horario nocturno. Se observa que para llegar en *Pumabús* al destino es necesario hacer grandes recorridos a pie que no es lo más cómodo para el usuario. El conjunto *Pareto optimal* se compone por un recorrido totalmente a píe de 34.5 minutos y un recorrido combinado peatonal y de autobús de 30 minutos. Ambos recorridos se realizaron en trabajo de campo en donde la ruta totalmente a píe arrojo un resultado de 34.2 minutos mientras que la ruta combinada, utilizando la ruta 5 del *Pumabús*, dio un tiempo de 29.8 minutos.

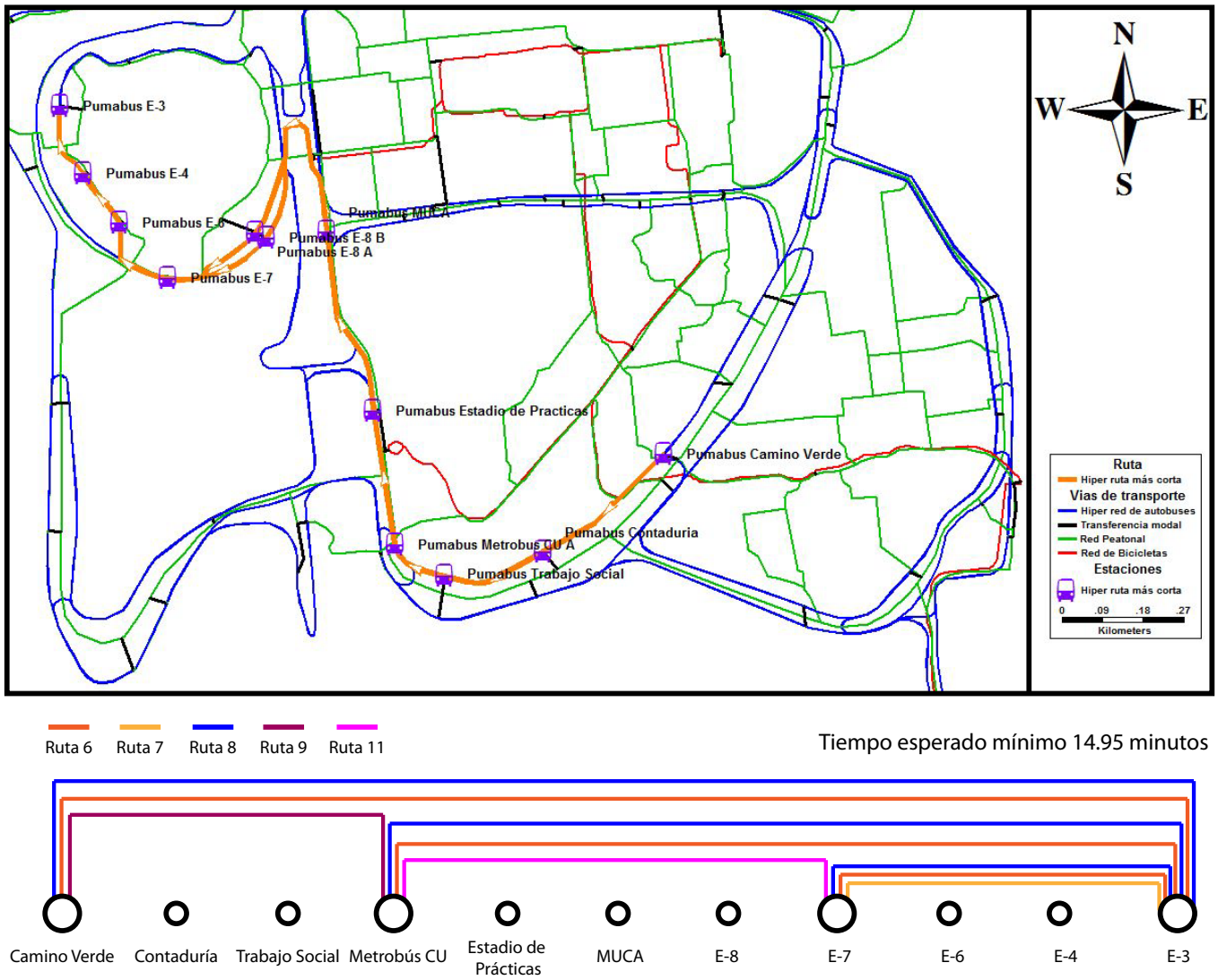
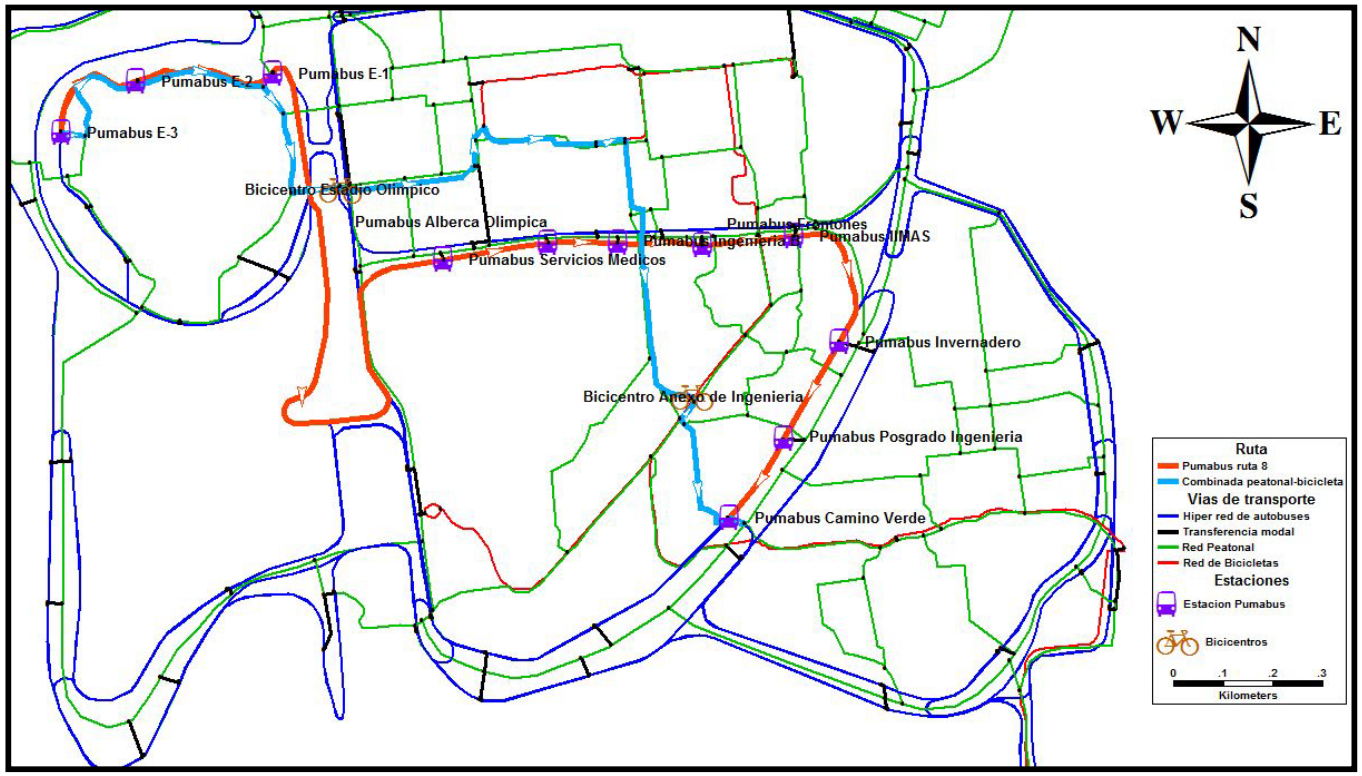


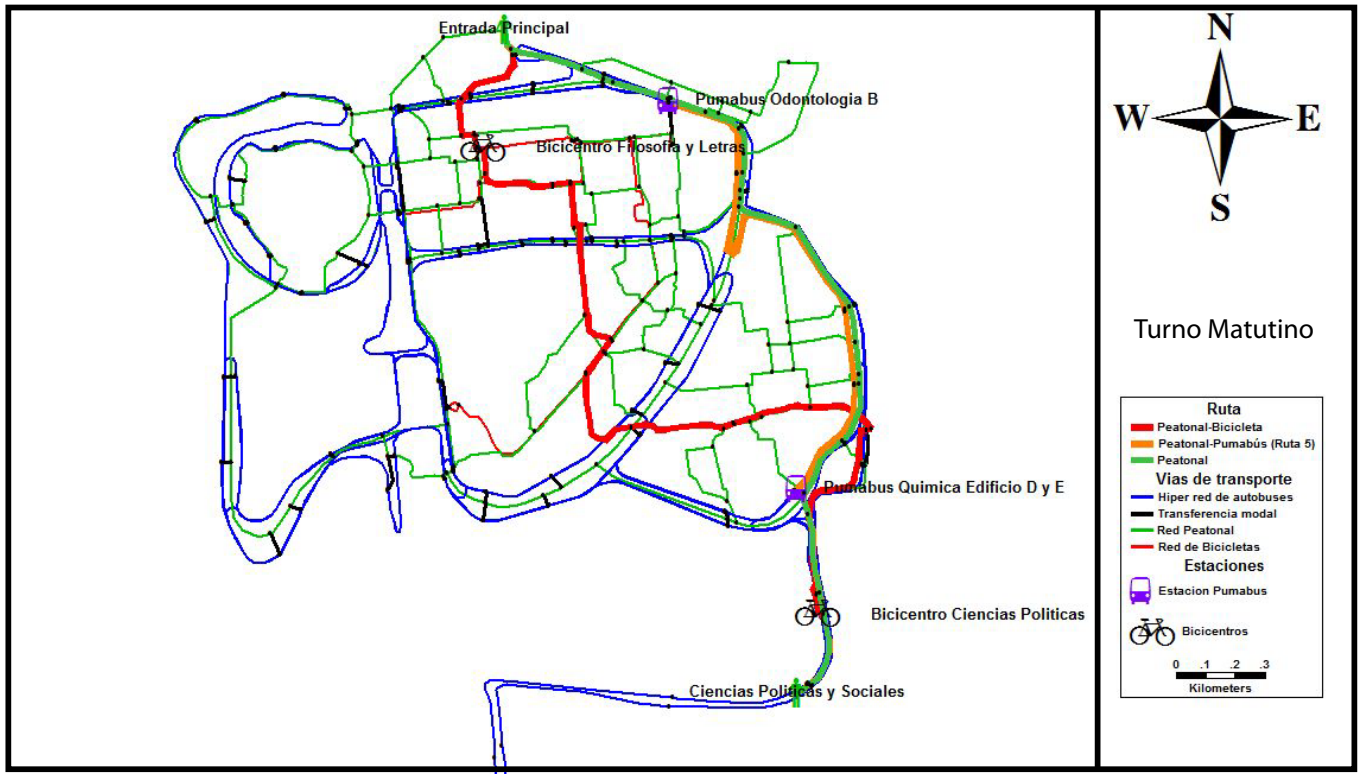
Figura 6.1. Híper ruta del Anexo de Ingeniería al Estadio Olímpico Universitario



Tiempo esperado mínimo en Pumabús 19.64 minutos y sin transferencias modales

Tiempo esperado mínimo en bicicleta y a pie 18.94 minutos con 2 tranferencias modales

Figura 6.2. Híper ruta del Estadio Olímpico Universitario al Anexo de Ingeniería

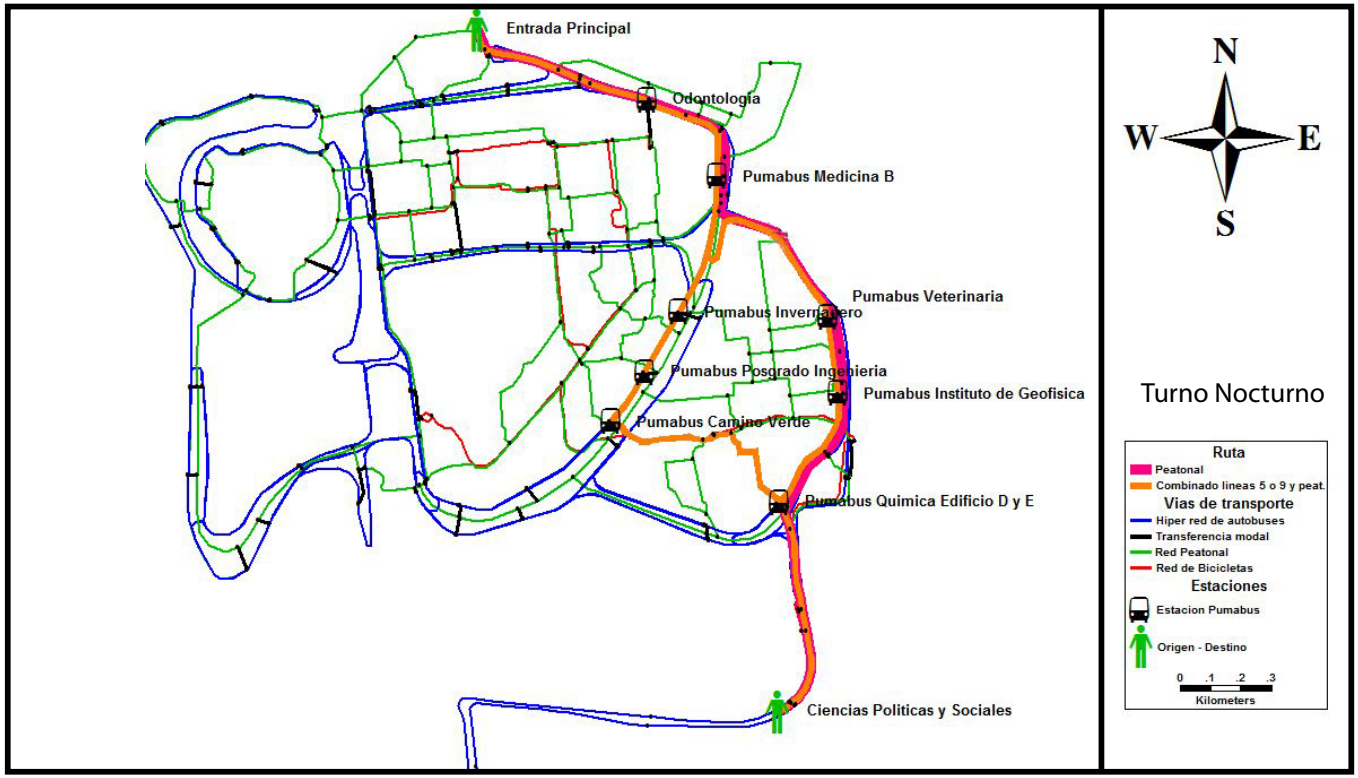


Tiempo esperado mínimo en ruta peatonal 34.5 minutos y sin transferencias modales

Tiempo esperado mínimo en bicicleta y a pie 20.13 minutos con 2 tranferencias modales

Tiempo esperado mínimo en Pumabús (Ruta 5) y a pie 25.4 minutos con 2 tranferencias modales

Figura 6.3. Híper ruta de la entrada principal a Ciudad Universitaria a la Facultad de Ciencias Políticas y Sociales



Tiempo esperado mínimo ruta peatonal 34.5 minutos

Tiempo esperado mínimo hiper ruta peatonal-autobús 30.6 minutos y dos transferencias modales

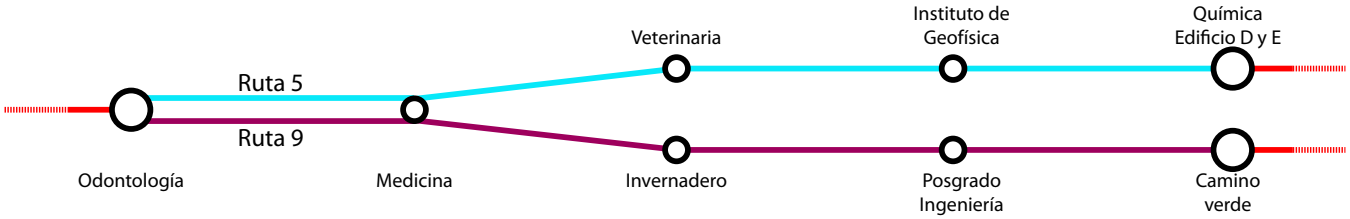


Figura 6.4. Híper ruta de la entrada principal a Ciudad Universitaria a la Facultad de Ciencias Políticas y Sociales en el turno nocturno

CAPÍTULO 7

Conclusiones

Se cumplió el objetivo principal de la tesis ya que la implementación del algoritmo de Lozano & Storchi (2002) para el caso de CU encuentra las rutas más rápidas dentro de CU si se considera que es posible trasladarse caminado, en autobús o en bicicleta. Además de que los resultados arrojados por el algoritmo se encuentran suficientemente cercanos a los tiempos de recorrido obtenidos en trabajo de campo. Encontrar resultados que sean exactos es difícil si se considera que los tiempos de espera en las paradas de autobús son inciertos. A continuación se presenta un serie de recomendaciones que se pueden hacer para mejora el transporte público, conclusiones acerca del algoritmo y los alcances de esta tesis.

Ciudad Universitaria es una zona con una gran densidad de población flotante, por lo que tener un sistema de transporte bien comunicado es de vital importancia. Se deben homogeneizar la frecuencia de los autobuses para evitar que éstos transiten con capacidades arriba de sus límites (un hecho altamente riesgoso) o que se encuentren vacíos. De ser posible sería excelente crear un itinerario de corridas basado en los niveles de demanda, para mejorar la calidad de viaje del usuario y el nivel de servicio del transporte universitario.

Se observó que en ocasiones se forman colas en los módulos de bicicletas y aunque esto no ocasiona demoras en el sistema, las filas pueden causar accidentes pues obstruyen el paso de los demás ciclistas. Como se mencionó en la Sección 3.2, los retrasos son ocasionados por usuarios que buscan su credencial, sin embargo, en algunos módulos no existe información alguna que explique el proceso de renta de bicicletas, por lo que sería recomendable poner carteles con dicha información e invitar a la comunidad universitaria a llegar preparada a los módulos de préstamo.

La creación de híper redes aunque es un trabajo laborioso no requiere de una gran diversidad de información, lo que lo hace una opción económica para implementarla en diversos sistemas de transporte. Además, actualmente existe una gran cantidad de *software* libre en donde se puede crear la híper red, cargar las bases de datos e implementar el algoritmo.

El algoritmo es suficientemente rápido si se considera que se está resolviendo un problema pseudo polinomial (a lo más un minuto), aunque este tiempo se puede disminuir con una computadora más potente y si se programa en un lenguaje de más alto nivel como *C*. Por otro lado, los resultados arrojados por el algoritmo son buenos ya que se encuentran muy cercanos a los obtenidos por trabajo de campo.

El trabajo que falta por hacer es la creación de un portal en Internet (funcional tanto para computadoras como para dispositivos móviles) para que el público en general tenga acceso a los resultados de la tesis. El portal mostrará un mapa de CU (similar al de *Google Maps*) en donde el usuario puede seleccionar su origen-destino y solicitar la búsqueda de las rutas más cortas. Entonces, se desplegará en el mapa el conjunto *Pareto-Optimal* de híper rutas más cortas, el tiempo de recorrido de cada una de las híper rutas, instrucciones escritas de los movimientos que se deben hacer y en el caso de que existan rutas en *Pumabús* se mostrará un mapa de las diferentes rutas que conducen al destino. La información mostrada en el mapa será similar a la mostrada en las figuras del capítulo 6. La arquitectura del portal puede ser alguna de las siguientes:

1. Se puede integrar el algoritmo al portal de manera que cada vez que se elija un origen y un destino, el algoritmo se ejecute y encuentre el resultado. Este sistema permite sin mayores complicaciones cambiar las características de la red o hacer ajustes a la base de datos para mantener actualizado el sistema. Sin embargo, para que el portal sea eficiente es necesario un servidor potente y además sería recomendable re programar el algoritmo en algún lenguaje de alto nivel para mejorar los tiempos de ejecución.

2. Se puede crear una base de datos entre los diferentes pares origen-destino, aproximadamente 25,000 entradas. Entonces, cuando se realiza una búsqueda de rutas más cortas el sistema consulta en la base de datos de origen-destino, los tiempos de recorrido y rutas, para después desplegar los resultados. Debido a que el portal funciona a base de búsquedas en bases de datos, los tiempos de espera necesarios para obtener los resultados son cortos. Sin embargo, si cambia la topología de la red cambiará en su totalidad la base de datos lo que implica un esfuerzo considerable si se desea mantener actualizada la información. Además, se requiere de una buena cantidad de memoria en el servidor para almacenar los datos.

3. Se puede hacer un sistema que resulte de la integración de los sistemas descritos en los puntos 1 y 2. Esto es, seleccionar un conjunto de los principales orígenes y destinos para crear una pequeña base de datos origen-destino, mientras que el resto de las rutas se calculan con el uso del algoritmo. Aunque este sistema parece una buena opción pues combina la rapidez de consulta en las bases de datos con la versatilidad del algoritmo, todavía es necesario hacer un análisis más concienzudo.

Actualmente muchas ciudades del mundo están haciendo fuertes inversiones y desarrollado tecnología para los Sistemas de Transporte Inteligentes (STI), este tipo de sistemas se valen de las telecomunicaciones y la informática para hacer reportes de tráfico, cobro de peajes, información en tiempo de real de ocupación de estacionamientos o información del servicio de transporte público. La herramienta propuesta en esta tesis, además ser una opción económica, es uno de los primeros avances en los STI que considero debe marcar el rumbo hacia donde se debe de dirigir la UNAM para que el transporte universitario sea de calidad y un ejemplo para el país.

Referencias

Ahuja, R., Magnanti, T. & Orlin, J. (1993), *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, United States.

Bang-Jensen, J. & Gutin, G. (2007), *Digraphs Theory, Algorithms and Applications*, Springer-Verlag, New York, chapter 2, pp. 45–58.

Berge, C. (1989), *Hypergraphs: Combinatorics of Finite Sets*, North-Holland, Amsterdam.

CalyMayor (2009), 'Semoforización intersecciones de paso peatonal', CalyMayor y Asociados. Presentación realizada a la U.N.A.M.

Diestel, R. (2005), *Graph Theory*, Springer-Verlag, New York.

Lozano, A. & Storchi, G. (2001), 'Shortest viable path in multimodal networks', *Transportation Research Part A* **35**, 225–241.

Lozano, A. & Storchi, G. (2002), 'Shortest viable hyperpath in multimodal networks', *Transportation Research Part B* **36**, 853–874.

Miller-Hooks, E. (1997), Optimal Routing in Time-Varying, Stochastic Networks: Algorithms and Implementations, PhD thesis, University of Texas at Austin.

Modestia, P. & Sciomachen, A. (1998), 'A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks', *European Journal of Operational Research* **111**, 495–508.

Nguyen, S., Pallotino, S. & Gendreau, M. (1998), 'Implicit enumeration of hyperpaths in a logit model for transit networks', *Transportation Science* **32**, 54–64.

Nie, Y. & Wu, X. (2009), 'Shortest path problem considering on-time arrival probability', *Transportation Research Part B* . doi:10.1016/j.trb.2009.01.008.

Nielsen, L., Andersen, K. & Pretolani, D. (2003), 'Bicriterion shortest hyperpaths in random time-dependent networks', *IMA Journal of Management Mathematics* **14**, 271–303.

Pallottino, S. & Schettino, A. (1999), *Il trasporto collettivo urbano*, Fanco Angeli, Milano, chapter VII, pp. 619–655.

Pallottino, S. & Scutellà, M. (1998), *Equilibrium and Advanced Transportation Modelling*, Kluwer Academic Publishers, Dordrecht, chapter Shortest path algorithms in transportation models: classical and innovative aspect, pp. 240–282.

Pallottino, S. & Scutellà, M. (2003), 'A new algorithm for reoptimizing shortest paths when the arc costs change', *Operations Research Letters* **31**, 149–160.

Pretolani, D. (2000), 'A directed hypergraph model for random time dependent shortest paths', *European journal of operational research* **123**, 315–324.

UNAM (2009*a*), 'Agenda estadistica 2009', Dirección General de Planeación. Obtenido el 18 de septiembre del 2009 en http://www.estadistica.unam.mx/agenda/agendas/2008/disco/index2.php.

UNAM (2009*b*), 'Campus central de la ciudad universitaria de la unam', Universidad Nacional Autónoma de México. Obtenido el 14 de septiembre del 2009 en http://www.unam.mx/patrimonio/.

UNAM (2009*c*), 'Servicio de transporte alternativo', Dirección General de Atención a la Comunidad Universitaria. Obtenido el 31 de agosto del 2009 en http://www.tucomunidad.unam.mx/Bicipuma/.

UNAM (2009*d*), 'Sistema de transporte interno pumabús', Dirección General de Servicios Generales. Obtenido el 31 de agosto del 2009 en http://www.pumabus.unam.mx/.

UNESCO (2009), 'Central university city campus of the universidad nacional autónoma de méxico', United Nations Educational, Scientific and Cultural Organization. Obtenido el 14 de septiembre del 2009 en http://whc.unesco.org/en/list/1250.

Voloshin, V. I. (2009), *Introduction to Graph and Hypergraph Theory*, Nova Science Publishers, United States.

Wachs, M., Samuels, J. M. & Skinner, R. E. (2000), *Highway Capacity Manual*, Transportation Research Board, United Satates of America.

ANEXO A

Clase Hiper Rutas

El código *Java* de todas la clases tiene la siguiente estructura de colores: en color rosa se muestran los constructores del algoritmo, en color azul se muestran las variables del algoritmo y en color verde se muestran los comentarios de las líneas sucesivas del código.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class HiperRutas {
  static List<ParNodoEstado> qNow = new ArrayList<ParNodoEstado>();
  static List<ParNodoEstado> qNext = new ArrayList<ParNodoEstado>();
  static int idNodoDestino = //número introducido por el usuario;
  static int idNodoOrigen = //número introducido por el usuario;
  static final int transferenciasMaximas = //número introducido por el usuario;
  static int turno = //número introducido por el usuario;
  static int contadorTransferencias;
  static List<Integer> nodosVisitados = new ArrayList<Integer>();
  static List<HiperCamino> losCaminosBuenos = new ArrayList<HiperCamino>();

  public static void main(String[] args) {
    Nodo nodoFinal = new Nodo(idNodoDestino);
    ParNodoEstado primerQNow = new ParNodoEstado(nodoFinal,0);
    HiperCamino primerHiperCamino = new HiperCamino(nodoFinal, 0);
    primerHiperCamino.setTiempoEsperado(0);
    //Se añaden los elementos sus respectivas lista para que pasen entren al while
    HiperRutas.nodosVisitados.add(nodoFinal.getnId());
    HiperRutas.losCaminosBuenos.add(primerHiperCamino);
    qNow.add(primerQNow);
    while(contadorTransferencias <= transferenciasMaximas && qNow.isEmpty() ==
        false){
      Iterator<ParNodoEstado> iteradorqNow = qNow.iterator();
      while(iteradorqNow.hasNext()){
          Nodo nodoEvaluando = qNow.get(0).getEsteNodo();
          int estadoEvaluando = qNow.get(0).getEsteEstado();
          qNow.remove(0);
```

```
            HiperCamino hiperCaminoProbar =
            nodoEvaluando.dameHiperCaminoEntrante(estadoEvaluando);
            if (hiperCaminoProbar.getTiempoEsperado() < hiperCaminoProbar.
                getLastLabel()){
              hiperCaminoProbar.setLastLabel(hiperCaminoProbar.getTiempoEsperado());
              nodoEvaluando.obtenArcos();
              List<Arco> misArcosEntrantes = new ArrayList<Arco>();
              misArcosEntrantes = nodoEvaluando.getArcosEntrantes();
              for(Arco arcoConcatenar : misArcosEntrantes ){
                  HiperCamino hiperCaminoConcatenar =
                  nodoEvaluando.dameHiperCaminoEntrante(estadoEvaluando);
                  if(arcoConcatenar.getSoyHiperArco() == 0){
                    hiperCaminoConcatenar.concatenameArco(arcoConcatenar);
                  }else{
                    hiperCaminoConcatenar.concatenameHiperArco(arcoConcatenar);
                  }
                }
              }
        }
        Iterator<ParNodoEstado> itVaciaQnext = qNext.iterator();
        while(itVaciaQnext.hasNext()){
          ParNodoEstado e = itVaciaQnext.next();
          qNow.add(e);
        }
        qNext.clear();
        contadorTransferencias = contadorTransferencias + 1;
      }

      //El siguiente segmento de código imprime los resultados en pantalla para que
          puedan se copiados en pgAdmin III y después visualizado en algún SIG
      int k = 0;
      for(HiperCamino i : losCaminosBuenos){
        if(i.getIdNodoFinal() == idNodoOrigen){
          System.out.println("\n Tiempo esperado de la ruta " + k + " es " + i.
              getTiempoEsperado()
           + " transferencias " + i.getTransferencias() + ", estados: " + i.
               getEstadosQueLlegaron()
           + ", estado Final " + i.getEstadoHiperCamino());
           k = k + 1;
        }
      }
        int j = 0;
        for(HiperCamino i : losCaminosBuenos){
          if(i.getIdNodoFinal() == idNodoOrigen){
            System.out.print("\nDROP TABLE IF EXISTS ruta" + j + ";\nCREATE TABLE
                ruta" +
            j + " AS SELECT * FROM arcos_copia WHERE ");
            j = j + 1;
            for (Arco a : i.getArcosHiperCamino()){
              System.out.print("id =" + a.getId() + " or ");
            }
          }
        }
    }
}
```

ANEXO B

Clase HiperCamino

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

//Cada una de las varibles de la clase puede ser accesada por las demás clase
    mediante getters y setters
public class HiperCamino {
  private double tiempoEsperado = Double.POSITIVE_INFINITY;
  private double tiempoEsperadoMinimoUsandoHiperArco = Double.POSITIVE_INFINITY;
  private int transferencias;
  private double lastLabel = Double.POSITIVE_INFINITY;
  private List<Arco> arcosHiperCamino = new ArrayList<Arco>();
  private int estadoHiperCamino;
  private List<Integer> estadosQueLlegaron = new ArrayList<Integer>();
  private Arco arcoSucesivo = new Arco();
  private int idNodoFinal;

  public HiperCamino() {
  }

  public HiperCamino(Nodo unNodo, int unEstado) {
    setEstadoHiperCamino(unEstado);
    setIdNodoFinal(unNodo.getnId());
  }

  public HiperCamino(Nodo unNodo) {
    if (unNodo.getnId() == HiperRutas.idNodoDestino) {
    }
  }

  public HiperCamino(Arco unArco, HiperCamino unHiperCamino) {
  }
  // Este es el constructor que se encarga de hacer la copia del hipercamino
  private HiperCamino(HiperCamino caminoCopiar) {
```

```
    this.setTiempoEsperado(caminoCopiar.getTiempoEsperado());
    this.tiempoEsperadoMinimoUsandoHiperArco = caminoCopiar.
        tiempoEsperadoMinimoUsandoHiperArco;
    this.setTransferencias(caminoCopiar.getTransferencias());
    this.setLastLabel(caminoCopiar.getLastLabel());
    this.setEstadoHiperCamino(caminoCopiar.getEstadoHiperCamino());
    this.setIdNodoFinal(caminoCopiar.getIdNodoFinal());
    if (caminoCopiar.getArcoSucesivo() != null){
      this.setArcoSucesivo(caminoCopiar.getArcoSucesivo().getCopy());
    }
    for(int l : caminoCopiar.getEstadosQueLlegaron()){
      this.getEstadosQueLlegaron().add(l);
    }
    for (Arco a : caminoCopiar.arcosHiperCamino){
       this.arcosHiperCamino.add(a.getCopy());
    }
  }

  //Estos son los métodos que permiten accesar (get) y modificar (set) los campos
      de la clase
  public void setTiempoEsperado(double tiempoEsperado) {
    this.tiempoEsperado = tiempoEsperado;
  }

  public double getTiempoEsperado() {
    return tiempoEsperado;
  }

  public void setLastLabel(double lastLabel) {
    this.lastLabel = lastLabel;
  }

  public double getLastLabel() {
    return lastLabel;
  }

  public void setArcosHiperCamino (List<Arco> arcosHiperCamino){
    this.arcosHiperCamino = arcosHiperCamino;
  }

  public List<Arco> getArcosHiperCamino(){
    return arcosHiperCamino;
  }

  public void setEstadosQueLlegaron(List<Integer> estadosQueLlegaron) {
    this.estadosQueLlegaron = estadosQueLlegaron;
  }

  public List<Integer> getEstadosQueLlegaron() {
    return estadosQueLlegaron;
  }

  public void setIdNodoFinal(int idNodoFinal) {
    this.idNodoFinal = idNodoFinal;
  }
```

```
 public int getIdNodoFinal() {
   return idNodoFinal;
 }

 public void setEstadoHiperCamino(int estadoHiperCamino) {
   this.estadoHiperCamino = estadoHiperCamino;
 }

 public int getEstadoHiperCamino() {
   return estadoHiperCamino;
 }

 public void setArcoSucesivo(Arco arcoSucesivo) {
   this.arcoSucesivo = arcoSucesivo;
 }

 public Arco getArcoSucesivo() {
   return arcoSucesivo;
 }

 public void setTransferencias(int transferencias) {
   this.transferencias = transferencias;
 }

 public int getTransferencias() {
   return transferencias;
 }

 //Constructor que crea una copia del hiper camino que se indique
 public HiperCamino getCopy() {
   return new HiperCamino(this);
 }

       //Operación que pega un arco al final del algún hiper camino. Se observa
           que last lable se vuelve a definir púes en teoría se construyó un nuevo
            hiper camino
 public void uneArco(Arco unArco) {
   this.arcosHiperCamino.add(unArco);
   this.setIdNodoFinal(unArco.getFromId());
   this.setLastLabel(Double.POSITIVE_INFINITY);
 }

//Esta operación corresponde al procedimiento Arc-Concatenation
public void concatenameArco(Arco unArco) {

   int cuentaTransferencias = 0;
   int estadoFinal = 0;
   int contador = 0;
   List<Integer> estadosPreferidos = new ArrayList<Integer>();
   Nodo unNodoFinal = new Nodo(unArco.getFromId());
   int estadoActual = this.getEstadoHiperCamino();
   HiperCamino unHiperCaminoConcatenado = this.getCopy();  //Copia el elemento
       para no sobre escribirlo

   if(HiperRutas.turno == 1 || HiperRutas.turno == 2){
     switch (unArco.getModo()) {
```

```
case 1:
  if (estadoActual == 0 || estadoActual == 3){
    estadoFinal = 3;
    break;
  }else if(estadoActual == 1){
    estadoFinal = 1;
    break;
  }else if(estadoActual == 4){
   estadoFinal = 4;
  }break;
case 3:
   if (estadoActual == 2) {
    estadoFinal = 2;
    break;
  }else if(estadoActual == 1 || estadoActual == 0 || estadoActual == 3){
    estadoFinal = 2;
    //Se agrega el tiempo de entrega de bicicleta al tiempo esperado, pues la
         acción representa la de tomar una bicicleta
    unHiperCaminoConcatenado.setTiempoEsperado(this.getTiempoEsperado() +
    unArco.getTiempoEntrega());
    break;
  }break;
case 4:
  if (unHiperCaminoConcatenado.getTransferencias() <
  HiperRutas.transferenciasMaximas) {
    cuentaTransferencias = 1;
    switch (estadoActual) {
    case 0:
      estadoFinal = 1;
      break;
    case 1:
      estadoFinal = 1;
      break;
    case 2:
      estadoFinal = 4;
      //Se agrega el tiempo de recolección de bicicleta al tiempo esperado,
          pues la acción representa la de entregar una bicicleta
      unHiperCaminoConcatenado.setTiempoEsperado(this.getTiempoEsperado() +
      unArco.getTiempoRecoleccion());
      break;
    case 3:
      estadoFinal = 3;
      break;
    case 4:
      estadoFinal = 4;
      break;
    default:
      System.out.println("Error Estado Actual: " + estadoActual);
      System.out.print("Error Estado Final: " + estadoFinal);
      break;
   }
  }break;
 default:
  if (estadoActual == 0) {
    estadoFinal = 1;
    break;
```

```
    }
    estadoFinal = estadoActual;
    break;
  }
}else{
  switch (unArco.getModo()) {
  case 1:
    if (estadoActual == 0 || estadoActual == 3){
      estadoFinal = 3;
      break;
    }else{
      estadoFinal = 1;
    }break;
   case 3:
   //Como se está analizando el turno nocturno y este no permite el uso de
       bicicletas, se asigna el cero a estadoFinal para evitar que se forme el
       hiper camino en el siguiente if.
    estadoFinal = 0;
    break;
   case 4:
    if (unHiperCaminoConcatenado.getTransferencias() <
    HiperRutas.transferenciasMaximas) {
      cuentaTransferencias = 1;
      switch (estadoActual) {
      case 0:
        estadoFinal = 1;
        break;
      case 1:
        estadoFinal = 1;
        break;
      case 3:
        estadoFinal = 3;
        break;
      default:
        System.out.println("Error Estado Actual: " + estadoActual);
        System.out.print("Error Estado Final: " + estadoFinal);
        break;
      }
   }break;
  default:
    if (estadoActual == 0) {
       estadoFinal = 1;
    break;
    }
    estadoFinal = estadoActual;
    break;
 }
}

//Se realiza este check para impedir que se concatene un arco de transferencia
    con un hipercamino cuyo último arco es de transferencia pues no tiene
    sentido práctico tener dos arcos de transferencia seguidos.
if(unHiperCaminoConcatenado.getArcoSucesivo().getModo() == 4 &&
unArco.getModo() == 4){
  estadoFinal = 0;
}
```

```
HiperCamino hiperCaminoEntrante = unNodoFinal.dameHiperCaminoEntrante(
    estadoFinal);
if (estadoFinal != 0 && unHiperCaminoConcatenado.getTiempoEsperado() +
unArco.getTiempoConElFlujo() < hiperCaminoEntrante.getTiempoEsperado()) {
   estadosPreferidos = dameEstadosPreferidos(estadoFinal);
}else{
  estadosPreferidos.clear();
}

Iterator<Integer> itEstados = estadosPreferidos.iterator();
while (itEstados.hasNext() && contador != 1) {
  int e = itEstados.next();
  HiperCamino algunHiperCaminoEntrante = unNodoFinal.dameHiperCaminoEntrante(e)
      ;
  if (unHiperCaminoConcatenado.getTiempoEsperado() + unArco.getTiempoConElFlujo
      () <
  algunHiperCaminoEntrante.getTiempoEsperado()) {
  //El chec impide que se cuente como transferencia si se inicia el recorrido
      en un arco de transferencia (pues técnicamente te encuentras en el lugar
      indicado) y además sólo se contará como transferencia una vez que se
      abordó un arco de transporte
    if(this.getArcoSucesivo().getModo() == 4 && this.getArcosHiperCamino().size
        () > 1){
      unHiperCaminoConcatenado.setTransferencias(this.getTransferencias() + 1)
          ;
    }else{
      unHiperCaminoConcatenado.setTransferencias(this.getTransferencias());
    }
    unHiperCaminoConcatenado.uneArco(unArco);
    unHiperCaminoConcatenado.setEstadoHiperCamino(estadoFinal);
    unHiperCaminoConcatenado.setTiempoEsperado(unHiperCaminoConcatenado.
        getTiempoEsperado() +
    unArco.getTiempoConElFlujo());
    unHiperCaminoConcatenado.setArcoSucesivo(unArco);
    unHiperCaminoConcatenado.setIdNodoFinal(unArco.getFromId());
    unHiperCaminoConcatenado.getEstadosQueLlegaron().add(estadoActual);
    //Se agrega la siguiente condición ya que si unHiperCaminoConcatenado es
        mejor que algunHiperCaminoEntrante y ambos tiene el mismo estado,
        algunHiperCaminoEntrante debe ser removido de los caminos buenos pues
        se encontró uno mejor.
    if(unHiperCaminoConcatenado.getEstadoHiperCamino() ==
    algunHiperCaminoEntrante.getEstadoHiperCamino()
       && algunHiperCaminoEntrante.getTransferencias() ==
       unHiperCaminoConcatenado.getTransferencias()){
      HiperRutas.losCaminosBuenos.remove(algunHiperCaminoEntrante);
    }
    HiperRutas.nodosVisitados.add(unNodoFinal.getnId());
    //Se verifica la siguiente para que pase la primera vez
    if(HiperRutas.losCaminosBuenos.isEmpty()){
     HiperRutas.losCaminosBuenos.add(unHiperCaminoConcatenado);
    }
    HiperRutas.losCaminosBuenos.add(0,unHiperCaminoConcatenado);
    if(unNodoFinal.getnId() != HiperRutas.idNodoOrigen && unNodoFinal.getnId()
        !=
    HiperRutas.idNodoDestino){
```

```
        if (cuentaTransferencias == 0 && unNodoFinal.estoyEnQnow(estadoFinal) ==
            false) {
          ParNodoEstado nuevoElementoQnow = new ParNodoEstado(unNodoFinal,
              estadoFinal);
          HiperRutas.qNow.add(nuevoElementoQnow);
        }
        if (cuentaTransferencias == 1 && unNodoFinal.estoyEnQnext(estadoFinal) ==
            false) {
          ParNodoEstado nuevoElementoQnext = new ParNodoEstado(unNodoFinal,
              estadoFinal);
          HiperRutas.qNext.add(nuevoElementoQnext);
        }
        }
        contador = 1;
      }
    }
  }

  //Esta operación corresponde a h-Arc-Concatenation
  public void concatenameHiperArco(Arco unArco) {
    int contador = 0;
    int estadoFinal = 0;
    List<Integer> estadosPreferidos = new ArrayList<Integer>();
    int estadoActual = this.estadoHiperCamino;
    Nodo unNodoFinal = new Nodo(unArco.getFromId());
    HiperCamino unHiperCaminoConcatenado = this.getCopy();
    List<HiperCamino> hiperCaminoSoporte = new ArrayList<HiperCamino>();
    HiperCamino entranteDeAutobus = unNodoFinal.dameHiperCaminoEntranteDeAutobuses
        ();

   //El ciclo 'for' comprueba si ya se utilizó algún hiper arco del hiper arco de
       soporte para así encontrar el resto de los arcos de soporte
   for(int i :unArco.getIdArcoSoporte()){
     for(HiperCamino h: HiperRutas.losCaminosBuenos){
       if(h.getArcoSucesivo().getId() == i){
         hiperCaminoSoporte.add(h);
       }
     }
    }

    if(unArco.getFrecuenciaCombinada() != 0){
       estadoFinal = dameHiperEstadoFinal(estadoActual, entranteDeAutobus.
           getEstadoHiperCamino());
    }else{
       estadoFinal = estadoActual;
    }

    HiperCamino hiperCaminoEntrante = unNodoFinal.dameHiperCaminoEntrante(
        estadoFinal);
    if(this.getTiempoEsperado() < hiperCaminoEntrante.getTiempoEsperado() ){
      //Si ya se utlizó el hiper arco en algún otro hiper camino este chec
          actualiza las etiquetas de los hiper arcos que inciden en el mismo
          vértice
      if(hiperCaminoSoporte != null ){
        for(HiperCamino h: hiperCaminoSoporte){
          if(h.getArcoSucesivo().getId() != unArco.getId()){
```

```
            unArco.setFrecuenciaCombinada(h.getArcoSucesivo().
                getFrecuenciaCombinada());
            unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
            h.tiempoEsperadoMinimoUsandoHiperArco;
          }else{
             unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
             h.tiempoEsperadoMinimoUsandoHiperArco;
          }
        }
    }

     //Se hace un 'switch' para que se consideren las diferentes frecuencias en
         turnos diferentes
     switch (HiperRutas.turno) {
     case 1:
       unArco.setFrecuenciaCombinada(unArco.getFrecuenciaCombinada() +
       unArco.getFrecuenciaMatutina());
       unArco.getNodosCabeza().add(unNodoFinal);
       unHiperCaminoConcatenado.getEstadosQueLlegaron().add(estadoActual);
       if(entranteDeAutobus != null){
          entranteDeAutobus.setEstadoHiperCamino(estadoFinal);
       }
       if(unArco.getFrecuenciaCombinada() == unArco.getFrecuenciaMatutina()){
         unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
         (60/unArco.getFrecuenciaMatutina()) + this.getTiempoEsperado();
       }else{
         unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
         unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco -
         (((unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco -
         this.getTiempoEsperado())*unArco.getFrecuenciaMatutina())/
         unArco.getFrecuenciaCombinada());
       }
       break;
    case 2
       unArco.setFrecuenciaCombinada(unArco.getFrecuenciaCombinada() +
       unArco.getFrecuenciaVespertina());
       unArco.getNodosCabeza().add(unNodoFinal);
       unHiperCaminoConcatenado.getEstadosQueLlegaron().add(estadoActual);

       if(entranteDeAutobus != null){
          entranteDeAutobus.setEstadoHiperCamino(estadoFinal);
       }
     if(unArco.getFrecuenciaCombinada() == unArco.getFrecuenciaVespertina()){
         unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
         (60/unArco.getFrecuenciaVespertina()) + this.getTiempoEsperado();
     }else{
         unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
         unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco -
         (((unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco -
         this.getTiempoEsperado())*unArco.getFrecuenciaVespertina())/
         unArco.getFrecuenciaCombinada());
       }
       break;
    case 3:
       unArco.setFrecuenciaCombinada(unArco.getFrecuenciaCombinada() +
       unArco.getFrecuenciaNocturna());
```

```
    unArco.getNodosCabeza().add(unNodoFinal);
    unHiperCaminoConcatenado.getEstadosQueLlegaron().add(estadoActual);
    if(entranteDeAutobus != null){
       entranteDeAutobus.setEstadoHiperCamino(estadoFinal);
    }
    if(unArco.getFrecuenciaCombinada() == unArco.getFrecuenciaNocturna()){
       unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
       (60/unArco.getFrecuenciaNocturna()) + this.getTiempoEsperado();
    }else{
      unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco =
      unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco -
      (((unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco -
      this.getTiempoEsperado())*unArco.getFrecuenciaNocturna())/
      unArco.getFrecuenciaCombinada());
    }
    break;
  default:
    System.out.print("Error al introducir horario");
    break;
  }
 if(unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco <
hiperCaminoEntrante.getTiempoEsperado()){
    estadosPreferidos = dameEstadosPreferidos(estadoFinal);
 } else {
    estadosPreferidos.clear();
 }
  Iterator<Integer> itEstados = estadosPreferidos.iterator();
  while (itEstados.hasNext() && contador != 1) {
    int e = itEstados.next();
    HiperCamino algunHiperCaminoEntrante = unNodoFinal.dameHiperCaminoEntrante(
        e);
    if(unHiperCaminoConcatenado.tiempoEsperadoMinimoUsandoHiperArco <
    algunHiperCaminoEntrante.getTiempoEsperado()){
      unHiperCaminoConcatenado.uneArco(unArco);
       //Se añaden arcos al hiper camino que pertenezcan pero que no estén en
           él
       for(HiperCamino j :hiperCaminoSoporte){
          for(Arco i: j.getArcosHiperCamino()){
             if(unHiperCaminoConcatenado.getArcosHiperCamino().indexOf(i) ==
             -1 && i.getModo() == 1){
               unHiperCaminoConcatenado.getArcosHiperCamino().add(i);
             }
          }
      }
      unHiperCaminoConcatenado.setEstadoHiperCamino(estadoFinal);
      unHiperCaminoConcatenado.setTiempoEsperado(unHiperCaminoConcatenado.
                                                    tiempoEsperadoMinimoUsandoHiperArco
                                                    );
      unHiperCaminoConcatenado.setTransferencias(this.getTransferencias());
      unHiperCaminoConcatenado.setArcoSucesivo(unArco);
      unHiperCaminoConcatenado.getEstadosQueLlegaron().add(estadoActual);
      HiperRutas.nodosVisitados.add(unNodoFinal.getnId());
      //La operacion permite que pase la primera vez
      if(HiperRutas.losCaminosBuenos.isEmpty()){
         HiperRutas.losCaminosBuenos.add(unHiperCaminoConcatenado);
      }
```

```
          HiperRutas.losCaminosBuenos.add(0,unHiperCaminoConcatenado);
          if(unNodoFinal.getnId() != HiperRutas.idNodoOrigen){
          if (unNodoFinal.estoyEnQnow(estadoFinal) == false) {
            ParNodoEstado nuevoElementoQnow = new ParNodoEstado(unNodoFinal,
                estadoFinal);
            HiperRutas.qNow.add(nuevoElementoQnow);
          }
          }
          contador = 1;
        }
    }
  }
}

      //Esta operación calcula los estados preferidos del estado indicado
public static List<Integer> dameEstadosPreferidos(int estado) {
  final List<Integer> losEstadoPreferidos = new ArrayList<Integer>();
  switch (estado) {
  case 1:
    losEstadoPreferidos.add(1);
    break;
  case 2:
    losEstadoPreferidos.add(2);
    losEstadoPreferidos.add(1);
    losEstadoPreferidos.add(3);
    break;
  case 3:
    losEstadoPreferidos.add(3);
    break;
  case 4:
    losEstadoPreferidos.add(4);
    losEstadoPreferidos.add(1);
    losEstadoPreferidos.add(3);
    losEstadoPreferidos.add(2);
    break;
  default:
    System.out.print("Error añadiendo estados ");
    break;
  }
  return losEstadoPreferidos;
}

//Esta operación corresponde a Determine-s
public int dameHiperEstadoFinal(int estadoActual,
int estadoHiperCaminoEntranteQueUsaHiperArco){
  int suma = estadoActual + estadoHiperCaminoEntranteQueUsaHiperArco;
  int estadoFinal;
  if(estadoHiperCaminoEntranteQueUsaHiperArco == 1){
    return estadoFinal = estadoActual;
  }else if (estadoActual == 1) {
     return estadoFinal = estadoHiperCaminoEntranteQueUsaHiperArco;
  }else if(suma == 6){
    return estadoFinal = 3;
  }else if(suma == 7 || suma == 8){
    return estadoFinal = 4;
  }
```

```
    return estadoFinal = 4;
  }
}
```

ANEXO C

Clase Arco

```java
import java.util.List;

public class Arco {
  private int id;
  private int soyHiperArco;
  private double tiempoConElFlujo;
  private double tiempoAbordaje;
  private double frecuenciaMatutina;
  private double frecuenciaVespertina;
  private double frecuenciaNocturna;
  private double tiempoRecoleccion;
  private double tiempoEntrega;
  private int modo;
  private int fromId;
  private int toId;
  private double frecuenciaCombinada = 0.0;
  private double costoCalculado = Double.POSITIVE_INFINITY;
  private List<Nodo> nodosCabeza = new ArrayList<Nodo>();
  private int ruta;
  private List<Integer> idArcoSoporte = new ArrayList<Integer>();

 // Este es el constructor que se encarga de hacer la copia del hipercamino
 private Arco(Arco arcoCopiar) {
    this.setId(arcoCopiar.getId());
    this.setSoyHiperArco(arcoCopiar.getSoyHiperArco());
    this.setTiempoConElFlujo(arcoCopiar.getTiempoConElFlujo());
    this.setTiempoAbordaje(arcoCopiar.getTiempoAbordaje());
    this.setFrecuenciaMatutina(arcoCopiar.getFrecuenciaMatutina());
    this.setFrecuenciaVespertina(arcoCopiar.getFrecuenciaVespertina());
    this.setFrecuenciaNocturna(arcoCopiar.getFrecuenciaNocturna());
    this.setTiempoRecoleccion(arcoCopiar.getTiempoRecoleccion());
    this.setTiempoEntrega(arcoCopiar.getTiempoEntrega());
    this.setModo(arcoCopiar.getModo());
    this.setFromId(arcoCopiar.getFromId());
```

```
    this.setToId(arcoCopiar.getToId());
    this.setFrecuenciaCombinada(arcoCopiar.getFrecuenciaCombinada());
    this.setCostoCalculado(arcoCopiar.getCostoCalculado());
    for (Nodo n : arcoCopiar.getNodosCabeza()){
      this.nodosCabeza.add(n.getCopy());
    }
    this.setRuta(arcoCopiar.getRuta());
    for (int i : arcoCopiar.getIdArcoSoporte()){
        this.getIdArcoSoporte().add(i);
    }

  }

  public Arco(int arcoID, int yoSoyHiperarco, double miTiempoConElFlujo,
   double miTiempoAbordaje, double miFrecuenciaMatutina,
  double miFrecuenciaVespertina, double miFrecuenciaNocturna,
   double miTiempoRecoleccion, double miTiempoEntrega, int miModo,
  int miFromId, int miToId, int miRuta, List<Integer> l) {
    setId(arcoID);
    setSoyHiperArco(yoSoyHiperarco);
    setTiempoConElFlujo(miTiempoConElFlujo);
    setTiempoAbordaje(miTiempoAbordaje);
    setFrecuenciaMatutina(miFrecuenciaMatutina);
    setFrecuenciaVespertina(miFrecuenciaVespertina);
    setFrecuenciaNocturna(miFrecuenciaNocturna);
    setTiempoRecoleccion(miTiempoRecoleccion);
    setTiempoEntrega(miTiempoEntrega);
    setModo(miModo);
    setFromId(miFromId);
    setToId(miToId);
    setRuta(miRuta);
    getIdArcoSoporte().addAll(l);
  }

  public Arco() {
  }

  //Estos son los métodos que permiten accesar (get) y modificar (set) los campos
      de la clase
  public void setSoyHiperArco(int soyHiperArco) {
    this.soyHiperArco = soyHiperArco;
  }

  public int getSoyHiperArco() {
   return soyHiperArco;
  }

  public void setFromId(int fromId) {
    this.fromId = fromId;
  }

  public int getFromId() {
    return fromId;
  }

  public void setToId(int toId) {
```

```
    this.toId = toId;
}

public int getToId() {
    return toId;
}

public void setModo(int modo) {
    this.modo = modo;
}

public int getModo() {
    return modo;
}

public void setTiempoConElFlujo(double tiempoConElFlujo) {
    this.tiempoConElFlujo = tiempoConElFlujo;
}

public double getTiempoConElFlujo() {
    return tiempoConElFlujo;
}

public void setFrecuenciaCombinada(double frecuenciaCombinada) {
    this.frecuenciaCombinada = frecuenciaCombinada;
}

public double getFrecuenciaCombinada() {
    return frecuenciaCombinada;
}

public Arco getCopy() {
    return new Arco(this);
}

public void setFrecuenciaMatutina(double frecuenciaMatutina) {
    this.frecuenciaMatutina = frecuenciaMatutina;
}

public double getFrecuenciaMatutina() {
    return frecuenciaMatutina;
}

public void setNodosCabeza(List<Nodo> nodosCabeza) {
    this.nodosCabeza = nodosCabeza;
}

public List<Nodo> getNodosCabeza() {
    return nodosCabeza;
}

public void setCostoCalculado(double costoCalculado) {
    this.costoCalculado = costoCalculado;
}

public double getCostoCalculado() {
```

```
    return costoCalculado;
  }

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

  public int getId() {
    return id;
  }

  public void setTiempoEntrega(double tiempoEntrega) {
    this.tiempoEntrega = tiempoEntrega;
  }

  public double getTiempoEntrega() {
    return tiempoEntrega;
  }

  public void setTiempoRecoleccion(double tiempoRecoleccion) {
    this.tiempoRecoleccion = tiempoRecoleccion;
  }

  public double getTiempoRecoleccion() {
    return tiempoRecoleccion;
  }

  public void setTiempoAbordaje(double tiempoAbordaje) {
    this.tiempoAbordaje = tiempoAbordaje;
  }

  public double getTiempoAbordaje() {
    return tiempoAbordaje;
  }

  public void setRuta(int ruta) {
  this.ruta = ruta;
  }

  public int getRuta() {
    return ruta;
  }

  //Este método verifica si el hiper camino contiene o no el arco especificado
  public boolean contains(List<Arco> arcosHiperCamino) {
    for(Arco i: arcosHiperCamino){
      if(this.getId() == i.getId()){
        return true;
      }
    }
    return false;
  }

  public void setIdArcoSoporte(List<Integer> idArcoSoporte) {
    this.idArcoSoporte = idArcoSoporte;
  }
```

```
  public List<Integer> getIdArcoSoporte() {
    return idArcoSoporte;
  }

  public void setFrecuenciaVespertina(double frecuenciaVespertina) {
    this.frecuenciaVespertina = frecuenciaVespertina;
  }

  public double getFrecuenciaVespertina() {
    return frecuenciaVespertina;
  }

  public void setFrecuenciaNocturna(double frecuenciaNocturna) {
    this.frecuenciaNocturna = frecuenciaNocturna;
  }

  public double getFrecuenciaNocturna() {
    return frecuenciaNocturna;
  }
}
```

ANEXO D

Clase Nodo

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;


public class Nodo {
private int nId;
  private char name;
  private List<Arco> arcosEntrantes = new ArrayList<Arco>();
  private boolean fuiVisitado = false;
  private List<Integer> estados = new ArrayList<Integer>();
  private HiperCamino hiperCaminoEntrante = new HiperCamino();

  public Nodo(int unID){
    setnId(unID);
  }
  //Este es el constructor que hace la copia
  private Nodo(Nodo nodoCopiar) {
    this.setnId(nodoCopiar.getnId());
    this.name = nodoCopiar.name;
    this.fuiVisitado = nodoCopiar.fuiVisitado;
    this.hiperCaminoEntrante = nodoCopiar.hiperCaminoEntrante.getCopy();
    for(int i : estados){
      this.estados.add(i);
    }
    for (Arco a : getArcosEntrantes()){
      this.getArcosEntrantes().add(a.getCopy());
    }
  }
```

```
//Este es el método para hacer la copia
public Nodo getCopy() {
  return new Nodo(this);
}
public void setnId(int nId) {
  this.nId = nId;
}

public int getnId() {
  return nId;
}

public void setArcosEntrantes(List<Arco> arcosEntrantes) {
  this.arcosEntrantes = arcosEntrantes;
}

public List<Arco> getArcosEntrantes() {
  return arcosEntrantes;
}


public Nodo() {

}

//Este constructor que regresa un hiper camino que termine en el nodo indicado
    con el estado pedido
public HiperCamino dameHiperCaminoEntrante(int unEstado) {
  if (HiperRutas.nodosVisitados.indexOf(this.getnId()) != -1) {
    for(HiperCamino h : HiperRutas.losCaminosBuenos){
        if (h.getIdNodoFinal() == this.getnId() &&
         h.getEstadoHiperCamino() == unEstado) {
        return h;
      }
    }
  }
  return hiperCaminoEntrante = new HiperCamino(this, unEstado);
}

//Este constructor que regresa un hiper camino compuesto por arcos de autobús y
    que termine en el nodo indicado
public HiperCamino dameHiperCaminoEntranteDeAutobuses() {
  if (HiperRutas.nodosVisitados.indexOf(this.getnId()) != -1) {
    for(HiperCamino h : HiperRutas.losCaminosBuenos){
      if (h.getIdNodoFinal() == this.getnId() &&
       h.getArcoSucesivo().getSoyHiperArco() == 1) {
        return h;
      }
    }
  }
  return null;
}

//Este método permite conectarse a la base de datos, para a su vez consultar los
    arcos que incidan en el nodo indicado
public Object obtenArcos() {
```

```
try {
  String url = "jdbc:postgresql://127.0.0.1:5432/RedCU";
  Class.forName("org.postgresql.Driver");
  Connection con = DriverManager.getConnection( url,"postgres","postgre");
  Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
   ResultSet.CONCUR_UPDATABLE);
  ResultSet rs;

  String query = "select id, h_arco, tiempo, tiempo_abo, frec_matu, frec_vesp,
   frec_noc, tiempo_rec, tiempo_ent, from_id, to_id, modos,
    ruta from arcos_copia where to_id=" + getnId();

  rs = stmt.executeQuery(query);

 while (rs.next()){
    int arcoId = rs.getInt("id");
    int yoSoyHiperarco = rs.getInt("h_arco");
    int miFromId = rs.getInt("from_id");
    int miToId = rs.getInt("to_id");
    List<Integer> l =  new ArrayList<Integer>();
    if(yoSoyHiperarco == 1){
      Connection con2 = DriverManager.getConnection( url,"postgres","postgre");
      String query2 = "select id from arcos_copia where from_id=" + miFromId;
      Statement stmt2 = con2.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
       ResultSet.CONCUR_UPDATABLE);
      ResultSet rs2 =stmt2.executeQuery(query2);
      while(rs2.next()){
         l.add(rs2.getInt("id"));
      }
      stmt2.execute("END");
      stmt2.close();
      con2.close();

    }
    double miTiempoConElFlujo = rs.getDouble("tiempo");
    double miTiempoAbordaje = rs.getDouble("tiempo_abo");
    double miFrecuenciaMatutina = rs.getDouble("frec_matu");
    double miFrecuenciaVespertina = rs.getDouble("frec_vesp");
    double miFrecuenciaNocturna = rs.getDouble("frec_noc");
    double miTiempoRecoleccion = rs.getDouble("tiempo_rec");
    double miTiempoEntrega = rs.getDouble("tiempo_ent");
    int miModo = rs.getInt("modos");
    int miRuta = rs.getInt("ruta");


  Arco esteArco = new Arco(arcoId,  yoSoyHiperarco,  miTiempoConElFlujo,
  miTiempoAbordaje,  miFrecuenciaMatutina,
  miFrecuenciaVespertina,  miFrecuenciaNocturna,  miTiempoRecoleccion,
  miTiempoEntrega,  miModo,  miFromId,  miToId, miRuta, l);
  getArcosEntrantes().add(esteArco);
  }
  stmt.execute("END");
  stmt.close();
  con.close();
  } catch (ClassNotFoundException e) {
  e.printStackTrace();
```

```
      } catch (SQLException e) {
      e.printStackTrace();
    }
    return null;
}
        //El método verifica si la pareja nodo-estado pertenece a Qnow
  public boolean estoyEnQnow(int unEstado){
    Iterator<ParNodoEstado> itQnow = HiperRutas.qNow.iterator();
    while(itQnow.hasNext()){
      ParNodoEstado elementoQnow = itQnow.next();
        if(this.getnId() == elementoQnow.getEsteNodo().getnId() &&
        unEstado == elementoQnow.getEsteEstado()){
        return true;
      }
    }
    return false;
  }

        //El método verifica si la pareja nodo-estado pertenece a Qnext
  public boolean estoyEnQnext(int unEstado){
    Iterator<ParNodoEstado> itQnext = HiperRutas.qNext.iterator();
    while(itQnext.hasNext()){
      ParNodoEstado elementoQnext = itQnext.next();
      if(this.getnId() == elementoQnext.getEsteNodo().getnId() &&
         unEstado == elementoQnext.getEsteEstado()){
        return true;
      }
    }
    return false;
  }
}
```

ANEXO E

Clase ParNodoEstado

```
public class ParNodoEstado {
  private Nodo esteNodo;
  private int esteEstado;

  public void setEsteNodo(Nodo esteNodo) {
    this.esteNodo = esteNodo;
  }
  public Nodo getEsteNodo() {
    return esteNodo;
  }
  public void setEsteEstado(int esteEstado) {
    this.esteEstado = esteEstado;
  }
  public int getEsteEstado() {
    return esteEstado;
  }
  public ParNodoEstado() {
  }
  public ParNodoEstado(Nodo unNodo, int unEstado) {
    setEsteNodo(unNodo);
    setEsteEstado(unEstado);
  }
}
```