UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio Politécnico

Desarrollo de un sistema de localización de rutas óptimas entre dos puntos geográficos

Eric Manuel Aguayo Moreno

Tesis de grado presentada como requisito para la obtención del título de Ingeniería de Sistemas

Quito, 15 de mayo del 2008

Resumen

La presente tesis trata sobre el desarrollo de una aplicación basada en web que permita encontrar la ruta óptima entre dos puntos geográficos de un área del Distrito Metropolitano de Quito en base a los tiempos promedios en que toma atravesar cada uno de los tramos que componen las vías. Se analizan algunas de las mejores herramientas que existen para el desarrollo de la aplicación y se estudian varios algoritmos que permitan el cálculo de la ruta óptima de manera eficiente para el propósito. El documento inicia con un análisis detallado de los requerimientos del sistema así como el diseño de la base de datos y luego describe el proceso seguido para la implementación de la aplicación.

Abstract

The present thesis is about the development of an web based application, which allows to find the optimal route between two geographic points within an area of the “Distrito Metropolitano de Quito” based on the mean time that takes to cross each section of the way. There is an analysis of some the best tools that can be used to develop such application and there is a study of the algorithms which may allow the calculation of the route. This document begins with a detailed analysis of the system requirements and database design then it describes the process followed for the application implementation.

Índice: **página**

Desarrollo de un
Sistema de localización de rutas óptimas

1. Introducción

En la actualidad, el crecimiento de la actividad comercial, la expansión tanto demográfica como geográfica de las personas, el incremento en el nivel de la calidad de vida entre otras razones y el temporal que deja muchas vías inhabilitadas han provocado un aumento en la necesidad trasladarse entre los diversos puntos geográficos en un tiempo menor al que tomaba en épocas anteriores. Todo esto, junto con el incremento en la adquisición de vehículos incrementa con el paso del tiempo la necesidad de buscar medios que permitan transportarse de manera eficiente.

En el presente documento se plantea una solución al problema brevemente mencionado a través del desarrollo de una aplicación basada en web que permita la localización de las rutas óptimas dentro de una región geográfica.

2. Antecedentes

Es evidente que el crecimiento acelerado del Distrito Metropolitano de Quito ha incrementado la variedad de rutas existente para llegar de un punto a otro dentro de la ciudad. Por otro lado, existen personas que necesitan llegar a tiempo a una determinada institución o lugar de trabajo y por tanto necesitan elegir la ruta que tome menor tiempo para trasladarse de un punto a otro dentro de la ciudad. Además, dado la creciente adquisición de vehículos de transporte terrestre, existe una gran cantidad de tráfico vehicular sobre ciertas zonas a determinadas horas del día, de la cual surge la necesidad de determinar una ruta eficiente para trasladarse entre dos puntos geográficos de la ciudad y se espera que debido al continuo crecimiento, ésta necesidad aumente.

Existen muchos beneficios y ventajas en el uso eficiente de las vías de tránsito. En primer lugar existe un beneficio evidente para los usuarios de las vías de transporte permitiendo reducir el tiempo para llegar a un determinado lugar y disminuyendo el nivel de estrés en los conductores y contaminación por ruido de la bocina. Una ventaja también es que permitirá reducir la congestión del tráfico vehicular general de la ciudad, reduciendo el consumo de combustible, evitando la contaminación por ruido y disminuyendo la tasa de accidentes de tránsito de personas que tratan de llegar a tiempo a un lugar específico. Es por esto que es importante encontrar maneras de mejorar la eficiencia del transporte vehicular.

Una solución a este problema es la posibilidad de determinar la ruta óptima entre un punto de origen y un punto de destino dado. Una ruta óptima se entiende por aquella ruta que toma la menor cantidad de tiempo para llegar de un lugar de origen a un lugar de destino entre un número variable de rutas posibles. La ruta será considerada como una colección de tramos sobre los cuales influyen ciertos factores. En el proyecto considerará el tiempo transcurrido sobre cada tramo a una determinada hora en un determinado día de la semana como factor principal para determinar la ruta optima.

En este proyecto se plantea como solución crear un sistema que permita encontrar la ruta óptima entre dos puntos geográficos de la ciudad. El proyecto está orientado a ser usado por las personas que necesitan trasladarse por distintas localidades de la ciudad de manera eficiente.

En la actualidad se cuenta con el software y la capacidad computacional suficiente como para desarrollar e implementar la solución al problema. Existen motores de bases de datos propietarios como Microsoft SQL Server, así también otras que son libremente distribuidas como PostgreSQL que permiten manejar información geográfica de diversos tipos y que pueden ser conectadas fácilmente con infraestructuras que permiten ejecutar aplicaciones web como IIS7

en el caso de Microsoft, ArcIMS en el caso de ESRI (Environmental Systems Research Institute) o UMN Mapserver/Apache desarrollado por la Universidad de Minnesota.

3. Objetivos

a. Objetivo General

El objetivo final del proyecto es crear un sistema que sirva como herramienta para determinar la ruta óptima entre dos puntos geográficos del distrito metropolitano de Quito a través de una aplicación web.

b. Objetivos específicos

- Es necesario que la aplicación sea fácil de usar para el usuario, es decir, que se despliegue información gráfica necesaria (e.g. un mapa) además de información que guíe al usuario sobre la ruta sugerida.
- La aplicación deberá tener un buen desempeño en el sentido de que la información debe presentarse de manera oportuna al usuario.
- La aplicación deberá permitir su mantenimiento y actualización sobre modificaciones en la información debido a que el comportamiento del tráfico vehicular es un fenómeno dinámico.
- Las herramientas utilizadas para desarrollar el sistema deberán ser de amplio uso y conocimiento para que el sistema pueda ser actualizado y mantenido en el largo plazo.
- Realizar un modelo de la base de datos de acuerdo a los requerimientos del sistema.
- Analizar los posibles algoritmos u opciones que permitan la localización de rutas óptimas (e.g. dijkstra, programación lineal, etc).

- Realizar un esquema de la representación del problema en estructuras de datos y los objetos a utilizarse con sus distintas funcionalidades así como un diagrama de flujo de datos y un modelo de la interfaz gráfica apropiado.
- Implementar funciones que permitan el mantenimiento de los tramos y las rutas de la base de datos.
- También es necesario definir interfaces claras y métodos que permitan ampliar o enlazar la aplicación con otros sistemas (e.g GPS o aplicaciones relacionadas con el análisis de tráfico).
- Documentar los pasos para el desarrollo y la implementación del sistema para facilitar su posterior uso y mantenimiento.
- Evaluar el trabajo realizado para determinar mejoras que podrían realizarse en un futuro

4. Contenido

a. Definiciones Generales

- **Nodo:** Punto geográfico con coordenadas específicas que se almacena en la base de datos
- **Tramo (arista):** Línea o arista continua que une dos nodos y cuya información se almacena en la base de datos. Un tramo no contiene más de dos nodos y ésta definido exactamente por dos nodos.
- **Grafo:** Conjunto de nodos y aristas las cuales conectan a algunos pares de nodos del conjunto de nodos.
- **Ruta o camino:** Subconjunto de nodos y tramos de un grafo en el que cada nodo pertenece a exactamente dos tramos excepto dos nodos, los nodos de inicio y fin que pertenecen a un tramo.
- **Grafo conexo:** Grafo en el cual se puede encontrar un camino desde cualquier nodo del conjunto de nodos del grafo a cualquier otro nodo del mismo conjunto
- **Grafo con dirección:** Grafo en el cual cada arista tiene una dirección que indica desde que nodo y hacia que nodo se puede viajar por la arista.

- **Ciclo:** Es un conjunto de nodos y aristas sobre que forman un camino que permite volver a un mismo nodo sin regresar por ninguna de las aristas.
- **Árbol:** Es un grafo conexo en el que no existen ciclos.
- **Ruta óptima:** Ruta o camino que toma el menor tiempo para trasladarse de un punto a otro.

b. Requerimientos del sistema

El sistema requiere ejecutarse como una aplicación web, es decir, que pueda obtenerse la información de la ruta óptima a través de un portal web mediante el uso de un navegador. La razón de esto es incrementar la facilidad de uso para el usuario puesto que la mayoría de usuarios tendrán un conocimiento previo con la utilización de aplicaciones web. También se podrá aprovechar la infraestructura de internet para la difusión de la aplicación.

La interfaz gráfica de la aplicación debe ser orientada a la facilidad de uso por parte del usuario, conteniendo elementos intuitivos que permitan al usuario el uso inmediato del sistema. Se debe mostrar un mapa que pueda orientar al usuario sobre la ubicación de la ruta. También es necesario desplegar el tiempo en que toma trasladarse por la ruta indicada.

El sistema deberá proveer de rutas alternas en caso de que algún tramo de una vía quede cerrado o inhabilitado temporalmente. Asimismo se deberá proveer de los medios necesarios como funciones para habilitar o deshabilitar los tramos de la vía y funciones que permitan el cálculo de las rutas alternas.

Otro requerimiento del sistema es que sea lo suficientemente rápido tanto en el cálculo de la ruta óptima como en el despliegue de los resultados de la información. Para mejorar la eficiencia en el despliegue de los resultados es necesario que las rutas sean precalculadas en la base de datos de manera que la información pueda ser entregada al usuario inmediatamente. Es necesario tener un tiempo razonable para el cálculo de la ruta óptima, de manera que si se cierra una vía o el tiempo en atravesar un tramo determinado cambia con el tiempo, las rutas asociadas

se puedan calcular nuevamente en cuestión de minutos. Éste cálculo no deberá sobrepasar más de un día para el cálculo de la ruta óptima entre cada par de nodos de un día y hora específicos. Éste tiempo es razonable debido a que va a ser necesario ejecutarse solo la primera vez.

El sistema también debe proveer un conjunto de funciones disponibles para facilitar el mantenimiento y la posible extensión de la aplicación para que pueda ser usada con otros sistemas como por ejemplo con un sistema basado en GPS o un sistema de entrega de pedidos.

c. Representación matemática del problema y consideraciones preliminares.

Una base de datos de un área considerable de la ciudad de Quito que va desde la calle Ladrón de Guevara al sur hasta la Avenida Francisco de Orellana al norte y desde la Avenida 6 de diciembre al este hasta la Avenida América al oeste fue proporcionada por la EMAAP-Q. Se asume que los datos de la base de datos proporcionados por la EMAAP-Q son consistentes en el sentido de que se asume que existe al menos un camino para llegar de un nodo cualquiera a otro nodo arbitrario. La verificación de la consistencia de estos datos está fuera del alcance de éste proyecto no obstante se han incluido en el código de la aplicación un mecanismo para manejar los casos en el que no exista un camino para llegar de un nodo a otro.

Para encontrar la ruta óptima entre dos puntos geográficos, es necesario organizar la información de manera que podamos distinguir cada una de las rutas posibles. Para esto es necesario introducir el concepto de grafo. Un grafo es un conjunto de nodos distribuidos sobre un plano que están conectados entre sí a través de aristas que en éste caso se llamarán tramos. De esta manera las intersecciones de las vías están representadas por los nodos, en donde una ruta podría tomar un camino distinto, y las vías que van de una intersección a otra inmediata estarán representadas por los tramos.

Matemáticamente los tramos se pueden representar por un conjunto de pares de nodos. Si es que un par de nodos, está en el conjunto, entonces eso quiere decir que una arista une ese par de nodos. En el caso de representar las vías de una ciudad por tramos es necesario considerar la dirección de las vías, por lo tanto se tiene que considerar no sólo un conjunto de pares de elementos, sino un conjunto de pares ordenados en donde el primer elemento podría representar el nodo de partida y el segundo elemento el nodo de llegada. Entonces podemos representar a los tramos como una tabla con pares de nodos de inicio y fin. Sin embargo algunas vías pueden tener doble sentido entonces se tiene dos opciones, agregar otra arista en sentido contrario o considerar una propiedad adicional de cada tramo que permita distinguir ésta vía en sentido doble. En éste proyecto se va a utilizar la segunda opción debido a que la base de datos proporcionada incluye un campo que permite distinguir las calles (que se considerarán como vías de un sólo sentido) de las avenidas (que se considerarán como vías de doble sentido)

Ahora, una ruta es entonces una colección de tramos, pero no cualquier colección de tramos sino una colección “secuencial” de tramos. Decimos que una ruta cumple con las siguientes propiedades:

1. Los tramos están ordenados
2. Existen un nodo de inicio y un nodo de fin, los cuales están asociados a un único tramo
3. El resto de nodos está asociado exactamente a dos tramos
4. La ruta es un grafo conexo, es decir todos los nodos están conectados entre sí o en otras palabras desde cualquier nodo de la ruta se puede llegar a cualquier otro nodo de la misma ruta y existe al menos un camino para llevar a cabo esto.

Luego del análisis anterior, el paso siguiente es guardar la información relevante con respecto a cada una de las rutas, tramos y nodos. Se podría entender a la ruta óptima como la ruta que recorre la menor distancia en términos geográficos. Sin embargo, desde el punto de vista del usuario es más conveniente considerar la ruta óptima como la ruta que toma el menor tiempo para trasladarse de un punto a otro que en éste caso no es necesariamente igual a la ruta que recorre la menor

distancia puesto que los niveles de tráfico son distintos en cada una de las rutas.

Ahora, éste concepto de ruta óptima conlleva a considerar otros aspectos puesto que el tiempo que toma en trasladarse de un punto geográfico a otro es variable con respecto a la hora y el día de la semana. Por lo tanto es necesario también considerar los tiempos en que toma cada ruta para cada hora del día en cada día de la semana.

El siguiente problema es decidir si el tiempo de recorrido (o costo en tiempo) es una propiedad de la ruta o de alguna otra entidad. Si consideramos un día específico de la semana y una hora específica de ese día entonces notamos que el costo en tiempo para cada tramo es relativamente constante en promedio por lo tanto, el promedio de costo en tiempo es una propiedad de cada tramo. Sin embargo éste varía con la hora y día de la semana que estemos considerando Por lo tanto el costo en tiempo es una variable del producto cartesiano de tres variables que son la hora, el día y el tramo.

El usuario del sistema también va a estar interesado en saber el tiempo promedio que le tomará recorrer la ruta, por lo tanto un tiempo total promedio calculado a partir de la suma de los tiempos promedios de cada tramo debe estar asociado a cada ruta.

Es también importante considerar que las vías no son estáticas en la realidad y necesitan mantenimiento o por alguna razón necesitan cerrarse temporal o permanentemente por lo tanto es necesario calcular rutas alternas. En el caso del cierre temporal de una vía, es necesario calcular otra ruta para el caso de algunas rutas que incluyen a la vía, estas deberán ser almacenadas sin reemplazar la ruta original puesto que luego de que se rehabilite la vía, se utilizará la ruta original.

d. Representación de las entidades y descripción de procedimientos

En base a las consideraciones realizadas en la sección previa se han podido identificar 9 entidades (o tablas) con sus respectivos atributos que se guardan como tablas en una base de datos con sus respectivos campos dentro de la base de datos. Con el fin de usar un mismo formato para la nomenclatura de los objetos, todos los nombres y denominaciones tanto de entidades como atributos están con letras minúsculas. En caso de ser una denominación compuesta de varias palabras, éstas se separarán con un subguión ("-"). Las entidades se encuentran detalladas a continuación con su denominación respectiva:

uioev: Consta de los tramos o vías que conforman todo el grafo o mapa de calles. La información fue proporcionada por la EMAAP-Q. Cada entrada de la tabla representa una vía simple, es decir un tramo de vía que no consta de intersecciones o nodos intermedios. Se especifica por lo tanto con un punto de entrada denominado f_node y un punto final denominado t_node. La información proporcionada por la EMAAP-Q también incluye la distancia de cada uno de los tramos dentro del campo length. También se incluye el nombre de la vía a la que pertenece dentro del campo name. La base de datos proporcionada por la EMAAP-Q está en formato shp correspondiente a un shapefile que es un tipo de archivo manejado por el software de ESRI. Al convertir la información para ser ingresada en la base de datos de postgreSQL se crea otro campo denominado the_geom, en donde se almacena la información de la geometría correspondiente a ser renderizada, en este caso, la geometría correspondiente a los tramos será una línea. Esta información sería suficiente si se asume que los tramos tienen doble sentido, pero esto no siempre es cierto puesto que existen calles de una vía y doble vía. Para esto se considera el valor del campo type, el cual indica si los tramos son calles o avenidas. A continuación se encuentra una tabla con los distintos campos y el tipo de datos asignado al campo para la tabla uioev.

Tabla d.1 Campos o atributos de la Tabla uioev

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
gid	Integer	Clave Primaria
f_node	Integer	Clave Extranjera
t_node	Integer	Clave Extranjera
length	Numeric	
name	varchar(35)	
type	varchar(8)	
the_geom	Geometry	
open	Boolean	

nodes: Ésta entidad contiene los nodos del grafo de calles, es decir las intersecciones de las vías en donde los vehículos pueden tomar otra ruta. La tabla es obtenida a partir de las aristas que se encuentran almacenadas en la tabla uioev ya que estas cuentan con la información de los nodos de inicio y fin. La tabla cuenta con un identificador único de tipo entero denominado node_id. A continuación se encuentra una tabla con los distintos campos y el tipo de datos asignado al campo para la tabla de nodos. También tiene como atributos los nombres de las dos calles que conforman las intersecciones correspondientes al nodo denominados respectivamente name1 y name2. Sin embargo podríamos tener que los nodos son intersecciones de más de una vía (un redondel o un intercambiador por ejemplo), entonces adicionalmente se agregarán 3 campos más denominados respectivamente name3, name4 y name5. También se agregó el campo other_desc para proporcionar información adicional de la locación del nodo. Se han incluido tres campos adicionales denominados distance, solved y pred_node, que serán utilizados por el algoritmo de Dijkstra para el cálculo de la ruta óptima.

Tabla d.2 Campos o atributos de la Tabla nodes

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
node_id	Integer	Clave Primaria
name1	varchar(35)	
name2	varchar(35)	
other_desc	varchar(100)	
distance	Numeric	
solved	Boolean	
pred_node	Integer	
name3	varchar(35)	
name4	varchar(35)	
name5	varchar(35)	

time_costs: Ésta tabla representa los tiempos asignados a cada uno de los tramos. No es una propiedad o atributo de los tramos debido a que depende de un día y una hora específica, por lo tanto se la ha considerado como una entidad adicional. Ésta entidad se puede identificar a través de tres atributos que son claves extranjeras que se refieren a otras entidades que son: gid_fk, hour_id_fk, day_id_fk relacionadas con las tablas uioev, hours y days correspondientes al tramo, la hora y el día respectivamente. Adicionalmente tiene un campo que contiene el costo en tiempo correspondiente denominado time_cost_avg. La obtención de los valores para éste último atributo están fuera del alcance de ésta tesis por lo tanto se van a calcular éstos valores en función de la distancia para realizar las pruebas respectivas. La tabla de campos de la entidad se muestra a continuación:

Tabla d.3 Campos o atributos de la Tabla time_costs

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
gid_fk	Integer	Clave Primaria/Extranjera
hour_id_fk	Integer	Clave Primaria/Extranjera
day_id_fk	Integer	Clave Primaria/Extranjera
time_cost_avg	Numeric	

hours: Está tabla representa las horas consideradas para el cálculo de la ruta óptima. Cada registro o fila de la tabla está identificada únicamente a través del campo de tipo entero denominado hour_id. Para ésta tabla se necesita especificar un intervalo de tiempo sobre el cual la entrada de la tabla time_costs correspondiente es válida, por lo tanto se necesita dar la hora de inicio y la hora de fin, las cuales están representadas por los campos denominados begin_time, end_time. Los intervalos se han especificado inicialmente para intervalos enteros de horas aunque no necesariamente están restringidos a estos intervalos, por ejemplo se podría definir intervalos de 30 minutos especificando respectivamente las horas de inicio y fin. Se han tomado las 24 horas del día para la tabla inicial. A continuación se muestra la tabla de campos para ésta entidad.

Tabla d.4 Campos o atributos de la Tabla hours

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
hour_id	Integer	Clave Primaria
begin_time	Time	
end_time	Time	

days: Ésta tabla contiene los días considerados para el cálculo de la ruta óptima. La tabla está identificada por un campo entero denominado day_id. Además contiene un campo denominado name, para proporcionar el nombre o descripción del día (por ejemplo: Viernes o Feriado). Los campos de la entidad se muestran a continuación:

Tabla d.5 Campos o atributos de la Tabla days

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
day_id	Integer	Clave Primaria
Name	varchar(35)	

routes: Ésta tabla contiene las rutas óptimas calculadas entre cada par de nodos (nodos de inicio y fin). Puesto que la entidad time_costs es dependiente de la hora y el día, entonces el cálculo de la ruta óptima está basado en el tiempo en costo de cada tramo para el día y hora especificados aparte de los nodos de inicio y fin. Como se espera que para cada par de nodos a exista una sola ruta un día y hora específicos, entonces se especifica que la clave primaria de la base de datos sea compuesta por claves extranjeras de t_node y f_node de la tabla nodes, hour_id de la tabla hours y day_id de la tabla days. El número de tramos es también guardado como parte de la entidad para establecer cuantos tramos en la tabla route_tracks existen pertenecientes a una ruta específica. Esto también podría haberse obtenido a través de la instrucción COUNT de SQL, sin embargo para optimizar el tiempo se decidió asignarla como atributo. Los tramos de los que están compuestos las rutas se especifican en una nueva entidad debido a que una ruta por lo general está compuesta por más de un tramo por lo cual forma parte de una relación uno a muchos. A continuación se muestra la tabla correspondiente a ésta entidad

Tabla d.6 Campos o atributos de la Tabla routes

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
f_node	Integer	Clave Primaria/Extranjera
t_node	Integer	Clave Primaria/Extranjera
hour_id_fk	Integer	Clave Primaria/Extranjera
day_id_fk	Integer	Clave Primaria/Extranjera
total_time	Integer	
num_edges	Integer	

route_tracks: En ésta entidad se encuentran especificados los tramos que corresponden a diversas rutas. Como clave primaria se encuentran claves extranjeras a los mismos campos de la clave primaria de la tabla routes más una clave extranjera correspondiente a gid, que es el identificador del tramo correspondiente de la tabla uioev. Adicionalmente existe un campo denominado edge_number que permite establecer el orden en que son recorridos los tramos en la ruta. Los campos de la tabla se encuentran a continuación

Tabla d.7 Campos o atributos de la Tabla route_tracks

Nombre del Campo	Tipo de Datos	Clave Primaria/Extranjera
gid_fk	Integer	Clave Primaria/Extranjera
f_node	Integer	Clave Primaria/Extranjera
t_node	Integer	Clave Primaria/Extranjera
hour_id_fk	Integer	Clave Primaria/Extranjera
day_id_fk	Integer	Clave Primaria/Extranjera
edge_number	Integer	

Adicionalmente se han considerado dos tablas o entidades denominadas alternate_routes y alternate_route_tracks que son similares a las tablas routes y route_tracks respectivamente y que cumplen el propósito de almacenar las rutas alternas y los tramos correspondientes a las mismas en el caso de cerrar temporalmente un tramo que esté contenido en la ruta original.

Se han identificado además los distintos procedimientos y funciones que se describen en la siguiente tabla de manera que la aplicación cumpla con los requerimientos especificados. El código de los procedimientos y funciones con su respectiva documentación, se encuentran descritas en mayor detalle en el Anexo e.

Tabla d.8 Descripción de los procedimientos y funciones

Nombre	Descripción
populate_nodes	Permite llenar la tabla nodes con nodos a partir de una tabla de tramos como la tabla uioev proporcionada por la EMAAP-Q
populate_time_cost	Permite llenar la tabla time_cost con información estimada de los costos en tiempo a partir de las distancias de los tramos almacenadas en la tabla uioev.
find_route	Aplica un algoritmo de optimización de rutas para encontrar la ruta óptima dado un nodo inicial, un día y una hora específicos.
find_routes	Encuentra todas las rutas óptimas entre cada par de nodos para un día y hora específicos
find_all_routes	Encuentra todas las rutas óptimas entra cada par de nodos para cada hora y cada día de la tabla hours y de la tabla days respectivamente.
find_alternate_route	Encuentra las rutas alternas dado un nodo de origen, estas rutas son guardadas en las tablas alternate_routes y alternate_route_tracks.
close_street	Cambia el valor del campo o columna open de la tabla uioev a falso indicando que el tramo está cerrado o inhabilitado temporalmente. Además recalcula las rutas alternas para las rutas que incluyen al tramo cerrado y establece el campo valid de las rutas originales como false, es decir, no es válida
open_street	Restablece el campo o columna open de los registros de la tabla uioev que tenga un tramo determinado y cambia el campo valid de la tabla de rutas a true para las rutas que incluyan ese tramo.
find_node	Dados dos nombres calcula al menos un nodo asociado a estos nombres. Devuelve los resultados como una tabla

extract_geometry	Dados dos pares de nombres de calles y los identificadores de un día y una hora específicas. Devuelve la geometría de la ruta correspondiente.
extract_route	Dados dos pares de nombres de calles y los identificadores de un día y una hora específicas. Devuelve una tabla con los nombres de los tramos que se atraviesa en orden del recorrido de la ruta.
find_route_nodes	Dados dos pares de nombres de calles y los identificadores de un día y una hora específicas. Devuelve los nodos de inicio y fin que son considerados para el cálculo de la ruta en caso de existir varios nodos asociados como en el caso de un redondel
get_route_extent	Dados dos pares de nombres de calles y los identificadores de un día y una hora específicas. Devuelve dos coordenadas representadas por cuatro puntos numéricos en una tabla que representan la extensión de la ruta
change_cost	Dados el identificador de un tramo, el día y la hora, cambia el costo en tiempo de ese tramo para el día y las horas especificadas y recalcula las rutas asociadas a ese tramo.

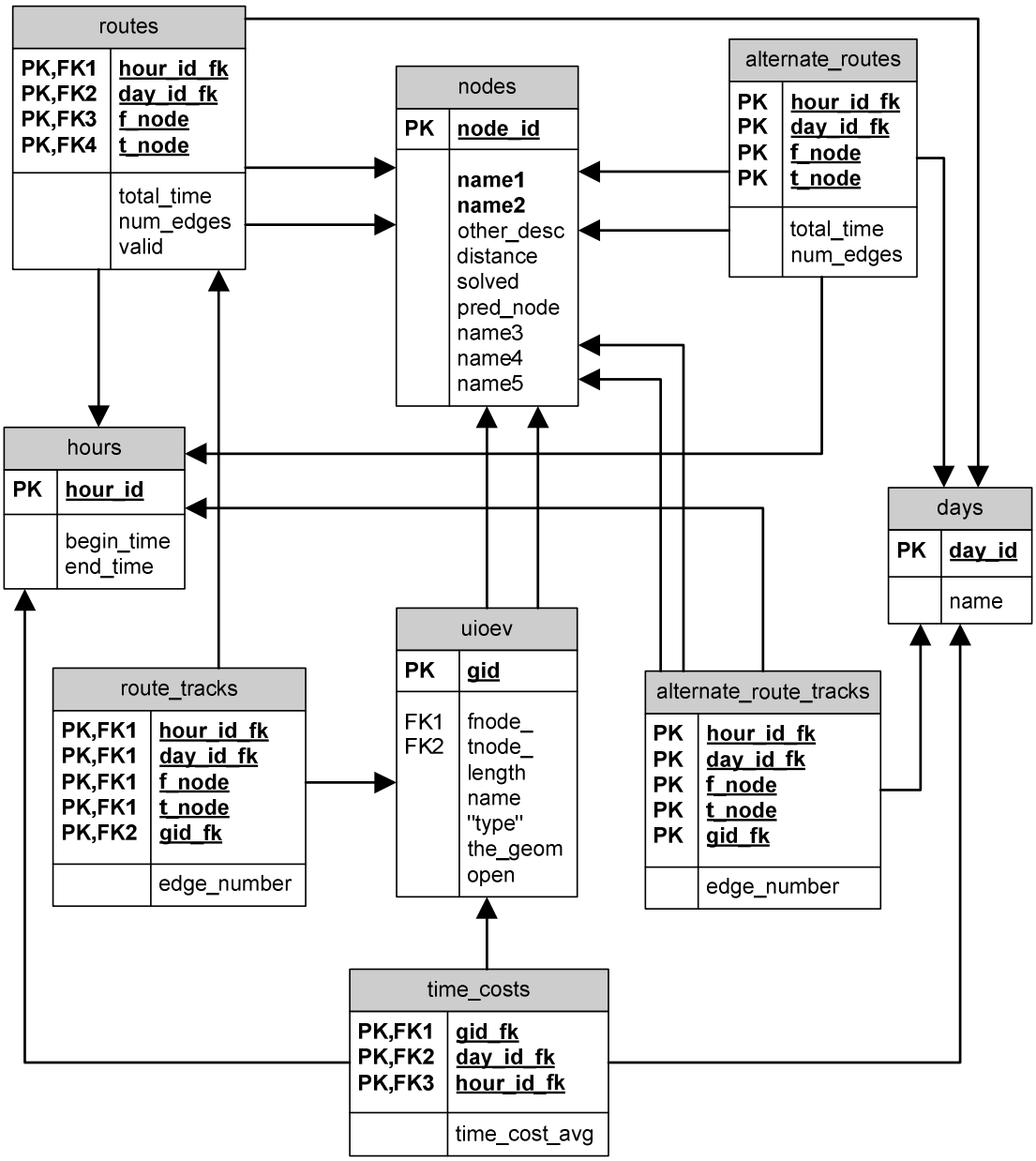
Además de las funciones mencionadas anteriormente existe un conjunto de funciones proporcionadas por las extensiones PostGIS que permiten la obtención de coordenadas geográficas, realizar operaciones geométricas, calcular distancias, puntos de intersección, etc. Estas funciones facilitarán una posible extensión de la aplicación para el uso con otros sistemas, como sistemas de entrega de paquetes o sistemas basados en GPS.

e. Diseño de la base de datos

Basado en las consideraciones anteriores desarrollamos el siguiente modelo para la base de datos independiente del motor de la base de datos que vaya a ser utilizado excepto por el campo the_geom que es específico del motor de PostgreSQL con extensiones Postgis. Más adelante se detallará el proceso de decisión en la selección del software y motor de base de datos a ser utilizado.

La información de los tramos proporcionada por la EMAAP-Q se encuentra en la tabla uioev. La tabla de nodos contiene todas las intersecciones de las vías en las que una ruta podría cambiar. La tabla time_costs contiene el tiempo promedio que toma atravesar cada uno de los tramos de la tabla uioev a una hora y día de la semana dado, por lo tanto es evidente que se encuentra relacionada directamente con las tablas hours, days y uioev. Las tablas hours y days contienen información relacionada con las horas y los días de la semana que se van a considerar para tomar los tiempos promedio. También se encuentran las tablas rutas, que contiene la ruta óptima para cada par de nodos a un día y hora específicos. La tabla route_tracks contiene información relacionada con los tramos que componen cada ruta y el orden de los mismos. En base a éstas consideraciones tenemos el siguiente esquema de la base de datos en el que se muestran las entidades con sus respectivas relaciones y atributos

Figura e1. Modelo de la base de datos



f. Análisis del algoritmo para el cálculo de la ruta óptima

Existen varios algoritmos para obtener la ruta óptima en un grafo con dirección, muchos de ellos son derivaciones o adaptaciones del famoso algoritmo de dijkstra. Vamos a analizar cuatro algoritmos que son ampliamente usados en la actualidad, estos son: Algoritmo de Floyd-Warshall, Algoritmo de Bellman-Ford, Algoritmo de Dijkstra, y Algoritmo de Dijkstra A*.

Analicemos primero el algoritmo de Floyd–Warshall, éste algoritmo permite en una ejecución del mismo encontrar la ruta óptima entre todos los pares de nodos de un grafo. La idea de éste algoritmo es iterar sobre todo el conjunto de nodos partiendo de rutas óptimas para cada par de nodos y en cada iteración verificar si es que una siguiente opción de ruta es mejor que la anterior. Una gran ventaja de éste algoritmo sobre otros es que es un algoritmo iterativo bastante simple como se puede apreciar en el anexo b.

Sin embargo, el algoritmo como tal sólo permite calcular la distancia más corta, es necesario por lo tanto incluir información adicional en cada iteración para llevar un registro de las rutas marcadas hasta el momento lo que incluiría una complejidad adicional. La complejidad del algoritmo es evidentemente $O(n^3)$ debido a los tres lazos anidados. Desafortunadamente el algoritmo sólo permite calcular las rutas óptimas entre cada par de nodos, en el caso de necesitar hallar el algoritmo sólo para algún nodo de origen específico, el algoritmo debe ejecutarse de nuevo para todos los nodos.

Otro aspecto de éste algoritmo que podría ser un inconveniente es que obtenemos las rutas óptimas sólo al final del algoritmo. En mapas demasiado grandes podría ser útil para determinar rutas aproximadas iterando un menor número de veces sobre el algoritmo.

También tenemos el algoritmo de Bellman-Ford que consiste básicamente en calcular la distancia de un nodo (aquel para el cual aplicamos el algoritmo) a los demás sin dar ningún salto, luego dando uno, luego 2 y así sucesivamente. Detalles del algoritmo se encuentran en el anexo b.

Una ventaja de este algoritmo es que permite manejar grafos que contengan aristas con pesos negativos. Sin embargo hay una desventaja muy grande la complejidad del algoritmo para calcular todas las rutas óptimas desde un nodo de inicio específico es $O(n^3)$, por lo cual para calcular la ruta entre todos los nodos la complejidad sería de orden $O(n^4)$. En un grafo demasiado grande el tiempo de ejecución tendría una diferencia considerable. Sin embargo puede ser interesante en el caso de que se necesite determinar las rutas óptimas aproximadas solo hasta n saltos.

El algoritmo de Dijkstra es un algoritmo recursivo que básicamente analiza las rutas óptimas desde un nodo de partida especificado a los nodos adyacentes a un árbol de rutas óptimas. En otras palabras, el algoritmo se ejecuta de la siguiente manera:

- Partimos de un árbol que contiene un sólo nodo (el nodo de origen)
- Buscamos entre los nodos que no están en el árbol el nodo adyacente que se encuentre a la menor distancia del nodo de origen.
- Agregamos el nodo y la arista que contribuye con la menor distancia al árbol y guardamos como propiedad del nodo la distancia desde el nodo de origen hasta el nodo agregado (también llamado nodo resuelto). Añadimos también la información del nodo predecesor para poder recuperar la ruta.
- Luego volvemos ejecutar el algoritmo hasta agregar al árbol para agregar otro nodo y así sucesivamente hasta agregar todos los nodos del grafo.

El pseudocódigo de éste algoritmo se encuentra detallado en el anexo b. La ventaja del algoritmo de Dijkstra es evidente, en cada paso el algoritmo encuentra la ruta óptima desde el nodo inicial al nodo agregado. Si es que la nueva ruta determinada hasta cada nodo resuelto no está agregada, la agregamos a la tabla de rutas, caso contrario el proceso continúa. De esta manera se mejora la eficiencia al no tener que volver a agregar la ruta y no tenemos que esperar hasta que finalice el algoritmo para ir encontrando rutas óptimas por lo cual las podemos ir agregando

directamente a la base de datos a medida que se ejecute el algoritmo. La complejidad del algoritmo es $O(n^2)$ para calcular las rutas óptimas desde un nodo específico, es decir, que para calcular todas las rutas entre todos los pares de nodos el algoritmo es de orden $O(n^3)$.

Existe otro algoritmo que se lo denomina algoritmo de Dijkstra A* que es una combinación del algoritmo de Dijkstra con el algoritmo A* que es utilizado para la resolución de juegos por turnos en el que se penaliza una jugada se aleje o perjudique el resultado final deseado. La idea de éste algoritmo es acelerar el proceso de búsqueda desde un nodo de partida hasta un nodo de llegada penalizando aquellos nodos que están más alejados geográficamente del nodo destino. A pesar de esto, no es evidente cuánto penalizar a cada nodo por estar más lejos del nodo destino en función de la distancia a ese nodo.

Por lo general, cualquiera que sea la política de penalización, éste algoritmo incrementa la velocidad para encontrar la ruta de la distancia más corta en entre un par de nodos determinados, sin embargo, los pesos de las aristas en nuestro caso no están determinados por las distancias sino por el tiempo que toma en atravesar las aristas, por lo tanto, este algoritmo no necesariamente mejora la eficiencia, además de que el algoritmo se vuelve más complicado al tener que manejar información geográfica al mismo tiempo lo que podría reducir el desempeño en lugar de mejorarlo con respecto al algoritmo de Dijkstra original. Otro punto en contra de éste algoritmo es que es únicamente útil para calcular la ruta óptima entre un nodo origen y un nodo destino. El algoritmo tendría que ejecutarse de nuevo si es que se cambia el nodo destino o tendría que complicarse más para no tener que ejecutarse de nuevo, en ese caso, el algoritmo de Dijkstra original sería más eficiente. En el caso de éste proyecto, es más probable que necesiten calcularse varias rutas alrededor de un mismo punto puesto que si se cierra temporalmente un tramo, pueden existir varias rutas que contenga un mismo nodo de origen y que contengan ese tramo. Por éstas razones, éste algoritmo queda descartado.

En síntesis, el algoritmo de Bellman-Ford a pesar de permite manejar pesos de aristas negativos a diferencia de los otros algoritmos, no se aplica en éste caso y además tiene una complejidad mayor que los otros

algoritmos analizados por lo cual ésta opción queda descartada. Ahora, se podría realizar una implementación del algoritmo de Floyd-Warshall para el caso de determinar todas las rutas entre cada par de nodos puesto que éste algoritmo es aparentemente más simple que el algoritmo de Dijkstra, sin embargo, estamos interesados en obtener la ruta y como se mencionó antes esto incluye una complejidad extra por lo cual su eficiencia sobre el algoritmo de Dijkstra no está garantizada. Además utiliza demasiados recursos puesto que es necesario mantener una matriz de dimensión nxn y en el caso de una ciudad completa n puede ser muy grande haciendo uso de una gran cantidad de memoria. Para la aplicación es necesario proveer un procedimiento para calcular la ruta óptima entre cada par de nodos de la tabla, sin embargo, esto se realizará una sola vez. Eventualmente se necesitará recalcular todas las rutas pero esto no es algo que suceda a menudo. Por lo cual descartamos el algoritmo de Floyd-Warshall y seleccionamos como algoritmo para el cálculo de las rutas óptimas al algoritmo de Dijkstra.

g. Selección del software y el motor de la base de datos

Para el propósito de ésta tesis existen un sinnúmero de paquetes computacionales que nos permiten manejar información geográfica y desplegarla de manera que pueda ser leída por el usuario en una página web. Ahora vamos a analizar dos de los paquetes computacionales de amplio uso que permiten manejar información geográfica para desarrollar aplicaciones basadas en la web, estos son:

- UNM Mapserver
- ESRI ArcIMS

Cada uno de estos paquetes de software ofrece la misma funcionalidad, sin embargo, Mapserver tienen algunas ventajas sobre ArcIMS. Existe evidencia suficiente acerca de que por lo general Mapserver responde significativamente más rápido que ArcIMS y debido a que se quiere satisfacer al usuario con un despliegue rápido de

información entonces ésta opción es conveniente. En el anexo a. se encuentra información detallada acerca de la descripción y comparación de ambas herramientas.

Una ventaja de ArcIMS sobre Mapserver es su facilidad de instalación y configuración lo que facilita la administración mantenimiento de la aplicación, sin embargo puede restar flexibilidad en las opciones proporcionadas. Mapserver por su parte se ha demostrado que no necesita recargar los servicios y que el servidor ser reinicia considerablemente rápido lo que es una ventaja sobre ArcIMS en los procesos administrativos

Mapserver trabaja en conjunto con el servidor de Apache, que es completamente configurable y de código abierto y compatible con varios formatos de scripting, mientras ArcIMS implementa su propio servidor web el cual está restringido a solo algunos formatos de scripting lo que representa una desventaja frente a Mapserver. Además Mapserver satisface mucho más los requerimientos para un WMS estándar según el OGC (Open Geospatial Consortium) que ArcIMS.

En conclusión debido a que Mapserver es compatible con muchas otras herramientas y además iguala o sobrepasa el desempeño de ArcIMS por lo tanto será la herramienta a ser utilizada en éste proyecto.

Ambos paquetes analizados anteriormente soportan una serie de motores de base de datos tanto propietarios, como Microsoft SQLServer, así también como de código abierto como MySQL y PostgreSQL. Sin embargo, PostgreSQL tiene una arquitectura bastante estable que permite manejar información geográfica que puede ser manejada por Mapserver. Esto se realiza a través de las extensiones Postgis. Actualmente las bases de datos de SQLServer y MySQL también han introducido la posibilidad de manejar información geográfica pero es una tecnología relativamente nueva mientras que PostgreSQL la información geográfica ya se ha venido manejando desde hace algún tiempo con Postgis lo que aumenta las probabilidades de que la aplicación sea estable. Es por eso que se utilizará PostgreSQL como motor de la base de datos de la aplicación

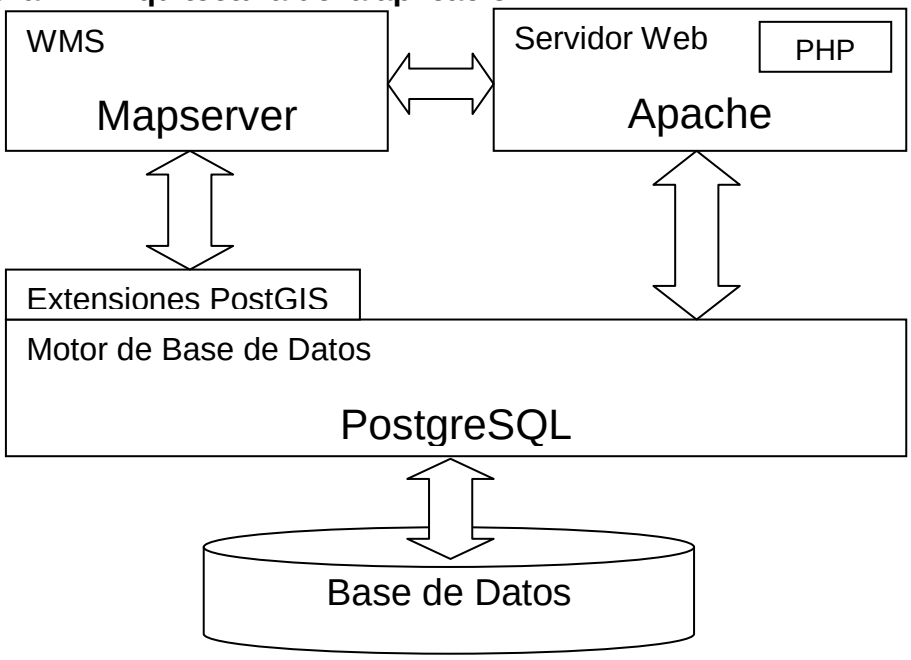
Mapserver también trabaja con varios servidores de aplicaciones, compatibles con J2EE y lenguajes de scripting como PHP, Python, Perl.

Mapserver provee de una librería llamada Mapscript que contiene un conjunto de funciones de PHP que permiten manipular la información geográfica a ser desplegada. Se ha decidido realizar este proyecto en PHP para reducir el tiempo de desarrollo de la aplicación permitiendo a su vez ejecutar la aplicación durante el proceso de desarrollo para verificar la correcta implementación de la aplicación.

h. Arquitectura de la aplicación

La arquitectura se basa de acuerdo a los requerimientos del sistema y va de acuerdo con las herramientas seleccionadas en la sección anterior. Por una parte en el backend tenemos que la información es almacenada en una base de datos de postgreSQL donde se encuentra toda la información relacionada con las rutas y las entidades descritas anteriormente. La base de datos se conecta directamente con el servidor web Apache puesto que en ocasiones es necesario desplegar información inmediatamente a la página web sin tener necesidad de pasar por Mapserver. Asimismo, Apache se comunica con Mapserver enviando la petición del usuario para desplegar determinada información geográfica. Mapserver a su vez se comunica con la base de datos para obtener la información geográfica almacenada en ésta y generar en base a esta información una imagen renderizada que será desplegada al usuario.

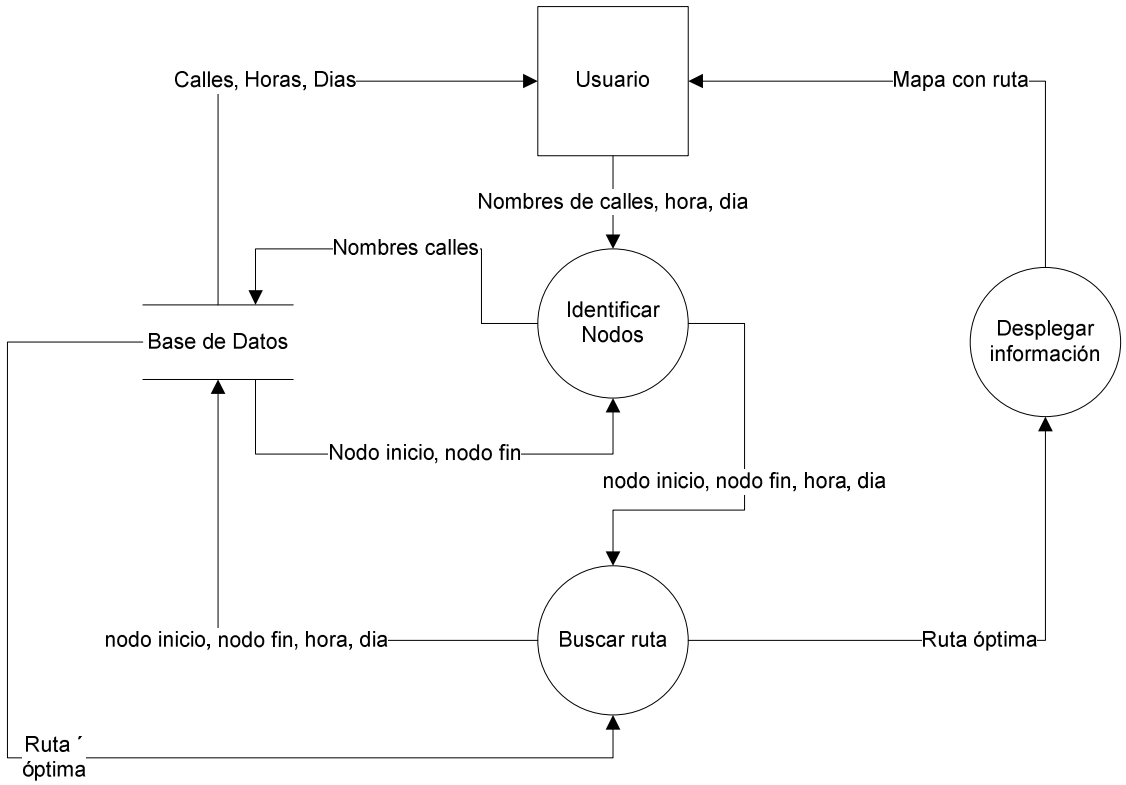
Figura h1. Arquitectura de la aplicación



i. Descripción del flujo de datos

El usuario ingresa a la aplicación a través de un navegador y en la interface se despliegan campos donde específica dos calles de origen y dos calles de destino además de la hora y día en el cual desea trasladarse. Con ésta información el usuario produce una petición de obtener la ruta óptima, la cual es dirigida hacia el servidor de páginas web, en éste caso, Apache. Luego en el lado del servidor se ejecuta el proceso de identificar los nodos correspondientes a los nombres de las calles dados en la base de datos. Luego con los nodos de inicio y fin encontrados se busca la ruta óptima en la hora y día indicados por el usuario. Ésta información de la ruta es utilizada por Mapserver para desplegar un mapa en la pagina del usuario.

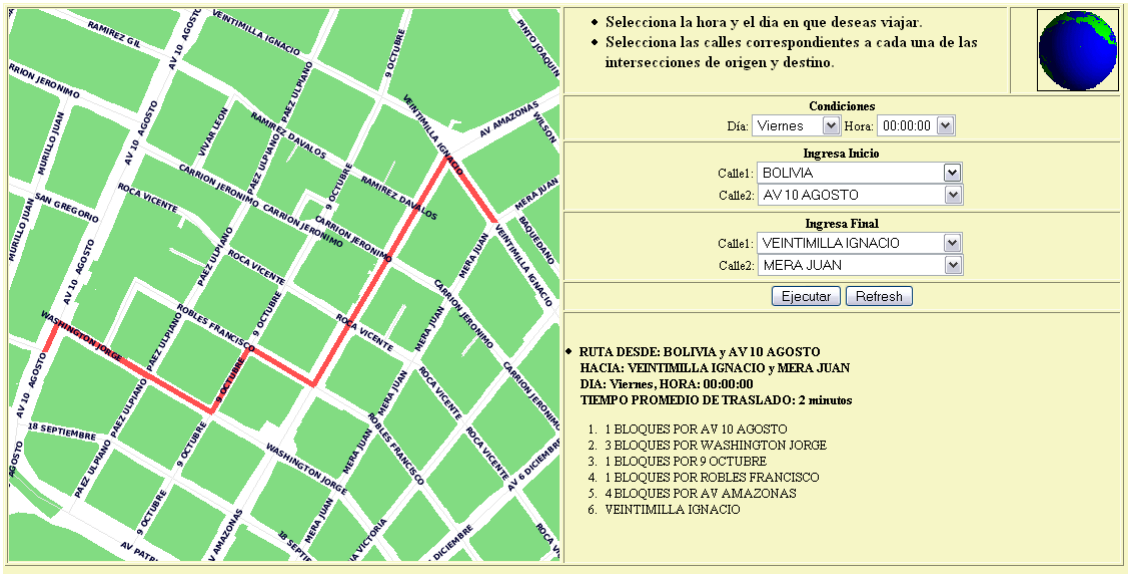
Figura i1. Diagrama de Flujo de datos



j. Esquema de la interfaz gráfica

La interfaz gráfica está diseñada de manera que sea intuitiva para el usuario. A la izquierda de la pantalla desplegada al usuario se muestra un mapa, mientras que a la derecha se despliegan campos en donde el usuario especifíca la hora, el día y las calles de intersecciones. En la parte superior derecha se encuentran instrucciones y la parte inferior derecha inicialmente se encuentra vacía. Luego de que el usuario ha ingresado los valores respectivos en los campos pulsa el botón de “Ejecutar” para obtener la ruta óptima. En ése instante se marca con una línea roja sobre el mapa la ruta óptima y se hace zoom automáticamente sobre el mapa. Además se indica el tiempo en que toma trasladarse por la ruta indicada, las vías que deben tomarse y cuantos bloques tienen que recorrerse en el orden del recorrido de la ruta desde el punto de inicio hasta el punto de destino. Toda ésta información se despliega en pocos segundos luego de haber presionado el botón de Ejecutar. Existe también un botón para resetear la página y volver al mapa al zoom original.

Figura j1. Esquema de la interfaz gráfica de la aplicación.



k. Implementación

i. Base de datos

La base de datos proporcionada por la EMAAP-Q contenía una tabla con la información geográfica de las rutas con nodos de inicio y fin para cada ruta. Esta base de datos estaba inicialmente en forma de shapefile que como ya se mencionó anteriormente es un formato leído por las aplicaciones de ESRI. Para convertir esta información a una base datos de PostgreSQL Mapserver proporciona la herramienta de línea de comandos shp2pgsql que permite crear código SQL de postgres para insertar información geográfica a través de las extensiones PostGIS.

El siguiente paso es crear la base de datos con las tablas descritas en el diseño de la base de datos. Para esto se escribió comandos SQL que permitan la creación de dichas tablas. Adicionalmente se agregó un campo adicional a la tabla uioev para permitir el cierre temporal de los tramos.

Luego se desarrolló una implementación del algoritmo de Dijkstra para el cálculo de rutas óptimas. El algoritmo se lo implementó en Plpg/SQL que es el lenguaje SQL recomendado que soporta estructuras lógicas y que permite manipular los datos de una base de datos postgreSQL. Se tomo esta decisión puesto que el algoritmo debía ser lo suficientemente rápido y eficiente, por lo cual, implementar el algoritmo en otro lenguaje hubiera implicado tener un overhead debido a la transmisión e interpretación de los resultados de las consultas.

Al principio el algoritmo era muy ineficiente tomando alrededor de una semana el cálculo de la ruta óptima en cada par de nodos. Luego se realizaron optimizaciones en el código donde se determinó la existencia de lazos que se podían evitar, esto provocó que el algoritmo terminase en cuestión de un par de días. Posteriormente se modificó la configuración del motor de la base de datos incrementando los recursos disponibles para la ejecución del motor de base de datos. Con esto se logró que las rutas óptimas entre cada par de nodos se determinaran en cuestión de horas.

Se realizó una optimización muy importante sobre el código de la implementación del algoritmo de Dijkstra que a través del uso de tablas dinámicas creadas a partir de la tabla de nodos permitió la paralelización de la ejecución del algoritmo para calcular simultáneamente el algoritmo para las rutas de distintas horas de un día.

El proceso de configurar el motor de la base de datos para mejorar la optimización no es un proceso fácil pues no hay una plantilla que se pueda llamar “la mejor configuración” para una determinada configuración de hardware y muchas veces tiene que ver con características específicas de cada base de datos y depende el tipo de procesamiento que se realice en ésa base de datos en éste caso se requiere una cantidad considerable de consultas a la tabla uioev y sobre todo comandos de inserción en la base de datos de rutas. El proceso de configuración de la base de datos se llevó a cabo siguiendo un proceso de ensayo y error en base a guías de los efectos de determinados por algunos parámetros.

Uno de los parámetros que se utiliza comúnmente para incrementar el desempeño en PostgreSQL son los “shared buffers” que definen un bloque de memoria sobre el cual se encolan peticiones que están esperando atención inmediata del cpu. Por default los shared buffers están configurados para un máximo de 32 MB. Al parecer luego de un tiempo de transcurrido el algoritmo, la velocidad de cálculo para cada nodo disminuye lo que puede indicar que los shared buffers están llenos. Luego de incrementar la memoria de los shared buffers a 128 MB, se obtuvieron incrementos de desempeño impresionantes, llevando a cabo el cálculo en alrededor de 1 día, es decir, aproximadamente 7 veces más rápido que al principio. El valor de éste parámetro se volvió a incrementar a 256 MB y se obtuvo un incremento en el desempeño pero no fue muy significativo con respecto al valor de 128 MB.

Otros parámetros que se modificaron fueron, sort memory, Effective Cache Size, Fsync Files y WAL Files, max_fsm_pages, max_fsm_relations, WAL_buffers, entre otros. Éstos parámetros básicamente definen cuantas relaciones pueden estar en memoria al mismo tiempo y cada cuanto se realizan escrituras al disco de las

inserciones o modificaciones de la base de datos. Con éstas modificaciones se obtuvo un incremento considerable en el desempeño que permitió ejecutar el cálculo de la ruta óptima en tan solo cuestión de horas.

Se implementaron funciones dentro del motor de la base de datos que permitan un retorno inmediato de consultas a la base de datos. Se implemento una función que permite determinar los nodos a los que pertenecen dos nombres de calles. Además se implementó un procedimiento que permite devolver una tabla con los tramos que componen a la ruta óptima para un par de nodos.

También se implementaron procedimientos para la administración eficiente de la base de datos, las cuales permiten, en caso de cerrarse uno o varios tramos calcular rutas óptimas alternas. Se trató de que la mayoría de procesos sobre todo los relacionados con el acceso directo a los datos almacenados en la base de datos se implementen como parte de las funciones almacenadas dentro de la base de datos con el fin de incrementar la eficiencia y reducir la complejidad en la implementación de la interfaz gráfica y la lógica de la aplicación en PHP. En el anexo e se encuentran detallados todas éstos procedimientos.

ii. Interfaz gráfica y Lógica de la Aplicación

La mayoría de funciones y procedimientos se implementaron como parte de los procedimientos y funciones almacenadas en la base de datos como se discutió en la anterior sección. Por lo tanto la complejidad en la implementación del código en PHP se redujo considerablemente, con lo cual se tomo la decisión de codificar la interfaz gráfica y la lógica de la aplicación en un solo archivo ya que el separarlo en varios archivos podría provocar tener partes de una misma secuencia lógica en archivos separados. Para evitar en lo posible confusión de la lógica de la aplicación con código de la interfaz gráfica, se trato de que el código relacionado con la aplicación esté al inicio del archivo, mientras que el código relacionado con la interfaz gráfica se encuentre al final del archivo.

La lógica de la aplicación cuenta con algunas funciones auxiliares para realizar el zoom del mapa de forma dinámica una vez que se realiza la consulta. Esto se debe a que el zoom dinámico del mapa es un proceso complicado y tener un sólo módulo que realice toda ésta tarea volvería al código demasiado difícil de mantener. La razón por la cual el hacer zoom es una tarea complicada es debido a que depende de muchos factores como son la extensión geográfica sobre la cual se desea hacer zoom (parte de éste trabajo se encarga una función almacenada en la base de datos), luego se debe considerar el tamaño de la imagen en relación a la extensión geográfica de la imagen. Cabe mencionar que la extensión geográfica se determina en relación a las coordenadas geográficas mientras que el tamaño de la imagen se mide en pixeles. Hay que también determinar cuál es el punto central sobre el mapa sobre el cual se va a hacer zoom.

Entre otros aspectos implementados a través del código de la lógica de la aplicación están la conexión con la base de datos y la realización de consultas a la misma. También se realiza una conexión con Mapserver a través de la librería php_mapscript.dll que se encarga básicamente del despliegue de la ruta óptima dentro de un mapa. PostgreSQL maneja lo que se conoce como mapfiles que son como plantillas que permiten cargar contenido geográfico en forma de capas. Un mapfile es utilizado para desplegar un mapa inicial del área de la ciudad. Luego a través de funciones que provee la librería de Mapscript se agrega una capa adicional que se recoge de la base de datos la información sobre la geometría de la ruta óptima para poder desplegar una línea que la distinga sobre el mapa.

También se encuentra implementado dentro del código de la lógica de la aplicación el proceso mediante el cual los nombres de las calles son desplegados en la interfaz gráfica indicando al usuario cuantos bloques tiene que recorrer antes de cambiar de vía. Esto se podría haber implementado como parte de una función almacenada dentro de la base de datos, sin embargo se decidió implementarlo en PHP debido a que se podía aprovechar las capacidades de manejo de cadenas que ofrece PHP.

Dentro del código correspondiente a la interfaz gráfica se encuentran codificados ciertos componentes que permiten cargar los nombres de las vías almacenadas en la base de datos dentro de la interfaz gráfica. Esto facilitará al usuario el momento de especificar los nombres de las calles en los campos respectivos.

5. Evaluación del Sistema

Durante ésta tesis lo más importante fue el desempeño en términos de tiempo de ejecución de la aplicación. El algoritmo de Dijkstra que calcula la ruta óptima fue evaluado utilizando un servidor Sun Blade 1500 de la Universidad San Francisco de Quito. Las especificaciones de éste servidor se encuentran en el anexo f.

Después de realizar las optimizaciones mencionadas anteriormente se logró reducir a aproximadamente 13 horas la ejecución del algoritmo para una sola hora y día específico lo cual ya es bastante razonable considerando que esto se va a correr una sola vez. Luego se modificó el código para que se pudiese calcular simultáneamente para varias horas. Para pocas ejecuciones simultáneas de el algoritmo, éste mejora considerablemente el tiempo de ejecución para 2 ejecuciones se ejecutó en aproximadamente 16 horas lo que implica un promedio de 8 horas por algoritmo. Ahora, 4 ejecuciones simultáneas se ejecutaron en aproximadamente 24 horas, lo que representa un promedio de 12 horas en el tiempo de ejecución lo que es una disminución de desempeño en relación al tiempo anterior. Finalmente, en la última prueba se realizaron 8 ejecuciones simultáneas, el algoritmo finalizó inesperadamente justo antes de terminar pero el servidor del PostgreSQL determinó que la causa de esto fue la falta de memoria en los Shared Buffers, lo que indica que existe una probabilidad muy alta de que la disminución progresiva de desempeño se deba a ésta razón o al parámetro max_transaction_locks que permite asignar una mayor cantidad de memoria a los procesos que adquieren el lock. Un mayor incremento en la memoria de los Shared Buffers requiere

que el kernel sea recompilado para permitir un mayor uso de memoria en éste campo. Ésta tarea está fuera del alcance de ésta tesis.

Adicionalmente se realizó una evaluación en términos de satisfacción de un grupo de usuarios en el cual se obtuvieron resultados favorables. Cabe mencionar que se les pidió a los usuarios que no consideraran dentro de su evaluación la consistencia de los datos proporcionados para la realización de la tesis puesto que la verificación de la misma está fuera del alcance.

Se les hizo tres preguntas a los usuarios del sistema para medir el grado de satisfacción en cuanto a facilidad de uso, información proporcionada y rapidez de la aplicación. Cada pregunta se evaluó sobre 10 puntos. En cuanto a la facilidad de uso se obtuvo un promedio de satisfacción de 8.625, mientras que en información proporcionada se obtuvo un promedio de 9.5. También se obtuvo un promedio bastante alentador de 10 en cuanto a rapidez. Estos resultados son bastante favorables puesto que se esperaba al menos un promedio de 80% en las calificaciones y se sobrepasaron éstas expectativas. Un detalle de los resultados obtenidos se encuentra en el anexo c.

6. Conclusiones y Recomendaciones

Existen un sinnúmero de herramientas que permiten el desarrollo de aplicaciones que contenga información geográfica, por lo tanto, es necesario realizar un estudio sobre las ventajas y desventajas que proveen cada una de éstas herramientas de acuerdo a las necesidades del problema. En este trabajo se ha realizado una comparación entre las herramientas de mayor uso sobre este tema.

Es también necesario investigar sobre los algoritmos existentes, puesto que muchos de estos algoritmos ya han sido optimizados para diversos propósitos. En éste proyecto se necesitaba un algoritmo que determinara la ruta óptima y se encontraron cuatro algoritmos importantes. No se puede decir que ninguno era mejor que otro, sin embargo sí se puede decir que un algoritmo es más conveniente que otro para determinados aspectos. Para el proyecto, el algoritmo de Dijkstra parecía ser el más equilibrado de acuerdo a los requerimientos planteados.

Un inconveniente durante el desarrollo de éste proyecto fue que se tenía que esperar mucho tiempo hasta obtener resultados durante las primeras versiones del algoritmo del cálculo de la ruta óptima. Por lo tanto se recomienda utilizar un computador con mucha capacidad de memoria y procesamiento para la base de datos para éste tipo de análisis.

Es importante realizar un análisis profundo sobre la implementación de algoritmos que necesiten ser ejecutados en el menor tiempo posible, de manera que puedan ser optimizados al máximo. También es necesario revisar la configuración del motor de la base de datos para destinar más recursos en caso de ser necesario. Además, es recomendable que estos algoritmos estén estrechamente integrados con la base de datos.

PHP puede ser un lenguaje de Scripting que ayude en incrementar la velocidad de desarrollo de las aplicaciones web, sin embargo tiende a volverse difícil de mantener en el largo plazo. Esto se debe a que a medida que se incluyen nuevas características en la interfaz gráfica, es más difícil diferenciar la lógica de la aplicación de la codificación de la interfaz gráfica.

La codificación de gran número de funciones como parte de la base de datos permitirá que la aplicación pueda ser fácilmente codificada en

otro ambiente distinto a PHP, como por ejemplo a través de un servidor de aplicaciones compatible con J2EE.

El trabajo presentado en éste documento es un inicio a un problema que puede ampliarse para poder ser utilizado con dispositivos de localización geográfica que permitan a los usuarios determinar las rutas mientras están en sus vehículos.

También se podría proveer una característica adicional que permita especificar que una vía esté cerrada periódicamente a determinadas horas o determinados días como un túnel por ejemplo. Sin embargo, esto se puede emular poniendo un costo en tiempo demasiado alto de tal manera que no sean tomadas en cuenta por el algoritmo, pero en tal caso la base de datos sería inconsistente.

Se recomienda también implementar una función que permita determinar la consistencia de la base de datos en el sentido de que existe al menos una ruta (no necesariamente óptima) entre cada par de nodos de la base de datos.

También se podría analizar la posibilidad de incluir no sólo el promedio del costo en tiempo para cada ruta a cada hora de cada día sino también información sobre una distribución de probabilidad sobre cada tramo para proporcionar al usuario información sobre el tiempo mínimo y el tiempo máximo que le tomaría recorrer la ruta óptima entre dos nodos.

Recientemente motores de base datos como Microsoft SQL Server y MySQL han provisto de soporte para manejo de información geográfica por lo cual se recomienda un estudio sobre la eficiencia entre los distintos motores de base de datos y su conectividad con Mapserver y otros paquetes de software.

7. Bibliografía

1. **S. J. Russell, P. Norvig**, Inteligencia Artificial: un enfoque moderno, Prentice Hall, México, 1999.

2. **R. Johnsonbaugh**, Matemáticas Discretas, 4ª edición, Prentice Hall, México, 1999.

3. **Behrouz A. Forouzan,** Data Communications and Networking, third edition, Tata McGraw Hill, 2003.

4. **Tanenbaum, Andrew S.**, *Computer Networks*, Prentice-Hall, 1996. **Bertsekas,**

5. **D. y Gallager, R.**, *Data Networks*, 2ª edición, Prentice-Hall, 1992.

6. **Comer, Douglas E.**, *Internetworking with TCP/IP*, 3ª edición, Volumen 1: *Principles, Potocols and Architectures*, Prentice-Hall, 1995.

7. Shridhar Daithankar, Josh Berkus, Tuning PostgreSQL for Performance
   http://www.varlena.com/GeneralBits/Tidbits/perf.html

8. Análisis de Brook Anderson sobre el desempeño de Mapserver con respecto a ArcIMS
   http://mapserver.gis.umn.edu/community/conferences/MUM3/present/session10/msvsarcims/view?searchterm=refractions

9. Artículo sobre una comparación entre ArcGIS y Mapserver
   http://www.spatiallyadjusted.com/2006/02/14/esri-arcims-vs-umn-mapserver/

10. Descripción del algoritmo de Bellman-Ford
http://es.wikipedia.org/wiki/Algoritmo_de_Bellman-Ford

11. Descripción del algoritmo de Dijkstra
http://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

12. Descripción del algoritmo de Floyd-Warshall
http://es.wikipedia.org/wiki/Algoritmo_de_Floyd

13. Descripción y Características de ArcIMS
http://www.esri-es.com/index.asp?pagina=5

14. Descripción y Características de UNM Mapserver
http://mapserver.gis.umn.edu/

8. Anexos

a. Descripción de herramientas disponibles

ESRI ArcIMS

"ArcIMS es el servidor de aplicaciones integrado dentro de la arquitectura ArcGIS que ha sido diseñado para la distribución y difusión de información geográfica, mapas y servicios GIS en entornos Internet / intranet. Tanto si se opera en un entorno limitado, como la intranet de una organización, como si se hace a través del entorno universal de Internet, es posible el empleo de ArcIMS para distribución de datos y funcionalidad GIS a múltiples usuarios. ArcIMS constituye una aplicación muy potente, escalable y basada en estándares que permite, de manera rápida y sencilla, diseñar y gestionar servicios de cartografía en Internet. " [8]

Entre las características principales están:

- Facilidades en la instalación y configuración del software
- Publicación en la web
    - Renderización de mapas
    - Consulta de información
    - Extracción y Descarga de datos
    - Geocodificación
    - Catalogo con metadatos de servicios
- Acceso integrado a Arcgis
- Capacidad de integración de datos de múltiples fuentes
- Arquitectura escalable
- Soporte a varios tipos de clientes
- Maneja repositorio de metadatos
- La comunicación integración y personalización es basada en estandars

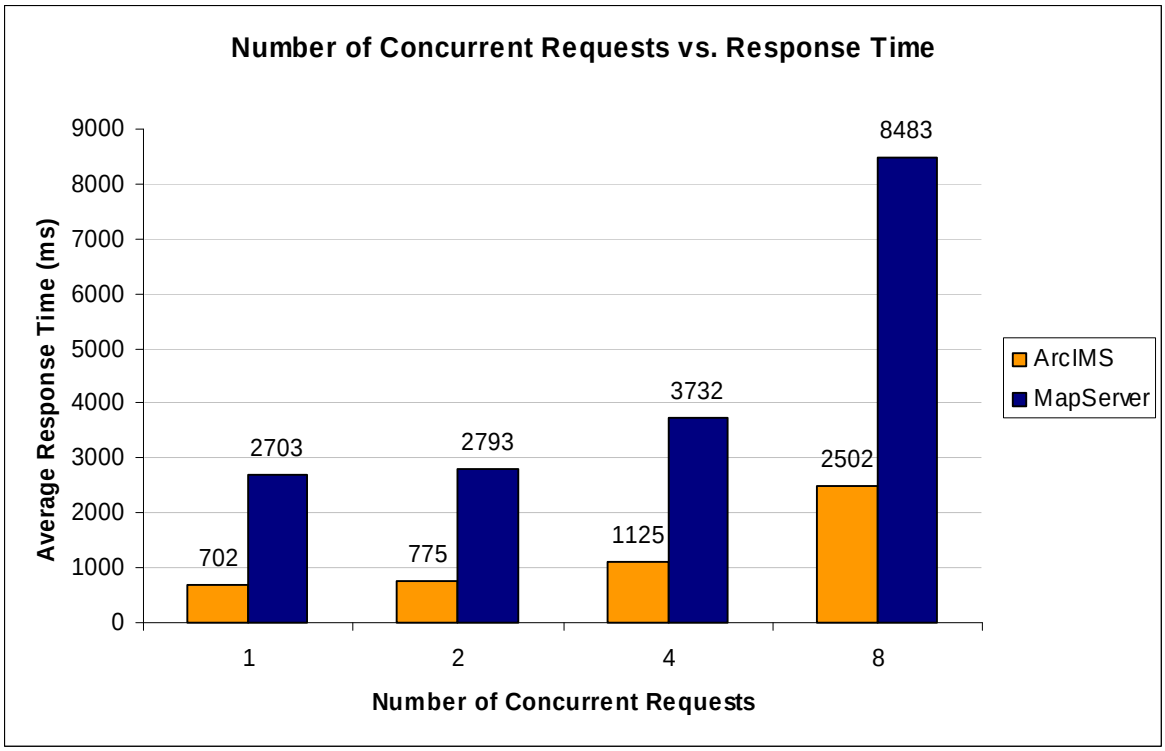
UMN Mapserver

Mapserver es un ambiente de desarrollo de código abierto para construir aplicaciones basadas en internet que permita manipular y desplegar información espacial. Se enfoca en renderizar datos con información espacial (mapas, imágenes y vectores de datos) en forma de mapas o diagramas para la web.

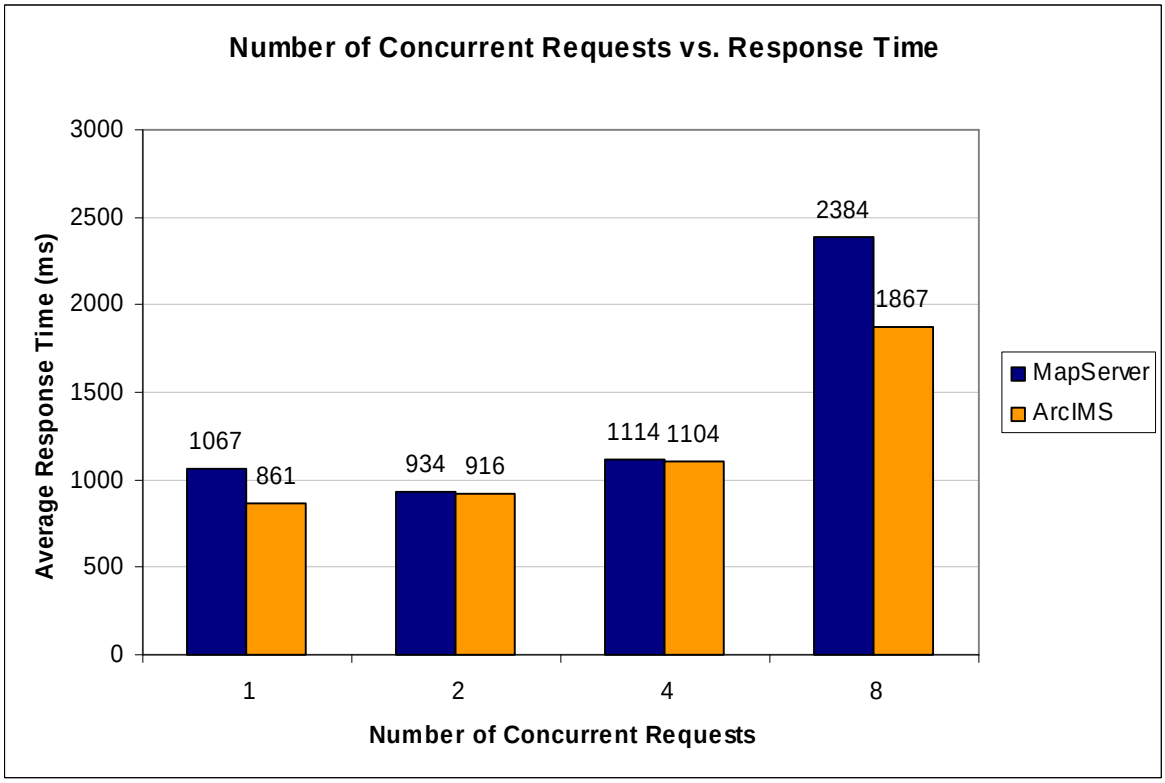
Entre sus características principales están

- Despliegue cartográfico avanzado
    - Renderización dependiente de la escala
    - Etiquetas y manejo de colisiones de etiquetas
    - Despliegue basado en plantillas personalizables
    - Soporta fuentes tipo TrueType
    - Automatización de elementos de mapa
- Soporta varios métodos de scripting y ambientes de desarrollo, entre ellos:
    - PHP, Python, Perl, Ruby, Java, and C#
- Soporta varias plataformas. Entre las más importantes
    - Linux, Windows, Mac OS X, Solaris.
- Soporta varios tipos de archivos vectoriales y raster.
    - TIFF/GeoTIFF, EPPL7, y otros a través de GDAL
    - ESRI shapfiles, PostGIS, ESRI ArcSDE, Oracle Spatial, MySQL y otros a través de OGR
    - Especificaciones de Open Geospatial Consortium (OGC)
        - WMS (client/server), non-transactional WFS (client/server), WMC, WCS, Filter Encoding, SLD, GML, SOS
- Soporte de proyección de mapas
    - Proyección de mapas “On-the-fly”

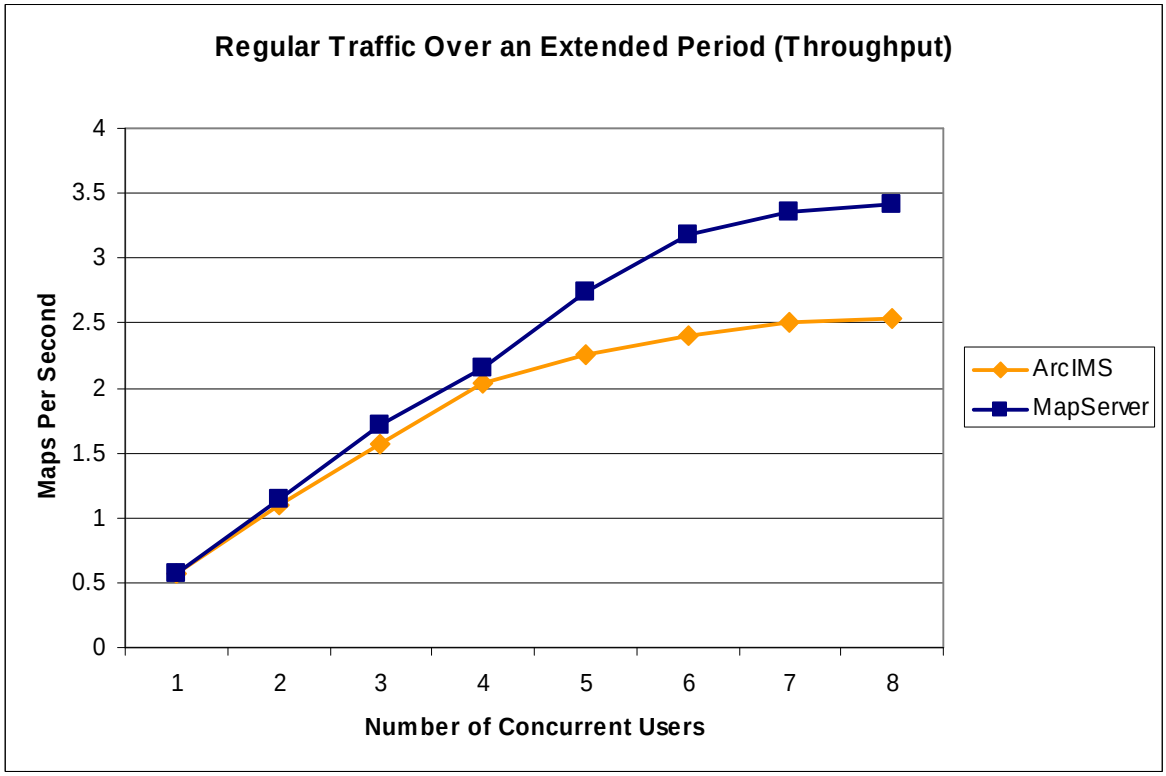
Un primer análisis en el desempeño de ambas herramientas realizado por Brock Anderson en Refractions Research Institute muestra los siguientes resultados:



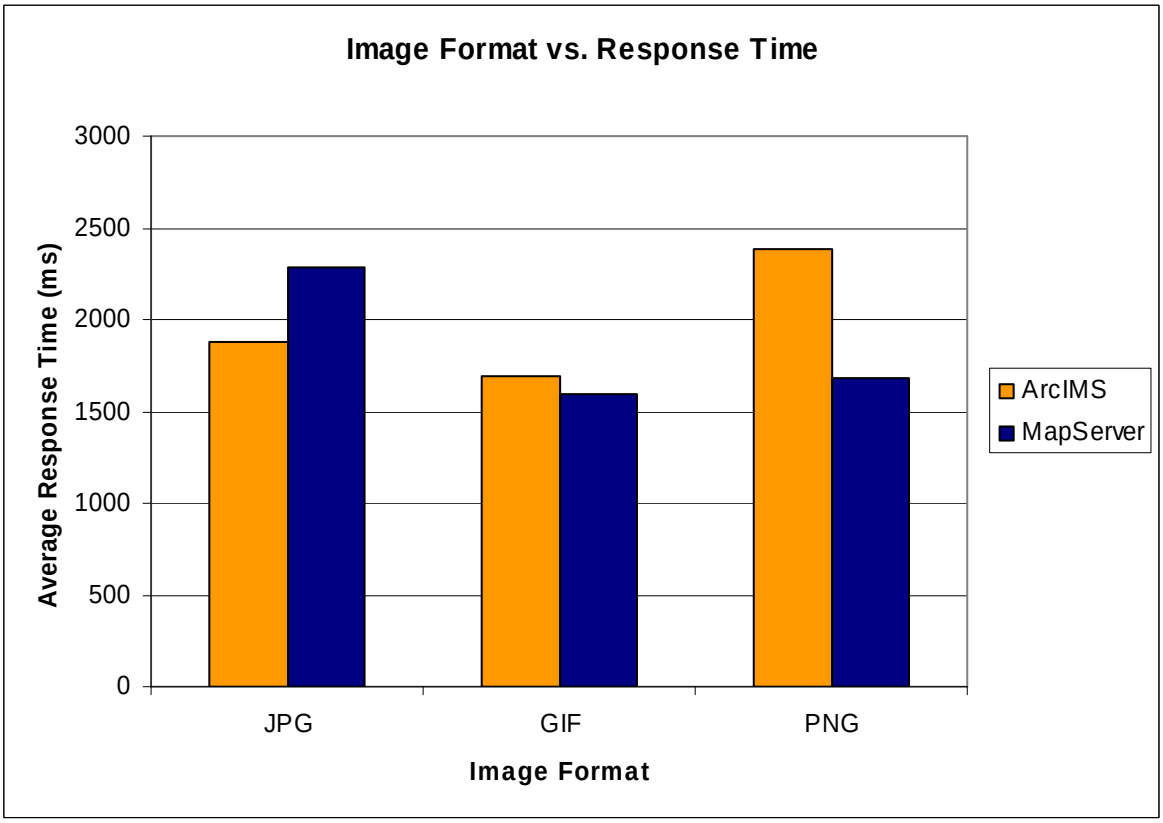
Luego de optimizaciones del código para ambos casos en el que ArcIMS ahora usa FastCGI se obtiene los siguientes resultados:



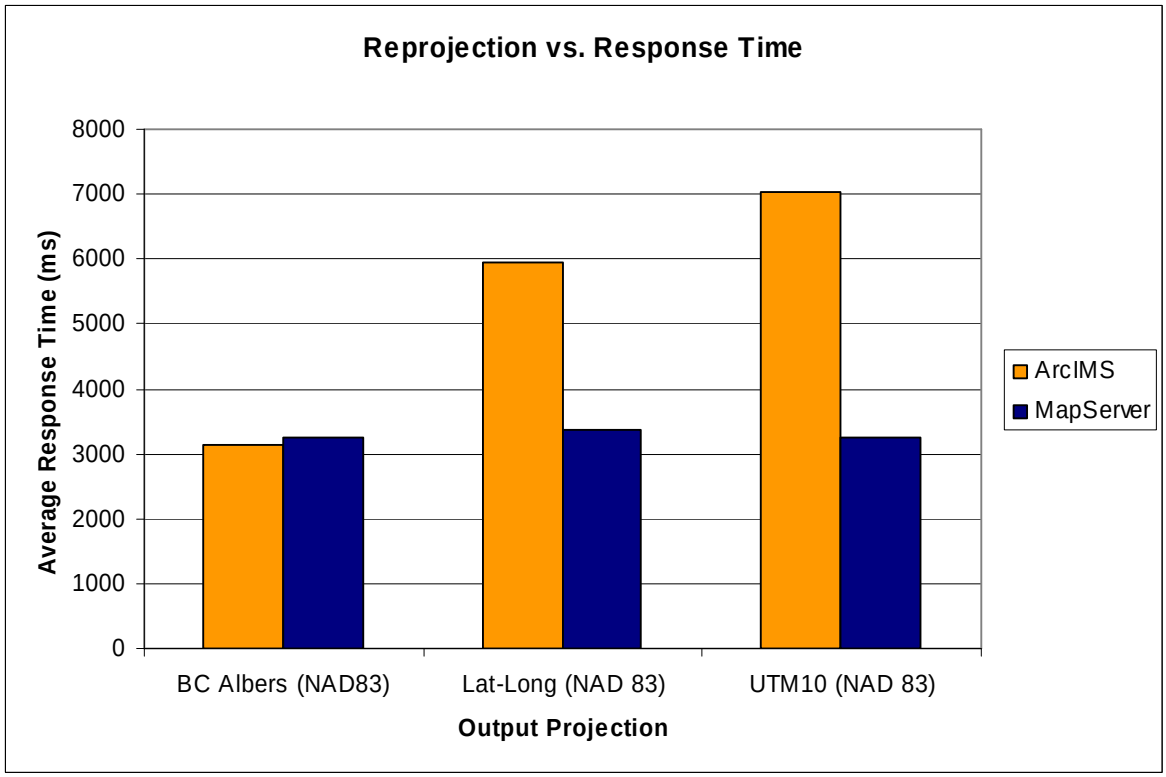
Bajo condiciones “regulares”, el desempeño de Mapserver es menor



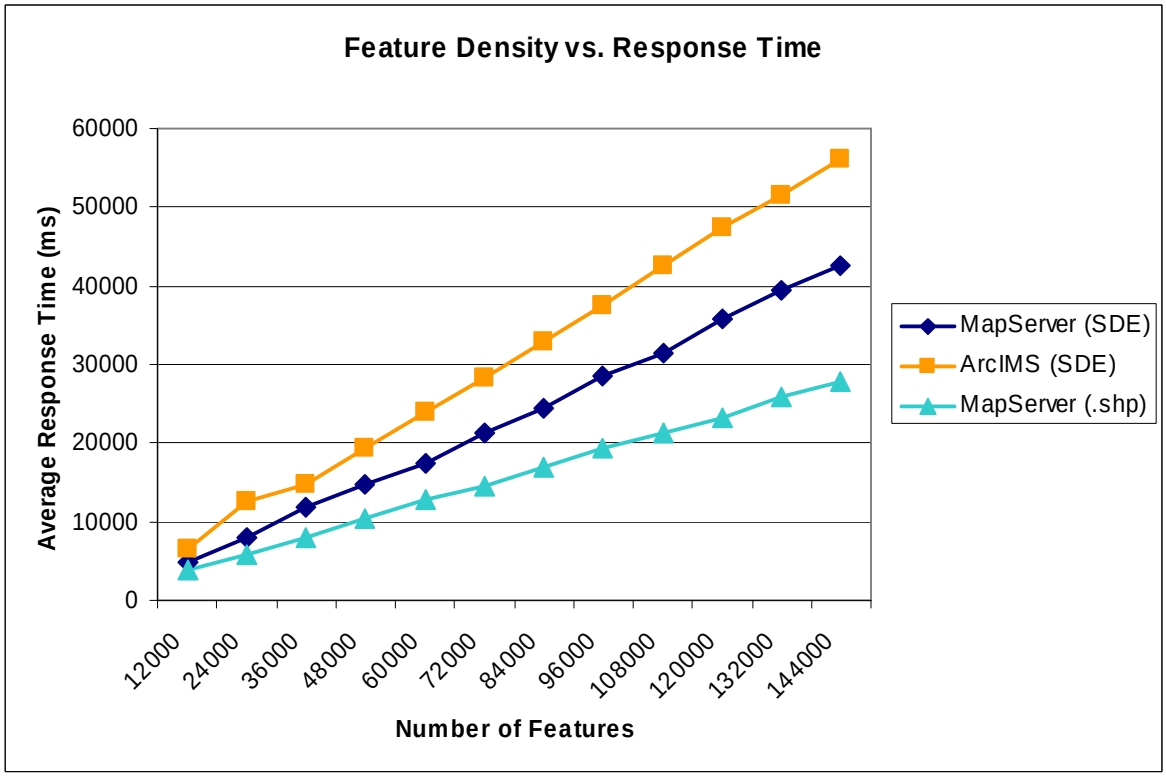
Mapserver es un poco más rápido en generar imágenes GIF y PNG



La reproyección en Mapserver es más rápida que en ArcIMS



Mapserver extrae los elementos SDE un poco más rápido que ArcIMS



Además de los resultados anteriormente presentados mapserver paso todos los tests estandarizados WMS CITE del OGC (Open Geospatial Consortium) mientras ArcIMS falló en 12 de éstos.

También se encontró que MapServer contaba con beneficios administrativos, como la facilidad en recargar los servicios y el que el servidor puede ser reiniciado rápidamente. Por su parte, ArcIMS permite un control más granular en los logs.

b. Pseudocódigo de algoritmos para el cálculo de la ruta óptima

i. *Pseudocódigo del algoritmo de Dijkstra*

El pseudocódigo a continuación describe los pasos para ejecutar el algoritmo de Dijkstra dado un grafo G y un nodo fuente s. V[g] es el conjunto de nodos en el grafo.

```
DIJKSTRA (Grafo G, nodo_fuente s)
    // inicializamos todos los nodos del grafo.
    //La distancia de cada nodo es infinita
    // y los padres son NULL
    for u ∈ V[G] do
        distancia[u] = INFINITO
        padre[u] = NULL
        distancia[s] = 0
        //encolamos todos los nodos del grafo
        Encolar (cola, V[G])
        mientras cola != 0 do
            // OJO: Se extrae el nodo que tiene distancia mínima y
            //se conserva la condición
             // de Cola de prioridad
             u = extraer_minimo(cola)
            for v ∈ adyacencia[u] do
                    if distancia[v] > distancia[u] + peso(u,v) do
                        distancia[v] = distancia[u] + peso(u,v)
                        padre[v] = u
```

ii. Pseudocódigo del Floyd-Warshall

Sea G=(V,A) un gráfo en el cual cada arco tiene asociado un costo no negativo. G=(V,A), V={1,...,n} y C[i,j] es el costo del arco que va de i a j. El algoritmo calcula la serie de matrices

$$A_0[i,j] = \begin{cases} 0 & \text{si } i = j \\ C[i,j] & \text{si } i \neq j \end{cases}$$

$$A_k[i,j] = min(A_{k-1}[i,j], A_{k-1}[i,k] + A_{k-1}[k,j])$$

$A_k[i,j]$ significa el costo del camino más corto que va de i a j y que no pasa por algún vértice mayor que k. El objetivo es calcular $A_n[i,j]$

La programación de este algoritmo puede realizarse de manera simple con tres ciclos anidados, de la siguiente manera:

Floyd-Warshall (Matriz de rutas)

```
// Hacer para los n nodos
For  k = 0 to n do
        // Los dos lazos anidados pasan por toda la matriz
        For i = 0 to n do
                For j = 0 to n do
                        //Se actualiza el costo
                        A[i,j] = mínimo(A[i,j],A[i,k] + A[k,j])
```

iii. Algoritmos de Bellman-Ford

El pseudocódigo siguiente corresponde a los pasos del algoritmo de Bellman-Ford no optimizado

```
BellmanFord(Grafo G, nodo_fuente s)
        // inicializamos el grafo.
        // Ponemos distancias a INFINITO menos el nodo fuente que
        // tiene distancia 0
        for v ∈ V[G] do
                distancia[v]=INFINITO
                predecesor[v]=NIL
                distancia[s]=0
                // relajamos cada arista del grafo tantas
                // veces como número de nodos -1 haya en el grafo
                for i=1 to |V[G]-1| do
                        for (u,v) ∈ E[G] do
                        if distancia[v]>distancia[u] + peso(u,v) then
                        distancia[v] = distancia[u] + peso (u,v)
                        predecesor[v] = u
    // comprobamos si hay ciclos de coste negativo
    for (u,v) ∈ E[G] do
       if distancia[v] > distancia[u] + peso(u,v) then
          print ("Hay ciclo de coste negativo")
          return FALSE
    return TRUE
```

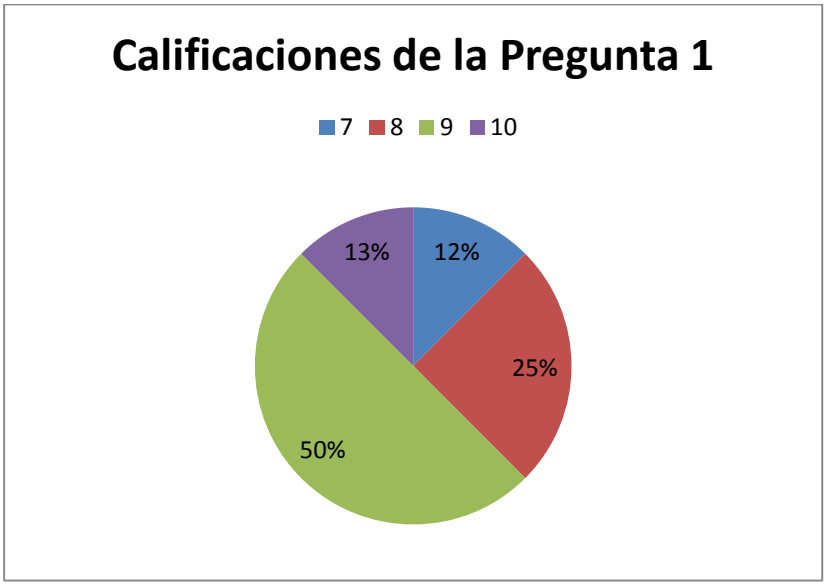
El pseudocódigo siguiente corresponde a los pasos del algoritmo de Bellman-Ford optimizado

```
BellmanFord_Optimizado(Grafo G, nodo_fuente s)
    // inicializamos el grafo. Ponemos distancias a INFINITO menos el nodo fuente que
    // tiene distancia 0. Para ello lo hacemos recorriéndonos todos los vértices del grafo
    for v ∈ V[G] do
        distancia[v]=INFINITO
        padre[v]=NIL
    distancia[s]=0
    encolar(s, Q)
    en_cola[s]=TRUE
    mientras Q!=0 then
        u = extraer(Q)
        en_cola[u]=FALSE
        // relajamos las aristas
        for v ∈ ady[u] do
            if distancia[v]>distancia[u] + peso(u,v) then
                distancia[v] = distancia[u] + peso (u,v)
                padre[v] = u
                if en_cola[v]==FALSE then
                    encolar(v, Q)
                    en_cola[v]=TRUE
```

c. Resultados de las evaluaciones a usuarios

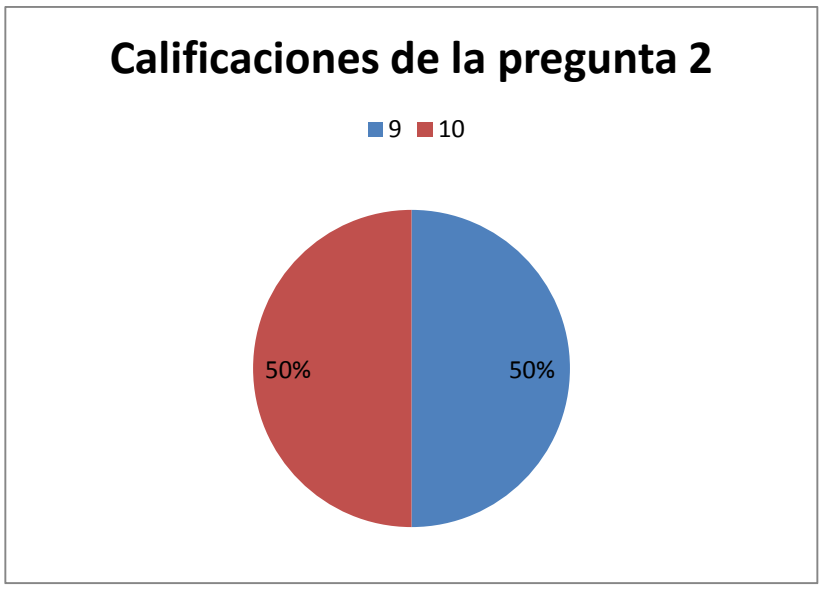
A continuación se muestra un resumen de los resultados de una pequeña encuesta realizada a 8 usuarios del sistema. Se les pidió que califiquen cada aspecto sobre 10:

1. Facilidad de uso de la interfaz gráfica



El promedio obtenido fue 8.625

2. Suficiencia de la información proporcionada



El promedio obtenido fue 9.5

3. Satisfacción con el tiempo de respuesta

El promedio fue de 10

d. Manual técnico

Contenido del DVD:

El DVD que acompaña a ésta tesis contiene las siguientes carpetas donde se encuentra la información indicada:

- Documento de Tesis: Contiene toda la información relacionada con el documento completo de la tesis
- Manual: Contiene el presente manual y el manual de referencia de Postgres.
- Codigo Fuente: Contiene la implementación de los distintos algoritmos y el código relacionado con cada una de las aplicaciones utilizadas (Postgres, Mapserver, PHP).
- Configuracion: Contiene la configuración utilizada por el motor de base de datos.
- Base de datos original: Contiene la base de datos proporcionada por la EMAAP-Q en forma de shapefile.
- Backup: Contiene las rutas precalculadas para el día el primer dia de la base de datos de días basada en costos de tiempo proporcionales a la distancia. Incluye una modificación en el costo un tramo de la Av. 6 de diciembre a las 12:00 para verificar la funcionalidad de la aplicación
- Software: Contiene el software de PostgreSQL, PostGIS y Mapserver necesarios para ejecutar la aplicación.

Requerimientos previos:

El sistema puede ser instalado sobre plataforma Windows XP, Linux o ambas ya que sus componentes están basados en herramientas de código abierto. Sin embargo se recomienda utilizar una sola plataforma para maximizar compatibilidad.

Los requerimientos de hardware dependen de los requerimientos del software a ser instalado que básicamente contiene Mapserver, Apache y PostgreSQL.

Se recomienda instalar el motor de base de datos en un servidor dedicado y con gran capacidad de memoria y procesamiento puesto que el consumo de recursos es alto para la determinación de la ruta óptima.

Procedimiento de instalación Mapserver, Apache y PostgreSQL sobre Windows:

Para instalar Apache descomprima el paquete ms4w e inicie el servidor Apache, el servidor ya está configurado automáticamente para trabajar con Mapserver y PHP.

Ahora, para instalar PostgreSQL, si se está instalando la versión 8.2 o inferior, se debe instalar sin las extensiones PostGIS puesto que éstas corresponden a una versión anterior. Luego, instalar la última versión de PostGIS que sea compatible con la versión 8.2.

Si está instalando la versión 8.3 o mayor de PostgreSQL, se debe instalar con las extensiones PostGIS puesto que ésta versión de Postgres incluye una versión actualizada de PostGIS que es compatible con la versión de Postgres

Procedimiento de instalación de Postgres sobre Solaris:

Se debe instalar PostgreSQL poniendo el valor correcto de la ubicación de la biblioteca requerida para la compilación en la variable de entorno de LD_LIBRARY_PATH.

El siguiente paso es configurar, compilar e instalar Postgres con make.o gmake.

Para poder instalar correctamente PostGIS es necesario instalar el paquete Geos. De igual forma es necesario configurar, compilar e instalar Geos utilzando make.

Luego, se necesita configurar PostGIS con la opción geos ya sea en la línea de comando o en el archivo de configuración de Make. Finalmente compilar e instalar PostGIS

Ahora es necesario inicializar un cluster de datos. Para esto puede ser necesario tener privilegios de super administrador dependiendo de la ubicación del repositorio. Esto se realiza a través de la siguiente línea de comandos.

initdb -D /usr/local/pgsql/data

Para inicializar la base de datos se utiliza la siguiente instrucción especificando el lugar donde se encuentra el cluster de datos

pg_ctl start -D /usr/local/pgsql/data

Para crear una base de datos nueva llamada quito en base a una plantilla template_postgis, se utiliza la instrucción la

createdb quito –T template_postgis

Creando una base de datos geográfica

Utilizamos template_postgis para crear una base de datos llamada quito desde el pgAdmin en el caso de Windows o desde la línea de comandos. Para Windows use la codificación SQL_ASCII

En el caso de instalar en Solaris es necesario ejecutar los siguientes comandos. Para crear la base de datos ejecutamos:

createdb quito

Para poder utilizar el lenguaje Plpg/SQL en el cual están escritas la mayoría de funciones de éste proyecto se necesita ejecutar el siguiente comando

createlang plpgsql quito

Finalmente para poder utilizar las facilidades de las extensiones geográficas de PostGIS es necesario ejecutar

psql –d quito –f /usr/local/pgsql/shar/lwpostgis.sql

El comando psql también permite ingresar a el ambiente de línea de comandos de Postgres que permite ejecutar comandos SQL y las funciones definidas.

Transformar los shapefiles en una base de datos de PostgreSQL

El siguiente paso es Transformar la base de datos proporcionada por la EMAAP-Q en una base de datos de Postgres. Para llevar a cabo esto usamos el programa de línea de comandos shp2pgsql.

Ésta aplicación está localizada en el directorio raíz de PostgreSQL, por lo tanto, es necesario crear un path a este directorio. En Windows por lo general el path estará en C:\Program Files\PostgreSQL\8.2\bin. En un ambiente basado en Unix por lo general estará en el directorio /usr/bin o incluso puede que ya esté agregado en el path en el momento de la instalación. Para convertir los shapefiles uioev.shp y uiopredios.shp utilizamos los siguientes comandos

shp2pgsql uioEV public.uioev > uioev.sql

shp2pgsql uioPredios public.uiopredios > uiopredios.sql

Éstas líneas producen archivos sql que pueden ser utilizados para insertar la información dentro de la base de datos quito utilizando el siguiente comando

psql -U postgres -d quito -f uioev.sql

Crear o Modficar el mapfile

Un mapfile es un mapa jerárquico utilizado para cargar la información geográfica y desplegarla en forma de imagen. Existe un modelo llamado test.map sobre el cual se trabajó en éste proyecto que puede ser utilizado como plantilla. Éste archivo se encuentra ubicado en la carpeta /Codigo Fuente/Mapserver.

Creando y llenando las tablas de la base de datos

Para crear las tablas se necesita ejecutar el siguiente comando que hace referencia a los archivos que contienen código SQL para crear tablas y agregar las restricciones necesarias a la base de datos.

psql –U postgres –d dbname –f create_tables.sql

Donde create_tables es el archivo que se encuentra en la carpeta /Codigo Fuente/Postgres del DVD que acompaña a ésta tesis.

Luego es necesario crear las funciones y procedimientos escritos en Plpg/SQL que se encuentran en agrupados en un solo archivo en la misma carpeta anteriormente mencionada

psql –U postgres –d dbname –f create_functions.sql

Dentro de la consola de línea de comandos psql usamos los siguientes comandos en el orden indicado para completar la base de datos con toda la información necesaria:

Para llenar las tablas de nodos y de time_cost utilizamos

SELECT populate_nodes();

SELECT populate_time_cost();

Finalmente utilizamos los siguientes comandos para precalcular las rutas óptimas

SELECT find_routes(1,1);

SELECT find_all_routes();

La primera línea calcula las rutas para el primer día y la primera hora, la segunda calcula todas las rutas para toda la base de datos

Configuración del Servidor de Base de Datos

Para el servidor de Base de Datos se recomienda usar la configuración del archivo postgresql.conf ubicado en la carpeta /Configuracion como base para configurar el motor de base de datos de PostgreSQL.

Para incrementar el desempeño de la aplicación se recomienda establecer valores altos para los parámetros shared_buffers y max_transaction_locks.

Iniciar la aplicación

Para iniciar la aplicación es necesario copiar la carpeta htdocs que contiene el archivo localizacion.php sobre la carpeta htdocs del directorio de Apache, que a su vez se encuentra dentro del directorio principal de Mapserver .

Es necesario especificar dentro del archivo de PHP donde se encuentra localizado el mapfile.

Luego nos aseguramos de que el servidor de Apache este levantado. De ser así podremos ingresar localmente en un navegador mediante la dirección

http://localhost/localizacion.php

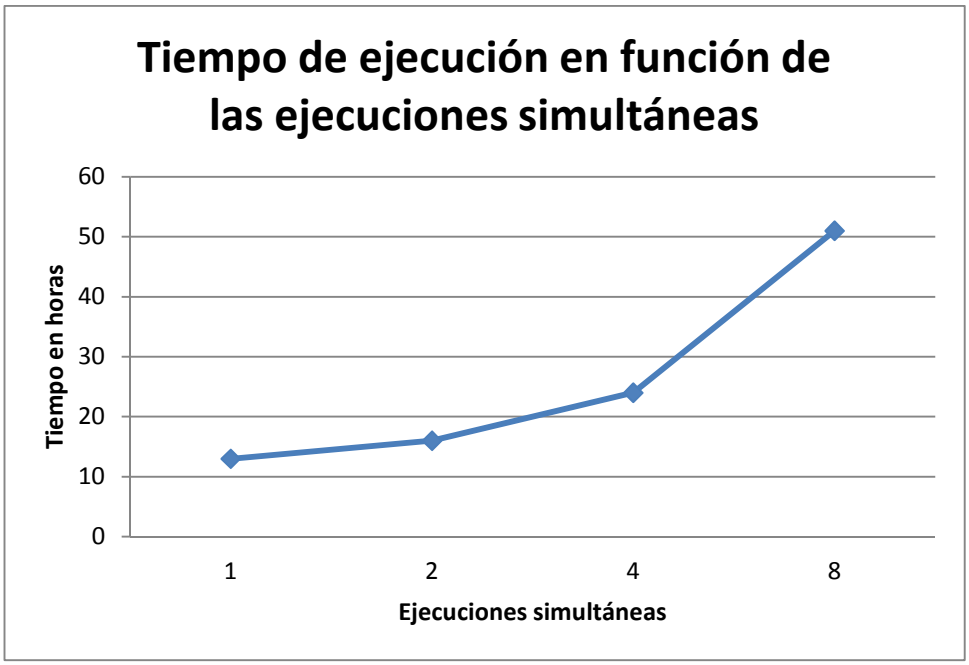
Backup

Para evitar la re calculación de las rutas óptimas se ha incluido un backup de las tablas routes y route_tracks de la base de datos que ya tiene precalculadas todas las rutas para el primer día, que se encuentra en el directorio Backup. Esta se puede recuperar a través del comando pg_restore.

e. Código Fuente del Sistema

f. Medidas de desempeño del algoritmo de Dijkstra en un servidor Sun Blade 1500

Figura Af. Tiempos de ejecución en función de tareas simultaneas



Los tiempos obtenidos fuero respectivamente:

- 1 ejecución: 13 horas 8 minutos
- 2 ejecuciones simultáneas: 16 horas 5 minutos
- 4 ejecuciones simultáneas: 24 horas 22 minutos
- 8 ejecuciones simultáneas: sobre las 51 horas

En la última ejecución no se terminó el cálculo debido a falta de memoria asignada a los shared buffers.

g. Especificaciones del servidor Sun Blade 1500