UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ANÁLISIS DE MOVILIDAD URBANA USANDO ALGORITMOS BIO-INSPIRADOS

TESIS

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA

HUMBERTO DEL ÁNGEL GARCÍA

TUTOR

DR. CARLOS GERSHENSON GARCÍA

IIMAS

MÉXICO, D.F., MAYO DE 2016

* * *

No he perdido ante la dificultad de los retos, sino ante el tiempo.

Leonardo da Vinci.

Quienes se inspiran en otra cosa que la naturaleza -maestra de los maestros- se agotan en vano.

Leonardo da Vinci.

Podemos y debemos hacer un buen trabajo. Hay que hacer las cosas con fe. Estamos compitiendo con Newton y con Galileo... Si no tenemos aparatos adecuados, los inventaremos.

Arturo Rosenblueth.

* * *

Agradecimientos

A mi madre Imelda García, por su amor, comprensión, consuelo y sobre todo su apoyo, y en general por todo su sacrificio a lo largo de mi vida.

A mis amigos y compañeros de generación por su ayuda, compañía y apoyo en esta etapa.

A Imelda Escamilla, gracias por todos sus consejos y ayuda desinteresada.

A Jorge, Karla, Abel, Evelyn, Sandra y Thalia, gracias por la motivación, el apoyo y por no desalentar mis locuras. Gracias por no dejarme naufragar en mi mismo.

A la UNAM por haberme dado la oportunidad de continuar mi formación académica en un posgrado de calidad. Al CONACYT por el financiamiento que me permitió realizar estos estudios; al personal del IIMAS, especialmente a Lulú, Amalia, Álvaro, y al Dr. Jorge Ortega, gracias por todo el apoyo brindado.

Contenido

Índice de figuras

Índice de tablas

Nomenclatura utilizada

Variables utilizadas en el capítulo 4.

Variable	**Significado**
$d(P_1, P_2)$	Distancia geográfica desde el punto P_1 hasta P_2
R_T	Díametro promedio de la tierra en Km con un valor de 6378.137 Km
lon_1,lat_1	Coordenadas para el punto P_1 en radianes
lon_2,lat_2	Coordenadas para el punto P_2 en radianes
Δ_{lon}	Diferencia entre lon_1 y lon_2 en radianes
h	Hora
h_{ini}	Hora de inicio para realizar una consulta sql
t_{reg}	Tiempo de cada registro en la base de datos
h_{fin}	Hora final para realizar una consulta sql

Principales variables utilizadas en el capítulo 5.

Variable	**Significado**
$\mathcal{B}(T_i, v_j)$	es la búsqueda binaria del elemento v_j entre los elementos de T_i
D	Arreglo que almacena la distancia total recorrida por cada hormiga
d_{max}	Distancia máxima hasta la cual se percibe el rastro de feromonas
$d\left(v_i, v_{i,k}\right)$	Distancia geográfica desde el vértice v_i hacia el k-ésimo vértice conectado a aquel
$d_{\overrightarrow{v_i v_j}}$	Distancia del vértice v_i al v_j
E	Arreglo de eventos de medidas $3 \times q_H$
e	Evento
$f\left(t - t_{(k,H_i)}\right)$,$f(\Delta t)$	Distancia recorrida por una hormiga en un intervalo de tiempo
$F_{(v_i \to v_j)}$	Cantidad de feromona perceptible desde el vértice v_i hacia v_j
$\mathcal{F}\left(\overrightarrow{v_A v_B}\right)$	Cantidad de feromona perceptible desde el vértice v_A en dirección a v_B
$G(v_i)$	Grado del vértice v_i
H_i	Hormiga i-ésima
M	Matriz de hormigas, cuyas medidas son $q_H \times 6$
M_S	Segunda columna de la matriz M, corresponde a los tiempos de salida de cada hormiga
M_L	Corresponde a la segunda columna de la matriz M
$\mathcal{M}_{\mathcal{S}}$	Variable de codificación para indicar la salida de un vértice
$\mathcal{M}_{\mathcal{L}}$	Variable de codificación para indicar la llegada a un vértice
n	Cantidad de vértices que tiene la gráfica
$\mathcal{N}(v_i, v_{i,k})$	Nivel de feromonas perceptibles desde el vértice v_i hacia el k-ésimo vértice conectado a v_i
$P_{v_i, v_{i,k}}$	Probabilidad de elegir la arista que conduce al k-ésimo vértice conectado a v_i

Variable	**Significado**
q_H	Cantidad de hormigas empleadas en el algoritmo *ACO*
R	Arreglo de listas ligadas que guarda los vértices visitados por cada una de las hormigas
R_i, R_{H_i}	Lista ligada que almacena los vértices visitados por la hormiga i-ésima
$\overrightarrow{R}$, $\overrightarrow{v_i v_f}$	Segmento de recta auxiliar que parte del vértice actual hacia la comida
$\overrightarrow{R_\perp}$	Recta perpendicular a $\overrightarrow{R}$
s_{g_f}	Sentido al que hay que girar para encontrar la comida
$T_{i,2}$	Distancia geográfica del vértice v_i hacia cada uno de los vértices almacenados en $T_{i,1}$
T	Lista de adyacencia que almacena a la triangulación *Delaunay* restringida
T_i, T_{v_i}	Información concerniente al vértice v_i
$T_{i,1}$	Vértices conectados al vértice v_i
$T_{i,3}$	Nivel de feromonas perceptible desde v_i hacia cada uno de los vértices guardados en $T_{i,1}$
t	Tiempo actual de la simulación
t_0	Tiempo inicial de la simulación
t_r	Tiempo de referencia usado en la simulación
$t_{(k,H_i)}$	Tiempo en que la hormiga i-ésima salió del k-ésimo vértice visitado por ella
$t_{\overrightarrow{v_i v_j}}$	Tiempo que le tomara a una hormiga cualquiera ir del vértice i al j
t_{S_k}	Tiempo de salida del vértice más reciente por parte de la hormiga k
t_{L_k}	Tiempo de llegada al siguiente vértice por parte de la hormiga k
U	Arreglo auxiliar para el proceso aleatorio de elección de aristas
$\widehat{u}$	Variable aleatoria con distribución uniforme
v_i	Vértice i-ésimo
$v_{i,k}$	k-ésimo vértice conectado al vértice i
$v_{(k,H_i)}$	k-ésimo vértice visitado por la hormiga i-ésima
v_{cte}	Velocidad constante a la que se mueve cada una de las hormigas
v_H	Vértice en el que se encuentra el hormiguero (nido)
v_f	Vértice en el que se encuentra la comida
V	Arreglo de listas ligadas que guarda las hormigas que han pasado por cada vértice
V_{v_H}, V_H	Lista de hormigas que han salido del vértice del hormiguero
$\overrightarrow{v_i v_{i,k}}$	Vector desde v_i hasta el k-ésimo vértice conectado a v_i
W	Nivel de feromonas perceptibles de las aristas que están del mismo lado que la comida
$\widehat{W}$	Arreglo W normalizado
x_{v_i}, y_{v_i}	Coordenadas geográficas del vértice i
x_{v_f}, y_{v_f}	Coordenadas geográficas del vértice donde se encuentra la comida
$x_{g_{min}}$	Longitud mínima de la gráfica
$x_{g_{max}}$	Longitud máxima de la gráfica
Δt	Intervalo de tiempo

Resumen

Los movimientos humanos exhiben altos niveles de regularidad, tanto temporales como espaciales, y son impulsados por la distribución geográfica de los lugares de interés a la vez que tienden a ser obstaculizados por la distancia que se recorre hacia ellos, sin embargo las oportunidades que dichos sitios ofrecen son un incentivo pese a la impedancia de transporte [1].

Los esfuerzos de investigación sobre los patrones de viaje individuales en períodos de tiempo largos se apoyan generalmente en una distancia de viaje, característica de cada persona, independiente del tiempo y una probabilidad significativa de volver a unos pocos lugares muy frecuentados; esto se supo a través de encuestas tradicionales, las cuales son costosas de implementar e implican limitaciones intrínsecas como cubrir un gran número de individuos y algunos problemas de fiabilidad. En contraste, los datos *GPS* (*Global Positioning System*, Sistema de posicionamiento global) se producen a un costo muy bajo como subproducto de una infraestructura de detección que es operativa. Los teléfonos inteligentes permiten la detección y colecta masiva de datos espaciotemporales, tales como los registros detallados de llamadas desde teléfonos móviles y las rutas *GPS* de los dispositivos de navegación, que representan un amplio espectro de la movilidad humana [2].

El trabajo expuesto en esta tesis tiene como principal objetivo hacer una búsqueda y análisis de patrones de movilidad urbana mediante una combinación de algoritmos de minería de datos, de ruteo y geométricos para procesar un conjunto de datos geo-espaciales, al hallar dichos patrones se hace posible hacer propuestas de movilidad entre distintas áreas de la ciudad que satisfagan en cierta medida a los sitios origen, destino y aquellos de interés mediante su agrupación.

En esta tesis se ha hecho uso de una base de datos abierta proporcionada por la teléfonica *Orange* a través del reto *D4D* (*Data for development*, http://www.d4d.orange.com/en/Accueil), la cual simplemente contiene el historial de llamadas de una parte de sus clientes junto con las posiciones geográficas de las antenas celulares que dan servicio a Senegal.

El núcleo del análisis que se presenta es un algoritmo de ruteo para inferir los trayectos más probables entre los distintos puntos geográficos que indica el historial de llamadas de cada usuario y con el uso de minería de datos obtener estadísticas de los sitios de interés más importantes, asimismo para visualizar mapas de calor de las regiones más transitadas y relaciones potenciales entre éstas.

Utilizando minería de datos se obtuvieron los puntos origen y destino más populares durante cada hora del día además de las rutas parciales por las que transita la gente, éstas no muestran el trazo completo a través de celdas adyacentes por lo que fue necesario el uso de un algortimo de ruteo y las aristas de la triangulación restringida para inferir los puntos intermedios en aquellas, esto permitió captar los movimientos urbanos los cuales son representados utilizando mapas de calor.

Por lo anterior se propuso un algoritmo de ruteo basado en el paradigma de optimización por colonia de hormigas (*ACO, Ant Colony Optimization*) para la búsqueda de las rutas que producen las distancias más cortas entre todos los pares de vértices en una gráfica en donde cada uno de estos representa la posición geográfica de una antena y que a su vez le corresponde una única región de *Voronoi* que es interpretada como una zona de interés en la ciudad. Se intuía que era posible diseñar un algoritmo que usara menos recursos que los algoritmos empleados tradicionalmente para el ruteo, y que mediante las optimizaciones correctas, la cantidad adecuada de agentes y relativamente poco esfuerzo, la calidad de las soluciones encontradas podrían compararse con la del algoritmo de *Dijkstra* [3]. Las pruebas del diseño propuesto mostraron que el algoritmo converge a una solución que es 10% (máximo) más larga que las soluciones que encuentra *Dijkstra*, además se observó que en ejecuciones continuas, para el mismo par de vértices, se pueden hallar distintas soluciones que igualmente no sobrepasan el máximo mencionado, sin embargo se consideró que esta diversidad en las soluciones halladas por el algoritmo *ACO* proporcionaría mapas de calor

inestables por lo que el algoritmo de ruteo que se empleo finalmente para inferir los sitios intermedios entre los puntos de las rutas encontradas en la base de datos fue el algoritmo de *Dijkstra*.

Así pues, se buscó la manera más computacionalmente eficiente de analizar el conjunto de datos mencionado y proponer una solución a la movilidad basada en los resultados obtenidos, mientras que el diseño y programación del *ACO* terminó siendo un producto secundario en el presente trabajo.

En el capítulo 1 se dan a conocer el planteamiento del problema, los objetivos del proyecto, sus alcances y limitaciones junto con la propuesta de solución.

En el capítulo 2 se introducen algunos de los modelos propuestos de sistemas de hormigas así como sus parámetros, al igual que algunos conceptos importantes de estructuras de datos y algoritmos geométricos. Asimismo es presentado el problema del análisis de movimientos urbanos y la importancia del uso de la tecnológia y la explotación de bases de datos.

Posteriormente, en capítulo 3 la congruencia metodológica utilizada en el presente trabajo es descrita.

En tanto, en el capítulo 4 se presenta el diseño algorítmico de la variante del algoritmo *ACO* el cual se describe, al igual que la lógica empleada para cada una de sus partes.

Los resultados estadísticos de los diversos experimentos en los que se comparó el desempeño del algoritmo *ACO* propuesto respecto al algoritmo de *Dijkstra* para realizar las mismas tareas de ruteo se exponen en el capítulo 5.

Finalmente las conclusiones y el trabajo futuro son mencionadas en el capítulo 6.

Los resultados obtenidos del análisis de datos geo-espaciales muestran la teselación de la urbe que fue objeto de estudio, usando el diagrama de *Voronoi* así como una triangulación *Delaunay* restringida que representa las relaciones existentes entre las divisiones del territorio. Como parte de los resultados se encontraron relaciones permanentes que existen entre algunas celdas de *Voronoi*, su agrupamiento junto con los puntos más populares de origen y destino hallados concentran los movimientos más importantes que ocurren en la ciudad, de una manera general. Finalmente se presenta a este agrupamiento como una propuesta de solución que satisface parte de las rutas encontradas en conjunto con la medicion de su posible impacto.

En el apéndice se adjunta un mapa ampliado de la ciudad que fue objeto de estudio así como la lista de adyacencias de los puntos geográficos utilizados para dividirla con una triangulación *Delaunay* restringida.

Capítulo 1

1.1.- Introducción

Los movimientos humanos exhiben altos patrones de regularidad, mismos que se ven obstaculizados por la distancia geográfica, hora del día, tráfico, entre otras variables.

Frecuentemente se hacen esfuerzos para mitigar los problemas causados por el desplazamiento humano constante, tales como el tráfico vehicular, el deterioro ambiental, uso de recursos, exceso de ruido, entre otros; mismos que merman la calidad de vida. Para conocer la raíz de aquellos hay que hacer primero un análisis origen-destino para corregir (o atenuar) el problema, una forma de hacerlo es mediante encuestas tradicionales de movilidad.

Por la cantidad misma de individuos, las respuestas de la población encuestada generalizan a las de la población entera. Es por ello que las encuestas tradicionales arrojan resultados insuficientes y son bastante caros en términos económicos y temporales, y que además ocasionan que sus logros no sean del todo satisfactorios a largo plazo.

Una alternativa bastante viable, es el uso de datos geo-espaciales brindados por la propia población; estos pueden ser almacenados de manera regular en una base de datos y ser analizados simultáneamente por un sistema informático. La recolección masiva de datos brinda una mayor claridad del amplio espectro de la movilidad y distintos algoritmos pueden obtener distintos resultados importantes en un tiempo bastante aceptable y sustancialmente mayor al de las encuestas tradicionales. Lo cual puede derivar directamente en la mejora o reestructuración de los sistemas de transporte público existentes, o en su desarrollo en el caso de su inexistencia, e incluso repercutir con alcances mayores como el desarrollo urbano. Para el presente trabajo, se ha utilizado una base de datos abierta proporcionada por la empresa telefónica *Orange* mediante el reto *D4D* (*Data for development*, http://www.d4d.orange.com/en/Accueil), aquella determinó la forma en que fueron recolectados los datos al igual que su procesamiento antes de ser liberados al público.

Por otro lado, el poder de los algoritmos bio-inspirados, en particular la optimización por colonia de hormigas, proviene del conocimiento colectivo que se genera por cada agente en cada iteración, mismo que es usado hasta la convergencia del algoritmo a un óptimo aceptable o incluso global.

El presente análisis de movilidad no solamente resultaría benéfico para la población que fue objeto de estudio si no que además el mismo procedimiento puede ser utilizado para cualquier otra urbe, cambiando únicamente la base de datos geo-espacial, un diagrama de *Voronoi* de sus respectivos puntos geográficos y restringir adecuadamente su triangulación *Delaunay*. Incluso sería suficiente con que la base de datos contenga las coordenadas de los sitios que son visitados, una estampa de tiempo y un identificador de usuario, sin siquiera llegar a recurrir a información personal.

Más aún, el algoritmo de optimización por colonia de hormigas que se propone es capaz de competir con algoritmos tradicionales de ruteo y sus alcances van más allá de la planificación de trayectos a través de una ciudad. Por sus características aleatorias, es apto para dar más de una solución óptima (al menos localmente) lo que es una ventaja frente a otros métodos, pues mientras los procedimientos clásicos buscan siempre la ruta más corta posible para entregar los paquetes en una red, aquella podrá ser la más corta, pero no la menos transitada, debido a su popularidad; lo que provoca que se sature fácilmente y se pierda la ventaja en la rapidez de entrega, por lo que la característica de *holgura* del algoritmo bio-inspirado, que en un principio parecía adversa, se transforma en un beneficio.

1.2.- Contribuciones

Los estudios sugieren que los datos de rastreo del teléfono móvil representan una aproximación razonable para la movilidad individual y muestran un enorme potencial como una fuente alternativa de datos y un complemento a las encuestas de viajes convencionales en el estudio de la movilidad. La combinación de algoritmos bio-inspirados, geométricos y la minería de datos proporcionan información rica para apoyar el modelado urbano y la planificación metropolitana. Siendo así, el presente trabajo representa una alternativa novedosa, más eficiente y considerablemente más barata que los métodos tradicionales respecto a la planeación de movilidad urbana, este nuevo enfoque, aunado a la profunda penetración de la telefonía móvil, ofrece a las ciudades la posibilidad de supervisar de manera oportuna las interacciones de los ciudadanos y el uso de conocimientos basados en bases de datos para mejorar la planificación y gestión de los servicios, lo que sin duda supera los límites de las encuestas. Además, utilizar los datos de teléfonos móviles para realizar el análisis y la optimización de tránsito representa una nueva frontera con impacto social significativo, sobre todo en los países en desarrollo.

Otra de las contibuciones es una nueva variante del algoritmo de optimización por colonia de hormigas, el cual se distingue de otros modelos basando su desempeño computacional en la cantidad de vértices que contiene un grafo, este algoritmo se planteó para resolver el problema de la búsqueda de la ruta más corta entre todos los pares de vértices presentes, ejecutándose en un tiempo menor o igual al que ocupan otros métodos más tradicionales como *Dijkstra*, y que además produce soluciones con una calidad bastante aceptable. El poder de los algoritmos bio-inspirados, en particular la optimización por colonia de hormigas, proviene a través del conocimiento colectivo que se genera en cada iteración, y que proporciona hasta su convergencia a un óptimo aceptable o incluso global.

1.3.- Objetivo general

Hacer una búsqueda y análisis de patrones de movilidad urbana mediante una combinación de algoritmos de minería de datos, bio-inspirados, geométricos para procesar un conjunto de datos geo-espaciales. Todo esto como un esfuerzo para mitigar los problemas causados por el desplazamiento humano constante, tales como el tráfico vehicular, el deterioro ambiental, uso de recursos, exceso de ruido, entre otros; mismos que merman la calidad de vida. Para conocer la raíz de aquellos hay que hacer primero un análisis origen-destino el cual develará los patrones de movilidad presentes, y con ellos será posible corregir (o atenuar) el problema mediante la agrupación de los puntos de interés más importantes en conjunto con los sitios de origen y destino más populares, lo cual derivará directamente en la mejora o reestructuración de los sistemas de transporte público existentes, o en su desarrollo en el caso de su inexistencia, e incluso repercutir con alcances mayores como el desarrollo urbano.

1.4.- Objetivos particulares

- Realizar una fase exploratoria del contenido de los datos para determinar el tipo de métodos que deben considerarse para obtener información de interés.
- Preprocesar los datos para hacer un filtrado eficiente.
- Utilizar minería de datos para dejar únicamente aquellos registros que aportan información de movilidad.
- Hacer un mapa de calor de las rutas obtenidas de los datos en bruto y determinar la necesidad de un procesamiento altenativo.

- Dividir a la ciudad usando el diagrama de *Voronoi* y restringir su gráfica dual, la triangulación *Delaunay*, de tal manera que se establezcan lazos aceptables entre las celdas de *Voronoi* y que estén manifestadas con la presencia de las aristas de la triangulación.
- Redirigir las rutas obtenidas de los datos en bruto a través de una triangulación *Delaunay* restringida para todos los vértices que son visitados.
- Hacer un mapa de calor de las relaciones potenciales más importantes entre celdas de *Voronoi*.
- Proponer y realizar un algoritmo de colonia de hormigas para encontrar las rutas más cortas entre todos los pares de vértices.
- Comparar el algoritmo de optimización por colonia de hormigas propuesto con el algortimo de *Dijkstra*.
- En adición a esto último se buscará trazar una o varias rutas alternativas de transporte con base en lo obtenido de la profundización en el conocimiento y minería de datos entre las celdas de *Voronoi* que, se descubra, sean sitios de interés y/o puntos de origen destino altamente potenciales. De igual manera se medirá el posible impacto en la cantidad de desplazamientos que satisfaría el desarrollo de infraestructura entre aquellas celdas.

1.5.- Planteamiento del problema

Senegal, un país en desarrollo en la costa occidental de África es el protagonista de este trabajo. Concretamente el estudio se centra en Dakar, la capital, al ser un centro importante de producción y la ciudad má grande de aquel país. El constante proceso de urbanización de las sociedades y la aparición de oportunidades y recompensas en diversos puntos geográficos deja en claro la necesidad de medios eficientes para transportarse a través de la urbe, la ausencia o ineficiencia de estos acarrea problemas sociales, económicos y ambientales, mismos que pueden ser prevenidos mediante el análisis del desplazamiento de los individuos.

Aun en una nación en vías de desarrollo la penetración de los teléfonos inteligentes está presente entre su población, un dispositivo así de cotidiano puede fácilmente convertirse en un aliado en el análisis de patrones de movilidad cuando un conjunto de datos específico es recolectado en forma masiva y continua. Su adecuado procesamiento revelará los comportamientos que se tienen en sectores, días y horas específicos y conducirá a resolver las necesidades que eran desconocidas o a aquellas situaciones que no se tenían previstas en una encuesta origen destino tradicional.

No se había presentado la oportunidad de empalmar tal diversidad de metodologías para tratar un problema que está presente desde las sociedades altamente industrializadas hasta las poblaciones menos favorecidas y en vías de desarrollo. Por un lado el poder de los algoritmos bio-inspirados, en particular la optimización por colonia de hormigas, proviene a través del conocimiento colectivo que se genera en cada iteración, y que proporciona hasta su convergencia a un óptimo aceptable o incluso global. Lo anterior sugiere que con la cantidad correcta de agentes y relativamente poco esfuerzo la calidad de las soluciones que se encuentren puede compararse con las conseguidas por otros medios. Y por otro lado, el uso de la minería de datos para obtener conocimiento que no es evidente a simple vista y el cual se aprovecha para detectar problemas y oportunidades

Todo esto rompe con los esquemas de las encuestas y análisis tradicionales que no proporcionan la información suficiente y que además son costosos en más de una forma y que en más de una ocasión se concluye que los resultados y propuestas de un análisis sólo satisfizo un problema en particular dejando a otros en espera pues ni siquiera se sabía de su existencia.

El presente trabajo se hizo para mejorar las condiciones de desplazamiento de Dakar, pues se trata de una ciudad con poca infraestructura vial, sistemas de transporte público improvisados en autos que normalmente consideraríamos de carga y sin rutas fijas. Pensando en todo ello, es posible mitigar el caos vial que provoca tal desorganización a la vez que se mejora la calidad de vida en general, aunado a que es una de las formas más baratas en las que podrían conseguir un análisis de calidad que destaque las mejoras en tiempo, desplazamiento y reordenación urbana que pueden conseguirse. Yendo más allá, este proyecto se pensó como una solución genérica para cualquier población que lo requiera haciendo uso de una intervención humana mínima.

1.6.- Justificación

Se hace especial énfasis en que los métodos tradicionales para encontrar la ruta más corta entre todos los pares de vértices en una gráfica conexa tienen un costo bastante alto algorítmicamente hablando, tanto en tiempo como en recursos computacionales.

El uso de datos geo-espaciales proporcionados por sistemas informáticos es una alternativa viable frente a las costosas encuestas origen-destino, puesto que aquellos ya existen y pueden ser programados para recolectar datos masivos en intervalos de tiempo determinados, aunado a la cantidad de usuarios que pueden cubrir sin que se vea reflejado en un costo mayor.

Uno de los primeros obstáculos de este tipo de proyectos es el procesamiento de un gran volumen de datos y el segundo mayor la complejidad algorítmica de los métodos que se usen para hacer el análisis de movilidad. El diseño de algoritmos y estructuras eficientes que conduzcan a un resultado óptimo es esencial para hallar una solución óptima.

Cuando se habla de optimización por colonia de hormigas para encontrar la ruta más corta entre un par de vértices en una gráfica, pocas veces se piensa en el tamaño de gráfica y la estructura computacional que la almacena, ni de las operaciones que se realizarán con ella. Muchos algorítmos de optimización por colonia de hormigas han sido propuestos pero rara vez analizados en cuanto a la cantidad de recursos temporales y espaciales que utilizan, y al ser implementados en aplicaciones muy pequeñas dan la falsa impresión de ser óptimas y poder extenderse a problemas más grandes conservando el mismo desempeño. En otros casos el reduccionismo aplicado en ellos predispone el resultado de los experimentos en los cuales se usan. Tanto en la forma en la que fueron diseñadas dichas variantes del algoritmo *ACO*, éstas no dan certeza de que sean adecuadas para un análisis de gran magnitud. Cuando se habla de movilidad urbana, hay que considerar el tamaño que ha de tener una red que represente adecuadamente una ciudad.

1.7.- Solución propuesta

A través del uso de la minería de datos, como una fase exploratoria, y una mezcla de algoritmos de calendarización, geométricos y bio-inspirados se pretende extraer las rutas en bruto que describe la población de Dakar a través de los desplazamientos que se hacen por la ciudad, a partir del adecuado procesamiento de estos se obtendrán las rutas que son transitadas durante todo el día. Intuitivamente la unión de éstas proporciona una idea de un conjunto de rutas potenciales que serían muy benéficas para la población de la urbe.

Este análisis es muy importante puesto que las rutas de transporte público en cualquier ciudad conllevan un gasto en infraestructura, misma que debe ser funcional durante años e incluso décadas; una mala planeación conlleva un gasto innecesario.

Aunado a lo anterior, se intuye que el uso de un algoritmo de optimización por colonia de hormigas para encontrar la ruta más corta entre todos los pares de vértices puede utilizar menos operaciones que los algoritmos voraces tradicionales proporcionando resultados igualmente, o al menos similarmente, óptimos.

1.8.- Alcances

- Utilizando la base de datos proporcionada por la empresa telefónica *Orange* se procesará de la manera más eficiente posible para obtener los puntos origen y destino más populares durante cada hora del día y también

para obtener las rutas parciales que se encuentren en los historiales de llamadas de cada usuario.

- Con base en el conocimiento adquirido por el estado del arte se diseñará un algoritmo de optimización por colonia de hormigas (*ACO, Ant Colony Optimization*) para encontrar la ruta más corta entre un par de vértices cualquiera.
- La calidad de las soluciones obtenidas con el algoritmo *ACO* propuesto se medirán comparando con aquellas que se obtienen usando el algoritmo de *Dijkstra* cuando se usa el mismo grafo y los mismos puntos de prueba.
- Para el análisis de movilidad se utilizará el mejor algoritmo, en donde los criterios para elegir a éste son la calidad de sus soluciones, complejidad algorítmica y estabilidad en las soluciones halladas.
- El algoritmo de ruteo será utilizado para inferir los puntos intermedios más probables entre los puntos geográficos que tengan las rutas parciales halladas.
- Se obtendrán mapas de calor para mostrar la cantidad de movimiento en Dakar para cada hora del día y de igual manera se hará una propuesta para unificar las celdas de *Voronoi* de los puntos origen y destino más populares junto con otros puntos de interés que sean encontrados, aunado a esto último se medirá la cantidad de relaciones entre celdas de *Voronoi* satisfechas, lo cual ayudaría en trabajos posteriores a la planificación de infraestructura.

1.9.- Límites

- No se hallarán las rutas más transitadas a través de las vialidades de Dakar, sino usando solamente la triangulación *Delaunay* restringida.
- Las relaciones potenciales entre celdas de *Voronoi* que resulten del análisis de los datos serán sólo un estimado que debería ser corroborado con un estudio de campo.
- No se realizarán estudios de campo para complementar este trabajo.
- No se implementará ni modelará el problema del agente viajero.
- No se pretende que el algoritmo *ACO* que se propone sirva para calcular rutas óptimas en tiempo real.
- El algoritmo *ACO* que se propone no busca simular ni reproducir los movimientos que hacen las personas en Dakar, ya que para eso se requiere información adicional que no fue incluida por la empresa Orange en la base de datos.
- El algoritmo *ACO* funcionará con sólo un vértice que servirá como hormiguero y otro en el cual se considera que está la comida que las hormigas buscan, es decir que para cada ejecución del algoritmo sólo se toman un par de vértices a la vez.
- No se ahondará en la combinación más óptima de las variables del algoritmo de optimización por colonia de hormigas.

Capítulo 2

2.1.- Estado del arte

Los movimientos humanos [1] exhiben altos niveles de regularidad y tienden a ser obstaculizados por la distancia geográfica. El origen de esta dependencia de la movilidad en la distancia, y la formulación de leyes cuantitativas que explican la movilidad humana sigue siendo, sin embargo, una cuestión abierta, la respuesta que daría lugar a muchas aplicaciones, por ejemplo, mejorar los sistemas de ingeniería, tales como la computación en nube y las recomendaciones basadas en la ubicación, mejorar la investigación en las redes sociales y dar una idea de una gran variedad de cuestiones sociales importantes, tales como la planificación urbana y la epidemiología.

La movilidad [1] es disuadida directamente por los costos (en tiempo y energía) asociados a la distancia física. Inspirado por la ley de gravedad de Newton, el flujo de personas se prevé que disminuya con la distancia física entre dos lugares, típicamente como una ley de potencias en función de la distancia. Además de la distancia, las versiones más complejas de modelos de gravedad también pueden considerar un parámetro que captura la *masa* del punto de partida y el destino de un viaje. En este caso, por lo general la población de una zona se utiliza como un *proxy* para cuantificarla. Estos llamados *modelos de gravedad* [1], tienen una larga tradición en la geografía cuantitativa y la planificación urbana y se han utilizado para modelar una amplia variedad de sistemas sociales, por ejemplo, migración humana, comunicación interurbana y análisis de los flujos de tráfico. Otro argumento es que no hay relación directa entre la movilidad y la distancia, y que la distancia es un sustituto para el efecto de oportunidades intermedias [1]. Así pues, los desplazamientos son impulsados por la distribución espacial de los lugares de interés, y por lo tanto por la respuesta a las oportunidades en lugar de la impedancia de transporte como se considera en los modelos de gravedad. Varios estudios estadísticos han demostrado que el concepto de intervención de oportunidades es una mejor explicación a una amplia gama de datos de movilidad.

Por otro lado, en contraste con las trayectorias aleatorias predichas por los modelos de vuelo y caminatas de Lévy predominantes, las trayectorias humanas muestran un alto grado de regularidad temporal y espacial, cada individuo se basa en una distancia de viaje característica independiente del tiempo y una probabilidad significativa de volver a unos pocos lugares muy frecuentados. Después de corregir las diferencias en las distancias de viaje y la anisotropía inherente de cada trayectoria, los patrones de viaje individual colapsan en una sola distribución de probabilidad espacial, lo que indica que, a pesar de la diversidad de su historial de viajes, los humanos siguen patrones simples que pueden ser reproducidos [4, 2].

Los esfuerzos de investigación anteriores sobre los patrones de viaje individuales en períodos de tiempo más largos se basan generalmente en los movimientos de las personas a través de encuestas tradicionales. La información obtenida se basa en cuestionarios que suelen ser costosas de implementar y con las limitaciones intrínsecas a cubrir gran número de individuos y algunos problemas de fiabilidad. En contraste, los datos *GPS* se producen a un costo muy bajo como subproducto de una infraestructura de detección que es operativa, mientras que las encuestas requieren grandes inversiones ad hoc. Estos esfuerzos, sin embargo, demostraron que los patrones de movilidad individuales están fuertemente relacionados con los patrones de uso del suelo, así como el entorno construido de una ciudad, y que los patrones de viaje diarios individuales exhiben una gran regularidad.

Se ha descubierto una relación entre la popularidad de una celda urbana y la probabilidad de que ésta sea visitada. En base al uso de métodos de estimación de densidad de *Kernel* se han construido mapas de densidad de la actividad en función del tiempo. El uso de este enfoque, también hace posible visualizar diferentes actividades

humanas en una ciudad y por lo tanto captar el pulso de las actividades humanas.

El análisis del movimiento ha sido fomentado por la amplia extensión en la difusión de las tecnologías inalámbricas, como el sistema de posicionamiento global y el de las redes de telefonía móvil. Estas infraestructuras de red, como un subproducto de sus operaciones normales, permiten la detección y colecta masiva de datos espacio-temporales, tales como los registros detallados de llamadas desde teléfonos móviles y las rutas *GPS* de los dispositivos de navegación, que representan un amplio espectro de la movilidad humana. Estos grandes datos de movilidad proporcionan un nuevo microscopio de gran alcance social, que puede ayudarnos a comprender la movilidad humana, y descubrir los patrones y modelos ocultos que caracterizan las trayectorias que los seres humanos siguen durante su actividad diaria [2].

La introducción de los servicios de localización en aplicaciones de medios sociales de los teléfonos inteligentes ha permitido a la gente a compartir sus opciones de actividad relacionadas (check-in) en sus redes sociales virtuales (por ejemplo, *Facebook*, *Foursquare*, *Twitter*, etc.) proporcionando una cantidad sin precedentes de datos generados por los usuarios sobre el movimiento humano y la participación de la actividad. Estos datos contienen información de geolocalización detallada, lo que refleja un amplio conocimiento sobre el comportamiento humano. Además, la información recolectada para cada registro de entrada se registra de tal manera que las actividades del usuario se pueden deducir. Por lo tanto los datos basados en localización nos ofrece una nueva dimensión de la información relacionada con las categorías de actividades humanas con mayores detalles [5].

El procedimiento automático de colecta de datos *GPS* (*Global Positioning System*, Sistema de posicionamiento global) se asegura de que todos los movimientos son capturados correctamente, mientras que las encuestas dejan espacio a omisiones o distorsiones. Los datos del *GPS* no proporcionan ninguna información semántica explícita sobre el propósito de los movimientos, el destino final, y los perfiles de los ciudadanos involucrados, mientras que las encuestas recogen explícitamente esta información [2]. Los estudios sugieren que los datos de rastreo del teléfono móvil representan una aproximación razonable para la movilidad individual y muestran un enorme potencial como una alternativa y con mayor frecuencia actualizable fuente de datos y un complemento a las encuestas de viajes convencionales en el estudio de la movilidad. Durante las dos últimas décadas, hemos visto una explosión en el despliegue de sistemas generalizados como las redes celulares, dispositivos *GPS*, y puntos de acceso *WiFi* que nos permiten capturar grandes cantidades de datos en tiempo real relacionados con las personas y las ciudades. El uso de estos conjuntos de datos podría permitir a los investigadores comprender mejor las leyes que rigen los movimientos de las personas y mejorar la eficiencia y la capacidad de respuesta de las políticas urbanas [6]. La profunda penetración de la telefonía móvil ofrece a las ciudades la posibilidad de supervisar de manera oportuna las interacciones de los ciudadanos y el uso de conocimientos basados en datos para mejorar la planificación y gestión de los servicios, sin embargo hay un largo camino desde los datos en bruto hasta las representaciones útiles de los comportamientos de movilidad, por lo que es necesario un proceso de búsqueda de conocimiento.

Utilizar los datos de teléfonos móviles para realizar el análisis y la optimización de tránsito representa una nueva frontera con impacto social significativo, sobre todo en los países en desarrollo. Los datos del teléfono móvil tienen una gran penetración incluso en los países en desarrollo, y ofrecen una alternativa más barata y más rápida de las encuestas costosas [7].

Visto desde otro enfoque, el análisis de la movilidad urbana puede verse como un problema de enrutamiento, tal como los protocolos de comunicaciones. El problema de enrutamiento para redes de comunicaciones se modela como un proceso dinámico de optimización combinatoria [8]. El problema a resolver por cualquier sistema de enrutamiento es dirigir el tráfico desde la fuente hacia sus destinos mientras algún criterio de eficiencia es maximizado. Actualmente, los algoritmos de enrutamiento se enfrentan a algunos importantes retos debido a la complejidad encontrada en las redes modernas. Por ejemplo, los algoritmos centralizados tienen problemas de escalabilidad, los algoritmos estáticos no pueden cambiar tan rápido como lo hace la topología de la red, los algoritmos distribuidos y dinámicos tienen problemas de estabilidad y oscilaciones. En general los algoritmos de enrutamiento deben hacer frente a los retos q2ue presentan las redes modernas; la ausencia de adaptación de los algoritmos a los cambios frecuentes en la red, a las capacidades del nodo, al modelo de tráfico, al volumen, a la disponibilidad de energía, entre otros, reducen la eficiencia de la red. El problema, sea tanto de movilidad urbana o tráfico de paquetes, se define como un problema de optimización combinatoria distribuida, el cual por sí mismo es un reto en el campo de optimización combinatoria.

Los algoritmos centralizados suelen tener problemas de escalabilidad, y la red es incapaz de recuperarse por sí misma en caso de un fallo de la fuente de alimentación. El enrutamiento estático asume condiciones invariantes en el tiempo en la red, lo que es poco realista en la mayoría de los casos. Los esquemas de encaminamiento adaptativo también tienen problemas, incluso inconsistencias cuando hay errores de nodo o una gran cantidad de oscilaciones en la carga de trabajo.

El software móvil y los agentes autónomos son capaces de ser adaptables, cooperativos, y se pueden mover de una

manera inteligente de un lugar a otro en la red. Las propiedades inherentes de los sistemas de hormigas artificiales incluyen la escalabilidad, la posibilidad de descubrir nuevas formas y soluciones, y la inteligencia, las interacciones locales simples y la comunicación a través del sistema, que son características deseables para la mayoría de los tipos de redes. Por un lado, el enrutamiento mínimo permite a los agentes seguir su camino con costos mínimos, mientras que el enrutamiento no-mínimo tiene más flexibilidad al elegir su camino mediante el uso de la heurística. Se puede pensar en la inteligencia artificial como una ciencia que trata de incorporar conocimiento a los procesos que realiza una máquina, para que estos se ejecuten con éxito. Los algoritmos bio-inspirados le imprimen naturalidad a dichos sistemas y poco a poco se van refinando para asemejarse a aún más a los métodos utilizados por la naturaleza. La importancia de estudiar estos algoritmos radica en que los métodos inspirados en el comportamiento biológico han probado ser eficaces en la solución de numerosos problemas de ingeniería que, si se aplicarán los métodos tradicionales, resultarían en problemas demasiado costosos de resolver.

Una de las concepciones de la inteligencia artificial es interpretar qué heurístico es el calificativo apropiado para los procedimientos que, empleando conocimiento acerca de un problema y de las técnicas aplicables, tratan de aportar soluciones (o acercarse a ellas) usando una cantidad de recursos razonable. En un problema de optimización, aparte de las condiciones que deben cumplir las soluciones factibles del problema, se busca la que es óptima según algún criterio de comparación entre ellas. A diferencia de las heurísticas, las metaheurísticas pueden ser vistas como un marco general algorítmico, que puede ser aplicado a diferentes problemas de optimización con mínimos cambios para ser adaptado a un problema específico. Las metaheurísticas son ampliamente reconocidas como una de las mejores aproximaciones para atacar problemas de optimización combinatoria.

2.2.- Optimización por colonia de hormigas

A continuación se expondrán en breve el funcionamiento general y las características más relevantes que distinguen a los diferentes modelos de optimización por colonia de hormigas hallados en la literatura.

La optimización por colonia de hormigas (*ACO, Ant Colony Optimization*) es un paradigma de optimización moderno inspirado por el comportamiento de las colonias de hormigas, y que en términos muy generales, funciona de la siguiente manera: cuando una hormiga busca comida en su medio ambiente, deposita feromonas en el suelo mientras se mueve alrededor, estos químicos pueden ser percibidos por otras hormigas que a su vez atraen a otras hormigas. Esto se ilustra en la figura 2.1.

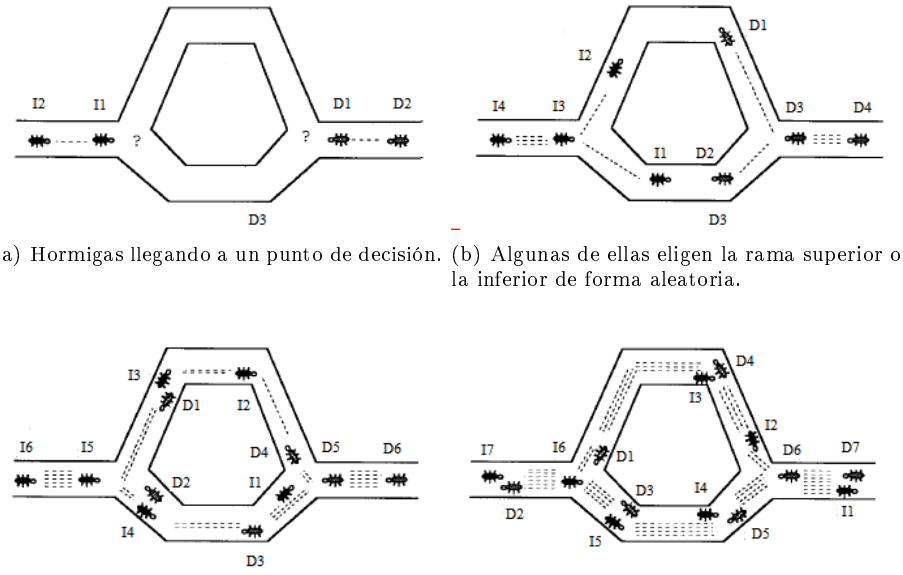
Cuando una hormiga encuentra la fuente de comida, tiende a caminar de regreso usando su propio rastro, invirtiendo en él más feromonas. Otras hormigas son atraídas por este rastro. Si el camino es corto, el trayecto se ve incrementado por feromonas rápidamente. De esta manera si varias hormigas encuentran diferentes caminos entre el hormiguero y la comida, es justamente la forma en la que la colonia entera converge hacia el camino más corto. Es esta inteligencia colectiva lo que ha inspirado el paradigma *ACO*.

La simulación en *ACO* sobre el comportamiento de las hormigas reales es mediante el uso de hormigas artificiales, las cuales son capaces de aprender sobre el espacio de búsqueda durante la ejecución del algoritmo y usan esta experiencia adquirida para construir, mejores soluciones en cada iteración. Este proceso de construcción puede entenderse como una toma secuencial de decisiones regida por una regla de transición estocástica. Parte esencial de los algoritmos *ACO*, es la combinación de información heurística y el rastro de feromona, el cual mide qué tan deseable es un nodo con base en lo que han aprendido las hormigas. [9].

Un parámetro τ refleja el rastro de feromonas, mientras que η refleja la información heurística (visibilidad). Ambos parámetros son utilizados en el algoritmo para calcular la probabilidad que perimitirá a cada hormiga decidir cómo moverse a través del grafo.

El primer algoritmo propuesto dentro de la metaheurística *ACO* fue el Sistema de hormigas (*AS, Ant System*), propuesto por *Dorigo*. Dado un conjunto de n ciudades, el problema del agente viajero (*TSP, Travelling Salesman Problem*) puede ser visto como el problema de encontrar la ruta más corta para visitar todo el conjunto de ciudades, cada una sólo una vez. Donde la distancia que separa a las ciudades es euclideana. De manera general, en el algoritmo, se posiciona un número de hormigas $b_i(t)$ en la ciudad i al tiempo t; de manera que la cantidad total de hormigas m [10] en el grafo está dado por la siguiente ecuación:

$$m = \sum_{i=1}^{n} b_i(t)$$



(a) Hormigas llegando a un punto de decisión. (b) Algunas de ellas eligen la rama superior o la inferior de forma aleatoria.

(c) Algunas llegan al otro lado usando la ruta más corta. (d) Las feromonas se acumulan mayormente en el trayecto más corto.

Figura 2.1: Dorigo, M. y Gambardella L., Hormigas buscando el camino más corto, 1997. Imagen tomada de `http://goo.gl/7mLbqZ` (Fecha de actualización 09 de mayo de 2016)

Para el modelo de *Dorigo*, cada hormiga es un agente simple que yendo de una ciudad i a una j deposita un rastro en la arista ij; que además puede escoger la ciudad a la cual ir con cierta probabilidad la cual está en función de la distancia y de la cantidad presente del rastro en la arista correspondiente; aunado a lo anterior se forza a los agentes a evitar visitar las ciudades en las que ya estuvieron hasta que el *tour* se completa.

Siendo $\tau_{ij}(t)$ la intensidad del rastro en la arista ij al tiempo t [10], cada iteración del algoritmo se convierte en:

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t, t+1)$$

Donde ρ es un coeficiente tal que $1-\rho$ representa la evaporación del rastro y donde la variable $\Delta\tau_{ij}(t, t+1)$ corresponde a la variación de la intensidad del rastro de feromonas en la arista ij del instante de tiempo t al instante $t+1$. El parámetro $\rho < 1$ evita la acumulación sin límite de feromonas. Con el objetivo de satisfacer la restricción del problema del agente viajero, cada hormiga se asocia a una estructura de datos, llamada la lista *tabú*, la cual alberga la lista de ciudades ya visitadas hasta el tiempo t y prohibe que la hormiga las visite nuevamente antes de que el *tour* esté finalizado. Cuando un *tour* es completado la lista *tabú* es vaciada y la hormiga es libre de elegir su camino nuevamente. Se define a $tabu_k$ como un vector que contiene la lista *tabú* de la hormiga k-ésima, y $tabu_k(s)$ es el s-ésimo elemento de la lista *tabú* de la lista que pertenece a la hormiga k-ésima.

Cada una construye una solución factible al problema del agente viajero, al aplicar de manera iterada la regla de transición [10] que combina τ y η para elegir con cierta probabilidad p_{ij} la siguiente ciudad j a visitar desde la ciudad i, esto se expresa en la siguiente ecuación:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}(t)]^\beta}{\sum_{j \in allowed} [\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}(t)]^\beta} & \text{si } j \in \text{ allowed} \\ 0 & \text{Otro caso} \end{cases}$$

Donde $allowed = \{j : j \in tabu_k\}$ y los parámetros α y β determinan la importancia de la feromona y la información heurística, respectivamente. Además se le llama visibilidad al parámetro η_{ij} el cual es el recíproco de la distancia de la ciudad i a la ciudad j, es decir $\frac{1}{d_{ij}}$ [9, 10].

La calidad de la solución obtenida se incrementa cuando el número de hormigas trabajando para resolver el problema se eleva, hasta alcanzar un punto óptimo [10].

Tiempo después fue propuesto el sistema de hormigas elitista (*EAS, Elitist Ant System*) [11] el cual es una variante de *AS*, en el cual se busca dar un reforzamiento adicional a la mejor solución conocida hasta el momento.

Para este modelo también se aprovecha la característica de estigmergia (la comunicación indirecta no simbólica en el medio ambiente), más allá de seguir el mismo paradigma lo que distingue a este modelo del anterior es que al término de una iteración, los valores de feromonas son modificados con el objetivo de influenciar a las hormigas en iteraciones futuras para construir soluciones de calidad similar a las que fueron anteriormente obtenidas.

Por otro lado, el sistema de hormigas Máx-Mín (*MMAS, Max-Min Ant System*) [11] es una mejora sobre el original *AS*, la principal característica es que únicamente la mejor hormiga actualiza los rastros de feromona y que el calor de ésta se encuentra acotado entre un límite inferior τ_{min} y el superior τ_{max}, esto es:

$$\tau_{ij} \leftarrow \left[(1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}^{best}\right]_{\tau_{min}}^{\tau_{max}}$$

Donde el operador $[x]_a^b$ se define como:

$$[x]_a^b = \begin{cases} a & \text{si } x > a \\ b & \text{si } x < b \\ x & \text{Otro caso} \end{cases}$$

y además $\Delta\tau_{ij}^{best}$ se expresa como:

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} & \text{si } i,j \text{ pertenece al mejor tour} \\ 0 & \text{Otro caso} \end{cases}$$

L_{best} se trata de la longitud del *tour* de la mejor hormiga [11], la cual puede ser tanto el mejor *tour* hallado en la iteración actual o el mejor encontrado desde el inicio del algoritmo, o una combinación de ambos. Mientras que los valores τ_{min}, τ_{max} son obtenidos empíricamente de acuerdo con el problema específico.

Se dice que este modelo converge si para cada punto de elección, uno de los componentes de la solución tiene τ_{max} como rastro de feromonas asociado, mientras que todos los componentes de solución alternativos tienen un rastro de feromona τ_{min}. Si *MMAS* ha convergido la solución construída al siempre elegir el componente de solución con máxima cantidad de feromona típicamente corresponderá con la mejor solución hallada por el algoritmo. Este concepto de convergencia difiere del concepto de estancamiento en un ligero pero importante aspecto. Mientras el estancamiento describe una situación donde todas las hormigas siguen la misma ruta, en el caso de la convergencia no corresponde con esto debido al uso de límites en el rastro de feromonas [12].

La cantidad máxima posible de feromona añadida después de cada iteración es:

$$\tau_{ij}^{max}(t) = \frac{1}{1-\rho}\cdot\Delta\tau_{ij}^{best}$$

Otro modelo, sistema de colonia de hormigas (*ACS, Ant Colony System*) [9], presenta una contribución interesante, se trata de la actualización local de feromonas aunado al reforzamiento hecho al final del proceso de construcción. Cabe señalar que la actualización local de feromonas se hace para todas las hormigas después de cada paso de construcción y cada hormiga lo aplica exclusivamente a la última arista atravesada:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_0$$

Donde τ_0 es el valor inicial de la feromona.

El objetivo principal de la actualización local es diversificar la búsqueda desempeñada por subsecuentes hormigas durante una iteración a través del decaimiento de la concentración de feromona por las aristas que se cruzan, esto ocasiona que las hormigas actuales motiven a otras hormigas a elegir diferentes aristas y con ello producir diferentes soluciones[11, 13, 14, 15].

Otro aspecto a destacar es que se usa una regla aleatoria-proporcional de tal manera que la hormiga k en la ciudad r escoge moverse a la ciudad s, es decir:

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)]^\beta}{\sum_{u\in J_k(r)}[\tau(r,u)]\cdot[\eta(r,u)]^\beta} & \text{si } s \in J_k(r) \\ 0 & \text{Otro caso} \end{cases}$$

Donde $J_k(r)$ es el conjunto de ciudades restantes por visitar por la hormiga k posicionada en la ciudad r.

Una vez que todas las hormigas han finalizado su *tour*, la cantidad de feromonas en las aristas es modificada de nueva cuenta aplicando una regla de actualización global [14]. Ésta se defina como sigue:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \sum_{k=1}^{m}\Delta\tau(r,s)$$

Donde

$$\Delta\tau_k(r,s) = \begin{cases} \frac{1}{L_k} & \text{si } (r,s) \in \text{tour hecho por la hormiga } k \\ 0 & \text{Otro caso} \end{cases}$$

Intuitivamente L_k es la longitud del *tour* hecho por la hormiga k y m el número de hormigas

Cuando una hormiga situada en r se encuentra dispuesta a elegir un arista para moverse hacia la ciudad s aplica la siguiente regla [16]:

$$s = \begin{cases} argmax_{u\in J_k(r)}\left\{[\tau(r,u)]\cdot[\eta(r,u)]^{\beta}\right\} & \text{si } q \leq q_0 \\ S & \text{Otro caso} \end{cases}$$

Donde q es una variable aleatoria uniforme en el intervalo $[0,1]$, y q_0 es un parámetro tal que $0 \leq q_0 \leq 1$ usado para controlar el nivel de exploración comprometido por las hormigas; S es una variable aleatoria seleccionada de acuerdo con la distribución de probabilidad dada por $p_k(r,s)$.

Sin embargo, una elección probabilística como la anteriormente expuesta no garantiza que una solución óptima sea encontrada. En algunos casos una solución ligeramente peor en comparación con la óptima puede ser encontrada al inicio y algunas aristas sub-óptimas son reforzadas por depósitos de feromonas. Este reforzamiento puede conducir a un comportamiento estancado resultado de que el algoritmo nunca encuentre la mejor solución [16].

Las pruebas empíricas iniciales [16] verificaron los dichos de *Dorigo* en los cuales se estipula que diferentes combinaciónes de parámetros (β, ρ, q_0) variarán el desempeño del algoritmo. Cuando a cada uno de estos parámetros se le asignan los rangos $0 < \beta < 15$, $0 < \rho < 1$ y $0 \leq q_0 \leq 1$, y al dividir estos tres rangos en catorce, nueve y once valores discretos, respectivamente. Esto proporcionó 1386 diferentes combinaciones de parámetros, que fueron probados en el algoritmo, lo cual tomó varias semanas en un servidor compartido con dos procesadores duales de 2.2 *GHz* y 2 *GB* en *RAM* haciendo uso de un lenguaje de programación híbrido funcional-imperativo de alto orden. Se encontró que los valores óptimos son $\beta = 6$, $\rho = 0.6$ y $q_0 = 0.2$. Esta combinación toma en promedio 34.2 iteraciones en hallar la mejor solución conocida, sin embargo en muchas situaciones soluciones sub-óptimas de gran calidad son suficientes. De igual manera se halló que no hay correlación estadística significativa entre ellos; generalmente el valor óptimo de ρ es cercano a 0.6, pero los otros parámetros varían considerablemente [16].

Usualmente los parámetros óptimos (o cercanamente óptimos) son escogidos por un procedimiento de prueba y error [17]. Como el número de intentos necesarios para afinar los parámetros es generalmente alto, es a menudo el caso en el que se utiliza un tiempo limitado para cada ejecución del algoritmo. Los mejores parámetros encontrados son usados entonces para lidiar con los problemas actuales. A su vez la solución del problema actual tiende a tomar mucho más tiempo [17].

2.2.- Marco teórico

Empecemos diciendo que un dato de tipo simple, no está compuesto de otras estructuras, que no sean los bits, y que por tanto su representación sobre el ordenador es directa, sin embargo existen unas operaciones propias de cada tipo, que en cierta manera los caracterizan. Una estructura de datos es, a grandes rasgos, una colección de datos (normalmente de tipo simple) que se caracterizan por su organización y las operaciones que se definen en ellos. Por tanto, una estructura de datos vendrá caracterizada tanto por unas ciertas relaciones entre los datos que la constituyen como por las operaciones posibles en ella.

Una decisión importante, a la hora de programar, es la selección de una estructura de datos frente a otra, ya que ello influye decisivamente en el desempeño del algoritmo que va a usarse para resolver un determinado problema, es por ello que continuación se presentan brevemente las estructuras y los algoritmos de geometría computacional que fueron utilizados para el desarrollo del presente trabajo.

2.2.1.- Listas simplemente ligadas

La principal desventaja de usar arreglos para almacenar datos, es que aquellos son estructuras estáticas y que por lo tanto no pueden fácilmente extenderse o reducirse para ajustarse al conjunto de datos.

Una lista simplemente ligada es una estructura de datos lineal y además dinámica que almacena cualquier cantidad de elementos, en donde cada dato es un objeto separado y es llamado nodo. Cada objeto de una lista ligada se comprende de dos partes, el dato por sí mismo y un apuntador hacia el siguiente objeto; particularmente el último elemento apunta hacía *null*. Mientras que el primero es llamado la cabeza de la lista. Para poder añadir un dato en una lista ligada es necesario recorrer la lista nodo por nodo hasta encontrar la posición adecuada. Se crea un nuevo nodo, se inserta el dato y se actualizan las ligas del nodo nuevo y del anterior para intercalar el nuevo nodo en la lista.

La ventaja principal es que la memoria que ocupa es solamente la necesaria. Cuando se agrega un nuevo elemento se reserva memoria para él y se añade a la lista; cuando se quiere eliminar el elemento simplemente se saca de la lista y se desocupa la memoria usada; a diferencia de un arreglo que puede desperdiciar memoria que no está en uso. Mientras que su principal desventaja, frente a los arreglos, es que no se permite un acceso directo a los elementos individuales. Cuando se requiere acceder a un elemento en particular se debe empezar por la cabeza y seguir las referencias hasta hallar el elemento buscado.

Desde el punto de vista abstracto, una lista es una secuencia de cero o más elementos de un tipo determinado. A menudo se representa como una sucesión de elementos entre paréntesis, separados por comas:

$$L = (a_0, a_1, \ldots, a_{n-1})$$

Donde $0 < n$ es el número de elementos que contiene, es decir la longitud de la lista, y cada a_i es de un tipo específico. Si $1 < n$ entonces se dice que a_0 es el primer elemento y a_{n-1} es el último elemento de la lista. Si $n = 0$ entonces la lista está vacía.

Una propiedad importante de una lista ligada es que sus elementos están acomodados en forma lineal, es decir que para cada elemento a_i existe un sucesor a_{i+1} (si $i < n-1$) y un predecesor a_{i-1} (si $0 < i$). Dos listas son iguales si tienen los mismos elementos y estos están acomodados en el mismo orden. Por ejemplo las siguientes listas no son iguales:

$$(1, 3, 7, 4) \neq (3, 7, 4, 1)$$

Mientras que en el caso de conjuntos, estos serían iguales. Una diferencia más respecto a los conjuntos es que en una lista se permite repetir elementos.

2.2.2.- Árboles y montículos binarios (heaps)

Un árbol binario es un conjunto finito de elementos que impone una estructura jerárquica sobre estos, el cual está vacío o dividido en tres subconjuntos separados:

- El primer subconjunto contiene un elemento único llamado raíz del árbol.
- El segundo subconjunto es en sí mismo un árbol binario y se le conoce como subárbol izquierdo del árbol original.
- El tercer subconjunto es también un árbol binario y se le conoce como subárbol derecho del árbol original.

El subárbol izquierdo o derecho puede o no estar vacío. Cada elemento de un árbol binario se conoce como nodo del árbol.

Un montículo binario (*binary heap* o simplemente *heap*) es un árbol binario completo, con la posible excepción de su último nivel, con elementos de un conjunto parcialmente ordenado, tal que el elemento de cada nodo es menor o igual que cualquiera de sus nodos hijo. Dado que un montículo es un árbol binario completo, los elementos pueden ser convenientemente ordenados en un arreglo. Si un elemento está en la posición i del arreglo, entonces su hijo izquierdo estará en la posición $2i$ y su hijo derecho estará en la posición $2i + 1$. Por las mismas razones, un elemento que no es la raíz del árbol, en la posición i tendrá a su padre en la posición $\frac{i}{2}$. Debido a esta estructura, un montículo con altura k tendrá entre 2^k y $2^{k+1} - 1$ elementos. De esta forma un montículo con n elementos

tendrá una altura de $logn$. Dado lo anterior, el elemento mínimo siempre estará presente en la raíz del árbol, así la operación para encontrarlo tendrá un costo en el peor caso de $O(1)$ en tiempo de ejecución.

La propiedad que permite que las operaciones sean hechas rápidamente es la propiedad de orden del montículo. Como se requiere tener la capacidad de encontrar al elemento mínimo rápidamente, tiene sentido que éste elemento se encuentre en la raíz. Si se considera que cualquier subárbol debería ser un montículo también, entonces cualquier nodo debería ser menor que todos sus hijos. Aplicando esta lógica se llega a la propiedad de orden del montículo. En un *heap*, para cada nodo X, el valor de su padre es menor o igual que X, con excepción de la raíz pues no tiene padre. Así las operaciones de inserción y eliminación implican asegurarse de que la propiedad de orden del montículo se preserva.

Para insertar un elemento X en el heap, se crea un espacio en la siguiente localidad disponible, puesto que de otra manera el árbol no estaría completo. Si X puede ser puesto en ese espacio sin violar la propiedad de orden, entonces la inserción queda hecha. De otra manera se intercambian los elementos que están en el espacio, y su nodo padre. Se continúa este proceso hasta que X pueda ser situado en el espacio. Esta estrategia general se conoce como filtrado hacia arriba, o percolado. El nuevo elemento es llevado hacia la cima del montículo hasta que la posición correcta se encuentra. Si el elemento agregado resulta ser el nuevo mínimo, éste será extraído del montículo. En algún momento el espacio creado del que se habla, será 1 y se terminará el ciclo. El tiempo requerido para hacer la inserción será de a lo más $O(logn)$, si el elemento que se inserta es el nuevo mínimo y es percolado hasta la raíz.

La operación de borrado, o de extracción, se maneja de for4ma similar a las inserciones. Encontrar al elemento mínimo es fácil, lo difícil es borrarlo. Cuando el mínimo es removido, un vacío es creado en la cima del montículo. Como ahora el *heap* es más pequeño, sigue que el último elemento X en el montículo deba moverse a algún otro sitio de éste. Si X puede ser situado en el vacío, entonces no hay más que hacer. Esto no es probable, por lo que se desplaza al vacío hacia abajo un nivel y percolando al elemento más pequeño. Se repite este paso hasta que X pueda ser acomodado. Para esta operación, en el peor caso se empleará $O(logn)$ en tiempo de ejecución [18].

2.2.3.- Cola de prioridad

Como se mencionó ya, un *heap* es una estructura de datos que almacena un árbol binario completo, además éste debe cumplir que para cada nodo su valor debe ser menor al valor de sus hijos en un *min-heap*, lo que garantiza que el nodo raíz es el menor de los elementos contenidos en el árbol. Éste es una de las mejores estructuras para implementar una cola de prioridad. Para obtener el siguiente menor elemento del *heap* se siguen los siguientes pasos:

1. Se retira la raíz del *heap*.
2. Se convierte el último nodo en la raíz del *heap*, y
3. Se restaura la propiedad del *heap*.

Una cola de prioridad es una estructura de datos cuyos elementos son claves las cuales soportan dos operaciones básicas: insertar un nuevo elemento y remover el elemento con la clave máxima. Sus aplicaciones incluyen sistemas de simulación donde las claves podrían corresponder con eventos temporales, para ser procesados en orden cronológico; calendarización de trabajo en sistemas de computadoras donde las claves podrían corresponder con prioridades indicando cuales usuarios deben ser atendidos primero; y computación numérica donde las claves podrían ser errores computacionales indicando que el más alto debería atenderse primero. En la práctica las colas de prioridad son más complicadas que la simple definición dada, debido a que hay otras varias operaciones que son necesarias para mantenerlas bajo todas las condiciones que podrían ocurrir mientras se usan. De hecho, una de las principales razones por las que muchas implementaciones de colas de prioridad son tan útiles es su flexibilidad para permitir programas de aplicación de clientes para desempeñar una variedad de diferentes operaciones en conjuntos de datos con claves [19].

Hay varias formas obvias de implementar una cola de prioridad, se podría usar una lista simplemente ligada, haciendo inserciones de elementos a través de la cabeza de aquella en $O(1)$ y atravesar la lista, lo cual requiere de $O(n)$ operaciones para eliminar al elemento mínimo. Alternativamente se podría insistir en que la lista se mantenga siempre ordenada; lo anterior hace que las inserciones sean costosas, usando $O(n)$, mientras que la operación de eliminar al elemento mínimo tomaría $O(1)$. La primer idea es probablemente mejor que la segunda, basándose en el hecho de que nunca hay más eliminaciones que inserciones. Otra forma de implementar una cola de prioridad es usar un árbol binario, esto proporciona un tiempo promedio de $O(logn)$ para ambas operaciones. Esto es verdadero

a pesar del hecho que aunque las inserciones sean aleatorias, las eliminaciones no lo son. El único elemento que siempre será borrado será el mínimo. Removiendo repetidamente al elemento que está en el subárbol izquierdo parecería afectar el balance del árbol haciendo pesado al subárbol derecho. Sin embargo el subárbol derecho es aleatorio. En el peor caso, cuando la operación de eliminación ha borrado el subárbol izquierdo, el subárbol derecho tendría a lo más el doble de elementos de lo que debería. Lo cual suma una pequeña constante a lo que sería su altura esperada.

Más que insertar y borrar elementos en un orden fijado, a cada elemento se le asigna una prioridad representada por un entero. Siempre se remueve un entero con la más alta prioridad, que está dada por la prioridad mínima entera asignada. Generalmente las colas de prioridad tienen un tamaño fijo. Algunas ventajas de usarla son:

- Permite disponer rápidamente del elemento con menor prioridad.
- Es una estructura dinámica porque permite insertar y quitar elementos en tiempo $O(logn)$, donde n es la cantidad de elementos en la cola, ejecutando las operaciones en cualquier orden.
- En una cola de prioridad es fácil obtener el elemento de mínima prioridad pero muy costoso obtener el de máxima, $O(nlogn)$.

2.2.4.- Ordenación por montículos (Heapsort)

Un montículo binario es una estructura que representa a un arreglo de objetos y que puede ser visto como un árbol binario completo, donde cada uno de sus nodos corresponde a un elemento del arreglo que almacena su valor en el nodo. El árbol está completamente lleno en todos sus niveles excepto tal vez en el último, el cual se ha llenado de izquierda a derecha hasta donde ha sido posible.

Es posible cualquier cola de prioridad para desarrollar un método de ordenación. Al insertar todos los elementos a ser ordenados en una cola de prioridad mínima, entonces repetidamente se puede remover el elemento más pequeño hasta tener a todos ordenados. Usando una cola de prioridad que representa a los elementos en un arreglo desordenado

El algoritmo *heapsort* se divide en dos fases: la construcción del montículo (*heap*), donde se reorganiza al arreglo original dentro de un *heap*; y la ordenación en sí misma, donde se toma a los elementos fuera del montículo en orden decreciente para construir el resultado ordenado.

Los métodos para insertar en el montículo y para reordenarlo son de orden logarítmico, o sea $O(logn)$, y los recorridos para insertar y eliminar los elementos del monticulo son de orden lineal, $O(n)$, por lo que la complejidad del algoritmo *heapsort* es de orden $O(nlogn)$ [19, 18].

2.2.5.- Área de un triángulo

Uno de los primeros problemas de la geometría computacional es el siguiente. Considérese una recta l en el plano con ecuación $y = ax + b$ y un punto $p(x_0, y_0)$ fuera de ella. Entonces ¿Dicho punto está a la derecha o a la izquierda de la recta?

Podemos suponer sin pérdida de generalidad que la recta no es horizontal ni vertical, es decir, $a \neq 0, \infty$. La forma más simple de conocer la respuesta a la pregunta es siguiendo los siguientes pasos:

- Trazar una recta horizontal que pasa por el punto p.
- Hallar el punto q de la intersección de l con la recta horizontal. O sea, sustituyendo $y = y_0$ en la ecuación de la recta y luego despejar para la coordenada x. Sea $q(x_1, y_1)$ el punto en cuestión y además notemos que $y_1 = y_0$.
- Se compara la coordenada $x = x_0$ del punto p con la coordenada $x = x_1$ del punto q. Entonces:
  - Si $x_0 = x_1$ el punto está sobre la recta l.

- Si $x_0 > x_1$ el punto p está a la derecha de l.
- Si $x_0 < x_1$ el punto p está a la izquierda de l.

De esta forma el problema queda resuelto. Una forma alternativa de resolver el mismo problema es transformándolo en el cálculo del área de un triángulo. Sea un triángulo con vértices en $a(a_1, a_2)$, $b(b_1, b_2)$ y $c(c_1, c_2)$, el área de éste viene dada por el producto vectorial de $\overrightarrow{A} = b - a = (b_1 - a_1, b_2 - a_2)$ y $\overrightarrow{B} = c - a = (c_1 - a_1, c_2 - a_2)$; el módulo de $\overrightarrow{A} \times \overrightarrow{B}$ es el área del paralelogramo de lados $\overrightarrow{A}$ y $\overrightarrow{B}$. El área es positiva si el ángulo entre ellos está dado en sentido contrario a las agujas del reloj, o negativa si el ángulo está en el sentido de las agujas del reloj. Por lo que el área que se busca es la mitad del determinante:

$$\Delta_{abc} = \frac{1}{2} \begin{Vmatrix} a_1 & a_2 & 1 \\ b_1 & b_2 & 1 \\ c_1 & c_2 & 1 \end{Vmatrix}$$

Si un punto b se encuentra a la derecha de la recta que pasa por a y c, entonces su área es positiva. Por el contrario si b está a la izquierda entonces el área es negativa. Para responder la pregunta inicial, basta con hacer $b = p(x_0, y_0)$ y tomar dos puntos a y c que se encuentren sobre la recta l y posteriormente obtener el signo del determinante descrito [20].

Esta lógica será usada ampliamente para la propuesta e implementación del algoritmo *ACO*.

2.2.6.- Diagrama de Voronoi

Todos los problemas de proximidad pueden ser resueltos usando una herramienta que contiene toda la información sobre la proximidad de los puntos. El diagrama de *Voronoi* es uno de los conjuntos mas importantes en geometría computacional.

Un diagrama de *Voronoi* para un conjunto $S = \{v_1, v_2, \ldots, v_n\}$ de puntos del plano es una partición del plano en n regiones poligonales convexas $V_1, \ldots, V_n$, tal que para cada i todos los puntos de la región V_i están más cercanos a v_i que a cualquier otro punto de $S - \{v_i\}$.

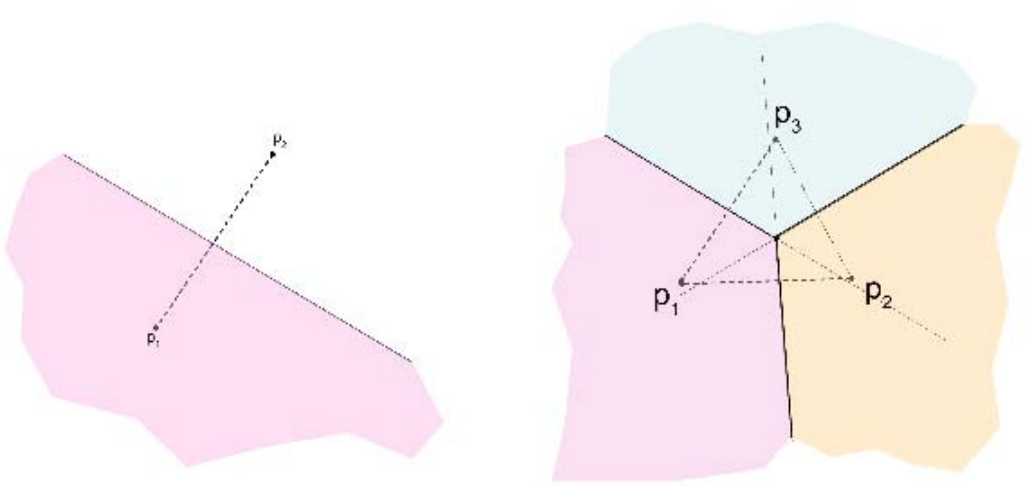
Si los puntos de S están uniformente distribuídos, una aproximación razonable sería usar una estructura basada en una cuadrícula. Así pues, con una rejilla de $\sqrt{n} \times \sqrt{n}$ se tiene en promedio un punto por cuadro, por lo tanto, se espera una petición en tiempo constante. Para una distribución general de puntos, en una dimensión, una aproximación efectiva sería justamente ordenar todos los puntos y hacer una búsqueda binaria para responder a la petición. Aquí el tiempo de cada petición es de $O(logn)$, el espacio utilizado de $O(n)$ y el tiempo de procesamiento de $O(nlogn)$ requerido por la ordenación.

Moviéndonos a dos dimensiones, se empieza con el caso base donde hay sólo dos puntos en S. La división del plano en dos regiones de influencia para cada punto se obtiene por el bisector perpendicular de la línea que une a los dos puntos. Entonces, una vez que se tiene la división del plano en dos regiones de influencia, se puede determinar en que región yace usando una petición de locación del punto. Fácilmente se puede extender esta idea a tres puntos construyendo los bisectores perpendicualres de cada par de puntos. Notemos que los tres bisectores se cruzan en un único punto (siendo que los tres puntos forman un triángulo) el cual es circuncentro de los 3 puntos. Los casos base se ilustran a continuación en la figura 2.2:

Haciendo una suposición más amplia, si se tienen 4 puntos en el conjunto S los cuales son co-circulares. Procediendo de una manera similar a la explicada por los casos base de 2 y 3 puntos, se puede extender esta técnica a una subdivisión planar para cualquier $n > 2$ construyendo bisectores perpendiculares para cada par de puntos vecinos. Dos puntos son considerados vecinos si el segmento que los une no cruza cualquier otro segmento más pequeño que él mismo. Esta definición de vecindad asegura que la subdivisión generada es planar. Tal subdivisión planar es llamada diagrama de *Voronoi*, diagrama de *Dirichlet* o diagrama de *Thiessen*. Los segmentos de la división se llaman aristas de *Voronoi* y los puntos de intersección de estos son los centros de *Voronoi*. Cada cara planar de la subdivisión es conocida como la región de *Voronoi* y cada una de estas regiones se puede asociar únicamente con un punto del conjunto S. Así, una región de *Voronoi* proporciona la región de influencia del punto en el plano, dados los otros $n - 1$ puntos de S [21].

Algunas características importantes de la teselación de *Voronoi* son:

- La región de *Voronoi* de un punto, $p_i \in S$, es $\cup_{i \neq j} H_{ij}$, donde H_{ij} es el semiplano que contiene a p_i y definido por el bisector perpendicular de p_i y p_j. Así, las regiones de *Voronoi* son intersecciones de semiplanos.



(a) Subdivisión planar en regiones de influencia de puntos individuales para el caso n=2

(b) Subdivisión planar en regiones de influencia de puntos individuales para el caso n=3

Figura 2.2: Sen, S. y Kumar, A, Casos base para la creación de diagramas de Voronoi, 2006. Imagen tomada de `http://goo.gl/aZIiGS` (Fecha de actualización 09 de mayo de 2016)

- Las regiones de *Voronoi* son regiones convexas, puesto que son la intersección de regiones convexas.
- Como se trata de una subdivisión planar del espacio, luego entonces el número de vértices, aristas y regiones de *Voronoi* están linealmente relacionados, esto es $O(n)$.

El número de aristas e del diagrama de *Voronoi* es $e \leq 3n - 6$, mientras que el número de vértices corresponde a $v \leq 2n - 4$.

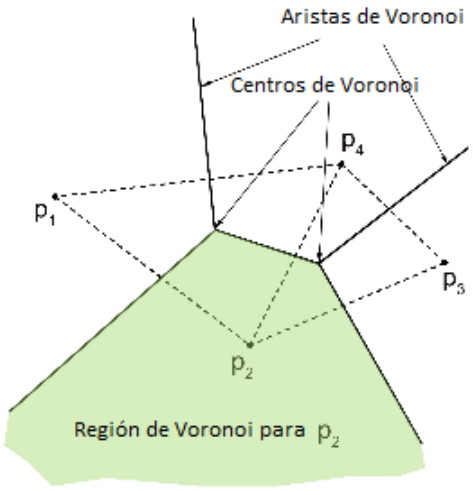


Figura 2.3: Sen, S. y Kumar, Componentes del diagrama de Voronoi, 2006. Imagen tomada de `http://goo.gl/aZIiGS` (Fecha de actualización 09 de mayo de 2016)

2.2.7.- Triangulación Delaunay

La triangulación *Delaunay* es un método de triangulación ampliamente usado en la generación de mallas no estructuradas. Es uno de los métodos más rápidos y relativamente fáciles de implementar, dando excelentes resultados para la mayoría de aplicaciones. Al igual que el diagrama de *Voronoi*, esta estructura es usada para fragmentar el espacio.

Para definir formalmente una triangulación de S, siendo éste un conjunto de puntos en el plano, primero se define una subdivisión planar máxima como una subdivisión de S tal que ninguna arista que conecta dos vértices puede ser añadida a S sin destruir su planaridad. En otras palabras, cada arista que no está en S cruza una de las aristas existentes. Una triangulación de S se define como una subdivisón planar máxima cuyo conjunto de vértices es S [22].

Primero definimos una triangulación de S como una subdivisión planar limitada con triángulos cuyos vértices son puntos de S y la frontera de la región es la envolvente convexa de S.

La triangulación *Delaunay* es una red de triángulos que se caracteriza por formar los triángulos más equiláteros posibles, es decir, maximiza el mínimo ángulo de cada triángulo. Además, posee la característica de ser el grafo dual al diagrama de *Voronoi*. Para cada triángulo de la triangulación, el centro del círculo circunscrito por el triángulo corresponde con un vértice generador del diagrama de *Voronoi* y las perpendiculares a los lados del triángulo forman las aristas del diagrama de *Voronoi*. Por lo tanto, se puede construir la triangulación de *Delaunay* a partir del diagrama de *Voronoi*, uniendo todos aquellos generadores que compartan un eje de *Voronoi*, y uniendo aquellos puntos vecinos en regiones de *Voronoi* abiertas (unión de los generadores de regiones adyacentes) [20].

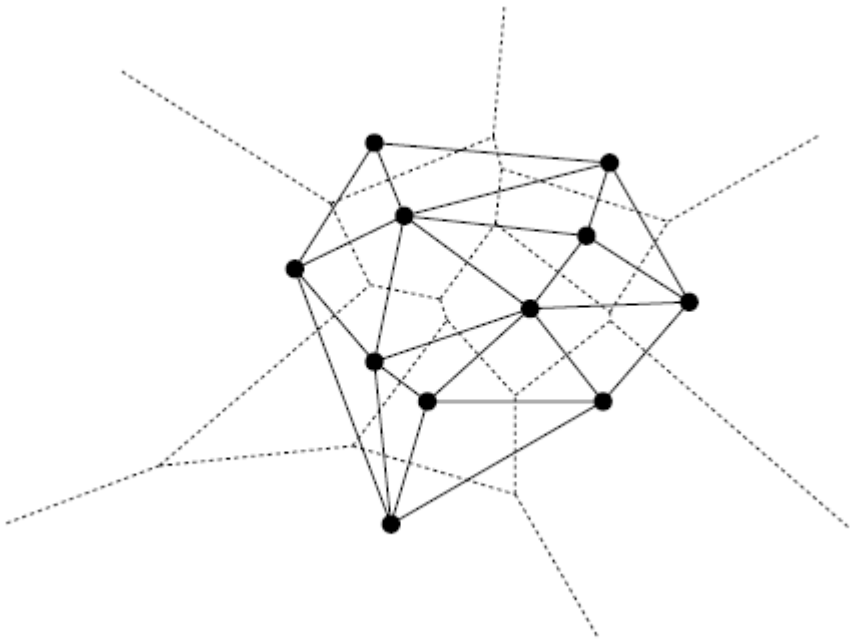


Figura 2.4: De Berg, M et al., Superposición del diagrama de Voronoi y la triangulación Delaunay, 2000.

Se tienen las siguientes observaciones:

- A cada vértice x de la triangulación *Delaunay*, denotada por D, le corresponde con la región de *Voronoi* V_x.
- Cada lado de D es un segmento recto con extremos p_1 y p_2, que corta perpendicularmente al lado de *Voronoi* que conecta las regiones donde se hayan los puntos dados.
- Cada cara de D es un tríangulo que contiene un vértice de *Voronoi*.

El proceso de triangulación ocurre de la siguiente manera. Si e_1, e_2 y e_3 son tres lados del diagrama de *Voronoi* que convergen en un vértice v, entonces hay tres regiones de *Voronoi* V_1, V_2 y V_3 tales que: e_1 separa a V_1 de V_2, e_2 a

V_2 de V_3 y e_3 separa a V_3 de V_1. Cada región V_i contiene un único punto de S, denotado por p_i. Luego los duales de los lados e_1, e_2 y e_3 son respectivamente p_1p_2, p_2p_3 y p_3p_1. El dual del vértice v es el triángulo $d(v) = \Delta(p_1, p_2, p_3)$.

Esta triangulación cumple las siguientes propiedades:

- Tres puntos p_i, p_j, p_k, pertenecientes a S son vértices de la misma triangulación *Delaunay* de S si y sólo si el círculo que pasa por estos tres puntos no contiene a ningún otro punto de S en su interior.
- Dos puntos p_i, p_j pertenecientes a S forman un lado de la triangulación *Delaunay* si y sólo si existe un círculo que contiene a p_i y p_j en su circunferencia y no contiene en su interior a ningún otro punto de S.

Las dos propiedades anteriormente descritas son resumidas por una tercera, llamada propiedad del círculo vacío. Si se dibuja un círculo utilizando los vértices de cualquier triángulo de *Delaunay*, este círculo no debe contener a ningún otro punto de la nube (vértice de otro triángulo). Para ver una demostración de estas propiedades se sugiere [22, 20].

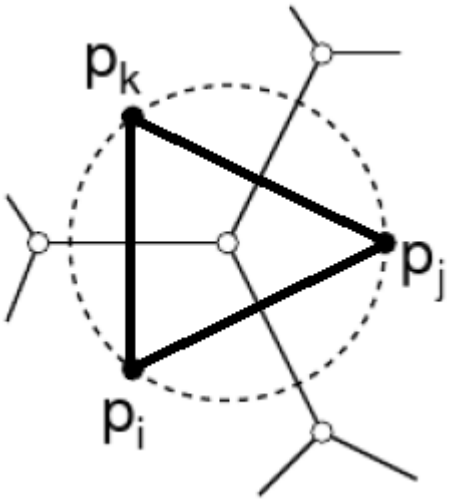


Figura 2.5: De Berg et al., Propiedad del círculo vacío de la triangulación Delauna, 2000.

Tanto la triangulación *Delaunay* como el diagrama de *Voronoi* tienen el mismo número de aristas y el contorno de la triangulación corresponde con el cierre convexo de la nube de puntos. La triangulación *Delaunay* puede encontrarse a partir del diagrama de *Voronoi* en tiempo lineal $O(e) = O(3n - 6)$.

2.2.8.- Búsqueda del camino más corto

Hay diversos problemas relacionados con la búsqueda de los caminos más cortos, como lo son:

- El camino mínimo entre los vértices P y Q.
- Las rutas más cortas entre el vértice P y todos los demás vértices de una gráfica conexa.
- Las rutas más cortas entre todos los pares de vértices.

La idea subyacente en el algoritmo de *Dijkstra* consiste en ir explorando todos los caminos más cortos que parten de un vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de costo negativo.

Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de ese vértice evaluaremos sus adyacentes. Entre todos los vértices adyacentes, se busca a aquel que esté más cerca del punto origen, se toma como punto intermedio y se revisa si es posible llegar más rápido a

través de este vértice a los demás. Posteriormente se elige al siguiente vértice más cercano (con las distancias ya actualizadas) y se repite el proceso. Esto se hace hasta que el vértice no utilizado más cercano sea el destino. En las palabras de Dijkstra se tiene el procedimiento siguiente [3]:

Usemos el hecho de que, si R es un nodo en el camino mínimo entre los vértices P y Q, el conocimiento de lo segundo implica el conocimiento de la ruta mínima de P a R. En la solución presentada, los caminos mínimos de P hacia el resto de los nodos es construída de tal forma que se incrementa la longitud hasta que Q es alcanzado. En el curso de la solución, los vértices son divididos en tres conjuntos:

A.- Los nodos para los cuales el camino de longitud mínima desde P es conocida; nodos que serán añadidos en este conjunto con el objetivo de incrementar la longitud del trayecto mínimo desde P.

B.- Los vértices desde los cuales el siguiente nodo a ser añadido al conjunto A serán seleccionados; este conjunto comprende todos aquellos nodos que están conectados a al menos un nodo del conjunto A pero no pertenecen a él aún por sí mismos.

C.- Los nodos restantes.

Así mismo las aristas son divididas en tres conjuntos:

I.- Las aristas que aparecen en el camino mínimo del nodo P a los nodos en el conjunto A.

II.- Las aristas desde las cuales la siguiente arista a ser situada en el conjunto I seán seleccionadas.

III.- Las aristas restantes (rechazadas o todavía no consideradas).

Empezando con todos los nodos en el conjunto C y todas las aristas en III. Ahora se transfiere el nodo P al conjunto A y desde ahora en adelante se efectúan repetidamente los siguientes pasos:

1. Considerando todas las aristas r conectadas con el nodo justo antes de transferirlo al conjunto A con nodos R en los conjuntos B o C. Si el nodo R pertenece al conjunto B, se investiga si el uso de la arista r proporciona un incremento a la ruta más corta de P a R comparado con el camino conocido que usa la correspondiente arista en el conjunto II. Si no es así, la arista r es rechazada; si lo es, entonces el uso de la arista r resulta en una conexión más corta entre P y R que la hasta ahora obtenida, esto reemplaza la correspondiente arista en el conjunto II y la última es rechazada. si el nodo R pertenece al conjunto C, se agrega al conjunto B y la arista r se añade al conjunto II.

2. Cada nodo en el conjunto B puede ser conectado al nodo P únicamente de una forma si nos restringimos a las aristas del conjunto I y a una del conjunto II. En este sentido cada nodo en el conjunto B tiene una distancia desde el nodo P: el nodo con distancia mínima desde P es transferido desde el conjunto B al A, y la arista correspondiente se mueve del conjunto II al I. Se regresa al paso 1 y se repite el proceso hasta que el nodo Q es transferido al conjunto A. Entonces la solución es hallada.

El algoritmo de *Dijkstra* es un ejemplo donde la voracidad funciona, en el sentido en que lo que parece localmente como lo mejor, se convierte en lo mejor de todo; en otras palabras utiliza el principio de que para que un camino sea óptimo todos los caminos que contiene también deben ser óptimos

El algoritmo de *Dijkstra* usa una cola de prioridad o *heap*, ésta debe tener la propiedad de *min-heap* es decir cada vez que extraiga un elemento de la cola de prioridad se debe devolver el de menor valor, que para el caso de las rutas el valor será el peso acumulado en los nodos.

Usando la cola de prioridad, este procedimiento tiene una complejidad algorítmica de $O(nlogn)$ para encontrar las rutas más cortas entre el vértice i y todos los demás vértices de una gráfica conexa. Extendiendo este mismo planteamiento, para encontrar las rutas más cortas entre todos los pares de vértices se tomará a lo más $O\left(n^2logn\right)$.

Capítulo 3

3.1.- Congruencia metodológica

Los datos analizados fueron proporcionados por la telefónica *Orange* con el reto *D4D* (*Data for development*, http://www.d4d.orange.com/en/Accueil), estos datos son de carácter abierto y su divulgación tiene el fin de impulsar mejoras para la población senegalesa mediante su análisis, detección de problemas u oportunidades no explotadas, y sus correspondientes propuestas de solución o mejora. Dichos datos contienen información geo-espacial de usuarios con teléfonos móviles inteligentes además de proporcionar información sobre posiciones geográficas de antenas celulares que dan servicio en todo Senegal; este conjunto de datos tiene las siguientes características:

- Los datos concernientes a las antenas celulares que proporcionan el servicio telefónico en Senegal contienen la posición geográfica de éstas además de ser nombradas con un identificador númerico único e indicar con otro identificador la zona a la cual pertenecen.
- Los datos correspondientes a los usuarios fueron procesados por la telefónica *Orange* de tal manera que cada usuario es anónimo. A cada usuario se le ha asignado un identificador numérico para un periódo de 15 días, transcurrido este tiempo, a cada usuario se le asigna otro identificador númerico; además estos identificadores pueden ser reutilizados en periodos siguientes. En cada periodo se quitaron o agregaron usuarios con nuevos identificadores o permutando los identificadores ya existentes. Este proceso se repite continuamente de tal forma que no se puede saber quién es el usuario ni tampoco rastrearlo.
- Los datos de los usuarios contienen, además del identificador, lo siguiente:
  - Estampa de tiempo que indica la fecha, hora, minuto y segundo en el cual cada usuario está realizando una llamada.
  - Posición geográfica de la antena a la cual se ha enlazado para hacer una llamada. La base de datos proporcionada no indica la posición de cada usuario, sino únicamente la posición geográfica de la antena que le está dando el servicio en ese momento.
  - Identificador único de la antena, la cual se han enlazado para hacer una llamada.
  - Identificador de la zona en la que se encuentra la antena utilizada por cada usuario en un momento específico.
- Como defecto del procesamiento hecho por la empresa telefónica hay una gran cantidad de datos repetidos.

Siendo Dakar la capital y ciudad más poblada de Senegal, el presente proyecto se enfoca solamente en el análisis de los movimientos de esta zona. El software utilizado fue el siguiente:

- **PostgreSQL.** Gestor de bases de datos de código abierto para almacenar las tablas de *Usuarios* y *Antenas*.
- **QGIS Chugiak.** Sistema de información geográfica de código abierto que sirve como visor de datos geo-espaciales y que está habilitado para tablas de PostgreSQL y que permite superponer diagramas de vectores.

- **Geoserver.** *Sofware* empleado como auxiliar para la visualización de mapas que entre sus funcionalidades incluye la carga de capas para páginas *web*.
- **OpenLayers.** Versión libre de Open Street Maps que permite mostrar mapas dinámicos.
- **MatLab.** Generalmente usado para hacer cálculos númericos, fue elegido por su rapidez y las múltiples optimizaciones que tienen sus funciones para diversas partes del procesamiento de datos y para los algoritmos de optimización por colonia de hormigas.
- **PHP, HTML y Javascript.** Utilizados para solicitar las consultas al gestor de bases de datos, solicitar el procesamiento de estos y la visualización para el usuario.
- **Apache.** Servidor web HTTP de código abierto para la creación de páginas y servicios web. Es un servidor multiplataforma, gratuito, muy robusto y que destaca por su seguridad y rendimiento.

Las etapas del proyecto junto con los algoritmos utilizados fueron los siguientes:

- Preprocesamiento de los datos de los usuarios.
- Filtrado de los datos de usuarios para eliminar repeticiones en cuanto a aquellos registros que no proporcionan información de movilidad.
- Filtrado de los datos de antenas para dejar sólo las ubicadas en Dakar.
- Obtención del diagrama de *Voronoi* de los puntos geográficos de todas las antenas ubicadas en Dakar.
- Obtención de la triangulación *Delaunay* de los puntos geográficos de todas las antenas ubicadas en Dakar.
  - Transformación de la triangulación *Delaunay* en una triangulación restringida.
- Implementación del algoritmo de *Dijkstra* para obtener las rutas más cortas entre todos los puntos geográficos de todas las antenas ubicadas en Dakar.
- Desarrollo e implementación de un algortimo de optimización por colonia de hormigas para encontrar la ruta más corta entre todos los puntos geográficos de todas las antenas ubicadas en Dakar.
- Filtrado de los datos de usuarios, hora por hora para obtener las rutas transitadas por cada uno.
  - Obtención de la lista de puntos de origen y sus correspondientes puntos destino.
- Redireccionamiento de las rutas obtenidas utilizando el algoritmo de *Dijkstra*.
- Desarrollo de una interfaz para la visualización de las rutas redirigidas.
- Visualización de mapas de calor de las rutas redirigidas, hora por hora.
- Búsqueda de relaciones permanentes entre celdas de *Voronoi*.
- Utilización de un algoritmo de optimización por colonia de hormigas para agrupar las rutas permanentes.

3.2.- Etapas y algoritmos utilizados

En seguida se explicarán cada una de las etapas del proyecto, cabe mencionar que sólo se hablará de los algoritmos más importantes y asímismo se mencionará la importancia y necesidad de cada etapa. Se evitará comentar explícitamente acerca del funcionamiento, configuración y puesta en marcha de cada software mencionado en la sección anterior, debido a que ello corresponde a una descripción bastante técnica. Concerniente a este escrito, no se pretende que sea un manual de referencia o un reporte meramente técnico, sino que se explicará lo mejor posible las ideas y la lógica empleada para dejar a otros diseñadores la libertad de elegir algún otro software que consideren más apropiado y posiblemente más fácil de utilizar.

3.2.1.- Preprocesamiento de los datos de los usuarios

Se separaron los datos de los usuarios que están dentro de los límites de Dakar, por lo que se obtuvieron 684,468,206 de registros para la tabla de *Usuarios*, llamémosle a estos los datos iniciales; cada uno de estos registros contiene una estampa de tiempo con formato *aaaa-mm-dd hh:mm:ss*. Para los propósitos del análisis, en los cuales es necesario hacer ordenaciones respecto a cada hora, hacer esta operación toma demasiado tiempo, pues es como ordenar una columna de tipo texto. Para un gestor de bases de datos es más fácil ordenar un conjunto de datos numérico que uno del tipo estampa de tiempo, por eso ésta se dividió en cuatro columnas que son: fecha, hora, minuto y segundo.

3.2.2.- Filtrado de los datos de usuarios para eliminar repeticiones

Haciendo consultas simples con el gestor de bases de datos se observó que los registros mostraban muchas repeticiones y que además estaban parcialmente agrupados por el identificador númerico del usuario, cabe mencionar que dichos identificadores presentan saltos en la secuencia numérica. Con esta simple exploración visual se encontró que había algunos identificadores con sólo una aparición en este conjunto inicial de datos; por lo cual estos son los primeros registros en ser eliminados pues no aportan información de movilidad al estar en un punto estático.

Para filtrar en su totalidad los datos iniciales se tomó en consideración el hecho de que los identificadores numéricos que se le asignan a los usuarios pueden ser reutilizados y como consecuencia existen agrupamientos aislados en la base de datos que tienen el mismo identificador. Se utilizó al *cluster* del IIMAS para almacenar esta base de datos, y se observó que en promedio una consulta *sql* simple tardaba alrededor de 4 minutos en ejecutarse (en promedio) tanto para obtener todos los registros de un único usuario como los de un conjunto de usuarios, por lo que se decidió aprovechar esta situación para procesar a los usuarios por bloques. Se procedió de la siguiente forma:

1. Se seleccionaron y ordenaron todos los identificadores numéricos de la tabla de *Usuarios* que son distintos y se conservaron únicamente aquellos que tienen más de una aparición en toda la tabla. Obteniendo aproximadamente 4 millones de usuarios.

2. Teniendo ordenado este conjunto de datos se sabe cuál es el identificador numérico mayor y menor de todos. Se hicieron consultas secuenciales por bloques a intervalos de cincuenta mil identificadores.

Si se hubieran seleccionado todos los registros de cada usuario, individualmente, para posteriormente procesarlos para eliminar repeticiones se hubieran tomado aproximandamente $4^6[usuarios] \times 4\left[\frac{minutos}{usuario}\right] \approx 30[años]$. Mientras que el proceso sencillamente descrito ocupó aproximadamente 76 horas en ejecución continua en el *cluster*. Después de este proceso quedan 467,831,931 registros de usuarios, es decir un 69.81% de los datos iniciales.

Figura 3.1: Posición geográfica de las antenas que dan servicio a Dakar.

3.2.3.- Filtrado de los datos de antenas para dejar sólo las ubicadas en Dakar

Haciendo un proceso sumamente más simple que el anterior, sólo se seleccionaron las posiciones geográficas de aquellas antenas que están a la izquierda del paralelo -17° 7' 12.8892" y del meridiano 14° 53' 57.336", dejando sólo 490 antenas de las más de 1600 existentes en todo el país. El resultado de esta etapa se puede apreciar en la figura 3.1.

3.2.4.- Obtención del diagrama de Voronoi de los puntos geográficos de todas las antenas ubicadas en Dakar

Una parte importante del presente trabajo es el cálculo de la teselación de *Voronoi* y su gráfica dual, la triangulación *Delaunay*. Como ya se explicó en el maco teórico hay diversas maneras de obtenerlas con una complejidad algorítmica bastante óptima, sin embargo el software *QGIS Chugiak* ya tiene implementada la función que hace el cálculo automático y se aprovechó ésta debido a que, aunado al cálculo, el *software* es capaz de convertirlo en una capa y colocarlo sobre un mapa como si fuera una transparencia.

La importancia de tener esta división de la ciudad de Dakar radica en que las celdas de *Voronoi* son una aproximación a posibles zonas de interés para la población.

Respecto a las posiciones geográficas que dan servicio a Dakar, el diagrama de *Voronoi* encontrado es el que se muestra en la figura 3.2.

El presente proyecto no busca hacer un análisis exhaustivo de los problemas de movilidad de Dakar en cada una de sus calles, por el momento se considera que es suficiente una aproximación a la movilidad que se cree que hay entre las celdas de *Voronoi*. Recordemos que la triangulación *Delaunay* es la parte dual de la teselación, entonces se puede interpretar a las aristas de cada triángulo como el flujo que entra o sale de una celda de *Voronoi* desde o hacia sus celdas vecinas. Este análisis a escala moderada revelará las zonas en las que debe considerar intervenir para hacer una revisión más profunda de las condiciones en que circula el tránsito. Posteriormente se explicará por qué se restringe a la triangulación *Delaunay* y la necesidad de que sea así.

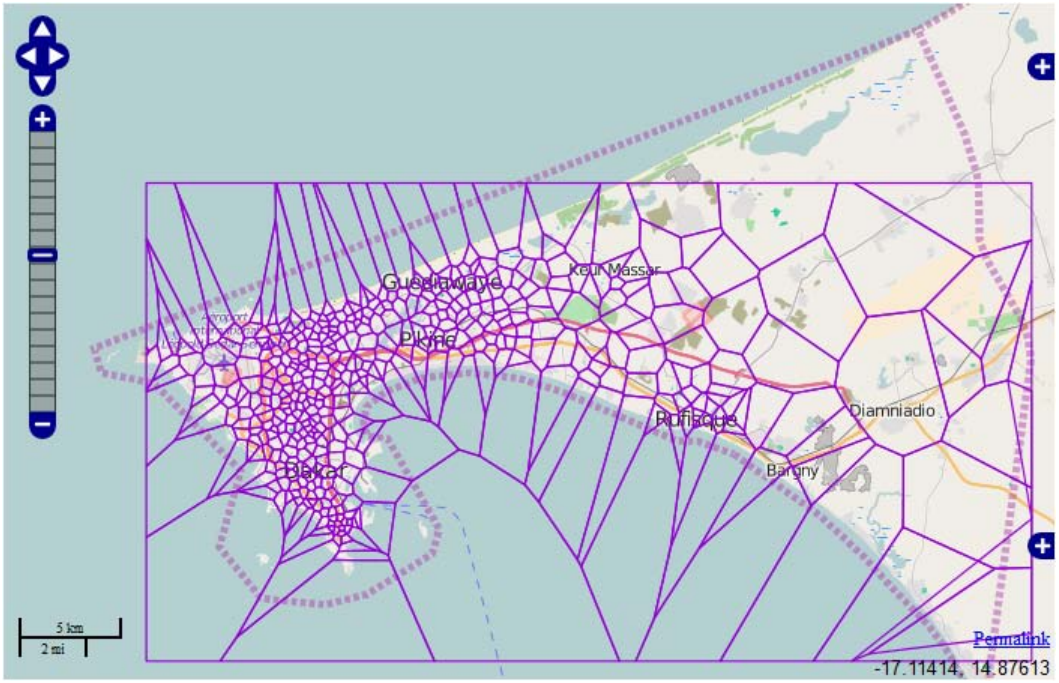


Figura 3.2: Diagrama de Voronoi que producen las antenas que dan servicio a Dakar.

Tanto para la capa correspondiente al diagrama de *Voronoi* como para la capa de la triangulación *Delaunay*, ambas se cargan en un mapa con ayuda de *Geoserver, javascript, HTML, PHP* y *apache*, lo que permite superponerlas entre sí sobre otros mapas y asignarles un estilo para permitir diferenciarlas cuando se muestran juntas, activarlas o desactivarlas cuando sea necesario.

En la figura 3.3 se muestra la triangulación obtenida.

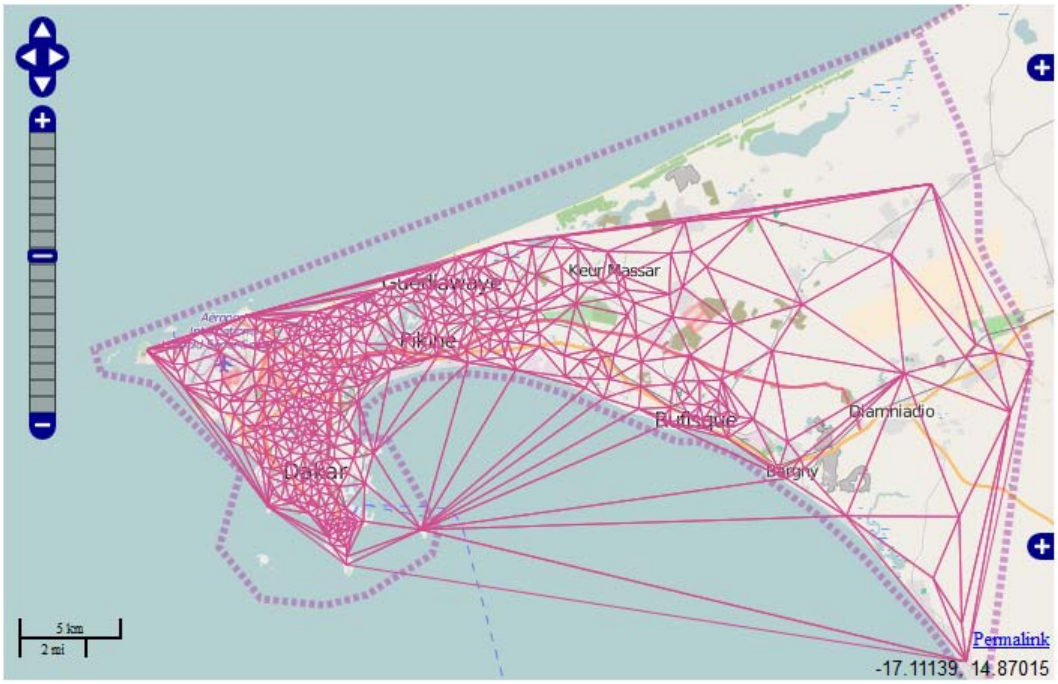


Figura 3.3: Triangulación Delaunay que producen las antenas que dan servicio a Dakar.

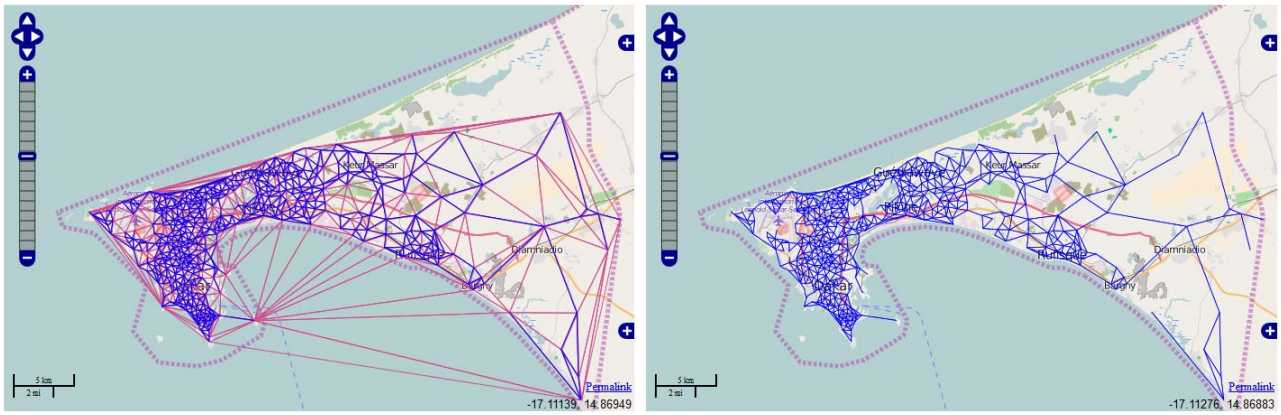
3.2.5.- Transformación de la triangulación Delaunay en una triangulación restringida

La triangulación *Delaunay* cubre todo el espacio sin excluir a ningún vértice; comparando el mapa original con el de las aristas obtenidas se observa claramente que algunas de éstas cruzan por áreas que consideramos restringidas como lo son las pistas de despegue del aeropuerto, parques grandes, el lago, algunas contrucciones grandes, zonas en donde no hay rutas vehiculares, entre otras; por este motivo es necesario crear una triangulación restringida desconectando manualmente algunas de las aristas de la triangulación y además permitiendo que la gráfica continue siendo conexa.

Se siguieron los siguientes criterios para elegir cuáles aristas desconectar y cuáles no:

- Se han desconectado las aristas que cruzan por las bahías de Dakar, excepto por la arista que va de la isla con posición geográfica *(-17.399363, 14667383)* hacia el punto geográfico *(-17.425784, 14.671906)* pues esta arista se considera como una aproximación a la ruta que sigue el *ferry* hasta la costa.
- Desconectar aquellas aristas por las que un individuo no podría transitar en un vehículo, por ejemplo las pistas del aeropuerto, lugares en donde no hay caminos, etcétera.
- Desconectar aquellas aristas que crucen zonas no transitables con una extensión muy amplia.
- No se deben desconectar aquellas aristas que están en el interior de un espacio no transitable y lo bastante cerca de alguna vía que sí lo es.
- No se deben desconectar aquellas aristas que cruzan un conjunto de vías transitables y que no siguen la dirección de éstas. Para este caso se considera que la triangulación *Delaunay* es una aproximación a moverse en diversas direcciones a través de este conjunto.

La triangulación resultante de este proceso en comparación con el conjunto de aristas inicial, se muestra en la figura 3.4 :



(a) Superposición de la triangulación Delaunay completa (rosa) y la triangulación restringida (azul).

(b) Triangulación Delaunay restringida.

Figura 3.4: Transformación de una triangulación Delaunay completa a una restringida.

3.2.6.- Implementación del algoritmo de Dijkstra para obtener las rutas más cortas entre todos los puntos geográficos de todas las antenas ubicadas en Dakar

Se tienen 490 antenas por toda la ciudad de Dakar, por lo cual es requerido calcular las rutas más cortas entre todos los pares de antenas, uno de los algoritmos que lo consegue más rápidamente es el de *Dijkstra* con montículos binarios, el cual tiene una complejidad algorítmica de $O(n^2 logn)$. Este algoritmo se ejecuta sólo una vez utilizando como matriz de pesos inicial la distancia de las aristas que se dejaron conectadas en la gráfica, la salida de este algoritmo es la matriz de precedentes, la cual muestra la secuencia de vértices que hay que visitar para ir de la posición i a al punto j usando el camino más corto encontrado; la otra salida del algoritmo es la matriz de pesos final, la cual indica la distancia total al ir de i a j a través del camino más corto.

Como se trata de puntos geográficos la distancia euclidiana entre dos puntos cualesquiera no corresponde con la distancia de dos puntos situados sobre el globo terráqueo debio a la curvatura de la tierra por lo que se ha usado la siguiente fórmula para calcular la distancia entre dos puntos de la gráfica:

$$d(P_1, P_2) = R_T \cdot acos(sin(lat_1) \cdot sin(lat_2) + cos(lat_1) \cdot cos(lat_2) \cdot cos(\Delta_{lon})) \qquad \text{ec. 3.1}$$

Donde:

$d(P_1, P_2)$ es la distancia geográfica calculada desde el punto P_1 a P_2
R_T es el díametro promedio de la tierra en Km con un valor de 6378.137 Km
lon_1,lat_1 son las coordenadas, en radianes, para el punto P_1
lon_2,lat_2 son las coordenadas, en radianes, para el punto P_2
Δ_{lon} es la diferencia, en radianes, entre lon_1 y lon_2
acos función arcocoseno
sin función seno
cos función coseno

Cabe mencionar que el uso de esta fórmula es importante para trabajos que se desarrollen posteriormente en los que se busque cuantificar la distancia promedio que recorren los usuarios entre otras medidas por el estilo. Sin embargo también se pudo haber utilizado la formula conocida para calcular la distancia euclidiana entre dos puntos.

3.2.7.- Desarrollo e implementación de un algoritmo ACO para encontrar la ruta más corta entre dos puntos geográficos cualesquiera

Como parte de los objetivos planteados se desarrolló e implementó un algoritmo de optimización por colonia de hormigas para encontrar la ruta más corta entre dos puntos geográficos cualesquiera y posteriormente extender éste mismo a la búsqueda de los caminos más cortos entre todos los pares de vértices de una gráfica, debido a que los algoritmos convencionales existentes tienen un costo algorítmico muy alto, que en el mejor de los casos es $O(n^2 logn)$, como una de las hipótesis se pensó que el algoritmo ACO podía dar los mismos resultados que los algoritmos tradicionales en un tiempo menor e incluso con menos recursos computacionales. La explicación detallada de este algoritmo se dará en el capítulo 4.

3.2.8.- Filtrado de los datos de usuarios, hora por hora para obtener las rutas transitadas por cada uno

La intención del análisis de los datos proporcionados es encontrar los patrones de movilidad que cotidianamente presenta Dakar y posteriormente proponer una, o varias rutas de transporte público, que beneficien a la mayor cantidad posible de usuarios. En primer instancia, para obtener dichos patrones de movilidad hay que hacer una serie de consultas al gestor de la base de datos y procesar los resultados.

Recordemos que los campos de la tabla *Usuarios* son: identificador de usuario, fecha, hora, minuto, segundo, posición geográfica de la antena a la cual el usuario correspondiente se enlaza para realizar una llamada, identificador único de la antena, identificador de la zona en la que se encuentra la antena. Por cuestiones de uso de recursos se solicitará al gestor que devuelva todos los registros de llamadas cuyo campo *hora* sea igual a h, con $h \in \mathbb{Z}^+ \cup 0$ tal que $0 \leq h < 24$, y que además únicamente se tome a los campos: identificador de usuario, fecha, hora, minuto e identificador único de la antena; en la misma consulta *sql*, se pedirá que los datos estén ordenados primero por el identificador de usuario y posteriormente por fecha, hora y minuto.

Siendo así, se obtendrán 24 consultas ordenadas que necesitan ser filtradas para eliminar la repetición de puntos geográficos repetidos cuando estos son adyacentes. Puesto que es posible que alguno de los usuarios haya realizado una serie de llamadas enlazándose con las antenas $A_1, A_2, \ldots, A_i$, esto indica que el individuo en cuestión se ha desplazado a través de diferentes celdas de *Voronoi*; y un tiempo después haya vuelto a llamar usando la misma antena A_i, lo que sugiere que continúa dentro de la misma celda de Voronoi., por lo que no aporta mayor información de movilidad; es por esta razón que deben eliminarse los registros que presentan situaciones de ese tipo.

Mediante el procesamiento completo de la base de datos, hora por hora, se obtendrán 24 mapas de calor que indicarán la afluencia a través de las celdas de *Voronoi* por parte de los ciudadanos de Dakar, la construcción de estos mapas se detallará más adelante, así como los resultados obtenidos.

3.2.9.- Obtención de la lista de puntos de origen y sus correspondientes puntos destino

Al tiempo que se construyen los mapas de calor hora por hora, el mismo proceso va construyendo listas que indican cuáles son los vértices origen y para cada uno de estos cuáles son sus destinos. Estos resultados serán utilizados para posteriormente crear las propuestas de rutas potenciales de transporte público que se cree podrían beneficiar ampliamente a la población de Dakar.

3.2.10.- Redireccionamiento de las rutas obtenidas utilizando el algoritmo de Dijkstra

Este también es un proceso simultáneo al de los dos puntos anteriores, una vez que se han separado los usuarios que sí se desplazan por la ciudad, se va calculando la ruta más corta a través de las celdas de *Voronoi* que va visitando. Esto es un paso necesario porque en ocasiones un usario hace una llamada en la celda i y tiempo después hace otra llamada en la celda j que se encuentra a una distancia considerable. El razonamiento empleado es que dicho usuario hace ese desplazamiento a tráves de las rutas más cortas; por lo que mediante el redireccionamiento se busca inferir la ruta más probable que ha seguido y consecuentemente las celdas de *Voronoi* que ha visitado.

3.2.11.- Desarrollo de una interfaz para la visualización de las rutas redirigidas

Se desarrolló una interfaz usando *PHP, Javascript, HTML y PostgreSQL*; se eligieron debido a todas las funciones que ya vienen implementadas y que son útiles para el procesamiento geo-espacial.

Se realiza una petición de consulta por medio de *PHP* a la base de datos de *PostgreSQL* para que devuelva todos los registros de llamadas que están entre las horas h_{ini} y h_{fin} de tal manera que el tiempo t_{reg} de cada registro, compuesto por hora, minuto y segundo, cumplan $h_{ini} \leq t_{reg} < h_{fin}$, donde h_{ini} es la hora de inicio y h_{fin} es la hora final. Éstas son llamadas rutas en bruto.

Capítulo 4

4.1.- Diseño del algoritmo ACO

Con base en el conocimiento adquirido por el estado del arte son evidentes ciertas particularidades de las variantes del algoritmo inspirado en el comportamiento de hormigas. Por un lado, para algunos de ellos, se predisponen los resultados del experimento cuando se le asigna una cantidad cero de feromonas a las aristas. En otros, la propuesta de los niveles mínimos y máximos de la cantidad de feromonas es algo que depende de realizar el mismo proceso de búsqueda de rutas cortas más de una vez, o de conocer la probabilidad de que una hormiga elija particularmente todas las aristas del camino más corto entre el hormiguero y la fuente de comida, esto no se puede saber *a priori*, mas que utilizando otros algortimos para obtener la probabilidad de que los agentes del sistema sigan el mismo camino y con ello asignar el nivel máximo de feromonas que debe tener el sistema bio-inspirado. Dicho de otra forma, primero habría que resolver el problema de interés con un método bastante estable y usar esos resultados para poder iniciar con el algoritmo de hormigas.

Aunado a las observaciones anteriores, sobre los algoritmos mencionados en el estado del arte, dados los experimentos realizados por los autores de los artículos consultados, salta a la vista la falta de información precisa sobre la cantidad de hormigas que tiene que ser usada para resolver las optimizaciones buscadas.

Sin embargo, varias de esas características serían bastante útiles aplicadas correctamente, así que se propone una variación del algoritmo *ACO*. Antes que nada, en esta sección, se mencionarán los elementos y características que hacen funcionar al algoritmo que se propone así como la necesidad y justificación de que aquellas estén presentes.

A grandes rasgos el algoritmo funciona de la siguiente forma: se coloca a todas las hormigas en un vértice origen (el hormiguero) y éstas saldrán una por una de él, a su debido tiempo, orientánose hacia la comida y eligiendo un arista con base en un proceso psudoaleatorio ponderado deacuerdo a una función de evaporación (percepción de hormonas), en donde las aristas que quedan detrás de cada hormiga no son tomadas en cuenta, a menos que retroceder sea la única opción para rodear algún obstáculo, Cuando la hormiga avanza a través de una arista va dejando un rastro de feromonas que cambia la probabilidad de elegir alguna de las aristas de los vértices por los que acaba de pasar recientemente por lo que esto ayudará probabilísticamente a otras hormigas a elegir aquel mismo camino. Cada vez que una hormiga llegue a un vértice y tenga que volver a elegir una arista que la acerque a la comida se vuelve a orientar hacia ésta, las sucesivas orientaciones y elecciones pseudoaleatorias permiten que las hormigas exploren un espacio acotado y que a su vez no predetermina las soluciones del problema. El modelo que se propone no busca ser total y biológicamente realista por lo que no se considera hacer regresar a las hormigas al hormiguero una vez que han llegado a la comida, en lugar de ello, en cada ocasión que una hormiga halla el alimento, se refuerzan las aristas visitadas por la hormiga que han producido la ruta más corta hasta el momento pues se considera como un equivalente a regresar a través del camino más corto que evidentemente tiene mayor presencia de feromnas en sus aristas; con el paso del tiempo y suficientes iteraciones se hallan rutas cada vez más cortas hasta alcanzar un mínimo local al menos.

Para reducir la complejidad computacional del algoritmo bio-inspirado, éste se ha transformado en un problema de calendarización, es decir, cada uno de los eventos corresponden a un instante de tiempo y el algoritmo analiza de cuál evento se atiende con base en el tiempo actual y los estados de las hormigas. Se consideran únicamente dos tipos de eventos: salida de un vértice y llegada a un vértice; además se tienen sólo los siguientes estados que pueden tener cada una de las hormigas de la colonia: inactiva, activada y desactivada.

A continuación se explicarán en conjunto los eventos y los estados de las hormigas:

Se considera que en un estado inicial todas las hormigas están inactivas, su interpretación es que las feromonas de éstas aún no ejercen efectos sobre las aristas de la gráfica pues no han salido del hormiguero.

- **Salida de un vértice.**
  - Cuando el vértice de salida es el hormiguero. Se activa la hormiga que corresponde al tiempo actual, y sale del hormiguero en busca del vértice de la comida.
  - Cuando el vértice de salida no es el hormiguero. La hormiga que corresponde al tiempo actual sale en busca del vértice de la comida.
- **Llegada a un vértice.**
  - Cuando se llega al vértice de la comida. Se desactiva a la hormiga que corresponde al tiempo actual; ésta ya no continuará avanzando más.
  - Cuando se llega a un vértice que no es el de la comida. Este evento automáticamente se convierte en la salida de un vértice que no es el hormiguero.

Por conveniencia, toda esta información estará contenida en una matriz M de medidas $q_H \times 6$, donde q_H es la cantidad de hormigas usadas en la colonia; cada renglón de la matriz tiene la información de una hormiga y cada columna almacena una variable diferente como se describe a continuación

- **Estado de cada hormiga.** Los valores que puede tomar esta variable y su significado son los siguientes:

Código	Significado
-1	Desactivada
0	Inactiva
1	Activa

Tabla 4.1: Codificación de los estados de las hormigas.

- **Tiempo de salida de cada hormiga.** Inicialmente esta columna guarda los tiempos de salida de cada hormiga del hormiguero.
- **Actual vértice de salida de cada hormiga.** En un inicio este vértice corresponde al del hormiguero, a medida que las hormigas van llegando a nuevos vértices, estos se convierten en los actuales vértices de salida y se van actualizando el correspondiente valor en la matriz.
- **Tiempo de llegada de cada hormiga.** En un principio, cada elemento de esta columna guarda el valor de ∞, la interpretación de esto es que si las hormigas no han escogido un vértice hacia donde dirigirse (que les acerque más a la comida), entonces no se puede saber el tiempo que tardarán en llegar a dicho lugar. Posteriormente cuando cada hormiga elige la ruta que va a tomar, ya es posible calcular el tiempo que le tomará desplazarse hasta el siguiente vértice y con ello actualizar este elemento en la matriz.
- **Próximo vértice al que llega cada hormiga.** En un principio, cuando aún no se ha escogido la arista que se recorrerá, el siguiente vértice v_i por visitar, para cada hormiga, es el vértice 0, éste no existe, esta asignación es únicamente para evitar que se calcule, antes de lo debido, el tiempo en que se llegará al siguiente vértice. Mientras no esté asignado el siguiente vértice, consevará el valor de 0.
- **Tiempo de salida del hormiguero para cada hormiga.** Es un respaldo de los tiempos de salida de cada hormiga del hormiguero. La utilidad de estos valores se explicará más adelante.

Más adelante se explicará el funcionamiento del algorimo completo, por ahora sólo se darán a conocer los procesos, estructuras computaciones y características que lo hacen funcionar. De igual forma, más adelante se dará a conocer el valor óptimo de q_H y las razones por las cuales se eligió que fuera éste.

4.1.1.- Calendarización de las salidas de hormigas

Desde un inico se programa el tiempo de salida de cada una de las hormigas, este tiempo es único para cada hormiga; para evitar problemas con la implementación de este algoritmo, no hay dos que salgan simultáneamente del hormiguero.

4.1.2.- Gráfica dirigida

Se toma a la gráfica de la triangulación *Delaunay* restringida como una gráfica dirigida cuyas aristas son los caminos que recorren las hormigas, además se considera que por cada arista de aquella, hay un camino que va hacia el vértice que tiene conectado al final y un camino que viene de regreso desde dicho vértice.

La gráfica se almacenará como una lista de adyacencia T para ahorrar espacio; la estructura computacional más adecuada para esto es un arreglo con 490 elementos (uno por cada vértice de la triangulación *Delaunay* restringida) donde cada uno de estos es una lista ligada de tres renglones, conformada como sigue:

- **Vértices.** El primer renglón contiene los vértices conectados a v_i; el elemento T_i representa al vértice v_i (de la triangulación *Delaunay* restringida). Cabe mencionar que este renglón se encuentra ordenado de menor a mayor. Es decir $T_{i,1} = \left[\begin{array}{cccc} v_{i,1} & v_{i,2} & \dots & v_{i,N_i} \end{array} \right]$ con $v_{i,1} < v_{i,2} < \ldots < v_{i,N_i}$, donde $v_{i,k}$ es el k-ésimo vértice conectado a v_i, con $k \in [1, N_i]$.

- **Distancias geográficas.** El segundo renglón almacena la distancia que hay entre el vértice v_i y cada uno de los vértices que están conectados a éste. Para ello se utiliza la ecuación 4.1 del capítulo anterior y a los elementos de $T_{i,1}$. Para el elemento i se tiene $T_{i,2} = \left[\begin{array}{cccc} d(v_i, v_{i,1}) & d(v_i, v_{i,2}) & \dots & d(v_i, v_{i,N_i}) \end{array} \right]$.

- **Niveles de feromonas.** El tercer renglón guarda los niveles de feromona perceptibles desde el vértice v_i hacia cada uno de los vértices que tiene conectados, entonces $T_{i,3} = \left[\begin{array}{cccc} \mathcal{N}(v_i, v_{i,1}) & \mathcal{N}(v_i, v_{i,2}) & \dots & \mathcal{N}(v_i, v_{i,N_i}) \end{array} \right]$. Estos niveles en un principio son inicializados con el nivel mínimo de feromona, del cual también se hablará más adelante.

Una lista de adyacencia permite recorrer los vértices conectados a v_i en un tiempo de a lo más el grado del vértice $O(G(v_i))$ mientras que al usar una matriz de adyacencia para representar el mismo grafo, aquella misma operación toma $O(n)$, con $G(v_i) \ll n$, esto aunado al poco espacio de almacenamiento requerido por la lista de adyacencia ($O(m+n)$), donde n es el número de vértices y m es la cantidad de aristas, son las características que se buscan explotar para el algoritmo ACO propuesto.

4.1.3.- Máximo nivel de feromona

Se propone que el nivel máximo de feromonas sea 7 y que este es perceptible en la posición actual de la hormiga H_i. El presente proyecto no busca simular a la perfección una colonia de hormigas, es por ello que el nivel de feromonas, para el algorimo propuesto, se considera que es una cantidad escalar sin unidades. se ha elegido que el nivel máximo sea 7 para evitar una lenta convergencia del algoritmo y de igual manera evitar la saturación de feronomas; sin embargo el valor óptimo puede ser un objeto de estudio posterior si se toma en consideración, por ejemplo, el tamaño de la gráfica.

4.1.4.- Listas de vértices visitados por cada una de las hormigas

Para la simulación se utiliza un arreglo de listas ligadas R, cada una de las cuales guarda los vértices visitados por cada hormiga, primer elemento de dichas listas es el vértice del hormiguero. Sea $R_{H_i} = \begin{bmatrix} v_{(1,H_i)} & v_{(2,H_i)} & \dots & v_{(k,H_i)} \end{bmatrix}$ la lista de vértices visitados por la hormiga H_i hasta el tiempo t, en donde $v_{(k,H_i)}$ con $i \in [1,k]$ es el vértice k-ésimo visitado por la hormiga H_i. Nuevamente la estructura más adecuada para guardarlos es un arreglo de listas ligadas.

4.1.5.- Velocidad constante

Para el modelo propuesto se considera que todas las hormigas avanzan a una velocidad constante durante todo el proceso, esta velocidad v_{cte} es de $50[\frac{m}{s}]$. Además al avanzar a través de una arista nunca cambia de dirección, siempre va hacia adelante. Tomando al conjunto V_{H_i} (vértices visitados por la hormiga H_i), el tiempo t y el instante de tiempo $t_{(k,H_i)}$, con $t_{(k,H_i)} < t$, en el cual H_i estuvo en el vértice $v_{(k,H_i)}$; la distancia recorrida durante el intervalo $t - t_{(k,H_i)}$ se calcula como:

$$f\left(t - t_{(k,H_i)}\right) = v_{cte} \cdot \left(t - t_{(k,H_i)}\right) \qquad \text{ec. 4.1}$$

De manera análoga se puede conocer el tiempo que le tomará a la hormiga H_i llegar de un vértice v_i a un vértice v_j, cuando estos están conectados, utilizando la siguiente ecuación:

$$t_{\overrightarrow{v_i v_j}} = \frac{d_{\overrightarrow{v_i v_j}}}{v_{cte}} \qquad \text{ec. 4.2}$$

Donde:
$t_{\overrightarrow{v_i v_j}}$, es el tiempo que tomará cruzar del vértice v_i al vértice v_j
$d_{\overrightarrow{v_i v_j}}$, es la distancia de la arista que une a v_i con el vértice v_j
v_{cte}, es la velocidad de desplazamiento de las hormigas, la cual es constante

Algorítmicamente, ambas operaciones toman siempre un tiempo constante, es decir $O(1)$.

4.1.6.- Detección y correspondencia de eventos

Las acciones y estados de las hormigas se basan en analizar que tipo de evento corresponde al instante de tiempo t actual, hacer lo que corresponde según el tipo de evento y el estado de la hormiga, e ir procesando cada nuevo instante de tiempo.

Los únicos eventos que se crean inicialmente son las salidas del hormiguero, todos los demás eventos se van creando conforme a la ejecución del algoritmo. Hay que notar que para cada evento de *salida de un vértice* siempre le corresponde uno de *llegada a un vértice*, además una vez que la hormiga ha salido de un vértice, siempre es posible saber en que momento llegará al siguiente, pues se puede calcular con la ecuación 5.2, mientras no haya elegido una arista a la cual dirigirse o aun no haya salido del hormiguero, se puede considerar el tiempo de llegada a un vértice como *infinito*.

Recordemos que en la matriz M hay una columna que corresponde al *tiempo de salida de cada hormiga* y otra para el *tiempo de llegada de cada hormiga*; ambas son consideradas como dos arreglos no ordenados M_S y M_L respectivamente, que tienen el mismo tamaño. Sean los arreglos $M_S = \begin{bmatrix} t_{S_1} & t_{S_2} & \dots & t_{S_{q_H}} \end{bmatrix}$ y $M_L = \begin{bmatrix} t_{L_1} & t_{L_2} & \dots & t_{L_{q_H}} \end{bmatrix}$, donde los elementos t_{S_k}, t_{L_k} con $k \in [1, q_H]$, corresponde al tiempo de salida y tiempo de llegada de la hormiga k, respectivamente. En un principio $M_L = \begin{bmatrix} \infty & \infty & \dots & \infty \end{bmatrix}$, debido al argumento del párrafo anterior.

El primer evento en ocurrir es simpre la salida de la hormiga H_1 del hormiguero, en el tiempo $t_0 = 0$, y sea además el tiempo de referencia $t_r = t_0$. Antes de elegir un vértice $M_{H_{1,5}} = M_{1,5} = \infty$ y después de ello $M_{H_{1,5}} = M_{1,5} = t_{L_1} = t_{\overrightarrow{M_{1,3} M_{1,5}}}$, resultado de usar la ecuación 4.2 con los vértices almacenados en $M_{1,3}$ y $M_{1,5}$; esto mismo, produce un efecto en el arreglo de *llegadas* que al actualizarse tiene los elementos $M_L = \begin{bmatrix} t_{L_1} & \infty & \dots & \infty \end{bmatrix}$. A

partir de aquí hay que considerar que el próximo evento puede estar en cualquiera de los dos arreglos, sabemos que $t_0 < t_{L_1}$ y puede ocurrir que el tiempo de salida de la hormiga H_2 sea posterior al tiempo de llegada de la hormiga H_1, o sea $t_{L_1} < t_{S_2}$, o que el tiempo de salida de H_2 ocurra primero que el tiempo de llegada de H_1, es decir $t_{S_2} < t_{L_1}$.

La situación expuesta indica que a medida que las hormigas hacen sus elecciones siempre hay que buscar en ambos arreglos para saber con certeza en que tiempo ocurrirá el evento siguiente y de qué se trata éste, si de la salida de un vértice o la llegada a uno. Implícitamente se da a entender que hacer la unión de ambos arreglos, $E = M_S \cup M_L = \begin{bmatrix} M_S & M_L \end{bmatrix}$ y ordenarlos representa un costo computacional elevado e innecesario durante todo el proceso, dado que la ordenación de los elementos siempre se consigue en $O(nlogn)$ y para este caso sería $O(q_H log q_H)$ y sólo será necesario tomar a $e \in E$ tal que $t_r < e$.

Una forma de obtener el tiempo en el que ocurre el evento siguiente y que tiene un costo computacional menor, se basa en los siguientes hechos:

- Construir el arreglo $E = M_S \cup M_L = \begin{bmatrix} M_S & M_L \end{bmatrix}$ depende del número de hormigas que se usen en la colonia, $O(q_H)$.
- Colocar a los elementos de E en las hojas de un árbol binario sólo toma $O(q_H)$.
- Tomar al elemento más pequeño de E, siempre tomará a lo más la profundidad del árbol binario, es decir $O\left(log q_H\right)$.
- Reemplazar un elemento de E siempre ocupa $O(1)$.

Con base en lo anterior tomar al k-ésimo elemento más pequeño de E tomará $O\left(klog q_H\right)$. Pensemos en el tiempo de referencia t_r como el $(k-1)$-ésimo elemento más pequeño de E, el objetivo es buscar al elemento k, el cual representa al instante de tiempo en el que ocurre el evento siguiente. Si al extraer al elemento más pequeño, éste es menor o igual a t_r entonces se le reemplaza en el arreglo E con ∞, y se continúa extrayendo y reemplazando sucesivamente a los elementos que son más pequeños o iguales a t_r, hasta obtener el tiempo en que ocurrirá el evento posterior a t_r.

El proceso brevemente descrito únicamente encuentra el instante de tiempo, pero no indica el tipo de evento al cual corresponde, si es la salida de un vértice o la llegada a uno, de igual manera tampoco señala cual es la hormiga a la que le corresponde dicho evento. Para solucionar esto y aprovechar el procedimiento descrito, se amplía la información de cada uno de los elementos del arreglo E de la siguiente manera:

$$E = \left[\begin{bmatrix} t_{s_1} \\ \mathcal{M}_{\mathcal{S}} \\ H_1 \end{bmatrix} \begin{bmatrix} t_{s_2} \\ \mathcal{M}_{\mathcal{S}} \\ H_2 \end{bmatrix} \ \ldots \ \begin{bmatrix} t_{s_{q_H}} \\ \mathcal{M}_{\mathcal{S}} \\ H_{q_H} \end{bmatrix} \begin{bmatrix} t_{L_1} \\ \mathcal{M}_{\mathcal{L}} \\ H_1 \end{bmatrix} \begin{bmatrix} t_{L_2} \\ \mathcal{M}_{\mathcal{L}} \\ H_2 \end{bmatrix} \ \ldots \ \begin{bmatrix} t_{L_{q_H}} \\ \mathcal{M}_{\mathcal{L}} \\ H_{q_H} \end{bmatrix} \right]$$

Cada uno de los vectores columna del conjunto E se coloca en las hojas de un árbol binario, y sucesivamente se extrae, vector tras vector, el elemento más pequeño comparando a t_r con el primer elemento de cada uno de los vectores columna que llegan a la raíz del árbol; hasta obtener al elemento k-ésimo más pequeño; cuando esto ocurre, ya se tiene el tiempo t_e en el que ocurrirá la salida de un vértice $\mathcal{M}_{\mathcal{S}}$ o la llegada a uno $\mathcal{M}_{\mathcal{L}}$, y cual es la hormiga que desencadena este evento H_i con $i \in [1, q_H]$. Por simplicidad se codifica al segundo renglón de cada elemento de E como $\mathcal{M}_{\mathcal{S}} = 2$ o $\mathcal{M}_{\mathcal{L}} = 4$, en referencia a la columna que les corresponde en la matriz M.

Este proceso resulta ser más completo y algorítmicamente toma la misma cantidad de tiempo que sólo buscar al instante de tiempo correspondiente al evento siguiente al actual, $O(klog q_H)$.

4.1.7.- Función de evaporación (percepción) de feromonas propuesta

Como se mencionó anteriormente en el estado del arte, a medida que las hormigas contruyen las soluciones van dejando un rastro de feromonas, mismo que se va evaporando conforme transcurre el tiempo y se incrementa momentáneamente en aquellas partes en donde otras hormigas ya han transitado. Visto de otra forma, para el modelo propuesto se considera que el nivel de feromona es máximo en la posición donde se encuentra actualmente la hormiga H_i y a medida que ésta se aleja de un punto fijo, la percepción de la feromona disminuye respecto a éste; entendiendo esto, se propone la siguiente función de evaporación (percepción) desde un punto fijo por el que ya pasó la hormiga H_i:

$$F_{(v_A \to v_B)}\left(f(\Delta t)\right) = \begin{cases} -\frac{6}{d_{max}} & \text{si } f(\Delta t) < d_{max} \\ 1 & \text{si } f(\Delta t) \geq d_{max} \end{cases} \qquad \text{ec. 4.3}$$

Donde:

$F_{(v_A \to v_B)}$, calcula la cantidad de feromona perceptible desde el vértice v_A mientras la hormiga H_i se dirige a v_B durante el intervalo de tiempo Δt. Se ha usado la notación $(v_A \to v_B)$ porque para llegar del vértice v_A al v_B puede haber una serie de vértices intermedios.

$f(\Delta t)$, utilizando la ec. 4.1, calcula la distancia en $[m]$ recorrida por la hormiga H_i en un intervalo de tiempo Δt específico.

d_{max} es la distancia máxima, en $[m]$, hasta donde el modelo propuesto considera que son percetibles las feromonas dejadas por las hormigas. Como parte de la lógica empleada, parece adecuado que la distancia máxima de percepción no sea fija, puesto que si se trata de encontrar la ruta más corta entre un par de vértices, los cuales distan mucho de sí, fijar una distancia máxima de percepción d_{max} que sea mucho menos que la distancia que separa a los vértices terminará por hallar resultados nada óptimos. En lugar de esto, se propone una distancia máxima dinámica en función de la mitad de la distancia geográfica que separa a los vertices v_i y v_j, para los cuales se requiere hallar la ruta más corta entre ellos. Esto dará libertad a los agentes de elegir entre mayores posibilidades de aristas por explorar sin predisponer los resultados de los experimentos.

La figura 4.1 muestra la función de evaporación (percepción) de feromonas para una distancia máxima de 400 metros.

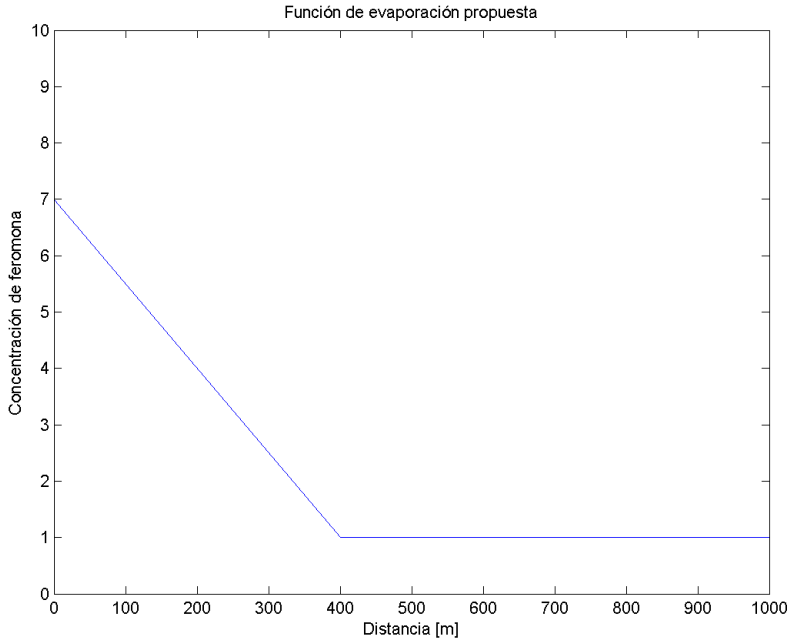


Figura 4.1: Función de evaporación en función de la distancia de la posición actual de la hormiga al hormiguero.

En la realidad la función de evaporación es una ecuación diferencial; para los propósitos del proyecto es más simple una ecuación que de igual manera decaiga con el tiempo, y para mayor simplicidad se eligió la ecuación por partes mostrada ya que es mucho más fácil de evaluar que una ecuación exponencial que decae conforme a la distancia recorrida por las hormigas. En esta parte se hace énfasis en que la distancia recorrida está en función del tiempo como se mencionó en la sección 4.1.5.

En ocasiones es mejor sólo tomar el nivel de feromonas que se percibe desde un vértice v_i hacía el vértice v_j, cuando ambos están conectados; hay que recordar que el arreglo de listas de adyacencias T almacena la cantidad de feromona perceptible desde cada uno de los vértices de la triangulación *Delaunay* restringida hacia los vértices que cada uno tiene conectados.

Se define a $\mathcal{F}\left(\overrightarrow{v_i v_j}\right)$ como el operador que toma la cantidad de feromona perceptible desde el vértice v_i en dirección a v_j, recordemos que este dato se encuentra almacenado en el arreglo T de listas de adyacencia; concretamente en

la lista de adyacencia T_{v_i}, la cual corresponde al vértice v_i; es necesario volver a señalar que dicha lista guarda, en su primer renglón, los vértices que están conectados a v_i; en su segundo renglón, la distancia desde v_i hacia cada vértice adyacente; y en su tercer renglón, el nivel perceptible de feromonas desde v_i hacia cada uno de los vértices que tiene conectados.

$$T_{V_i} = T_i = \begin{bmatrix} T_{v_i,1} \\ T_{v_i,2} \\ T_{v_i,3} \end{bmatrix} = \begin{bmatrix} T_{i,1} \\ T_{i,2} \\ T_{i,3} \end{bmatrix} = \begin{bmatrix} v_{i,1} & v_{i,2} & \dots & v_{i,N_i} \\ d(v_i, v_{i,1}) & d(v_i, v_{i,2}) & \dots & d(v_i, v_{i,N_i}) \\ \mathcal{N}(v_i, v_{i,1}) & \mathcal{N}(v_i, v_{i,2}) & \dots & \mathcal{N}(v_i, v_{i,N_i}) \end{bmatrix}$$

Para conocer $\mathcal{F}(\overrightarrow{v_i v_j})$ sólo el primer y tercer renglón de T_i son útiles en este proceso. Para extraer el nivel de feromona que se percibe desde v_i hacia v_j hay que encontrar a éste y para ello se usará el operador $\mathcal{B}(T_i, v_j)$, el cual busca al elemento $v_{i,k} \in T_{i,1}$ con $k \in [1, N_i]$ tal que $v_{i,k} = v_j$, esto es, tal que el k-ésimo vértice conectado a v_i sea igual a v_j . Como se mencionó anteriormente, los elementos de $T_{i,1}$ se encuentran ordenados de menor a mayor, y los elementos del segundo y tercer renglón están ordenados respecto al primer renglón; por ello para encontrar a v_j se utiliza una búsqueda binaria, la cual tiene una complejidad algorítmica de a lo más el logaritmo del grado del vértice v_i, o sea $O(logG(v_i))$.

La búsqueda binaria nos dará toda la información del elemento v_j, por lo que sólo bastará con tomar el dato almacenado en su tercer renglón, que es precisamente el resultado que persigue este proceso. La siguiente ecuación resume el argumento anterior:

$$\mathcal{F}(\overrightarrow{v_i v_j}) = \mathcal{B}(T_i, v_j) \qquad \text{ec. 4.4}$$

Donde:
$\mathcal{F}(\overrightarrow{v_i v_j})$ es el nivel de feromona que se percibe desde v_i en dirección a v_j
$\mathcal{B}(T_i, v_j)$ es la búsqueda binaria del elemento v_j entre los elementos de T_i

Como se observó, se trata de una consulta optimizada a la lista de adyacencia T_i. Cabe mencionar que en general $\mathcal{F}(\overrightarrow{v_i v_j}) \neq \mathcal{F}(\overrightarrow{v_j v_i})$.

4.1.8.- Orientación

Para acelerar el proceso de convergencia del algoritmo, cada vez que una hormiga sale del hormiguero o llega a un nuevo vértice, se orienta hacia la comida. Este proceso se describe a continuación:

1. Supongamos que la hormiga H_i en el vértice v_i, la hormiga toma el conjunto de vértices conectados a su posición actual, $V = \begin{bmatrix} v_{i,1} & v_{i,2} & \dots & v_{i,N_i} \end{bmatrix}$, este arreglo tiene dimensión $dim(V) = G(v_i) = N_i$, es decir el grado del vértice v_i.

2. La hormiga H_i se ayuda trazando un segmento de recta auxiliar, $\overrightarrow{R}$, que parte del vértice actual (v_i) hasta el vértice en el cual está la comida (v_f), esto es $\overrightarrow{R} = \overrightarrow{v_i v_f}$.

3. La hormiga H_i calcula la ecuación de la recta perpendicular al segmento de recta auxiliar $\overrightarrow{R}$; teniendo el vértice actual las coordenadas (x_{v_i}, y_{v_i}) y la comida (x_{v_f}, y_{v_f}), sea $\overrightarrow{R_\perp}$ la recta perpendicular a $\overrightarrow{R}$, la pendiente y ordenada al origen de esta recta vienen dados por las siguientes ecuaciones:

$$\begin{gathered} \Delta_y = y_{v_f} - y_{v_i} \\ \Delta_x = x_{v_f} - x_{v_i} \\ m_\perp = \begin{cases} -1 & \text{si } \Delta_y = 0 \\ -\frac{\Delta_x}{\Delta_y} & \text{si } \Delta_y \neq 0 \end{cases} \\ b_\perp = y_{v_f} - m_\perp \cdot x_{v_f} \\ \overrightarrow{R_\perp(x)} = m_\perp \cdot x + b_\perp \end{gathered} \qquad \text{ec. 4.5}$$

4. Sean los puntos $P_0(x_0, y_0)$, $P_1(x_1, y_1)$ y $P_2(x_2, y_2)$ la siguiente ecuación determina si para encarar el punto P_1 hay que girar a la derecha (cuando el resultado es 1) o la izquierda (cuando el resultado es -1) del vector formado por $\overrightarrow{P_0P_2}$:

$$s_g\left(P_0, P_1, P_2\right) = sign\left(\left|\begin{smallmatrix} P_2 - P_0 \\ P_1 - P_0 \end{smallmatrix}\right|\right) \qquad \text{ec. 4.6}$$

5. Sean los puntos $x_{g_{min}}$ y $x_{g_{max}}$ las longitudes mínima y máxima, respectivamente, usadas para definir los límites en la gráfica, se evalúan en la ecuación 4.5, obteniendo los puntos: P_0 con coordenadas $\left(x_{g_{min}}, \overrightarrow{R_\perp(x_{g_{min}})}\right)$ y P_2 con coordenadas $\left(x_{g_{max}}, \overrightarrow{R_\perp(x_{g_{max}})}\right)$; además, sustituyendo el punto P_1 con las coordenadas del vértice donde se localiza la comida (x_{v_f}, y_{v_f}), y evaluando la ecuación de la subsección anterior con estos tres puntos se obtiene el sentido de giro hacia la comida s_{g_f}.

6. Referente a la Figura 4.2, las aristas de la triagulación *Delaunay* restringida se muestran en azul; además, sea el vértice v_i (que corresponde al círculo rojo) la posición actual de la hormiga H_i y v_f el vértice donde se posiciona la comida (mostrado como un triángulo rojo en la misma figura), entonces el segmento de recta $\overrightarrow{R}$ (línea entrecortada en verde) representa la dirección hacía la cual debe dirigirse. Recordemos que V es el arreglo donde se almacenan los vértices conectados a v_i (la posición actual de la hormiga H_i), las aristas que los enlazan se muestran en rojo; los elementos de V son mostrados con círculos negros; y se determina cuales de aquellos se localizan del mismo lado que el vértice de la comida v_f (señalado con un triángulo rojo), para averiguarlo, el punto P_1 será reemplazado por las coordenas de los vértices de V (uno a la vez); y únicamente se tomarán en consideración aquellos cuyo sentido de giro sea el mismo que s_{g_f}. En otras palabras el segmento de recta $\overrightarrow{R_\perp}$ (línea entrecortada naranja) descarta las aristas que están detrás del vertice donde actualmente se encuentra la hormiga H_i y que por lo tanto no la conducen más cerca de la comida.

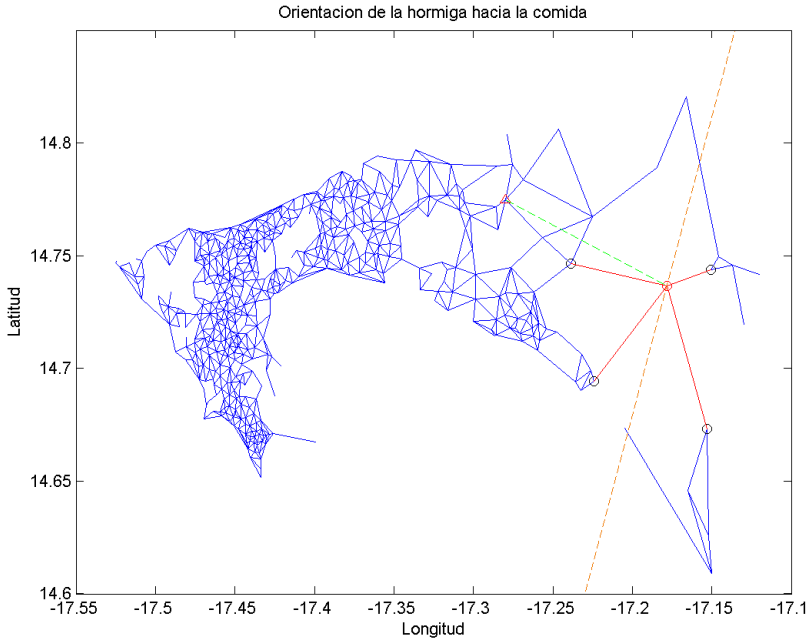


Figura 4.2: Orientación de una hormiga cualquiera (círculo en rojo) hacia la comida (triángulo rojo).

Algorítmicamente esta parte depende directamente del grado del vértice $G(v_i)$ en el que actualmente se encuentra la hormiga H_i; es decir que este subproceso toma a lo más toma $O(G(v_i))$. Cada vértice de una triangulación *Delaunay* tiene en promedio 6 aristas, por lo que en promedio este paso tomará $\Theta(6)$.

4.1.9.- Normalización del nivel de feromonas perceptible

Al salir de un vértice, cada hormiga escoge una arista mediante un proceso aletorio ponderado con base en la cantidad de feromonas que tienen las aristas candidatas. Imaginemos que la hormiga H_1 ha salido en el tiempo t_0 del hormiguero ubicado en el vértice v_i, y que está conectado con los elementos del arreglo $V = \left[\begin{array}{cccc} v_{i,1} & v_{i,2} & \dots & v_{i,N_i} \end{array} \right]$ cuya dimensión es $dim(V) = N_i$. El producto cartesiano $v_i \times V = \left[\begin{array}{cccc} \overrightarrow{v_i v_{i,1}} & \overrightarrow{v_i v_{i,2}} & \dots & \overrightarrow{v_i v_{i,N_i}} \end{array} \right]$ son las aristas por las que nadie ha cruzado, esto implica que cada una de éstas tiene la misma cantidad de feromona (el mínimo, el cual corresponde a 1 y de lo cual se hablará más ampliamente en la sección 4.1.11), y por lo tanto la posibildad de cada una de ser elegida es equiprobable, dicho de otro modo, la probabilidad de elegir a la arista $\overrightarrow{v_i v_{i,j}}$ con $j \in [1, N_i]$, está dada por la ecuación:

$$P_{v_i, v_{i,j}} = \frac{1}{dim(V)} = \frac{1}{N_i} \qquad \text{ec. 4.7}$$

Ahora imaginemos que la hormiga H_2 ha salido del hormiguero en el instante de tiempo t_1, en la diferencia de tiempo $\Delta t = t_1 - t_0$ transurrido, la hormiga H_1 ya ha avanzado $f(\Delta t)$ metros (utilizando la ecuación 4.1), y en este punto ahora se encuentra el máximo nivel de feromona, pues es en donde se encuentra la hormiga H_1 actualmente, y como consecuencia el nivel de feromona de H_1 que se percibe desde el hormiguero es menor al máximo. La arista que escogió H_1 tiene una cantidad mayor de feromonas mientras que las demás continúan teniendo el nivel mínimo. Ahora la probabilidad de escoger la arista $\overrightarrow{v_i v_{i,j}}$ viene dado por:

$$P_{v_i, v_{i,j}} = \frac{\mathcal{F}\left(\overrightarrow{v_i v_{i,j}}\right)}{\sum_{k=1}^{N} \mathcal{F}\left(\overrightarrow{v_i v_{i,k}}\right)} \qquad \text{ec. 4.8}$$

Donde:

$P_{v_i, v_{i,j}}$ es la probabilidad de elegir la arista que une a v_i con $v_{i,j}$
$\mathcal{F}\left(\overrightarrow{v_i v_{i,j}}\right)$es nivel de feromona perceptible desde el vértice v_i hacia $v_{i,j}$

Como se puede intuir, el proceso de elección de aristas está relacionado con el de orientación. Aunque algunos modelos se basan en hacer que las hormigas escojan las aristas que tienen una mayor cantidad de feromonas, se considera que esto predispone los resultados de los experimentos (cuando únicamente se sigue este criterio), pues cuando varias hormigas han pasado por una cierta arista, no es indicio de que ésta sea parte del conjunto solución de aristas que da la ruta más corta del hormiguero a la comida; dicho de otro modo, el siempre elegir las aristas con niveles mayores de feromonas limita la construcción de soluciones por parte de la colonia y sesga el potencial conocimiento colectivo que se podría generar y aprovechar. El proceso de orientación acelera la convergencia del algoritmo descartando soluciones no factibles y a su vez le da la oportunidad a las hormigas de continuar explorando soluciones que les conduzcan a un óptimo local (al menos).

Partiendo del final del proceso de orientación, sea el arreglo W el que almacena la lista de vértices conectados a v_i, y que acercan a la hormiga H_i a la comida, notemos que $dim(W) \leq dim(V)$, pues $W \subseteq V$, es decir que puede ocurrir que todas las aristas conectadas a v_i tengan el potencial de formar la ruta más corta, pues están del mismo lado que la comida.

A continuación, reasignamos el contenido de W tomando el nivel de feromonas perceptible desde v_i hacía cada uno de los vértices que contiene W, esto es:

$$W \leftarrow \left[\begin{array}{cccc} \mathcal{F}\left(\overrightarrow{v_i v_{i,1}}\right) & \mathcal{F}\left(\overrightarrow{v_i v_{i,2}}\right) & \dots & \mathcal{F}\left(\overrightarrow{v_i v_{i,dim(W)}}\right) \end{array} \right]$$

y posteriormente dividir al arreglo W entre la suma de la cantidad de feromona que tiene cada uno de sus elementos, lo cual evidentemente normalizará este arreglo a 1, como se ilustra en la siguiente ecuación

$$\widehat{W} = \frac{\left[\mathcal{F}\left(\overrightarrow{v_i v_{i,1}}\right)\ \mathcal{F}\left(\overrightarrow{v_i v_{i,2}}\right)\ \dots\ \mathcal{F}\left(\overrightarrow{v_i v_{i,dim(W)}}\right)\right]}{\sum_{k=1}^{dim(W)} \mathcal{F}\left(\overrightarrow{v_i v_{i,k}}\right)} \qquad \text{ec. 4.9}$$

La ecuación 4.9 representa la probabilidad porcentual que tiene cada una de las aristas de $\widehat{W}$ de ser elegida.

4.1.10.- Elección de aristas

Como se mencionó anteriormente, el procedimiento de elección de aristas es un proceso aleatorio ponderado. Se puede interpretar a cada elemento del arreglo $\widehat{W}$ como el ancho de un subintervalo semiabierto dentro de una distribución aleatoria uniforme en el intervalo $[0,1]$; sea U el arreglo que almacena los límites de dichos intervalos, con dimensión $dim(\widehat{W}) = dim(U)$, los elementos $U = \left[\begin{array}{cccccc} u_1 & u_2 & \dots & u_i & \dots & u_{dim(\widehat{W})} \end{array} \right]$, con $i \in \mathbb{Z}^+$ tal que $1 \leq i \leq dim(\widehat{W})$, se calculan con la siguiente ecuación:

$$u_i = \sum_{k=1}^{i} \widehat{W}_k$$

Notemos que la ecuación anterior obtiene al elemento u_i haciendo i sumas y al elemento que le sigue (u_{i+1}) haciendo $i+1$, y así sucesivamente; entonces la cantidad de operaciones empleada para encontrar a todos los elementos que conforman a U serán $\frac{dim(\widehat{W})\cdot(dim(\widehat{W})+1)}{2}$ y por lo tanto esta parte tiene una complejidad algorítmica de $O(dim(\widehat{W})^2)$. Resulta evidente que la forma de calcular a estos elementos necesita ser optimizada debido a la gran cantidad de veces que se ejecutará este proceso en el algoritmo *ACO*. Sea $u_1 = \widehat{W}_1$, los demás elementos de U se consiguen usando la siguiente ecuación:

$$u_i = \widehat{W}_i + u_{i-1} \text{ con i } \in \mathbb{Z}^+ \text{ tal que } 2 \leq i \leq dim(\widehat{W}) \qquad \text{ec. 4.10}$$

De esta manera es posible construir al arreglo U en $O(dim(\widehat{W}))$. Como se dijo, los componentes de U representan el ancho de subintervalos numéricos, dado que sus valores están ordenados de forma ascendente, se considera al primer límite inferior $lim_{inf_1} = 0$ y el primer límite superior $lim_{sup_1} = \widehat{W}_1$, con ellos se construye el subintervalo semiabierto por la izquierda $(lim_{inf_1}, lim_{sup_1}] = (0, \widehat{W}_1]$; para los límites del subintervalo I_i, con $i \in \mathbb{Z}^+$ tal que $1 \leq i \leq dim(\widehat{W})$, hay que considerar que el límite inferior de I_i es el límite superior del subintervalo anterior; y que el límite superior de I_i es el elemento u_i; adicionalmente se contempla que estos subintervalos sean semiabiertos por la izquierda.

Hay que señalar que ahora hemos partido al intervalo $[0,1]$, en $dim(\widehat{W})$ subintervalos (ordenados) más pequeños cuyo ancho corresponde con la probabilidad porcentual que tiene cada una de las aristas de $\widehat{W}$ de ser elegida; dicho de otro modo, si la arista $\overrightarrow{v_i v_j} \in V$ tiene un nivel de feronoma mayor mientras que las demás aristas del arreglo V conservan aún el mínimo; a dicha arista le corresponderá un subintervalo numérico más amplio.

Para simular un proceso de elección aleatorio por parte de las hormigas, se genera una variable aleatoria $\hat{u}$ con distribución uniforme, y dependiendo del valor de ésta, se buscará entre los elementos de U empleando una búsqueda binaria, con complejidad $O(logdim(W))$, hasta averiguar en que subintervalo está contenido $\hat{u}$; de esta manera encontraremos al elemento u_j tal que $u_{j-1} < \hat{u} \leq u_j$, que implícitamente indica que estando la hormiga actualmente en el vértice v_i (como se indicó en los procesos anteriormente descritos) ha elegido dirigirse hacia el vértice v_j.

4.1.11.- Nivel mínimo de feromonas para cada arista

Advirtamos que la construcción de los subintervalos descritos en la sección 4.1.10 es posible sólo porque cada $w_i > 0$; notemos que si alguno de los elementos de W fuera igual a cero, entonces no se podrían crear tantos subintervalos como son requeridos, y los experimentos estarían comprometidos por aquellas aristas que se eligen primero pues serían las únicas que tendrían un nivel de feromonas diferente de cero, ocasionando que las hormigas siempre escojan los mismos caminos que elige la primer hormiga,

Computacionalmente es más simple que el nivel mínimo de feromonas para todas las aristas sea de 1.

4.1.12.- Lista de ocupación de vértices y actualización de feromonas

Hay que hacer énfasis en la existencia de n vértices que conforman a cualquier gráfica, con $n \in \mathbb{Z}^+$, para que exista la triangulación *Delaunay* se requieren $n \geq 3$ puntos geográficos. Supongamos que en el tiempo actual todas las hormigas que forman la colonia están activas, cuando es el turno de alguna de ellas para escoger un camino, hay que actualizar los niveles de feromonas presentes en el grafo.

Actualizar los niveles de feromonas de todas las hormigas en todo momento, es computacionalmente ineficiente, pues basta con suponer que la última en salir del hormiguero q_H, tiene el turno de elegir un camino, si asumimos que ninguna de las hormigas ha llegado aún al vértice de la comida y que la mayoría de ellas ha cruzado ya r vértices, con $r < n$. Esta situación indica que actualizar los niveles de feromonas en este momento para todas las hormigas activas, necesitará a lo más de $O((q_H - 1) \cdot r)$ operaciones, lo que implica que hasta ahora se habrán hecho $O\left((q_H - 1) \cdot r^2\right)$ actualizaciones en los rastros de feromonas, si además consideramos que $r = q_H$, se tendría un desempeño algorítmico de $O\left({q_H}^3\right)$. Por lo que si se trata de encontrar una ruta entre dos vértices muy lejanos con una gran cantidad de hormigas; el algoritmo ACO tendría un desempeño bastante pobre.

Ahora pensemos que de antemano se sabe que la ruta más corta entre dos vértices en particular se obtiene al cruzar $\frac{n}{4}$ vértices del total disponibles, y que usamos una colonia con q_H hormigas, con base en esto, para cada hormiga que tenga que elegir un camino habría que hacer $O\left(q_H \cdot \frac{n^2}{16}\right)$ actualizaciones de los niveles de feromona, suponiendo que todas han ido por el camino más corto. Como las etapas descritas anteriormente demoran mucho menos tiempo que ésta, se puede decir que el algoritmo ACO propuesto sería de orden $O\left(q_H \cdot n^2\right)$ únicamente para encontrar la ruta más corta entre un par de vértices, si se actualizan en todo momento los niveles de feromonas para todas las hormigas activas. Si el número de agentes iguala al número de vértices, el orden de este algoritmo es de $O\left(n^3\right)$.

La cantidad de recursos se incrementa si en lugar de buscar la ruta más corta entre dos vértices se quiere la ruta entre todos los pares de vértices, por lo que una etapa como esta provocaría que el algoritmo ACO tuviera una complejidad algorítmica de $O\left(\left(n^3\right)^2\right) = O\left(n^6\right)$, si además asumimos que en la gráfica de interés existe la presencia de cavidades convexas únicamente, lo cual es realmente inviable.

Por otro lado, actualizar los niveles de feromonas de los agentes que recientemente cruzaron por el vértice en el que ahora se encuentra la hormiga, es una alternativa más factible, como se explica a continuación. Definamos primero a V como el arreglo de listas ligadas que almacena a las hormigas que han cruzado por los vértices de la triangulación y en el orden en el cual lo han hecho, es decir la ocupación de cada vértice. Para el caso del hormiguero, le corresponde la lista que está en la posición v_H del arreglo V, es decir $V_{v_H} = V_H$; imaginemos que las hormigas han salido en orden una a la vez y cada una en su debido tiempo, se tiene entonces $V_H = \left[\begin{array}{cccc} 1 & 2 & \ldots & q_H \end{array}\right]$. Es lógico pensar que el rastro de feromonas dejado por la primer hormiga que se marchó del hormiguero, ya se habrá evaporado significativamente en comparación con el que ha dejado la penúltima hormiga q_{H-1} que muy recientemente ha salido del mismo sitio. Hay que recordar que cada vértice en una triangulación *Delaunay* está conectado en promedio a 6 vértices más y también que el proceso de orientación sólo considera a los vértices que están del mismo lado que la comida y debido al proceso aleatorio ponderado usado para elegir un camino, no todas las hormigas eligen partir hacia el mismo vértice. Cuando q_H acaba de salir del hormiguero tiene que elegir un vértice al cual dirigirse y para ello deben estar actualizados los niveles de feromonas perceptibles desde el hormiguero en dirección a los vértices que están conectados a él, por lo que hay que revisar las correspondientes listas de estos vértices en el arreglo V; imaginemos que se puede elegir uno de entre 6 vértices; como ya se explicó, es suficiente con actualizar los niveles de feromona de las hormigas que se encuentran en el último elemento de cada lista ligada, tomar al elemento final de cualquier lista toma $O(1)$, estos elementos representan a las hormigas que pasaron de último por aquellos lugares, en este ejemplo implicaría tomar a esos 6 individuos y buscar sus respectivas listas de vértices visitados en el arreglo de listas ligadas R; y con ello actualizar el rastro que han dejado. Si además especulamos que cada hormiga ha pasado por aproximadamente por $\frac{n}{4}$ vértices, entonces, para una colonia con q_H hormigas, la actualización llevará $O\left(6 \cdot q_H \cdot \frac{n}{4}\right)$, lo que en notación algorítmica corresponde a $O\left(q_H \cdot n\right)$, para el peor de los casos, es decir cuando el número de agentes sea un múltiplo de la cantidad de vértices en la gráfica, se tendrá una complejiad algorítmica de $O\left(n^2\right)$. En palabras más simples, lo anterior es una especie de transformación que nos lleva primero al *espacio* de vértices V, luego al de hormigas R y por último al de feromonas presentes en la gráfica en general T, para cada toma de decisión.

Ahora pensemos en un ejemplo de mayor alcance, que conocemos previamente que la ruta más corta entre el hormiguero y la comida tiene $\frac{n}{4}$ vértices y que la mayor parte de la colonia ha cruzado ya, a lo más, por $q_H = \frac{n}{8}$ vértices. Al actualizar los niveles de feromona, hasta este punto del proceso, y hacerlo como se acaba de describir les ha tomado al menos $O\left(6 \cdot \frac{n^2}{32}\right)$ que corresponde a $O\left(n^2\right)$ cuando se utiliza una cantidad de hormigas que es

múltiplo del número de vértices, si lo que se busca es la ruta más corta entre dos puntos cualesquiera. Para obtener la ruta más corta entre todos los pares de vértices se emplearían a lo más $O\left(n^4\right)$ operaciones.

Sin lugar a dudas esta manera de actualizar los niveles de feromonas es mucho más eficiente que la primera de la que se habló; así se evita una serie de operaciones innecesarias y que perjudican el desempeño del algoritmo.

4.1.13.- Cantidad óptima de hormigas

Hasta ahora, una de las etapas más costosas para el algoritmo ACO ha sido la de actualización del nivel de feromonas, y aunque ya se describió el método más simple para lidiar con ese problema, en apariencia hace creer que un algoritmo de orden $O\left(q_H \cdot n\right)$ con facilidad de convertirse en $O\left(n^2\right)$ para obtener la ruta más corta entre un par cualquiera de vértices no está lo suficientemente optimizado. Cabe señalar un hecho importante del apartado anterior es que la cantidad de hormigas es un peso enorme para el propio algoritmo, por lo que se debe escoger correctamente el número de agentes que conforman a la colonia. Se propone que para q_H se debe tomar una cantidad en el intervalo $[2, n]$, y lo más conveniente es tomar:

$$q_H = c\lfloor\sqrt{n}\rfloor \text{ con } c \in \mathbb{Z}^+ \text{ tal que } 1 \leq c \leq \lfloor\sqrt{n}\rfloor \qquad \text{ec. 4.11}$$

Cuando una hormiga busque el camino más corto entre dos vértices cualesquiera, el desempeño del algoritmo por colonia de hormigas esté acotado por:

$$O\left(\lfloor\sqrt{n}\rfloor \cdot n\right) \leq O\left(ACO\right) \leq O\left(n^2\right)$$

De manera análoga, el camino más corto entre todos los pares de vértices depende igualmente del coeficiente c, por lo que la complejidad algorítmica estará comprendida entre los límites:

$$O\left(\lfloor\sqrt{n}\rfloor \cdot n^3\right) \leq O\left(ACO\right) \leq O\left(n^4\right)$$

Hay que hacer énfasis en el hecho de que *a priori* los únicos datos que se conocen de una gráfica son la cantidad de vértices que tiene y la posición de estos, por lo que esa idea fue la base de la lógica empleada.

4.1.14.- Reforzamiento del camino más corto

Parte esencial del funcionamiento de la optimización por colonia de hormigas es reforzar las aristas de los caminos más cortos que se han descubierto con el fin de guiar poco a poco a las hormigas hacia una solución óptima.

Sea D el arreglo que almacena la distancia total recorrida por cada una de las hormigas donde cada elemento se conforma como $\begin{bmatrix} Distancia \\ Hormiga \end{bmatrix}$; lo más conveniente es inicializar cada uno de los q_H elementos de D como $D = \begin{bmatrix} \infty & \infty & \dots & \infty \\ 1 & 2 & \dots & q_H \end{bmatrix}$, pues en un inicio se desconoce la distancia total que recorrerán las hormigas.

Cada vez que alguna hormiga llega al vértice de la comida basta con tomar el tiempo en el cual ha salido del hormiguero t_{v_H,H_i} y el tiempo en el cual han llegado a la comida t_{v_f,H_i} y utilizar la siguiente ecuación:

$$D_{H_i} = v_{cte} \cdot \left(t_{v_f,H_i} - t_{v_H,H_i}\right) \qquad \text{ec. 4.12}$$

Cuando la hormiga H_i ha llegado a la comida, se actualiza el elemento $D_{H_I} = D_i$ usando la ecuación 5.11.

Una vez almacenada la nueva distancia calculada, usando un árbol binario, se procede a comparar las trayectorias de ese arreglo y de las cuales la más corta llegará a la raíz del árbol junto con la hormiga que recorrió esa distancia (la cual no es necesariamente la hormiga H_i), esto llevará a lo más $O\left(logq_H\right)$ comparaciones entre los componentes de D. El elemento del cual se habla es $D_{min} = \begin{bmatrix} d_{min} \\ H_j \end{bmatrix}$, ahora se busca a las aristas que ha recorrido esta hormiga en el arreglo R, recordemos que en éste se almacenan los vértices que ha visitado , hasta el momento, cada una de las hormigas. La ruta $R_{H_j} = R_j$ es la que se procede a reforzar.

4.2.14.- Última iteración de cada hormiga

Con los procedimientos descritos hasta ahora, cada hormiga se aproximará lo más posible al vértice donde se ha depositado la comida y mediante el reforzamiento de los caminos más cortos es probable que llegue a través de las rutas más óptimas posibles.

Si consideramos que para cualquier hormiga, a medida que se aproxima a la fuente de comida, va descartando soluciones poco factibles, cuando esté a punto de llegar al vértice destino, concretamente cuando desde el vértice que es su posición actual la hormiga en cuestión pueda *oler* el alimento, no hace falta hacer el proceso aleatorio para elegir una arista. Será suficiente con tomar aquella arista que une al vértice actual con la comida; si se toma un camino distinto a éste, se ocasionará que la ruta gerenarada sea más larga. El argumento anterior se basa en el *teorema de la desigualdad del triángulo*. Esta consideración es válida debido a que una de las restricciones del algoritmo propuesto es que la gráfica usada haya sido dividida con la triangulación *Delaunay*.

4.3.- Caso particular

En caso de que la hormiga H_i no haya llegado al vértice destino y no queden delante de ella aristas disponibles para elegir, se le permite a la hormiga elegir cuaquiera de las aristas conectadas al vértice al que ha llegado.

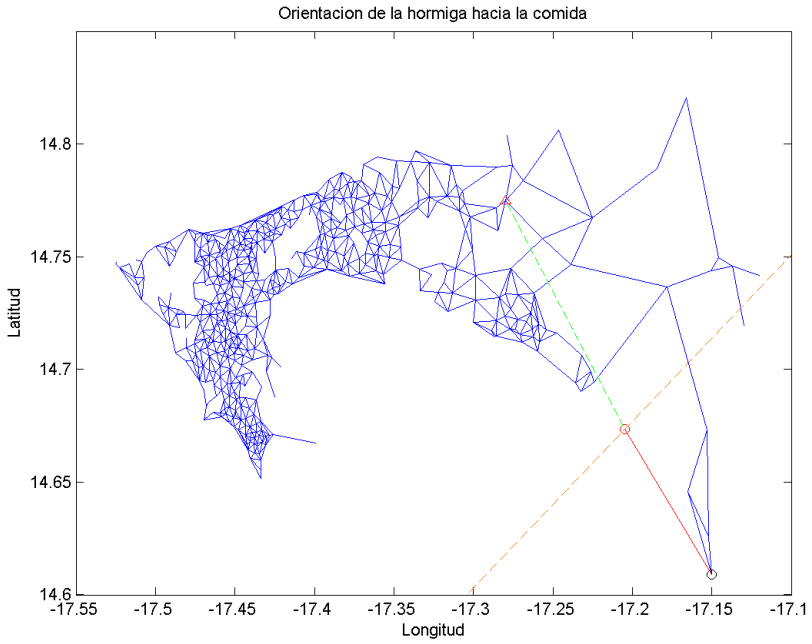


Figura 4.3: Orientación de una hormiga cualquiera hacia la comida sin más aristas por elegir.

En la figura 4.3 se muestra este caso en el que la hormiga H_i está actualmente en el vértice mostrado con un círculo rojo y se ha orientado correctamente hacía el vértice de la comida (simbolizado con un triángulo rojo) trazando el vector auxiliar $\overrightarrow{R}$ (mostrado con entrecortado en verde) y la perpendicular $\overrightarrow{R_{\perp}}$ (línea entrecortada naranja). Se observa que en ese punto a la hormiga le es imposible continuar avanzando en esa dirección por lo que su única opción es tomar el único camino disponible (arista en rojo) aunque éste vaya en la dirección contraria al

vector auxiliar; una vez que la hormiga llegue al vértice mostrado en negro le será más fácil llegar al vértice de la comida.

De igual manera para este caso, en el que hay una cavidad no convexa en la gráfica, el número de operaciones ya no conserva la garantía de que se descubrirá una ruta corta, subóptima por lo menos, con una complejidad algorítmica de $O(n)$.

4.4.- Funcionamiento del algoritmo ACO propuesto

Ahora se procede a explicar el funcionamiento de las etapas descritas en la sección 4.1 y el orden de ejecución de éstas. Antes que nada, se aclara que la triangulación *Delaunay*, completa y restringida, el diagrama de *Voronoi* se producen una sola vez, con el objetivo de tener esta información precargada y lista para que el algoritmo *ACO* haga uso de ellas, por ello en el cálculo de la complejidad del algoritmo *ACO* propuesto se excluyen sus respectivas complejidades algorítmicas.

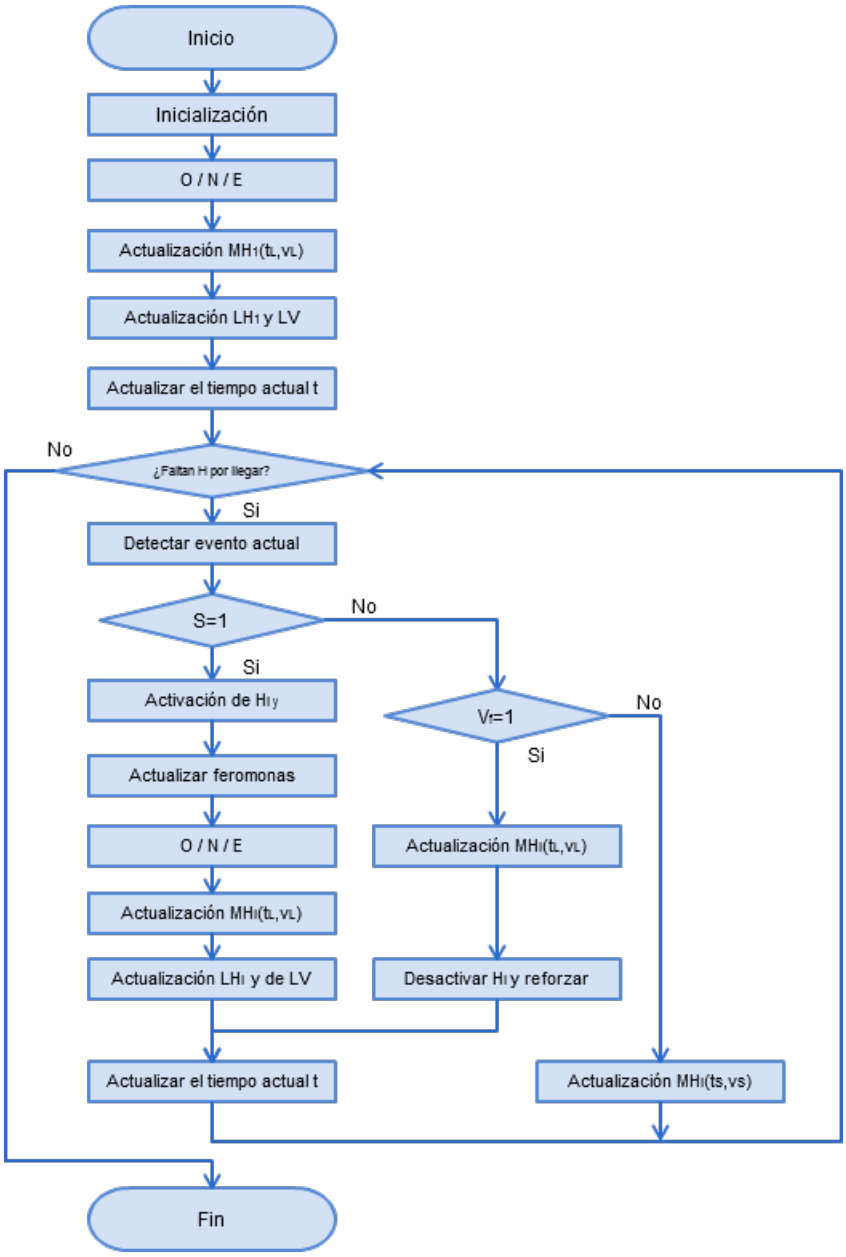


Figura 4.4: Diagrama de flujo que integra los procesos descritos en la sección 4.1

4.4.- Análisis algorítmico

En el presente trabajo se ha dado especial importancia a optimizar la cantidad de recursos temporales y espaciales del algoritmo *ACO* con la hipótesis de que este algoritmo puede ser tan bueno y eficiente como los algoritmos voraces tradicionales y la programación dinámica. Finalmente al término de esta sección se revelará el valor óptimo de q_H. Uniendo a los procesos descritos en la sección 4.1 en el orden en que se han especificado en el diagrama de flujo de la figura 4.4, la complejidad algorítmica de cada etapa se describe como sigue:

Proceso de *Inicialización*. Esta etapa corresponde a la carga e inicialización de las siguientes estructuras de datos y variables:

- Cargar la triangulación *Delaunay* restringida en una lista ligada: $O(n+m)$
- Calcular la distancia entre los vértices que componen a la lista de adyacencia: $O(n+m)$
- Asignar el nivel mínimo de feromonas para todas las aristas de la lista de adyacencias: $O(n+m)$
- Número de hormigas que tendrá la colonia: $O(1)$
- Cargar e inicializar las listas de vértices visitados por hormigas $O(q_H)$
- Calendarización de las salidas de hormigas: $O(q_H)$
- Velocidad de desplazamiento de las hormigas: $O(1)$
- Distancia máxima de percepción de las feromonas $O(1)$

Proceso *O/N/E (Orientación/Normalización/Elección).* Como ya se mencionó, la triangulación *Delaunay* ocasiona que en promedio cada vértice esté conectado a otros seis, por lo cual, cuando una hormiga cualquiera, que recién sale del hormiguero, o llega a un nuevo punto de desición, haga esa misma cantidad promedio de operaciones para las siguientes etapas:

- Orientación: $\Theta(6)$
- Normalización: $\Theta(6)$
- Elección: $\Theta(6)$

Proceso de *Actualización de la matriz de hormigas* ($MH_1(t_L, v_L)$). Sea cual sea el camino que se elija, se actualiza la matriz de hormigas, este proceso se desglosa en las siguientes operaciones:

- Calcular el tiempo de llegada t_L al siguiente vértice, conn base en el tiempo actual y a la velocidad constante que llevan las hormigas: $O(1)$
- Actualización de la matriz de hormigas, lo que implica actualizar en la matriz de hormigas al vértice al cual eligió dirigirse v_L y el instante de tiempo en el cual habrá de llegar: $O(2)$

Proceso de *Actualización de listas de vértices* (LH_1) De manera similar, una vez que la hormiga ha escogido un nodo al cual dirigirse, éste debe ser agregado a la lista de vértices que visita la hormiga.

- Mientras haya hormigas que no han llegado a la comida

Proceso de *Actualización de listas de vértices que son ocupados por hormigas (V).* De igual manera hay que indicar a esta lista que del hormiguero ha salido el primer agente y que éste mismo se dirige hacia el correspondiente vértice v_L que eligió.

- Actualización de la lista de vértices: $O(1)$

Proceso de *Actualización del tiempo presente (t).* Dado que ya se crearon los tiempos de salida para todas las hormigas, y que la primera de ellas siempre sale del nido en el tiempo $t = 0$, hay que encontrar en que momento ocurre el siguiente evento, el cual puede ser una salida del hormiguero o la llegada a un vértice.

- Detección y correspondencia del evento actual $O(klogq_H)$

Ciclo while *el cual se reprite mientras falten hormigas por llegar al vértice de la comida.* Recordemos que se especificaron ciertas codificaciones para los estados de las hormigas, luego entonces la forma más fácil de averiguar esto es comparando la suma de las hormigas que están desactivadas con el total de hormigas, es decir se tiene que $\sum_{i=1}^{q_H} MH_{i,1} > -q_H$, al cumplirse esta condición lógica se sabe que no todas las hormigas han encontrado el camino del punto inicial al final. La verificación de esta condición siempre toma un tiempo constante: $O(q_H)$

Todavía faltan hormigas por llegar.

Proceso de *detección del evento actual.* Como ya se mencionó, esto simplemente corresponde a un subproducto del proceso de detección, por lo que no requiere de tiempo extra. Se ejecuta siempre en tiempo constante: $O(1)$

Verificar condición: *¿El evento actual se trata de la salida de un vértice (S)?* Esto depende del resultado de la etapa anterior, la cual sólo puede derivar en dos resultados, en una salida de un vértice (ya sea del hormiguero o cualquier otro) o la llegada a un vértice por parte de alguno de los agentes.

Se trata de la salida de un vértice $S = 1$.

Proceso de *activación de la hormiga correspondiente al tiempo y evento actuales* H_i. Como se dijo en la sección de diseño, se considera que en un principio todas las hormigas tienen un estado inactivo, el cual está codificado, por lo que cambiar de estado a éste en su respectivo lugar en la matriz de hormigas $MH_{i,1}$ algorítmicamente toma: $O(1)$

Proceso de *Actualización de listas de vértices que son ocupados por hormigas (V).* La actualización lleva: $O(1)$

Proceso de *Actualización de los niveles de feromonas presentes.* Antes de que la hormiga H_i pueda elegir una arista por la cual caminar, es necesario que las aristas que están conectadas al vértice en el que actualmente se sitúa estén actualizadas. Para esto hay que:

- Verificar cuales son los vértices que son candidatos a actualizarse revisando a la lista de adyacencia T_i, lo cual lleva: $O\left(G\left(v_i\right)\right)$ que en promedio es de $\Omega(6)$
- Revisar en R cuales hormigas han transitado recientemente por aquellos. Del conjunto tomado en el paso anterior se verifica la lista de vértices ocupados por hormigas V, en las posiciones correspondientes, y se toman a los últimos elementos de estos. En promedio se hace uso de $\Omega(6)$
- Cuando ya se conocen las hormigas que pasaron por las aristas que tiene conectadas el vértice actual, se toma como referencia el tiempo actual t y el tiempo de salida del hormiguero de este grupo de hormigas del que se habla, y se usa la función de percepción (evaporación de feromonas) para determinar la cantidad que le corresponde a cada arista. Lo cual conlleva en promedio $\Omega(6)$ operaciones.

Proceso *O/N/E (Orientación/Normalización/Elección).* Como ya se mencionó, para cualquier agente tomará en promedio $\Omega(6)$ para cada subetapa.

Proceso de *Actualización de la matriz de hormigas* ($MH_i(t_L, v_L)$). Sus dos subprocesos toman siempre un tiempo constante de $O(1)$ para calcular el tiempo t_L y $O(2)$ para actualizar la matriz de hormigas para los elementos correspondientes en el renglón i.

Proceso de *Actualización de listas de vértices* (LH_i) Toma: $O(i)$

Proceso de *Actualización de listas de vértices que son ocupados por hormigas (V).* Actualización de la lista de vértices: $O(1)$

Proceso de *Actualización del tiempo presente (t).* Detección y correspondencia del evento actual $O(klogq_H)$

Se trata de la llegada a un vértice $S = 0$.

Checar si el vértice al cual se arriva es el de la comida. Esto se hace comparando directamente al elemento de la matriz de hormigas que corresponde al vértice al cual se llega, con el vértice que se designó como el de la comida. Lo cual toma $O(1)$

Se ha llegado al vértice de la comida.

Proceso de *Actualización de la matriz de hormigas* ($MH_i(t_L, v_L)$).

Proceso de *Desactivación de la hormiga* H_i. Conlleva una única operación $O(1)$

Proceso de *Reforzamiento del camino más corto hasta ahora.* Este procedimiento implica:

- Calcular la distancia recorrida por la hormiga H_i, lo cual se puede hacer en tiempo constante, pues se especificó que en ningún momento los agentes detienen su marcha a través del grafo, y que además se conoce el instante

de tiempo en el cual salieron del nido y se conoce también el tiempo en el cual han terminado su recorrido.Por otro lado, para conocer el camino más corto hasta ahora requiere usar una cola de prioridad con todas las distancias totales calculadas y seleccionar al elemento mayor prioridad, el cual almacena la distancia más corta recorrida y la hormiga que lo caminó H_{sh}; esto toma: $O(logq_H)$

- Actualizar el rastro de feromonas de las aristas por las que cruzó la hormiga H_{sh}, lo cual depende únicamente de la cantidad de vértices que haya atravesado, por lo que tomará $O(R_{sh})$

Se ha llegado a un vértice que no es el de la comida.

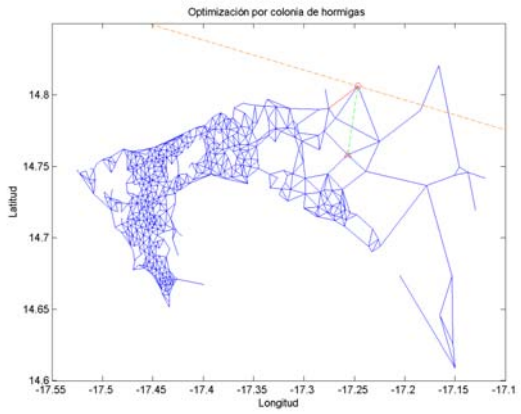
Proceso de *Actualización de la matriz de hormigas* ($MH_i(t_L, v_L)$). Al tratarse de un vértice cualquiera, esto indica que hay que volver a elegir un camino que lleve a la hormiga H_i hacia el vértice de la comida, por lo que este evento de llegada, se transforma inmediatamente en la salida de un vértice. Entonces sólo basta con indicar en la matriz de hormigas que se está saliendo de un nuevo vértice y se desconoce el tiempo de llegada al siguiente, pues aún no ha escogido al siguiente punto geográfico por visitar; esto necesita de $O(4)$ operaciones.

Ya han llegado todas las hormigas al destino. FIN DEL ALGORITMO.

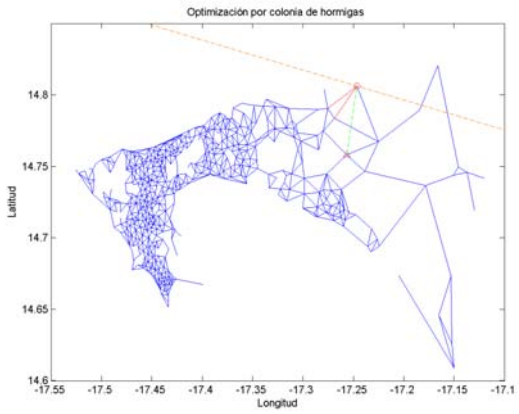
Capítulo 5

5.1.- Pruebas realizadas y análisis de resultados

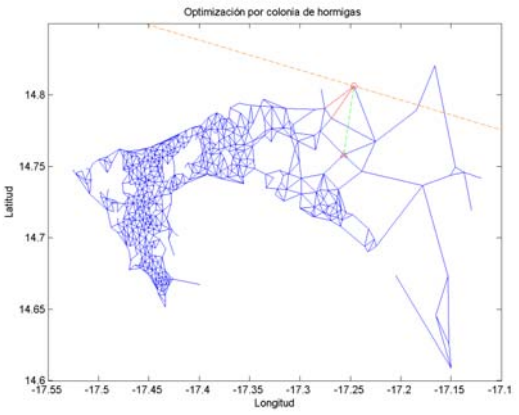
Siguiendo el diagrama de flujo mostrado en el capítulo anterior, a continuación se muestra un breve ejemplo de una colonia de 3 hormigas que buscan el camino más corto hacia la comida a través de las aristas de la triangulación *Delaunay* restringida.



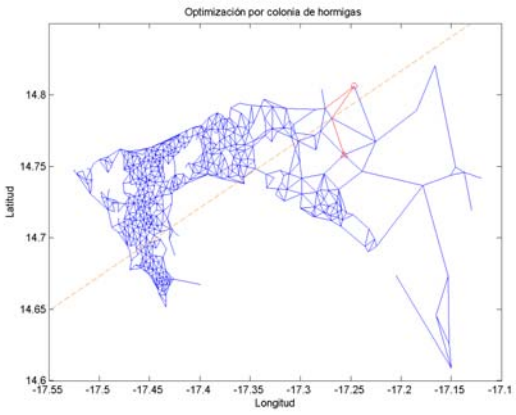
(a) Primer hormiga saliendo del nido



(b) Segunda hormiga saliendo del nido

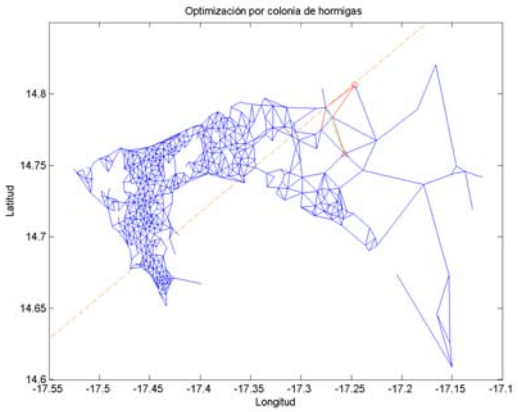


(c) Tercera hormiga saliendo del nido

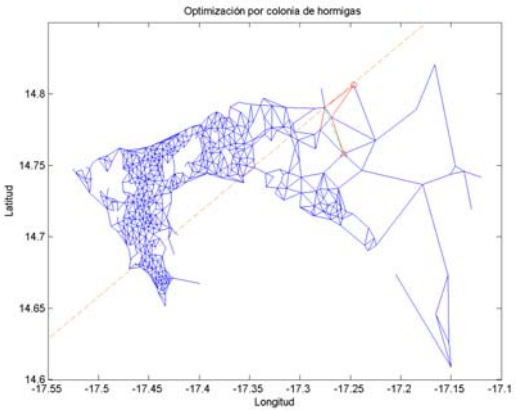


(d) La segunda hormiga elige una arista nueva

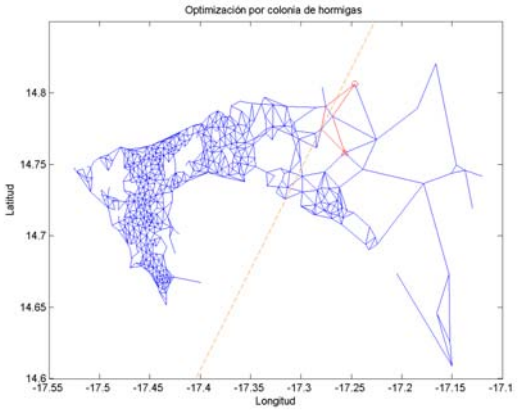
Figura 5.1: Búsqueda del camino más corto usando 3 hormigas



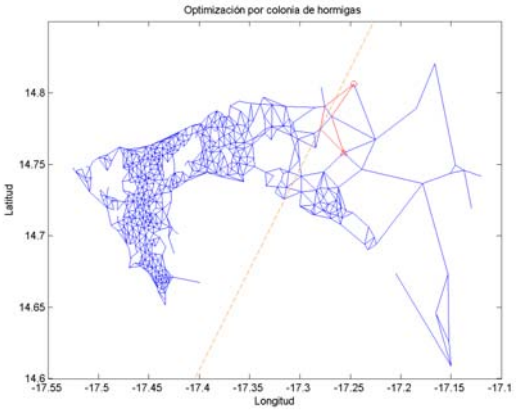
(a) La primer hormiga elige una arista nueva



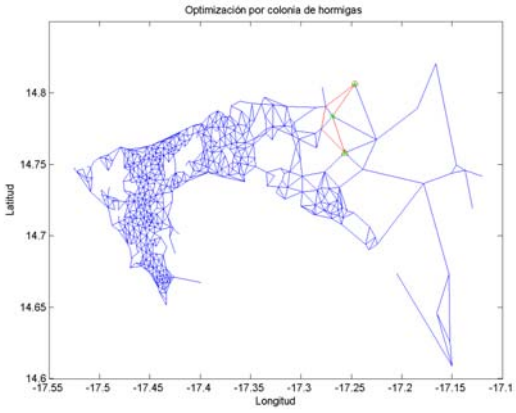
(b) La tercer hormiga elige una arista nueva



(c) La segunda hormiga llega a la comida



(d) La primer y tercer hormigas llegan a la comida



(e) El camino más corto señalado con asteriscos verdes

Figura 5.2: Búsqueda del camino más corto usando 3 hormigas

En experimentos posteriores se medirá la efectividad del algoritmo propuesto para encontrar la ruta más corta entre todos los pares de vértices en una gráfica.

5.2.- Pruebas con un grafo aleatorio

Para probar la utilidad del algoritmo propuesto se ha creado una gráfica con 40 vértices con coordenadas aleatorias uniformemente distribuídas, para los cuales se ha construído su correspondiente triangulación *Delaunay* y se ha mantenido intacta la gráfica, es decir no se han desconectado ni agregado aristas. El objetivo de este experimento es obtener la ruta más corta entre todos los pares de vértices y comparar su longitud con las distancias mínimas que da como resultado aplicar el algoritmo de *Dijkstra* para el mismo grafo; así como también comparar el número de iteraciones empleadas por la optimización por colonia de hormigas. A continuación se muestra la triangulación de la gráfica aleatoria generada:

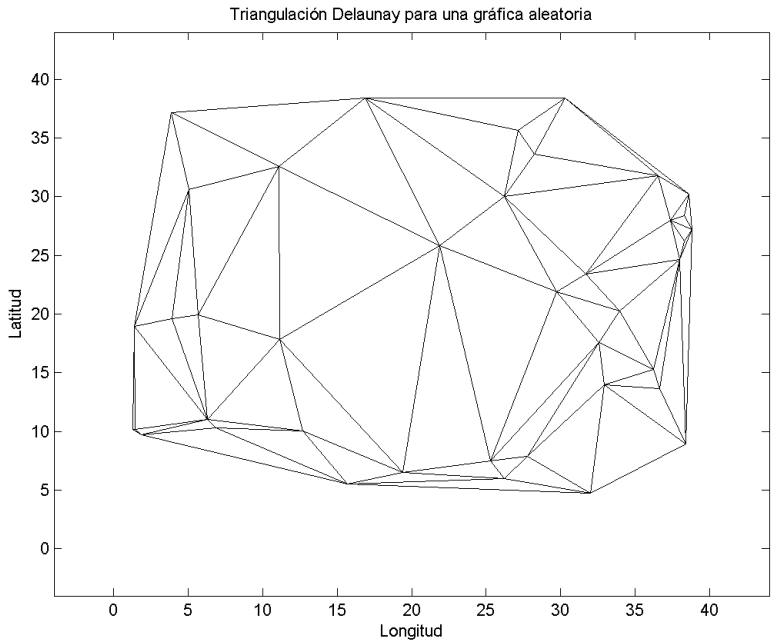


Figura 5.3: Triangulación Delaunay de una gráfica aleatoria.

El algoritmo de *Dijkstra* proporciona la ruta más corta entre todos los pares de vértices usando a lo más $O(n^3)$ operaciones, que para el caso generado son 64000. Como parte del análisis estadístico para medir la efectividad del algoritmo de optimización por colonia de hormigas propuesto, en cada experimento se utilizaron diferentes cantidades de hormigas, en el rango de $\lfloor\sqrt{40}\rfloor \leq q_H \leq 8 \cdot \lfloor\sqrt{40}\rfloor$, a su vez, se efectúa la búsqueda de la ruta más corta entre todos los pares de nodos, y éste proceso se vuelve a realizar 30 veces con la misma cantidad de hormigas para proseguir con una colonia más numerosa. En total se trata de un conjunto de 240 experimentos.

Los resultados promedio del número de iteraciones utilizadas por cada colonia de hormigas muestran una tendencia altamente lineal para encontrar los caminos más cortos entre todos los pares de vértices, esto viene representado por la línea continua negra de la figura 6.4. Mientras que la línea punteada nos muestra el máximo número de iteraciones que utilizaría como máximo el algoritmo de *Dijkstra* en su implementación de $O(n^3)$ para resolver el mismo problema.

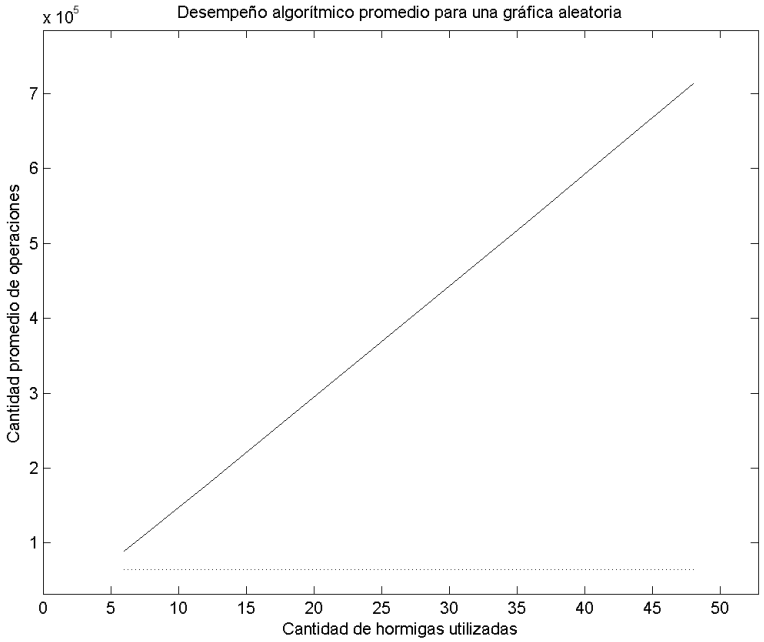


Figura 5.4: Desempeño algorítmico promedio para una gráfica aleatoria.

Cuando se halla la ruta más corta por parte de las hormigas, ésta se compara con la que se encuentra usando el algoritmo de *Dijkstra*. Sea la matriz Q una matriz cuadrada de $n \times n$ inicializada en ceros, aquella que almacenará la calidad de las rutas descubiertas y sea $d_{i,j}$ la distancia más corta descubierta por las hormigas para ir del vértice i al j y sea $D_{i,j}$ la distancia más corta obtenida por el algoritmo de *Dijkstra*, entonces la razón:

$$Q_{i,j} = \frac{d_{i,j}}{D_{i,j}}$$

mide la calidad de la solución encontrada en comparación con un algoritmo bastante conocido y empleado. La calidad promedio de un conjunto de experimentos en el cual se usa la misma cantidad de hormigas se obtiene al sumar todos los valores de sus respectivas matrices Q y dividir este resultado entre 30, que es el número de repeticiones que se hizo para cada colonia de hormigas.

Los resultados promedio de la calidad de las soluciones halladas por cada colonia de hormigas se muestran en la siguiente gráfica:

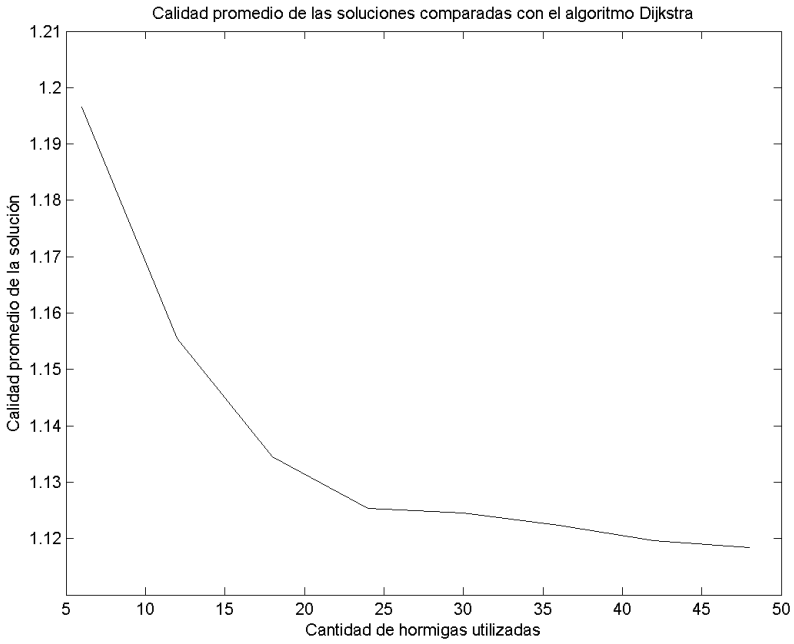


Figura 5.5: Calidad promedio de las soluciones comparadas con el algoritmo Dijkstra.

La gráfica anterior nos indica que a mientras se incrementa el número de hormigas en cada colonia mejores resultados son hallados. Sin embargo, como se puede notar estos resultados son subóptimos comparados con los resultados obtenidos con el algoritmo de *Dijkstra*.

La siguiente tabla proporciona un poco más de información al respecto:

n	Hormigas	I. totales	C. con Dijkstra	I. promedio	C. rutas cortas	D. E. rutas cortas
40	6	89429.20	3.31	102.79	1.1966	0.0124
40	12	176693.53	6.54	203.10	1.1555	0.0065
40	18	265185.93	9.82	304.81	1.1344	0.0044
40	24	353605.00	13.10	406.44	1.1253	0.0058
40	30	443028.93	16.41	509.23	1.1245	0.0038
40	36	532751.73	19.73	612.36	1.1223	0.0031
40	42	622726.53	23.06	715.78	1.1196	0.0031
40	48	713497.27	26.43	820.11	1.1184	0.0029

Tabla 5.1: Experimentos realizados con diversas graficas aleatorias comparando la calidad de las soluciones halladas y número de iteraciones usando el algoritmo ACO y el de Dijkstra

La columna *C. con Dijkstra* (Comparación de iteraciones con el algoritmo de *Dijkstra*) indica la razon de iteraciones promedio usadas comparadas con el peor caso que considera el algoritmo de *Dijsktra*, esta columna se obtiene al dividir los valores de la columna I. totales (Iteraciones totales promedio) entre 64000, que corresponde con el número de iteraciones máximas que usaría el algoritmo de *Dijkstra* en su implementación menos óptima, $O(n^3)$, para $n = 40$ vértices. Notemos que al tener un crecimiento aproximadamente lineal a medida que se incrementa el número de hormigas en la colonia, se puede considerar algorítmicamente que ambos algoritmos tienen la misma complejidad. Por otro lado, la columna *I. promedio* (Iteraciones promedio para cada par de vértices por cada hormiga) hace referencia a las iteraciones promedio que emplea cada hormiga para encontrar el camino más corto entre dos vértices cualesquiera, ésto se calculó tomando los valores de la columna *I. totales* y dividiéndolos entre 1560 que corresponde a la cantidad de trayectorias que le es posible calcular a un algoritmo de ruteo, es decir

$n \cdot (n-1)$. Otra cosa que se debe remarcar, es la calidad de las soluciones encontradas, las cuales se muestran en la columna *C. rutas cortas* (Calidad de las rutas más cortas comparadas con *Dijkstra*), ésta tiende a aumentar mientras más cercano sea su valor a 1, pues indicará que se encontró la misma solución que *Dijkstra*; la calidad de las soluciones aumenta conforme lo hace el número de hormigas y sin embargo continúan siendo subóptimas, acercándose asintóticamente a ser sólo un 10% más largas que las soluciones proporcionadas por *Dijksktra*, la desviación estándar en la columna *D. E. rutas cortas*, nos indica que con el incremento de agentes cada vez más se acerca a la distancia media de las rutas más cortas.

5.3.- Pruebas con un grafo que tiene cavidades no convexas

A diferencia del experimento anterior, en esta ocasión no se consideró crear una gráfica aleatoria y desconectarle aristas para crear cavidades no convexas, en lugar de ello se tomó la gráfica de la triangulación *Delaunay* restringida, la cual ya les incluye. Se realizó el mismo procedimiento de búsqueda de las rutas más cortas entre todos los pares de vértices del grafo.

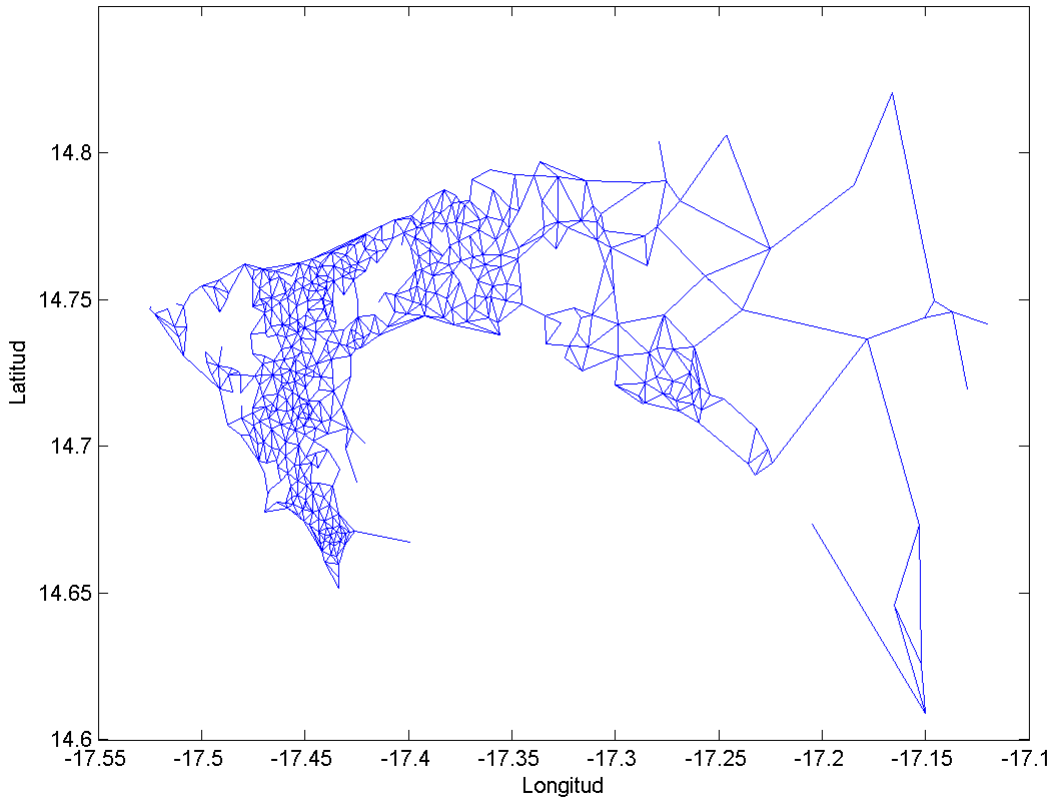


Figura 5.6: Triangulación Delaunay restringida que resulta ser un grafo con cavidades no convexas.

Se esperaba que el algoritmo *ACO* hallará todas las soluciones del problema a pesar de la presencia de obstáculos que dificultan tanto la orientación como la exploración de los caminos, sin embargo no le fue posible completar la tarea debido a que en varias ocasiones las hormigas se estancan cerca de estos obstáculos lo que les complica avanzar hacia el vértice destino con normalidad. Se observó además que para algunos casos, con la presencia de las cavidades no convexas, ir del vértice i al j le es más difícil que ir en el sentido inverso.

Para este experimento no se muestran la cantidad de iteraciones empleadas, debido a que cuando las hormigas llegan a estancarse en una parte de la gráfica las iteraciones incrementan desmesuradamente sin que un límite promedio haya sido registrado. Se repitió el mismo experimento para tratar de hallar un límite pero los procesos de elección aleatorios impiden que se repitan las mismas condiciones.

Además el experimento anterior resalta el hecho de que no siempre se encuentra la ruta más corta, por lo cual el promedio de la calidad de las rutas encontradas varía. Por ello se decidió no usar el algoritmo *ACO* para obtener las rutas más cortas entre todos lo pares de nodos, en su lugar se ha usado el algoritmo de *Dijkstra* debido a su estabilidad en las soluciones contruídas, es decir, se eligió porque para cada ejecición aquel hallará el mismo conjunto de soluciones. De esta manera la solución de movilidad que se propone en secciones posteriores resulta ser fácilmente repetible y estable. Recordemos que el algoritmo de ruteo se emplea para inferir los sitios intermedios más probables entre los puntos geográficos encontrados en el historial de cada usuario.

5.4.- Visualización de mapas de calor por hora

El propósito de este experimento es visualizar tanto la actividad que se tiene en el transcurso del tiempo como las celdas de *Voronoi* adyacentes que presentan una gran cantidad de tránsito. El procedimiento así como la lógica utilizados se describe a continuación.

Como se mencionó anteriormente mediante el uso de consultas a la base de datos y el correspondiente filtrado de aquellas, se obtienen los usuarios que se desplazan a través de la ciudad en el intervalo de tiempo que se le indique al gestor. Las llamadas rutas en bruto, son redirigidas usando la matriz de precedentes del algoritmo de *Dijkstra*, debido a las inestabilidades encontradas en el algoritmo de optimización por colonia de hormigas cuando se trata de hallar rutas entre puntos muy distantes entre sí. Como ya se explicó el redireccionamiento es necesario. Inmediatamente se almacenan estas nuevas rutas en un único arreglo bidimensional, cuya primera columna guarda el identificador del usuario que se está desplazando, y la segunda que guarda los vértices que el respectivo usuario va visitando. Por lo que el arreglo $\mathcal{B}$, a grandes rasgos, tendrá la siguiente forma:

$$\mathcal{B} = \begin{bmatrix} U_1 & V_1 \\ U_2 & V_2 \\ \vdots & \vdots \\ U_{n-1} & V_{n-1} \\ U_n & V_n \end{bmatrix}$$

De manera estricta, cada U_i, con $i \in \mathbb{Z}^+$, es un vector en el cual el identificador de usuario se repite tantas veces como vértices contenga su correspondiente conjunto vértices de ruta redirigida V_i.

La razón de que ambos campos estén disponibles es que de esta manera se puede tener una cantidad distinguible de n rutas que pertenecen a esa misma cantidad de usuarios, y de igual forma, como se dijo en el capítulo 4, cada una de estas rutas tiene al menos 2 vértices. El procedimiento y la lógica usados para construir los mapas de calor se explica a continuación.

Un primer paso para crear los mapas de calor, es tener una matriz $\mathcal{H}$, de tamaño $490x490$, rellena con ceros, que servirá como una matriz de adyacencia a la vez que de ponderaciones; además se debe considerar tener una variable pivote k, igualmente inicializada en cero, para encontrar rápidamente a la arista $\overrightarrow{v_i v_j}$ por la que más usuarios han cruzado ya sea en el sentido $\overrightarrow{v_i v_j}$ o en el sentido inverso, yendo de $\overrightarrow{v_j v_i}$. Posteriormente se toman los vértices visitados por cada uno de los usuarios, el usuario U_i transita por el conjunto de vértices $V_{U_i} = V_i$, cuya dimensión mínima es $dim_{min}(V_i) = 2$, evidentemente porque la cantidad mínima de veces que alguíen puede pasar por una arista de la triangulación *Delaunay* restringida es 1; aun cuando la dimensión máxima sea desconocida es posible tomar a los elementos de V_i por pares consecutivos desde el inicio hasta el final del arreglo.

Sea $V_i = \begin{bmatrix} v_{i,1} & v_{i,2} & \cdots & v_{i,dim(V_i)-1} & v_{i,dim(V_i)} \end{bmatrix}$, donde el elemento $v_{i,j}$ es el vértice j-ésimo visitado por el usuario U_i, es posible tomar al par inicial $v_{i,1}$ y $v_{i,2}$ e incrementar en 1 el valor que guarda la matriz del mapa de calor en la posición $\mathcal{H}_{(v_{i,1},v_{i,2})}$ y $\mathcal{H}_{(v_{i,2},v_{i,1})}$ lo que representa que se ha cruzado por la arista que une a aquellos vértices. Dado que ahora ambas posiciones contienen el mismo valor, se compara a cualquiera de éstas con el valor actual del pivote, en caso de cumplirse que $k < \mathcal{H}_{(v_{i,1},v_{i,2})}$ es necesario actualizar el valor del pivote a $k = \mathcal{H}_{(v_{i,1},v_{i,2})}$.

Se debe proceder así con todos los demás elementos tomando un $v_{i,j}$ y un $v_{i,j+1}$ sin exceder $dim\,(V_i)$ y repetir secuencialmente todo esto para cada uno de los n usuarios que otrora se filtraron. Por lo que al término de este procedimento se tendrá una matriz de adyacencia ponderada con la cantidad de veces que los usuarios involucrados pasaron por ciertas aristas y adicionalmente también se habrá conseguido la cantidad máxima de veces que se cruzó por alguna arista. Este sencillo algoritmo efectúa la creación de una matriz de un mapa de calor, dentro de un intervalo de tiempo específico que se le indica al gestor de bases de datos, con una complejidad algorítmica de $O\left(\sum_{i=1}^{U_n} dim(V_i)\right)$, es decir un orden lineal respecto al total de vértices almacenados en el arreglo $\mathcal{B}$.

Como al final de esta etapa ya se conoce la cantidad mínima y máxima de veces que se transitó por las aristas de la triangulación *Delaunay* restringida, es posible normalizar a los valores almacenados en $\mathcal{H}$ respecto al pivote encontrado y reasignarlos a esa misma matriz, es decir:

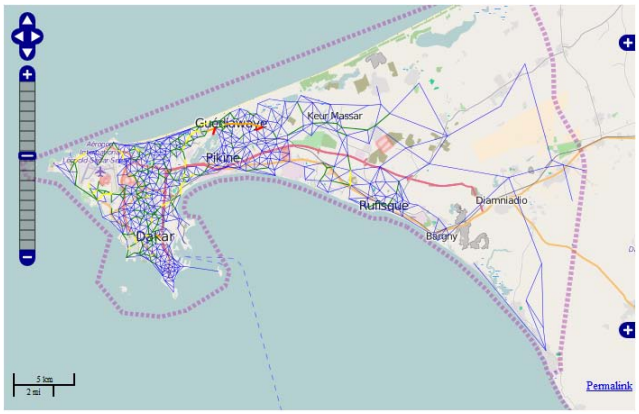
$$\mathcal{H} \leftarrow \tfrac{1}{k} \cdot \mathcal{H}$$

Por lo que ahora cada $\mathcal{H}_{i,j} \in \mathcal{H}$ se encuentra en el intervalo $[0, 1]$, esto hace más fácil separarlo en una escala de 5 intervalos para representar la cantidad de tránsito que lleva cada arista, esto será indicado mediante colores como se describe a continuación:

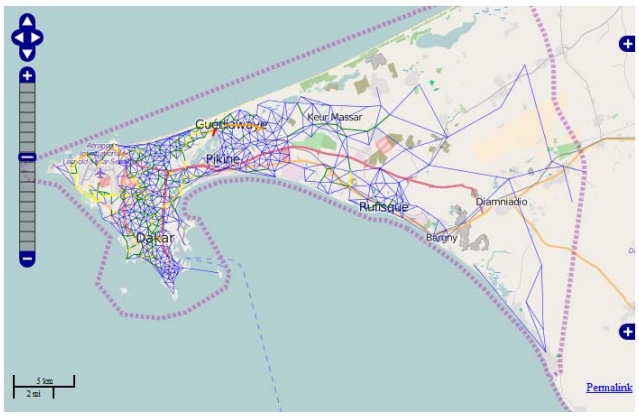
Color	**Significado**	**Intervalo numérico**
Transparente	Aristas inexistentes o descartadas de la triangulación restringida	0
Azul	Líneas frías	(0,0.2]
Verde	Líneas no tan frías	(0.2,0.4]
Amarillo	Líneas templadas	(0.4,0.6]
Naranja	Líneas no tan cálidas	(0.6,0.8)
Rojo	Líneas cálidas	(0.8,1]

Tabla 5.2: Codificación de colores para los mapas de calor

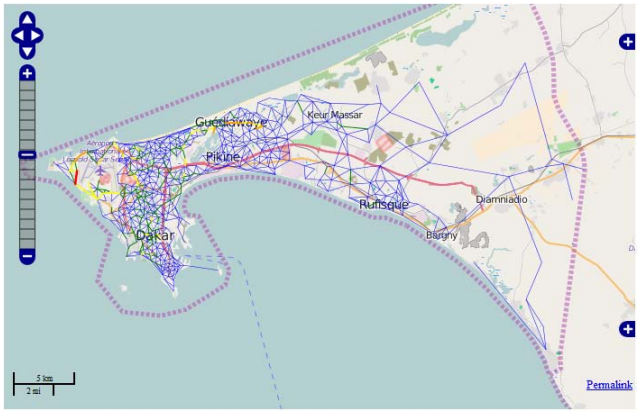
Entre más cálida sea una arista una cantidad de usuarios mayor habrá transitado a través de ella, y entre más fría sea, ocurre lo contrario. En seguida se muestran los 24 mapas de calor obtenidos para ilustrar los movimientos durante todo el día en Dakar.



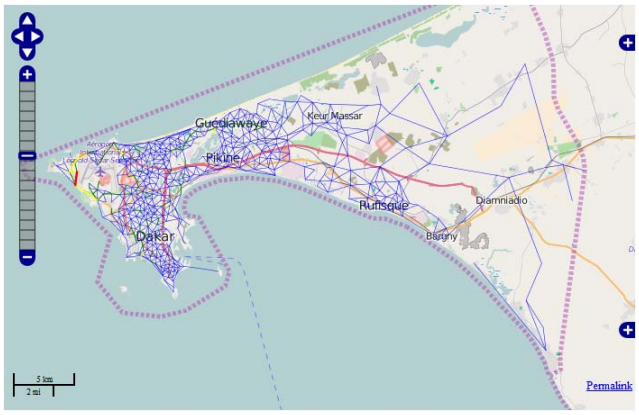
(a) Mapa de calor (entre las 0 hrs y 1 hrs)



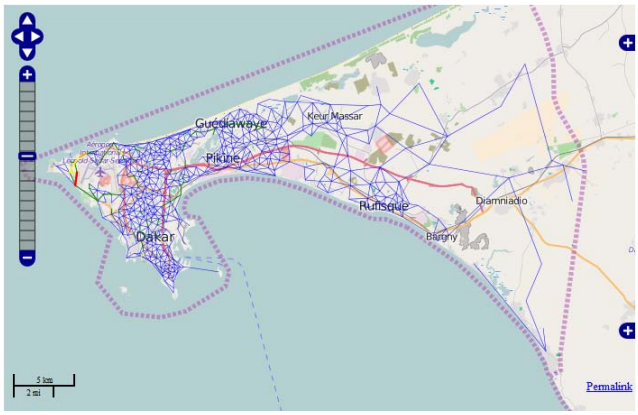
(b) Mapa de calor (entre las 1 hrs y 2 hrs)



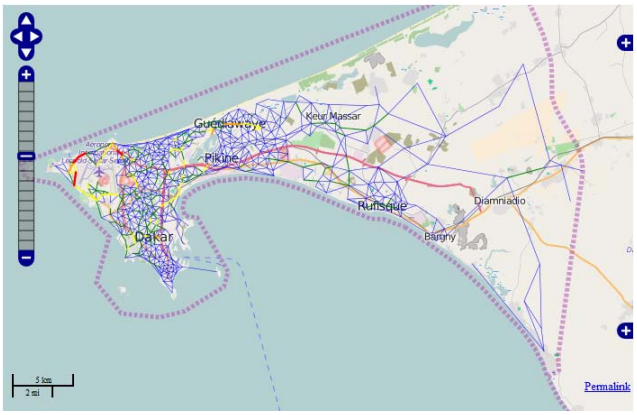
(c) Mapa de calor (entre las 2 hrs y 3 hrs)



(d) Mapa de calor (entre las 3 hrs y 4 hrs)

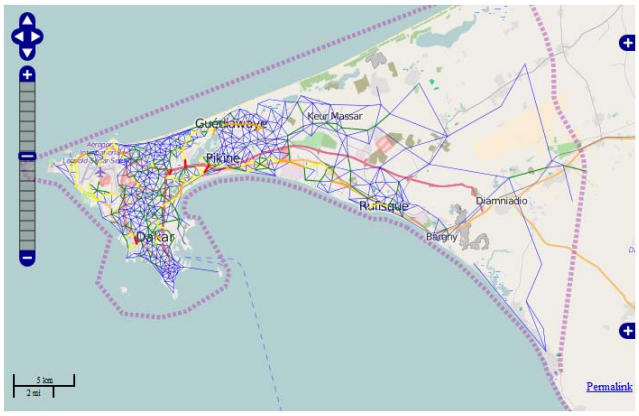


(e) Mapa de calor (entre las 4 hrs y 5 hrs)

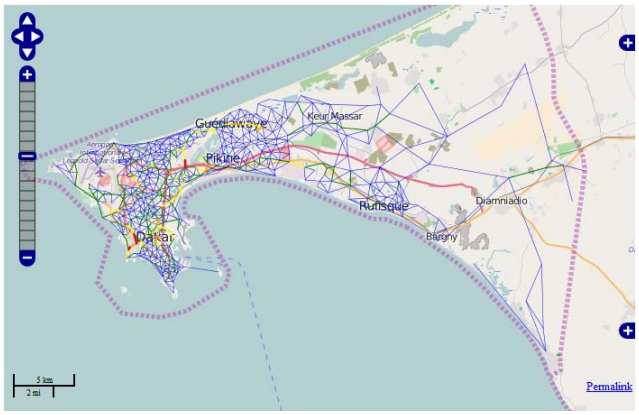


(f) Mapa de calor (entre las 5 hrs y 6 hrs)

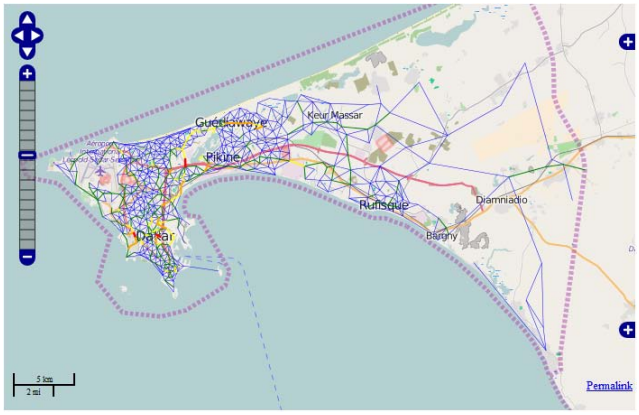
Figura 5.7: Mapas de calor de la movilidad de Dakar entre las 0:00 hrs y las 5:00 hrs



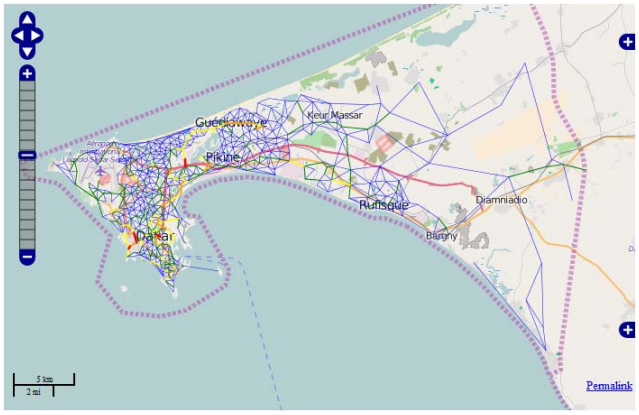
(a) Mapa de calor (entre las 6 hrs y 7 hrs)



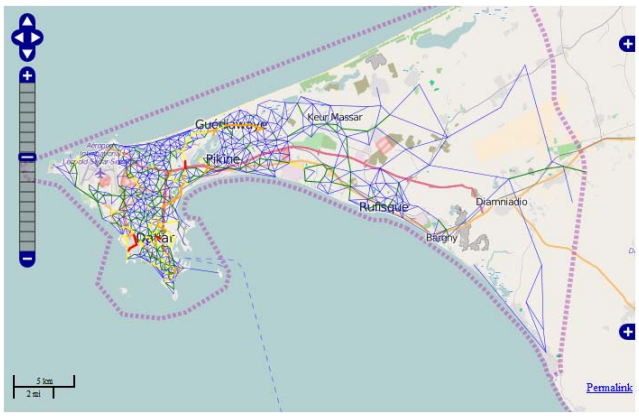
(b) Mapa de calor (entre las 7 hrs y 8 hrs)



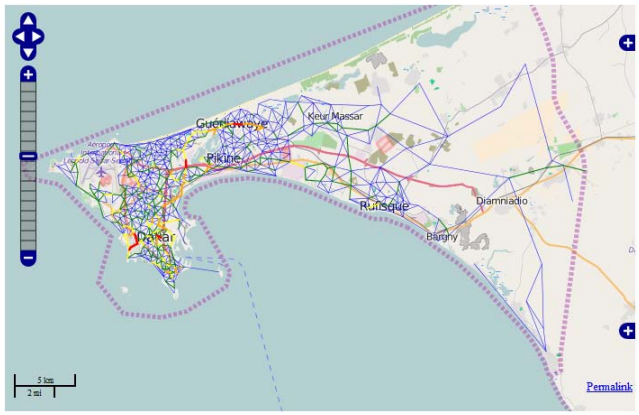
(c) Mapa de calor (entre las 8 hrs y 9 hrs)



(d) Mapa de calor (entre las 9 hrs y 10 hrs)

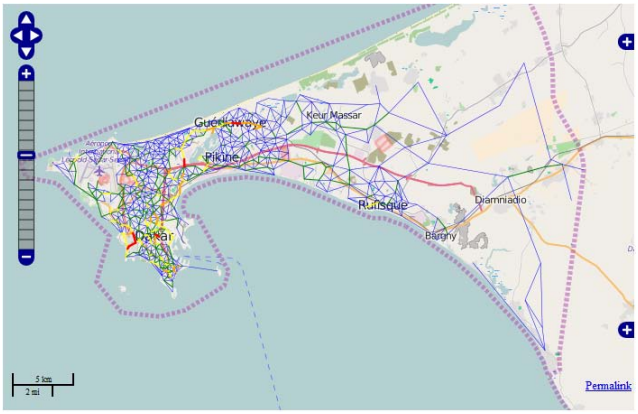


(e) Mapa de calor (entre las 10 hrs y 11 hrs)

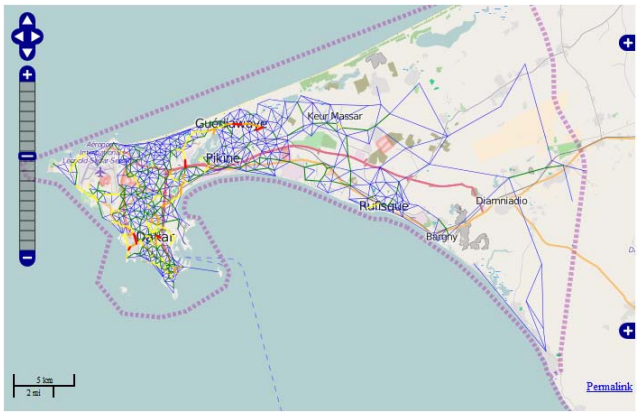


(f) Mapa de calor (entre las 11 hrs y 12 hrs)

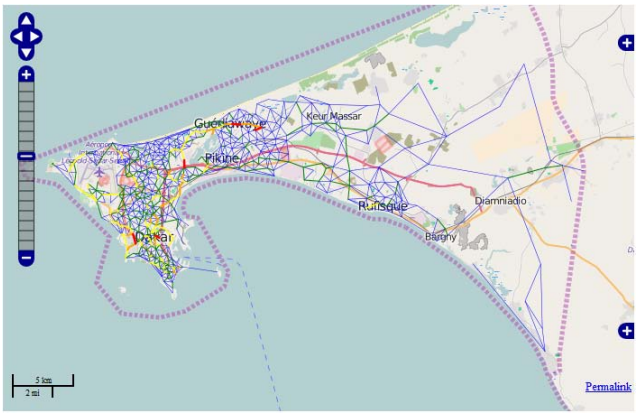
Figura 5.8: Mapas de calor de la movilidad de Dakar entre las 6:00 hrs y las 11:00 hrs



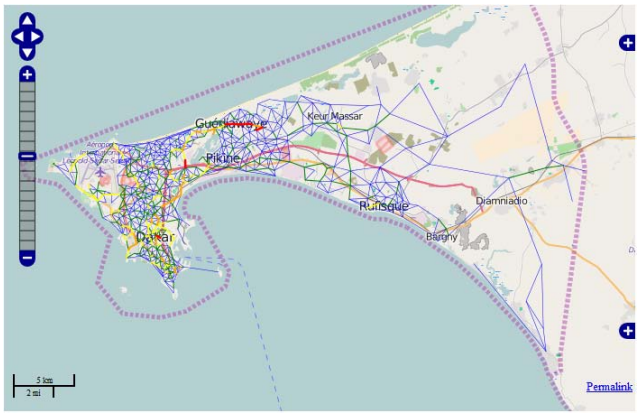
(a) Mapa de calor (entre las 12 hrs y 13 hrs)



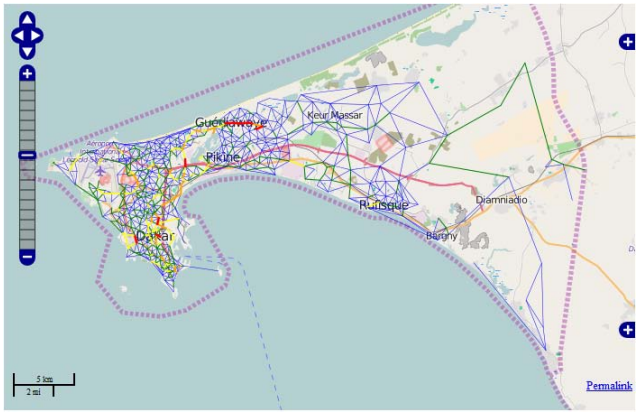
(b) Mapa de calor (entre las 13 hrs y 14 hrs)



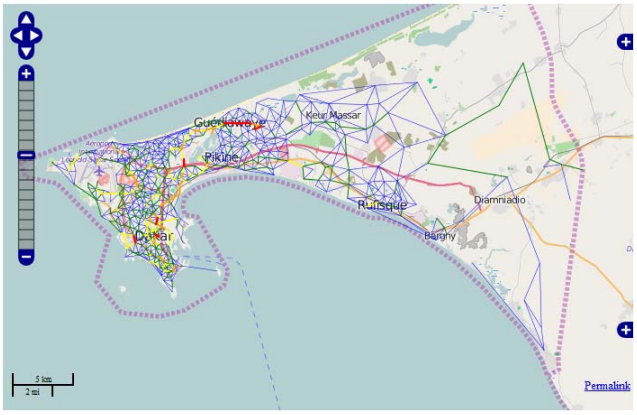
(c) Mapa de calor (entre las 14 hrs y 15 hrs)



(d) Mapa de calor (entre las 15 hrs y 16 hrs)

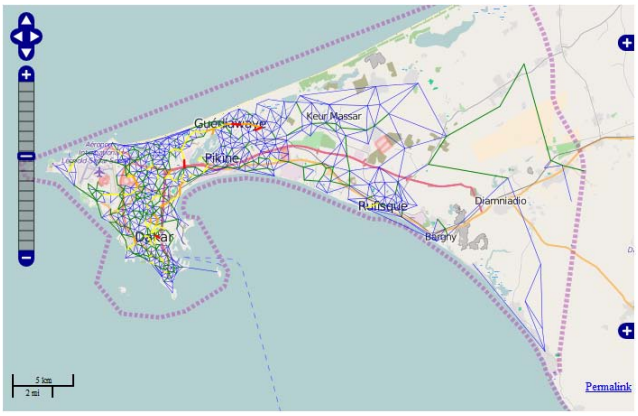


(e) Mapa de calor (entre las 16 hrs y 17 hrs)

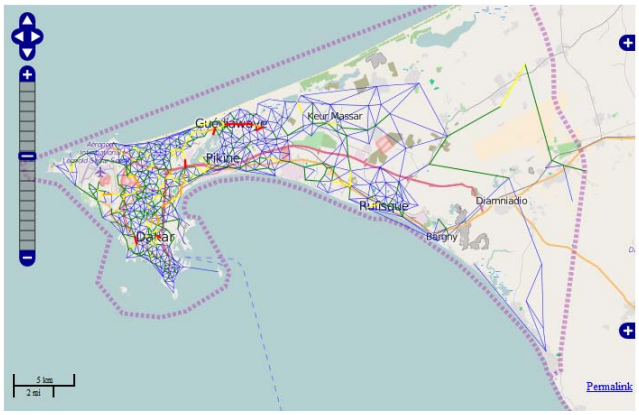


(f) Mapa de calor (entre las 17 hrs y 18 hrs)

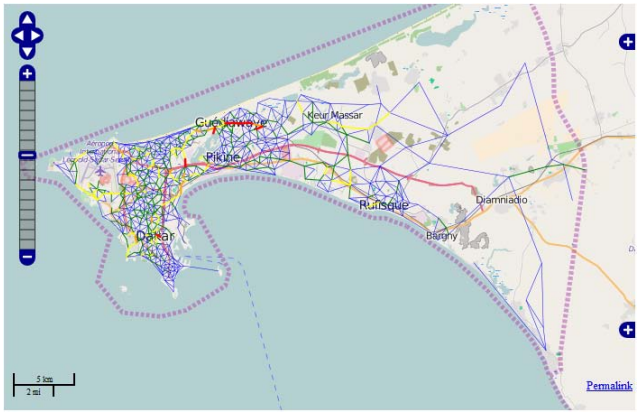
Figura 5.9: Mapas de calor de la movilidad de Dakar entre las 12:00 hrs y las 17:00 hrs



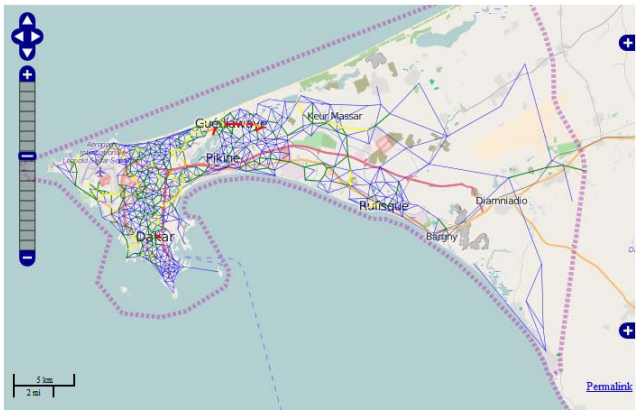
(a) Mapa de calor (entre las 18 hrs y 19 hrs)



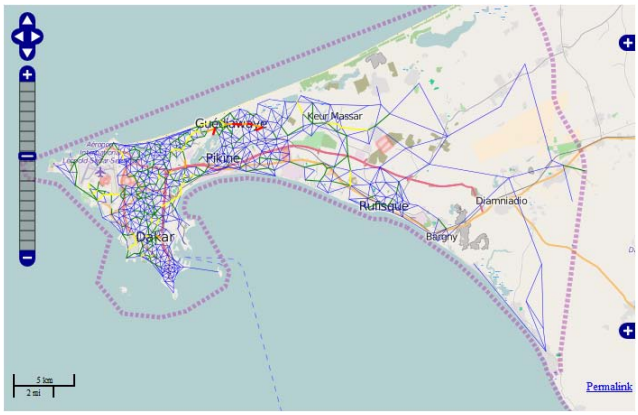
(b) Mapa de calor (entre las 19 hrs y 20 hrs)



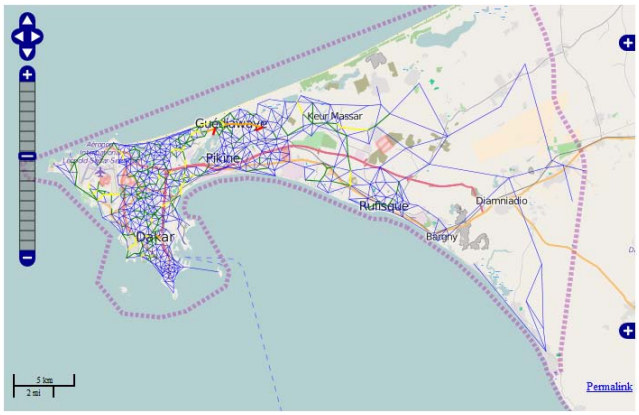
(c) Mapa de calor (entre las 20 hrs y 21 hrs)



(d) Mapa de calor (entre las 21 hrs y 22 hrs)



(e) Mapa de calor (entre las 22 hrs y 23 hrs)



(f) Mapa de calor (entre las 23 hrs y 24 hrs)

Figura 5.10: Mapas de calor de la movilidad de Dakar entre las 18:00 hrs y las 24:00 hrs

5.5.- Relaciones permanentes entre celdas de Voronoi

En diferentes horas del día hay diversos niveles de actividad a lo largo de Dakar, mientras que algunas aristas de la triangulación *Delaunay* restringida presentan poca afluencia en cierto periodo del día, éstas mismas pueden tener un uso masivo en un horario distinto o de igual manera podrían desaparecer por completo del mapa de calor, o viceversa. Ante estos escenarios es fácil intuir que la existencia de una arista en el futuro no depende exclusivamente del nivel de calor con el cual se le asocia en el momento en que se está observando, por este motivo se intuye que debe haber un conjunto de aristas que tienen una presencia permanente a lo largo del tiempo, es decir se sospecha de la existencia de relaciones permanentes entre celdas de *Voronoi* adyacentes, representadas mediante aristas.

Una forma sencilla de verificar la permanencia de aristas durante todo el día es mediante el uso de operadores lógicos. Nótese que si tenemos a los arreglos $X = [0, 7, 0]$ y $Y = [8, 1, -9]$ y a estos le aplicamos la operación de intersección lógica obtenemos $X \wedge Y = [0, 1, 0]$, debido a que sólo los segundos elementos de cada arreglo son distintos de 0; al usar esta misma operación para las matrices donde se almacenan los mapas de calor se vuelve sencillo descubrir los elementos permanentes y guardarlos en $\mathcal{H}_P$. Por ello, para esta etapa es necesario que primero se hayan creado 24 matrices $\mathcal{H}_h$ con $h \in \mathbb{Z}^+ \cup 0$ tal que $0 \leq h < 24$ que almacenen los valores de sus correspondientes mapas de calor, y excluir las aristas frías de la matriz $\mathcal{H}_0$, es decir del mapa de calor que corresponde al intervalo de tiempo entre las 0:00 a.m. y la 1:00 a.m, de no hacerlo se reconstruirán casi en su totalidad las aristas de la triangulación debido a que las líneas frías siempre son mayoría a comparación de las demás. Al aplicar la operación de intersección sucesivamente con cada mapa de calor se dejan sólo aquellas aristas que subsisten al paso del tiempo. Matemáticamente esto se expresa de la siguiente forma:

$$\mathcal{H}_P = \mathcal{H}_0 \bigwedge_{h=1}^{23} \mathcal{H}_h$$

A continuación se muestran las aristas permanentes halladas:

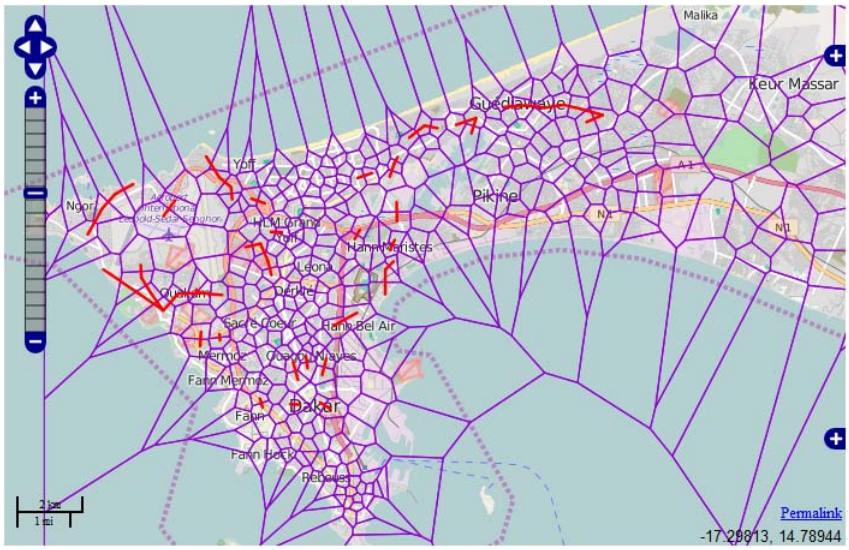


Figura 5.11: Rutas permanentes detectadas.

El resultado anterior nos da un ligero indicio del comportamiento de movilidad que está presente, sin embargo como se mira en la imagen anterior esto no es suficiente para la propuesta de una ruta de transporte. Con base en la observación puede intuirse que varias de las aristas cercanas entre sí pueden unirse y formar una camino, no obstante, esto debe respaldarse con otros experimentos y otras mediciones.

En la siguiente tabla se muestran las coordenadas geográficas de los vértices que conforman a estas aristas.

No. Arista	No. Vertice	Coordenadas	No. Vertice	Coordenadas
1	5	(-17.512870,14.740658)	10	(-17.508766,14.747767)
2	8	(-17.508395,14.730968)	19	(-17.491293,14.719656)
3	10	(-17.508766,14.747767)	11	(-17.505067,14.751808)
4	11	(-17.505067,14.751808)	12	(-17.499875,14.754555)
5	13	(-17.497565,14.728712)	14	(-17.497759,14.732262)
6	13	(-17.497565,14.728712)	16	(-17.494845,14.724453)
7	16	(-17.494845,14.724453)	19	(-17.491293,14.719656)
8	19	(-17.491293,14.719656)	24	(-17.486972,14.724319)
9	21	(-17.487934,14.750047)	23	(-17.491255,14.745721)
10	24	(-17.486972,14.724319)	28	(-17.480833,14.724458)
11	28	(-17.480833,14.724458)	29	(-17.481918,14.725431)
12	28	(-17.480833,14.724458)	38	(-17.474466,14.723921)
13	30	(-17.480514,14.713803)	33	(-17.480766,14.709816)
14	31	(-17.479050,14.762229)	42	(-17.475318,14.756450)
15	39	(-17.475090,14.711362)	41	(-17.475263,14.713093)
16	42	(-17.475318,14.756450)	50	(-17.471946,14.753731)
17	50	(-17.471946,14.753731)	51	(-17.471624,14.750367)
18	68	(-17.466153,14.750553)	83	(-17.462295,14.749303)
19	71	(-17.467852,14.737170)	88	(-17.463548,14.738111)
20	79	(-17.463679,14.695285)	85	(-17.462928,14.692848)
21	88	(-17.463548,14.738111)	93	(-17.461361,14.733214)
22	93	(-17.461361,14.733214)	97	(-17.460410,14.729414)
23	95	(-17.460571,14.741360)	111	(-17.457424,14.741267)
24	125	(-17.454238,14.705465)	131	(-17.452843,14.700483)
25	128	(-17.455101,14.693734)	139	(-17.452326,14.693656)
26	157	(-17.450096,14.703719)	159	(-17.450545,14.707033)
27	170	(-17.446510,14.694214)	188	(-17.443712,14.692505)
28	183	(-17.444736,14.706359)	192	(-17.445921,14.701803)
29	204	(-17.442743,14.715371)	244	(-17.435958,14.718992)
30	245	(-17.436633,14.739359)	255	(-17.435168,14.741912)
31	251	(-17.435862,14.756231)	264	(-17.431715,14.757625)
32	270	(-17.427909,14.724121)	278	(-17.427857,14.730956)
33	277	(-17.426597,14.756671)	280	(-17.425364,14.760166)
34	278	(-17.427857,14.730956)	285	(-17.425537,14.733195)
35	280	(-17.425364,14.760166)	288	(-17.424340,14.762116)
36	281	(-17.426557,14.736299)	284	(-17.425205,14.739142)
37	287	(-17.424548,14.744221)	292	(-17.424649,14.749654)
38	296	(-17.421033,14.767416)	299	(-17.416721,14.770730)
39	299	(-17.416721,14.770730)	304	(-17.412967,14.770074)
40	310	(-17.407699,14.770926)	315	(-17.401944,14.772847)
41	314	(-17.403287,14.768607)	315	(-17.401944,14.772847)
42	330	(-17.394190,14.775865)	341	(-17.388106,14.776314)
43	341	(-17.388106,14.776314)	351	(-17.382058,14.775739)
44	351	(-17.382058,14.775739)	358	(-17.376922,14.776504)
45	358	(-17.376922,14.776504)	366	(-17.373589,14.775756)
46	366	(-17.373589,14.775756)	377	(-17.365803,14.773671)
47	367	(-17.370399,14.771740)	377	(-17.365803,14.773671)

Tabla 5.3: Rutas permanentes halladas

5.6.- Búsqueda de los puntos origen y destino recurrentes

Al tiempo que se crean las matrices correspondientes a cada mapa de calor, para el conjunto de celdas de *Voronoi*, se crea un arreglo $\mathcal{O}$ en donde cada $\mathcal{O}_h$ con $h \in \mathbb{Z}^+$ tal que $1 \leq h \leq 490$, se contabiliza la cantidad de veces que cada vértice ha servido como punto de partida para todos los usuarios que se desplazan en una hora en particular. De manera similar, para el arreglo $\mathcal{D}$, para cada $\mathcal{D}_h$ con $h \in \mathbb{Z}^+$ tal que $1 \leq h \leq 490$ se cuenta el número de ocasiones que cada vértice fue un punto de llegada. Construir cualquiera de los dos arreglos, para cada hora procesada depende directamente de la cantidad de usuarios que transitan a través de las celdas de *Voronoi* en un lapso de tiempo de interés; por lo que a lo más tomará $O(U_n)$

Usando ambos arreglos es posible obtener los puntos geográficos de mayor actividad colocando a los respectivos elementos de cada vector en una cola de prioridad, como ya se mencionó en el marco teórico, las operaciones de construcción y extracción toman a lo más $O(logn)$, que para este caso corresponde a $O(log490)$.

Una observación importante es que para algunas horas en particular puede darse el caso en el que el vértice origen más porpular sea también el destino, por lo que cuando sucede esta situación hay que tomar el segundo elemento mayor, lo que algorítimicamente también toma $O(log490)$.

La tabla 5.4 muestra los puntos origen y destino más usados a lo largo del día, debe interpretarse que para cada hora del día hay un vértice origen, en la columna del mismo nombre, que tiene una mayor concentración de usuarios (que inician sus trayectos en ese sitio) respecto a los demás vértices a esa misma hora, asimismo para cada hora del día existe un vértice destino que tiene una mayor concentración de personas quienes finalizan sus trayectos en él. También recordemos que cada vértice en la gráfica corresponde con las coordenadas de una antena en Dakar y que a su vez se le asocia con una única celda de *Voronoi*, cuando se dice que los usuarios inician o terminan su ruta en un vértice se hace referencia a que salen o llegan a una celda de *Voronoi* específica.

Hora	Origen	Coordenadas	No. usuarios	Destino	Coordenadas	No. usuarios
0	171	(-17.44783,14.70949)	3137	161	(-17.45040,14.69030)	2714
1	171	(-17.44783,14.70949)	1491	28	(-17.48083,14.72446)	1289
2	50	(-17.47195,14.75373)	771	171	(-17.44783,14.70949)	725
3	9	(-17.50704,14.74067)	665	8	(-17.50840,14.73097)	507
4	9	(-17.50704,14.74067)	532	8	(-17.50840,14.73097)	444
5	9	(-17.50704,14.74067)	369	5	(-17.51287,14.74066)	355
6	85	(-17.46293,14.69285)	570	433	(-17.29886,14.73077)	523
7	77	(-17.46676,14.68550)	1956	85	(-17.46293,14.69285)	1714
8	85	(-17.46293,14.69285)	3427	77	(-17.46676,14.68550)	3389
9	77	(-17.46676,14.68550)	5890	85	(-17.46293,14.69285)	5889
10	77	(-17.46676,14.68550)	8396	85	(-17.46293,14.69285)	7687
11	77	(-17.46676,14.68550)	9130	85	(-17.46293,14.69285)	8839
12	77	(-17.46676,14.68550)	9746	85	(-17.46293,14.69285)	9698
13	85	(-17.46293,14.69285)	8886	188	(-17.44371,14.69251)	8375
14	188	(-17.44371,14.69251)	7560	85	(-17.46293,14.69285)	7271
15	188	(-17.44371,14.69251)	7661	171	(-17.44783,14.70949)	7337
16	188	(-17.44371,14.69251)	7522	171	(-17.44783,14.70949)	7045
17	188	(-17.44371,14.69251)	7570	171	(-17.44783,14.70949)	7217
18	171	(-17.44783,14.70949)	7704	188	(-17.44371,14.69251)	7313
19	171	(-17.44783,14.70949)	7705	188	(-17.44371,14.69251)	7012
20	171	(-17.44783,14.70949)	8840	277	(-17.42660,14.75667)	7451
21	171	(-17.44783,14.70949)	8134	367	(-17.37040,14.77174)	7364
22	171	(-17.44783,14.70949)	7404	367	(-17.37040,14.77174)	6524
23	171	(-17.44783,14.70949)	5224	161	(-17.45040,14.69030)	4722

Tabla 5.4: Puntos origen y destino más populares a lo largo del día

Visualmente los puntos de partida y llegada junto con los vértices permanentes, a lo largo de todo el día, se muestran con indicadores amarillos en la figura 5.12:

De esta manera se complementa la idea que se tenía de unir ciertas rutas en el mapa mostrado de aristas

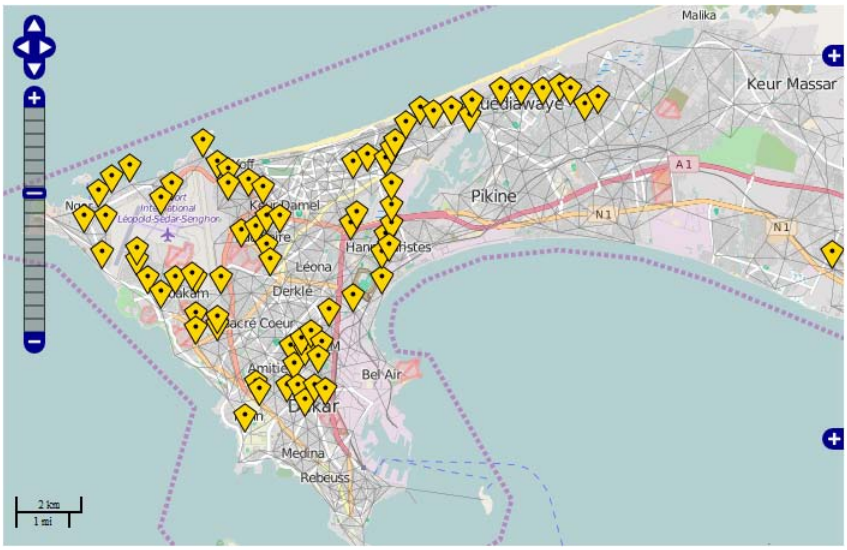


Figura 5.12: Puntos origen y destino recurrentes junto con los vértices permanentes.

permanentes ya de una manera justificada con las estadísticas obtenidas. En una sección futura se tratará el análisis que se hizo para agrupar a los puntos origen y destino recurrentes junto con los vértices permanentes.

5.7.- Búsqueda de horas pico

Una vez más, se aprovecha la utilidad de tener almacenados los movimientos que se hacen por cada hora en las matrices de adyacencia ponderadas para analizar la cantidad de flujo que Dakar presenta en diferentes intervalos del día. Recordemos que dichas matrices de adyacencia guardan los movimientos registrados en intervalos de tiempo de 1 hora habiendo procesado previamente todos los registros de la base datos que involucran a ese lapso específico.

Dado que cada elemento de las matrices $\mathcal{H}$ se encuentra en el intervalo $[0, 1]$, obtener la suma acumulada de los elementos de todos los renglones de la matriz $\mathcal{H}$; lo adecuado es tener un arreglo $\mathcal{C}$ en el cual almacenar los valores resultantes de la cantidad de movimiento para cada mapa de calor, donde cada elemento del vector se calcula como:

$$\mathcal{C}_h = \sum_{i=1}^{490} \sum_{j=1}^{490} \mathcal{H}_{i,j,h} \text{ con } h \in \mathbb{Z}^+ \cup \{0\} \text{ tal que } 0 \leq h \leq 23$$

Donde $\mathcal{H}_{i,j,h}$ debe interpretarse como el elemento en el renglón i, en la columna j de la matriz de calor para la hora h.

Lo anterior se obtiene en un tiempo constante de $O\left(490^2\right)$. Cuando se aplica este mismo proceso para todos los mapas de calor, y se almacenan los resultados obtenidos en un arreglo $\mathcal{C}$, algorítmicamente siempre toma $O\left(24 \cdot 490^2\right)$, lo cual resulta en una operación bastante costosa, sin embargo se hace una única vez y al tratarse exlusivamente de sumas el tiempo empleado resulta despreciable.

Sea además k un pivote inicializado con el valor de $\mathcal{C}_0$, para cada iteración siguiente de h, para $1 \leq h \leq 23$ se compara si $k < \mathcal{C}_h$, de ser así entonces se debe hacer la asignación $k = \mathcal{C}_h$, de esta manera aprovechando los mismos recursos temporales se obtiene la cantidad máxima de movimientos que se realizaron en una hora en específico; lo cual servirá de parámetro de comparación con el resto de las horas después de normalizar al arreglo $\mathcal{C}$, respecto al valor de movimientos máximo encontrado. Esto último toma un tiempo constante $O(24)$, matemáticamente esto es:

$$\mathcal{C} \leftarrow \tfrac{1}{k} \cdot \mathcal{C}$$

El resultado de este procesamiento sencillamente descrito se muestra en la tabla 5.5.

Hora	P. de movimientos	Hora	P. de movimientos
0	0.3593	12	0.9057
1	0.1586	13	0.8366
2	0.0727	14	0.7163
3	0.0316	15	0.8024
4	0.0218	16	0.7803
5	0.0274	17	0.7865
6	0.0452	18	0.8090
7	0.1347	19	0.8588
8	0.2324	20	1.0000
9	0.4336	21	0.9848
10	0.6511	22	0.9607
11	0.8096	23	0.6832

Tabla 5.5: Porcentaje de movilidad durante todo el dia, normalizada a 1

La información contenida en la tabla anterior puede apreciarse en la figura 5.13.

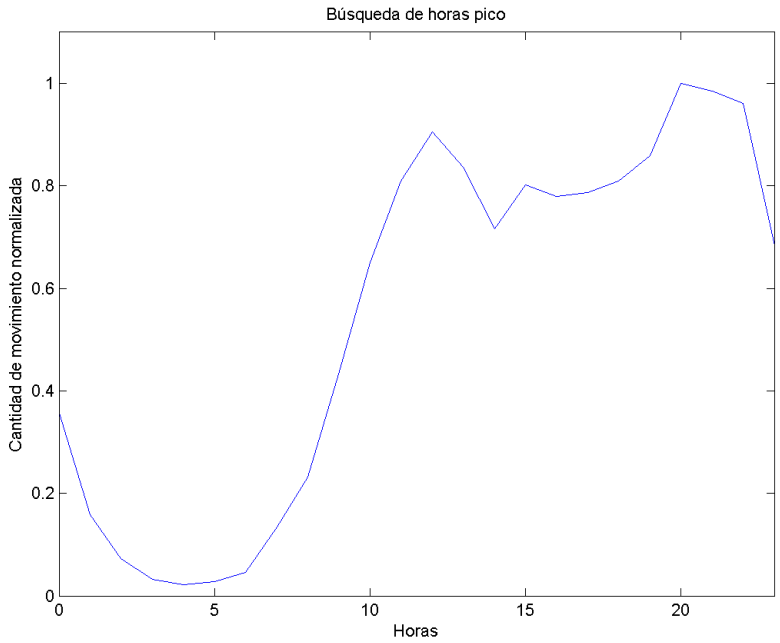


Figura 5.13: Horas pico detectadas.

El porcentaje de movilidad debe interpretarse como la normalización de la cantidad de usuarios que se desplazan entre los distintos vértices respecto al intervalo de tiempo en el cual más usuarios se movieron.

Se observa que a partir de la medianoche y hasta aproximadamente las 4:00 a.m. la movilidad decrece principalmente porque son las horas en las que la mayor parte de la población está durmiendo; y a partir de las 5:00 a.m. se incrementa casi de una manera constante hasta alcanzar un máximo local aproximadamente entre el mediodía y las 13 horas. Es notorio que durante las horas matutinas no se tiene una cantidad de movimientos importante, mientras que la mayor parte de estos ocurren durante las horas vespertinas y nocturnas, alcanzando un máximo absoluto entre las 20 y 21 horas. Lo anterior es una observación parcial puesto que no es lo mismo revisar las posiciones geográficas que se obtienen de un historial de llamadas que de un monitoreo periódico y constante de los movimientos de los usuarios.

5.8.- Agrupación de vértices permanentes y puntos geográficos populares

Antes que nada es necesario señalar que los siguientes resultados son meramente sugerencias que pueden tomarse en cuenta para la planeación de infraestructura de transporte público, cualquiera de ellas debe ser corroborada por un análisis de campo. La idea principal de cualquier proyecto de movilidad es optimizar los recursos que se tienen a la mano a la vez que se explota la mayor cantidad de información disponible.

Llamemos $\mathcal{S}$ el conjunto de vértices que están presentes de forma permanente, recordemos que los vértices corresponden con los puntos geográficos de las antenas que dan servicio a Dakar, y que a su vez éstas corresponden al centro de su respectiva celda de *Voronoi*. Además sea $\mathcal{O}_{max}$ el conjunto de vértices que resultan ser las celdas de *Voronoi* de las que parte una mayor cantidad de usuarios y sea $\mathcal{D}_{max}$ el conjunto de celdas a la cual arriva una mayor cantidad de usuarios, ambos por cada hora, cada una con dimensión $dim\left(\mathcal{O}_{max}\right) = dim\left(\mathcal{D}_{max}\right) = 24$

La finalidad de agrupar al conjunto de vértices descritos, es planificar las relaciones potenciales que hay entre sus respectivas celdas de *Voronoi* de manera tal que se satisfagan la mayor cantidad de rutas con un mínimo de relaciones entre las celdas de *Voronoi* involucradas y con ayuda de otros estudios determinar las rutas de transporte a través de las calles de Dakar. Es momento oportuno para mencionar que el propósito del presente proyecto no es encontrar este mínimo del que se habla, sino simplemente resaltar el potencial que tiene este tipo de análisis.

Una primera impresión para agrupar al conjunto de vértices mencionado es obtener las rutas más cortas para el producto cartesiano $\{\mathcal{S} \cup \mathcal{O}_{max} \cup \mathcal{D}_{max}\} \times \{\mathcal{S} \cup \mathcal{O}_{max} \cup \mathcal{D}_{max}\}$, es decir las rutas más cortas entre todos los pares de vértices. Esto evidentemente resulta inviable debido a la distribuición geográfica de todos estos puntos de interés, se terminaría reconstruyendo gran parte de la triangulación *Delaunay* restringida, lo que sugeriría hacer una planificación de infraestructura masiva.

Por otro lado si se obtienen las trayectorias más cortas de los elementos de $\{\mathcal{S} \cup \mathcal{O}_{max} \cup \mathcal{D}_{max}\}$ hacia el elemento que sea el más cercano, dentro del mismo conjunto; ocasionaría que estos vértices no se agruparan por completo.

Una manera más inteligente de agrupar a aquellos es tomar a los vértices permanentes $\mathcal{S}$ y calcular el camino más corto hacia el vértice más cercano del conjunto $\mathcal{O}_{max}$ o de $\mathcal{D}_{max}$, exclusivamente sólo a uno de ellos. La lógica detrás de esto es que los vértices permanentes representan puntos de interés, como se explicó en el estado del arte, que no desmotivan el desplazamiento geográfico, pues de antemano se sabe que siempre son usados; y la idea de unirlos a los puntos origen y destino más comunes es que son rutas altamente potenciales, de acuerdo con la estadística obtenida, para así encontrar el equilibrio entre las rutas que en algún momento del día serán muy concurridas con puntos geográficos aparentemente atractivos.

Los resultados de este experimento se muestran a continuación en la figura 5.14

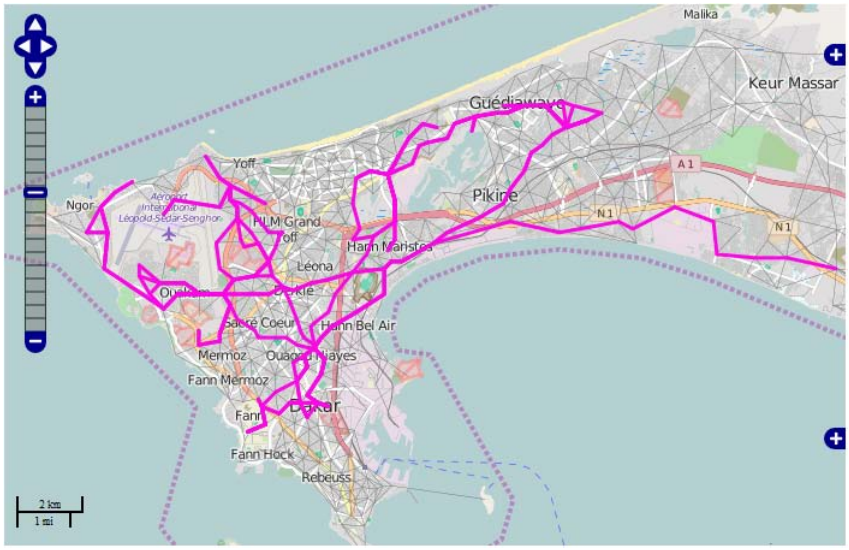


Figura 5.14: Agrupación de vértices permanentes y puntos geográficos populares.

Este experimento revela, tentativamente las relaciones que deben establecerse entre las celdas de *Voronoi* involucradas, pues planificando la infraestructura necesaria sería una forma barata de tener interconectadas a las celdas que son más activas durante todo el día y que incluso con modificaciones menores en los trazos, es posible

tenerlas comunicadas a todas usando menos recursos de equipamiento que si se hicieran adecuaciones del producto cruz entre todas las celdas involucradas.

5.9.- Porcentaje de rutas satisfechas

Como se vio anteriormente, la mayor parte de las aristas en cualquier mapa de calor corresponden a las líneas frías, aquellas por las que un número muy reducido de usuarios transitan; por lo que la mayor parte de éstas siempre quedará insatisfecha por cualquier bosquejo de infraestructura que se proponga. A continuación se muestra el porcentaje de aristas de cada color respecto del total en cada hora.

Hora	A. azules	A. verdes	A. amarillas	A. naranjas	A. rojas
0	51.63	35.05	8.78	3.35	1.19
1	49.20	33.01	12.59	4.57	0.63
2	55.18	29.63	10.49	3.98	0.73
3	72.46	22.08	3.65	0.69	1.11
4	76.85	18.33	2.86	0.75	1.21
5	51.89	31.11	11.57	3.00	2.44
6	45.48	30.51	15.34	6.64	2.03
7	48.67	34.72	10.98	3.38	2.25
8	47.17	34.58	9.90	6.72	1.62
9	44.03	32.95	14.49	6.01	2.52
10	40.80	34.84	13.65	7.78	2.94
11	39.03	35.65	13.97	8.07	3.28
12	39.16	36.22	13.93	7.38	3.30
13	38.20	36.00	15.81	6.90	3.10
14	37.93	36.90	15.67	6.88	2.62
15	36.73	37.57	15.24	7.53	2.92
16	36.25	38.24	14.99	6.75	3.77
17	37.95	40.00	14.39	5.86	1.80
18	38.86	39.10	14.45	5.81	1.78
19	38.15	37.88	16.48	4.09	3.40
20	38.54	38.26	16.36	3.25	3.59
21	43.07	38.86	12.94	2.26	2.86
22	48.52	35.98	10.81	2.58	2.11
23	51.85	33.53	10.20	3.26	1.16
Promedio	46.15	34.21	12.48	4.90	2.27

Tabla 5.6: Porcentaje de aristas de cada color respecto al total

Una forma de medir el impacto que tendría la implementación de una infraestructura que interconecte las celdas de *Voronoi* de manera tal como se ilustró en puntos anteriores es superponiendo cada uno de los mapas de calor sobre las rutas propuestas y contabilizar el número de aristas de cada tipo (frías, no tan frías, etc.) respecto al total de cada una para cada hora.

Una forma sencilla de cuantificar y porcentualizar la cantidad de aristas satisfechas por las relaciones propuestas entre celdas es teniendo una matriz auxiliar $\mathcal{I}$ de medidas 490×490, la cual indicará las aristas existentes en la agrupación hecha de vértices permanentes y puntos geográficos populares, como se mostró en la figura 5.14.

Para cada mapa de calor se hace una multiplicación de su correspondiente matriz $\mathcal{H}_h \text{con} h \in \mathbb{Z}^+ \cup 0$ tal que $0 \leq h < 24$, elemento a elemento, con $\mathcal{I}$. Es decir, para cada hora del día se efectuará el producto $\mathcal{I}_h = \mathcal{H}_h \cdot \mathcal{I}$. Dado que la matriz sólo almacena los valores 0 o 1 exclusivamente, la multiplicación descrita únicamente dejará aquellas aristas del mapa de calor que correspondan exactamente con la agrupación propuesta en la figura 5.15. Además,

para aquellos elementos que continúen presentes, la matriz $\mathcal{H}_h$ conserva sus valores ponderados despues de realizar el producto, basta con cuantificar a aquellas aristas que permanecieron y obtener su porcentaje respecto a la cantidad que había de su mismo tipo antes de la multiplicación de ambas matrices.

De manera análoga, para mostrar las rutas que no fueron satisfechas es necesario utilizar una matriz complemento $\mathcal{I}^c$, en la cual cada elemento $\mathcal{I}^c_{i,j}$ es el complemento a 2 del elemento $\mathcal{I}_{i,j}$, con $i, j \in \mathbb{Z}^+$ tal que $1 \leq i \leq 490$ y además $1 \leq j \leq 490$, y efectuar una multiplicación elemento a elemento con cada matriz $\mathcal{H}_h$, o sea $\mathcal{I}^c_h = \mathcal{H}_h \cdot \mathcal{I}^c$.

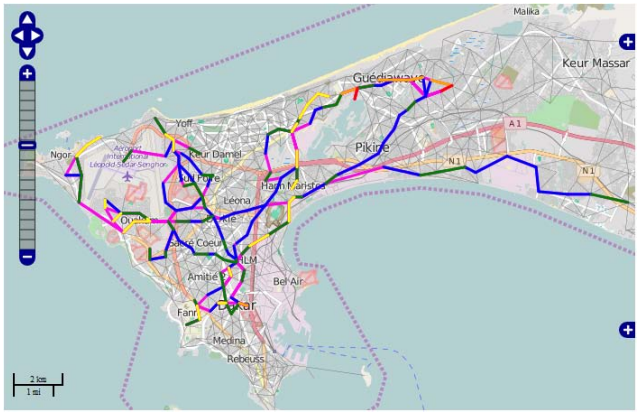
El resultado obtenido se muestra en la tabla 5.7.

Hora	**A. azules**	**A. verdes**	**A. amarillas**	**A. naranjas**	**A. rojas**
0	11.01	33.75	73.60	100.00	100.00
1	11.49	26.90	79.18	100.00	100.00
2	12.58	33.04	89.83	100.00	100.00
3	15.28	71.11	87.48	100.00	100.00
4	16.41	79.94	82.14	100.00	100.00
5	11.11	27.75	89.12	100.00	100.00
6	9.96	25.29	60.07	94.61	73.89
7	11.37	31.47	65.34	75.64	75.41
8	11.35	28.11	52.99	70.12	100.00
9	10.80	27.45	45.60	69.20	81.20
10	10.15	25.30	42.82	68.57	82.93
11	9.34	25.01	42.54	66.12	85.40
12	9.09	26.32	39.60	71.88	85.45
13	8.60	24.19	43.41	76.17	86.93
14	8.58	23.28	47.39	76.31	100.00
15	8.58	20.91	48.49	75.36	100.00
16	8.66	22.38	43.20	72.81	100.00
17	8.97	22.84	48.87	94.75	100.00
18	8.73	21.86	53.13	94.58	100.00
19	7.23	22.83	47.63	100.00	100.00
20	7.84	22.90	51.11	100.00	100.00
21	7.84	27.61	60.31	100.00	100.00
22	10.03	28.99	67.10	100.00	100.00
23	11.42	31.12	66.79	100.00	100.00
Promedio	10.27	30.43	59.49	87.75	94.63

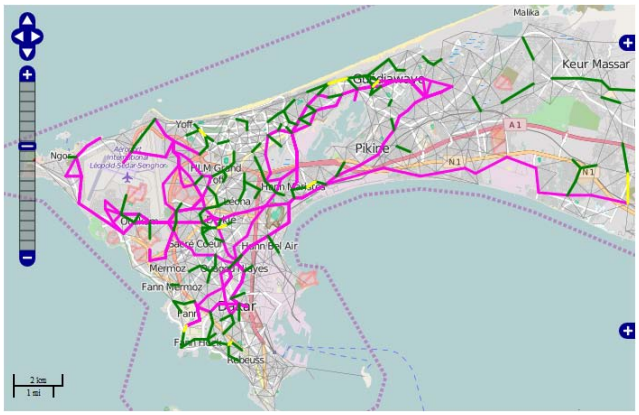
Tabla 5.7: Rutas permanentes halladas

A continuación se muestra el contraste entre la superposición de los mapas de calor, por cada hora, con la agrupación de vértices propuesta en comparación con las rutas no satisfechas. Como se mencionó recién, el principal grupo de aristas no satisfechas por las relaciones propuestas siempre serán las aristas frías, lo que falsamente da la impresión de que la agrupación no es útil. Se considera que las aristas más importantes a satisfacer son todas exceptuando a las líneas frías; por lo que en las siguientes imágenes únicamente se han eliminado éstas, para la parte de rutas insatisfechas. Se han usado los mismos códigos de color especificados en la tabla 5.2 (página 62) y además se han agregado aristas en color rosa que corresponden con la agrupación de la figura 5.14 de la sección anterior.

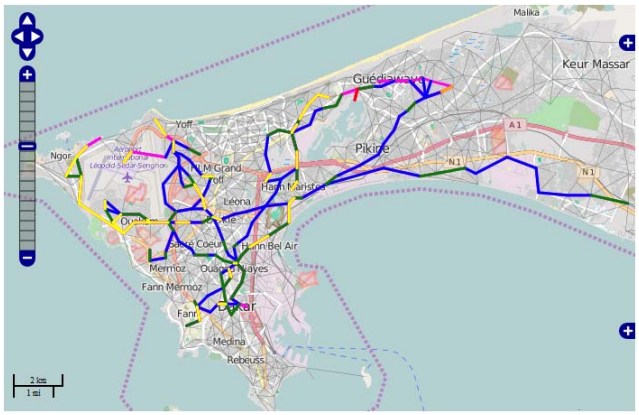
Las siguientes gráficas deben interpretarse del siguiente modo: el conjunto de aristas rosas es una especie de transparencia o máscara tal que si se coloca como base y se le agregan encima las aristas de los mapas de calor mostrados anteriormente, y sólo se dejan las que se superponen con la máscara, hace visibles los movimientos que ocurren sobre ella., esta situación corresponde a las gráficas de las columnas izquierdas. Por otro lado si a cada mapa de calor se le suprimen las aristas frías (las líneas azules que indican poca movilidad respecto al resto) y esto se deja como una transparencia y ahora a ésta se le pone encima la capa máscara deja visibles aquellas aristas que tienen movimientos importantes que la agrupación de vértices (líneas rosas) no satisfizo, esto corresponde con las gráficas de las columnas derechas.



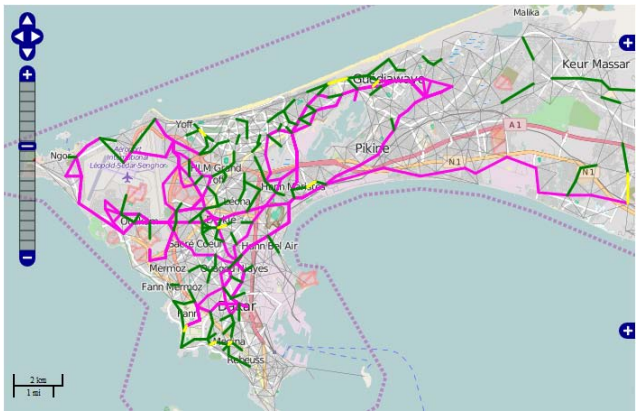
(a) Rutas satisfechas entre las 0:00 hrs y las 1:00 hrs



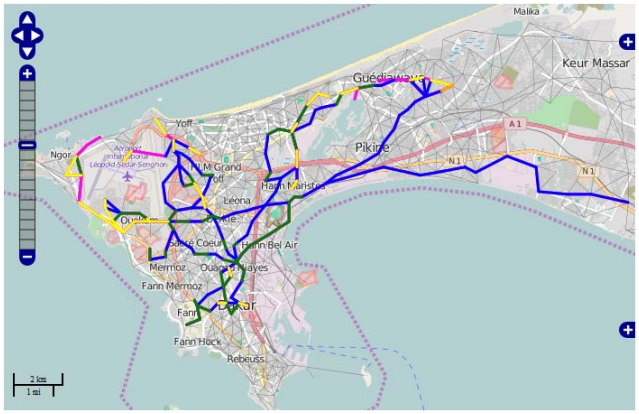
(b) Rutas no satisfechas entre las 0:00 hrs y las 1:00 hrs



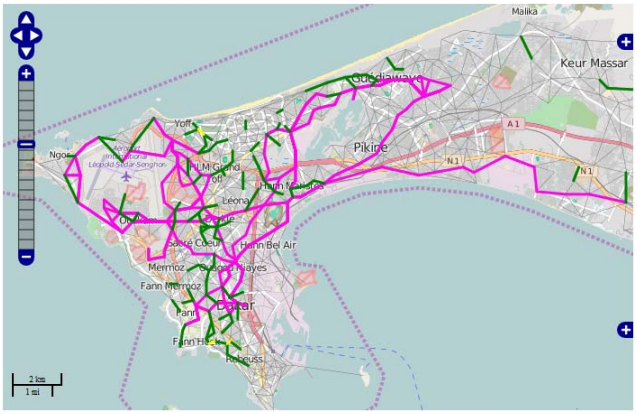
(c) Rutas satisfechas entre las 1:00 hrs y las 2:00 hrs



(d) Rutas no satisfechas entre las 1:00 hrs y las 2:00 hrs

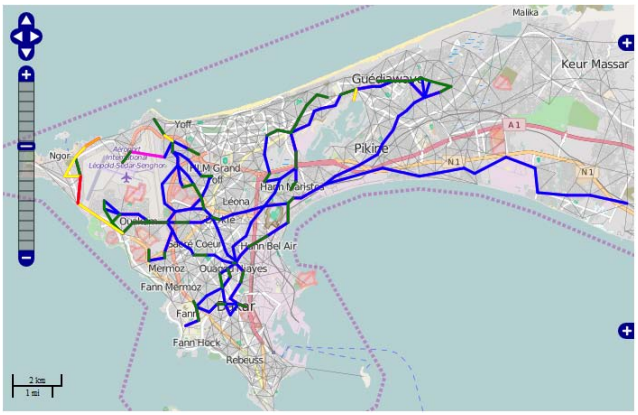


(e) Rutas satisfechas entre las 2:00 hrs y las 3:00 hrs

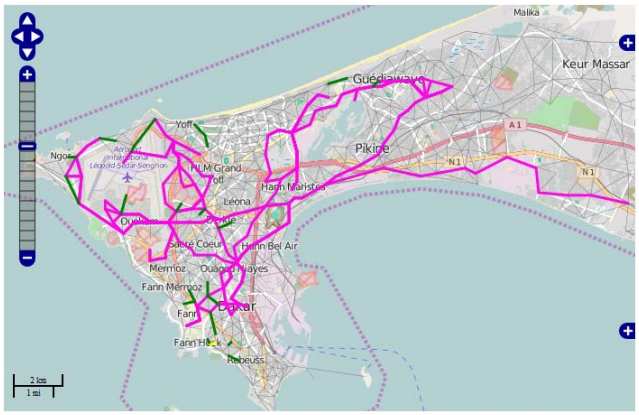


(f) Rutas no satisfechas entre las 2:00 hrs y las 3:00 hrs

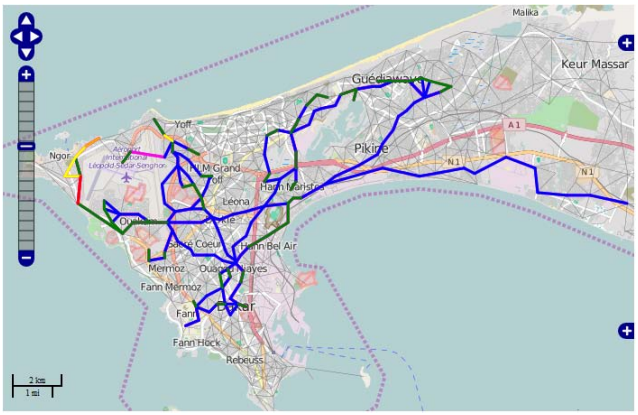
Figura 5.15: Comparación entre las rutas satisfechas y las que no entre las 0:00 hrs y las 3:00 hrs



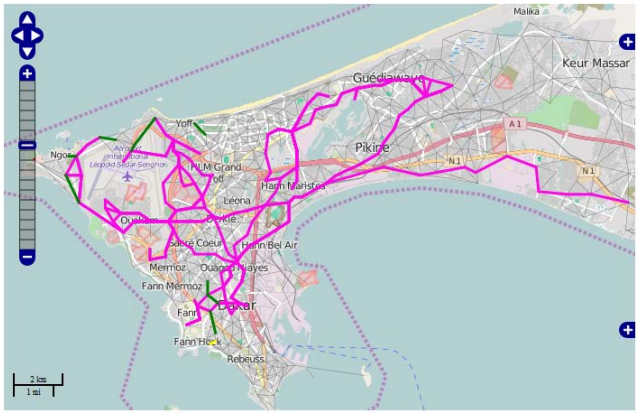
(a) Rutas satisfechas entre las 3:00 hrs y las 4:00 hrs



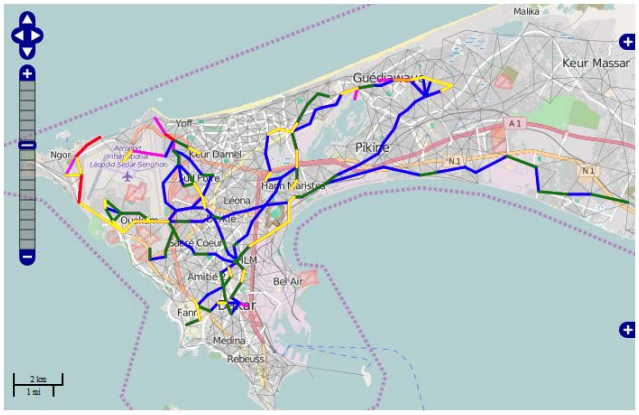
(b) Rutas no satisfechas entre las 3:00 hrs y las 4:00 hrs



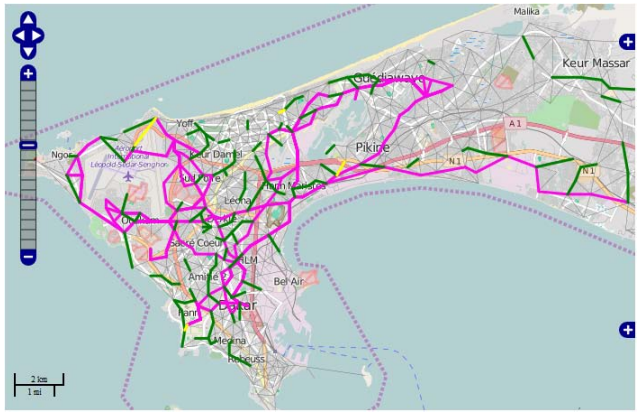
(c) Rutas satisfechas entre las 4:00 hrs y las 5:00 hrs



(d) Rutas no satisfechas entre las 4:00 hrs y las 5:00 hrs

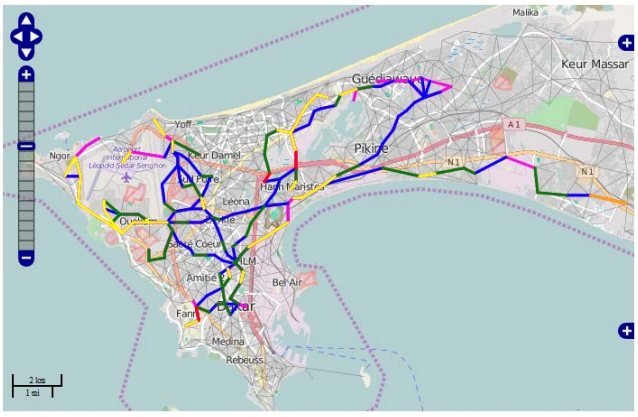


(e) Rutas satisfechas entre las 5:00 hrs y las 6:00 hrs

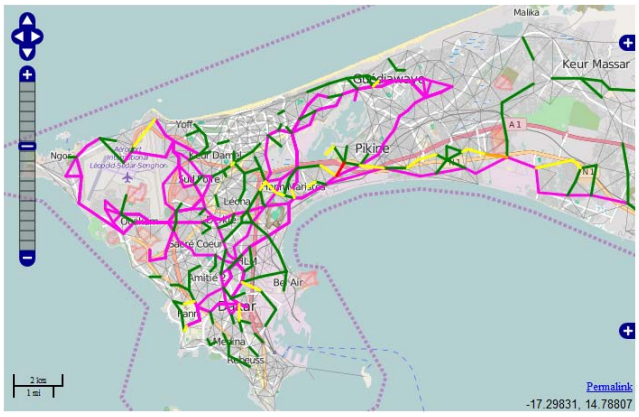


(f) Rutas no satisfechas entre las 5:00 hrs y las 6:00 hrs

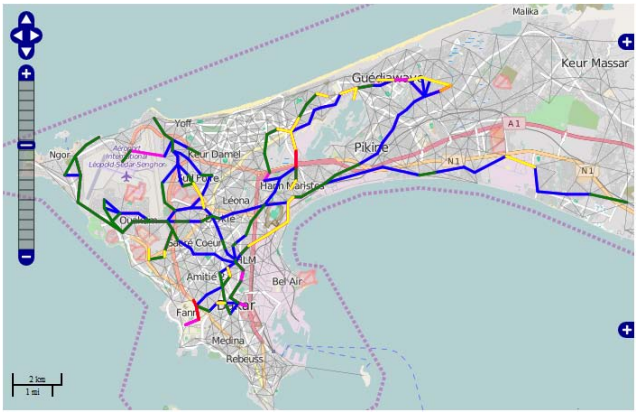
Figura 5.16: Comparación entre las rutas satisfechas y las que no entre las 3:00 hrs y las 6:00 hrs



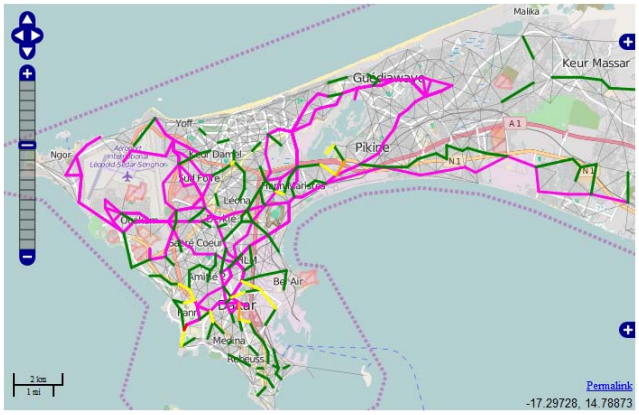
(a) Rutas satisfechas entre las 6:00 hrs y las 7:00 hrs



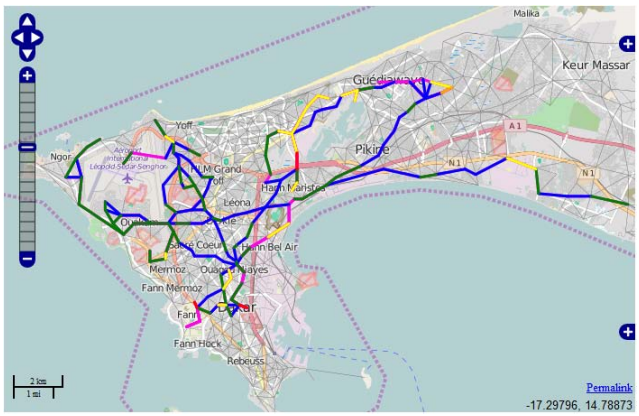
(b) Rutas no satisfechas entre las 6:00 hrs y las 7:00 hrs



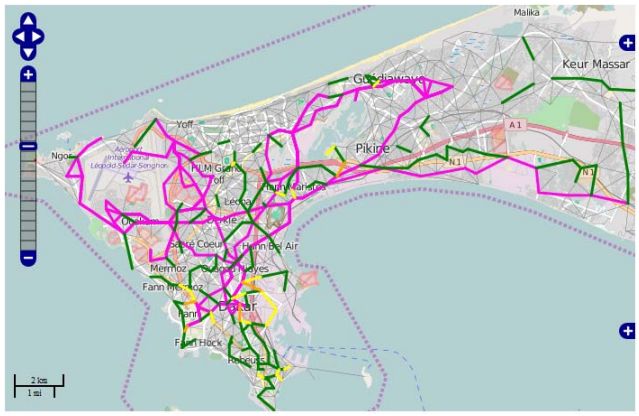
(c) Rutas satisfechas entre las 7:00 hrs y las 8:00 hrs



(d) Rutas no satisfechas entre las 7:00 hrs y las 8:00 hrs

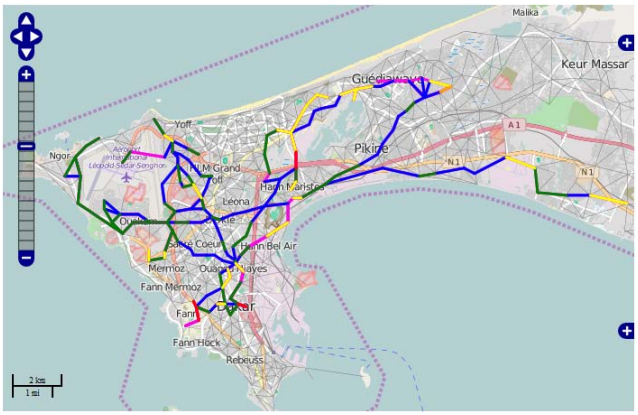


(e) Rutas satisfechas entre las 8:00 hrs y las 9:00 hrs

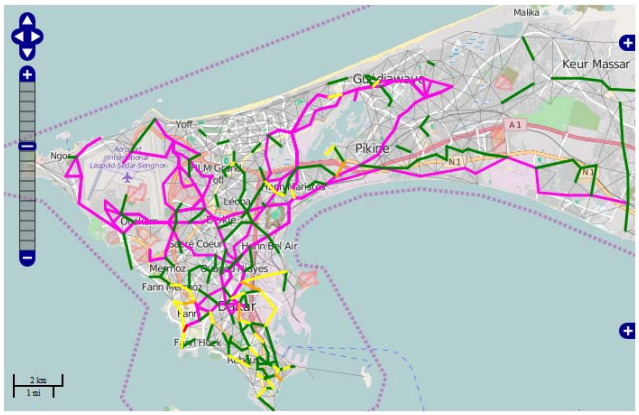


(f) Rutas no satisfechas entre las 8:00 hrs y las 9:00 hrs

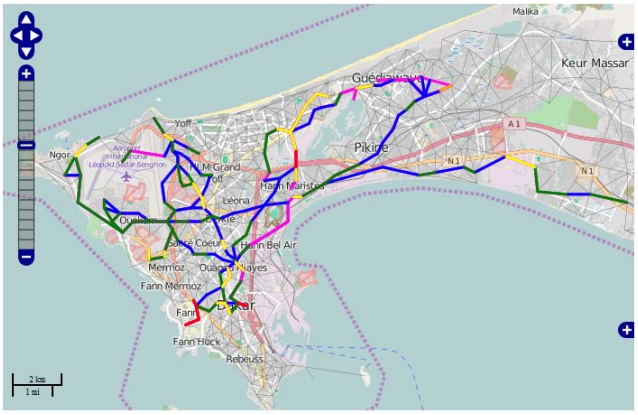
Figura 5.17: Comparación entre las rutas satisfechas y las que no entre las 6:00 hrs y las 9:00 hrs



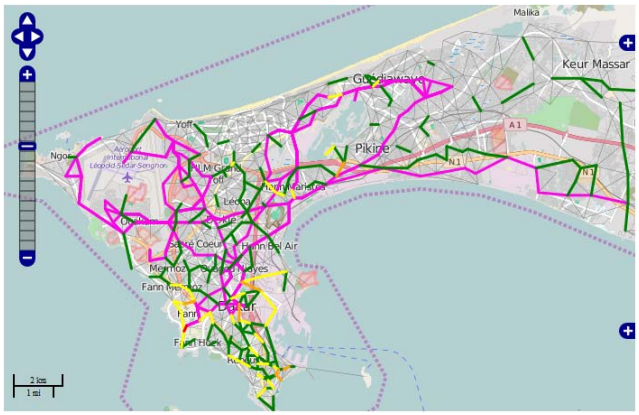
(a) Rutas satisfechas entre las 9:00 hrs y las 10:00 hrs



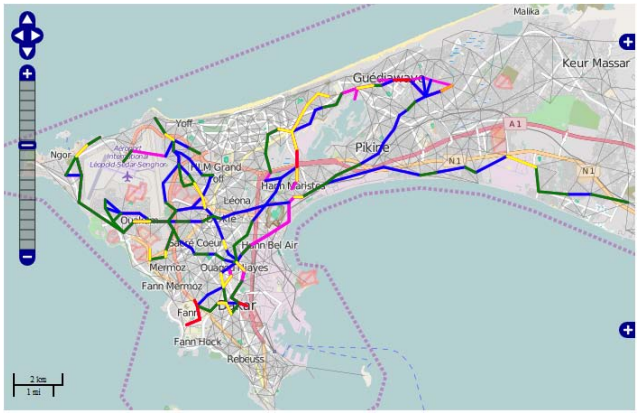
(b) Rutas no satisfechas entre las 9:00 hrs y las 10:00 hrs



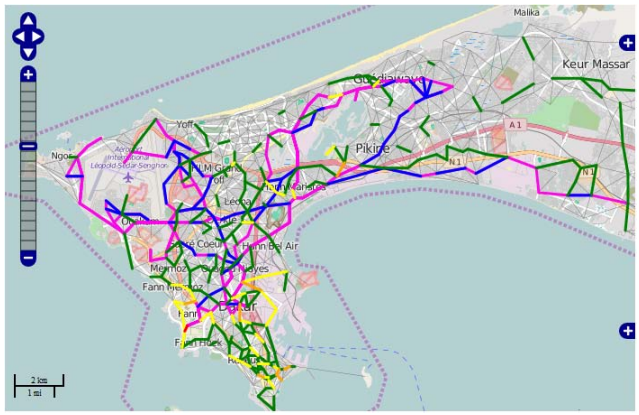
(c) Rutas satisfechas entre las 10:00 hrs y las 11:00 hrs



(d) Rutas no satisfechas entre las 10:00 hrs y las 11:00 hrs

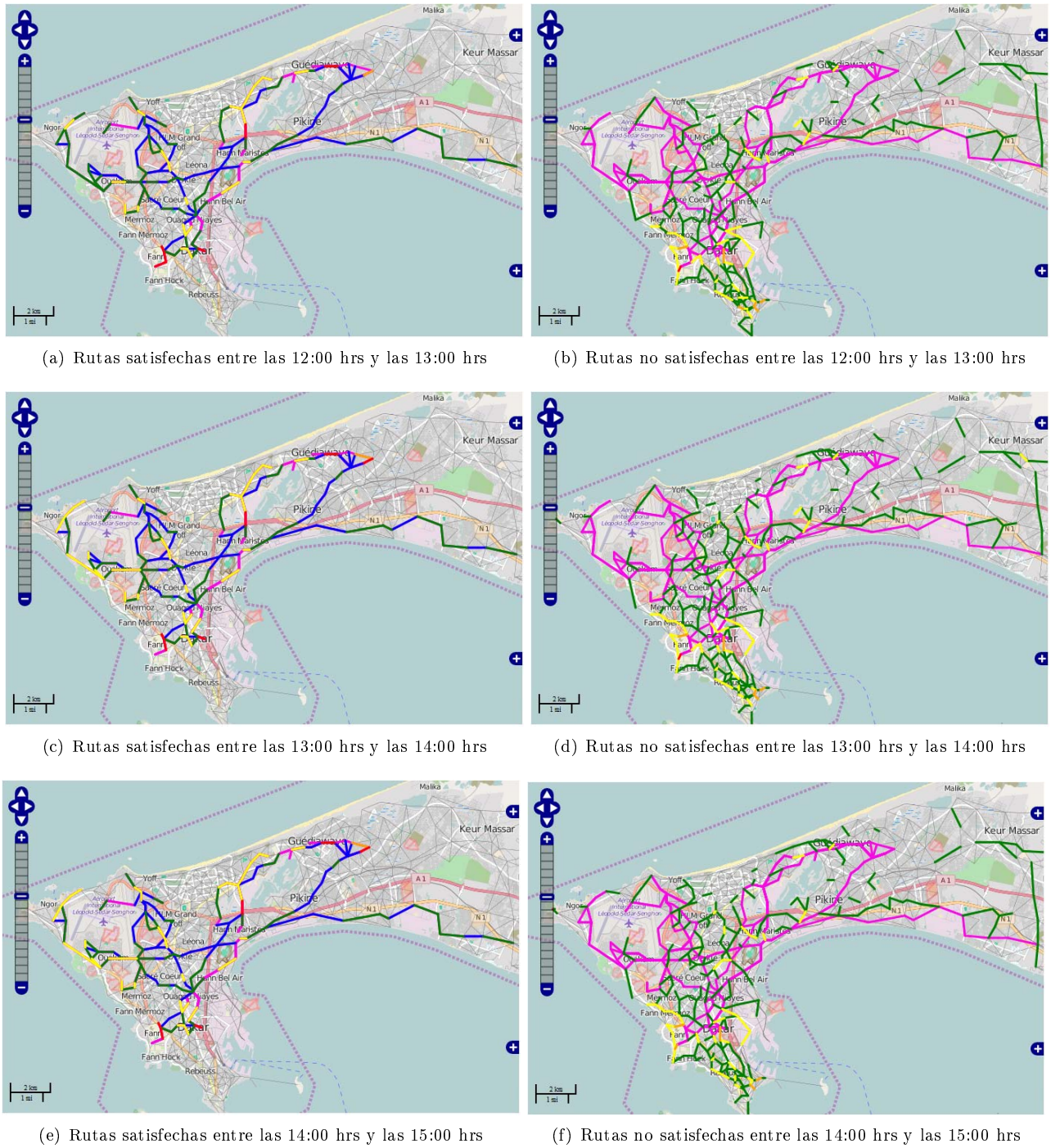


(e) Rutas satisfechas entre las 11:00 hrs y las 12:00 hrs



(f) Rutas no satisfechas entre las 11:00 hrs y las 12:00 hrs

Figura 5.18: Comparación entre las rutas satisfechas y las que no entre las 9:00 hrs y las 12:00 hrs



(a) Rutas satisfechas entre las 12:00 hrs y las 13:00 hrs

(b) Rutas no satisfechas entre las 12:00 hrs y las 13:00 hrs

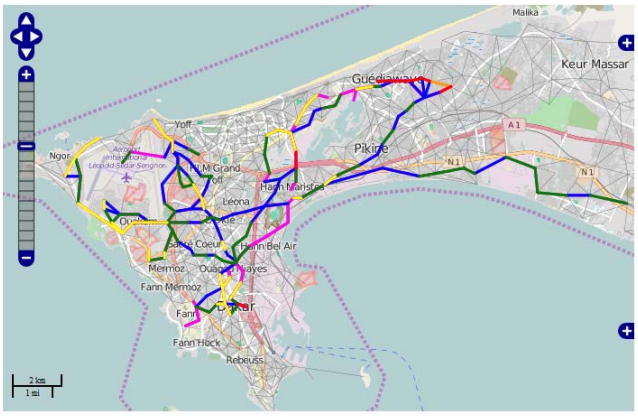
(c) Rutas satisfechas entre las 13:00 hrs y las 14:00 hrs

(d) Rutas no satisfechas entre las 13:00 hrs y las 14:00 hrs

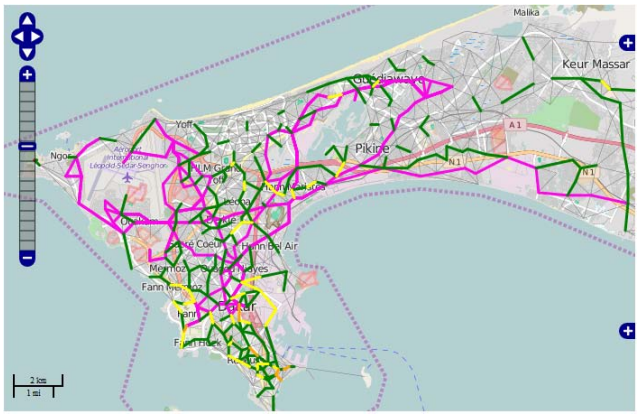
(e) Rutas satisfechas entre las 14:00 hrs y las 15:00 hrs

(f) Rutas no satisfechas entre las 14:00 hrs y las 15:00 hrs

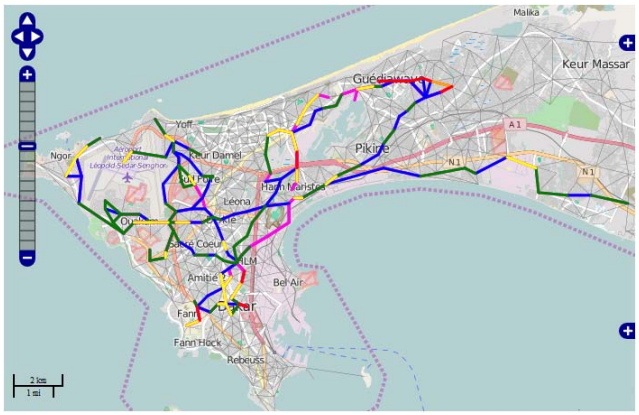
Figura 5.19: Comparación entre las rutas satisfechas y las que no entre las 12:00 hrs y las 15:00 hrs



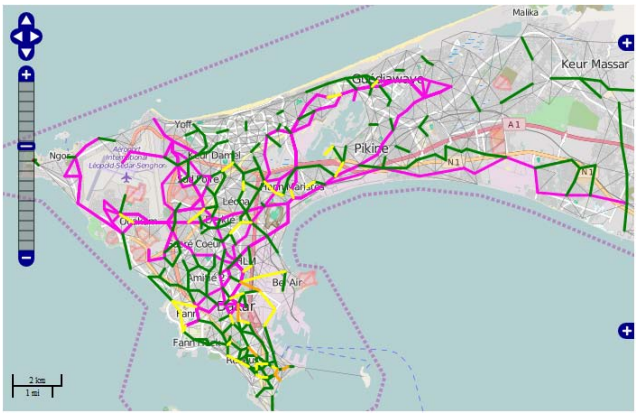
(a) Rutas satisfechas entre las 15:00 hrs y las 16:00 hrs



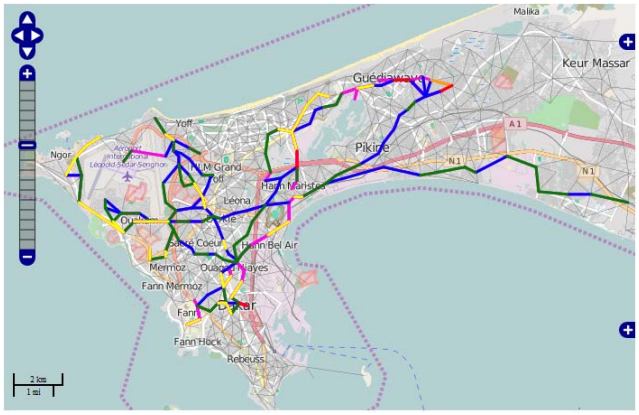
(b) Rutas no satisfechas entre las 15:00 hrs y las 16:00 hrs



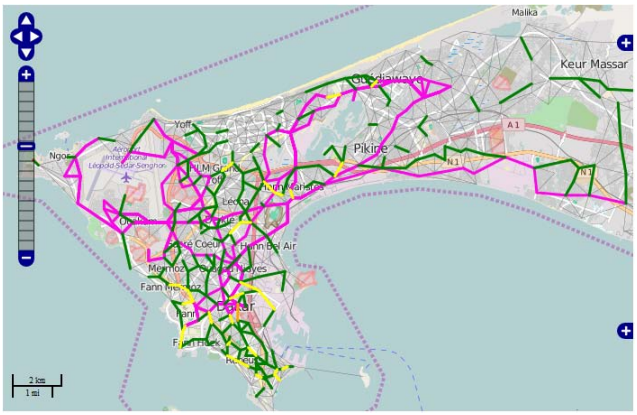
(c) Rutas satisfechas entre las 16:00 hrs y las 17:00 hrs



(d) Rutas no satisfechas entre las 16:00 hrs y las 17:00 hrs

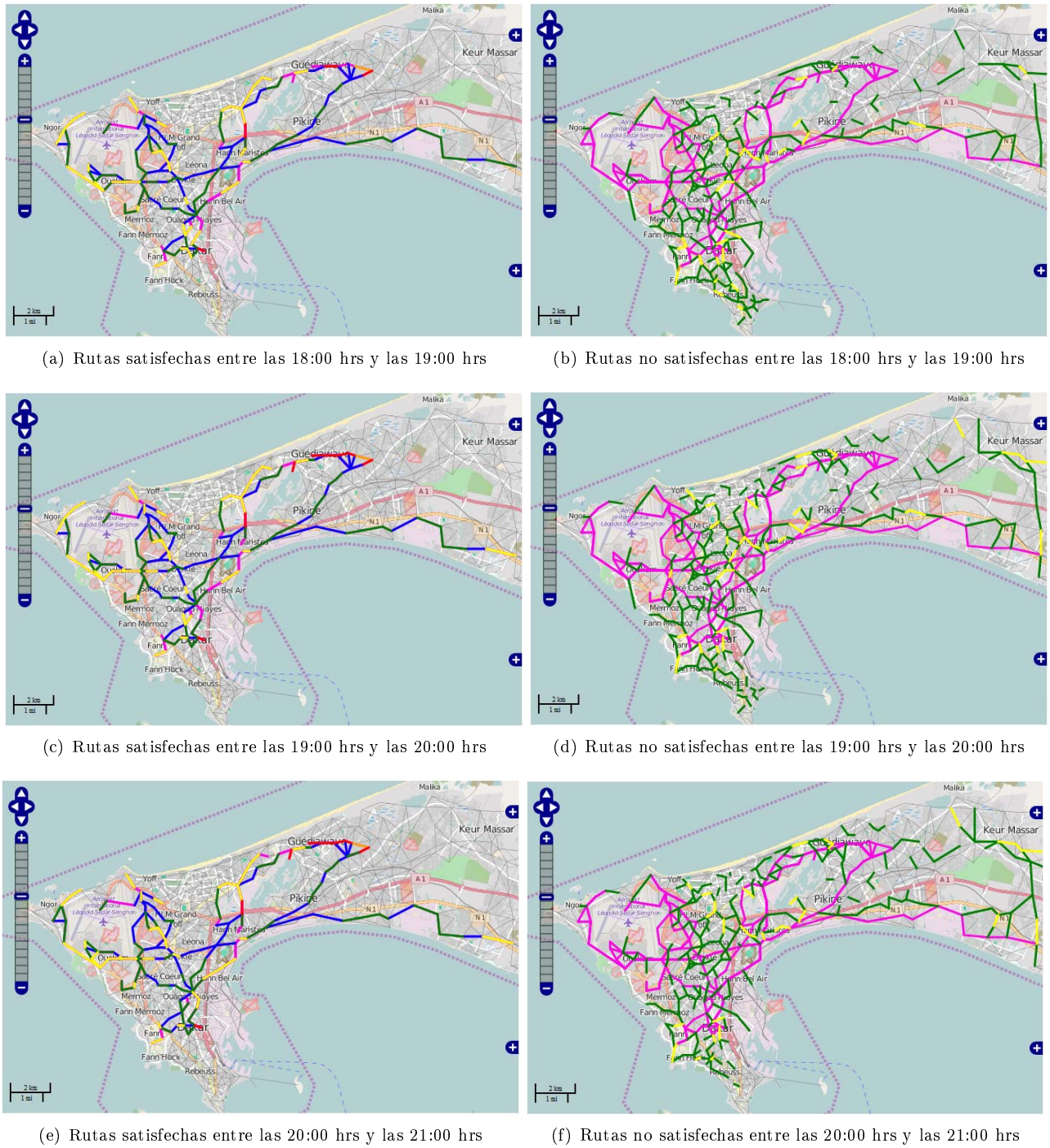


(e) Rutas satisfechas entre las 17:00 hrs y las 18:00 hrs



(f) Rutas no satisfechas entre las 17:00 hrs y las 18:00 hrs

Figura 5.20: Comparación entre las rutas satisfechas y las que no entre las 15:00 hrs y las 18:00 hrs



(a) Rutas satisfechas entre las 18:00 hrs y las 19:00 hrs

(b) Rutas no satisfechas entre las 18:00 hrs y las 19:00 hrs

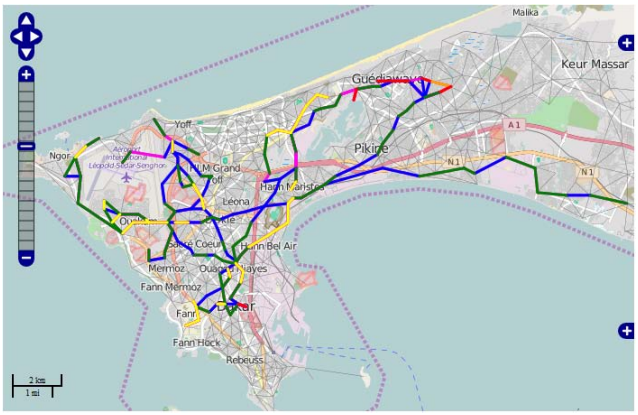
(c) Rutas satisfechas entre las 19:00 hrs y las 20:00 hrs

(d) Rutas no satisfechas entre las 19:00 hrs y las 20:00 hrs

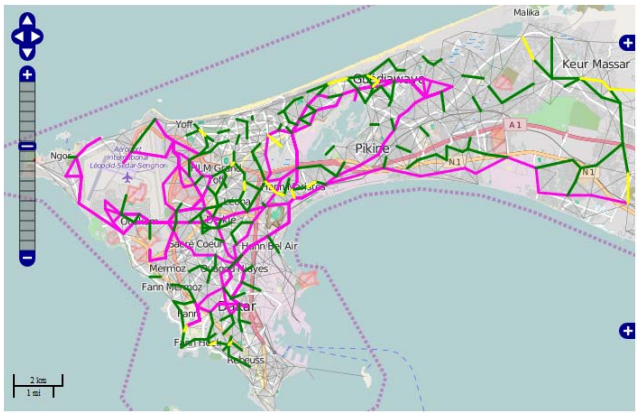
(e) Rutas satisfechas entre las 20:00 hrs y las 21:00 hrs

(f) Rutas no satisfechas entre las 20:00 hrs y las 21:00 hrs

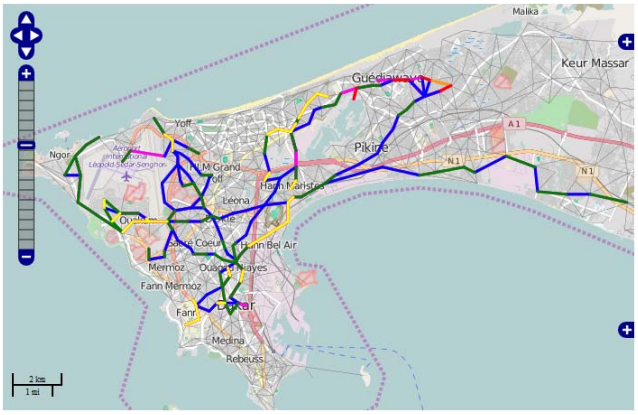
Figura 5.21: Comparación entre las rutas satisfechas y las que no entre las 18:00 hrs y las 21:00 hrs



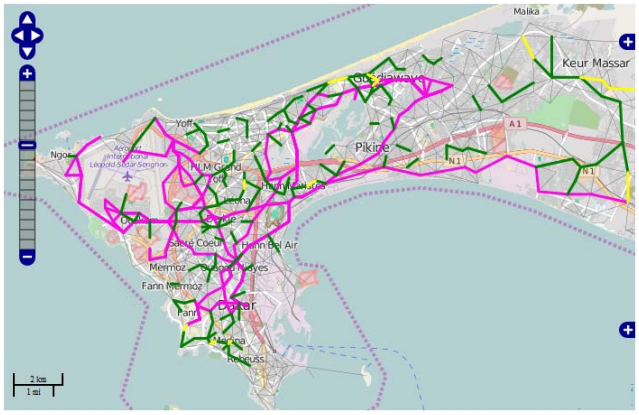
(a) Rutas satisfechas entre las 21:00 hrs y las 22:00 hrs



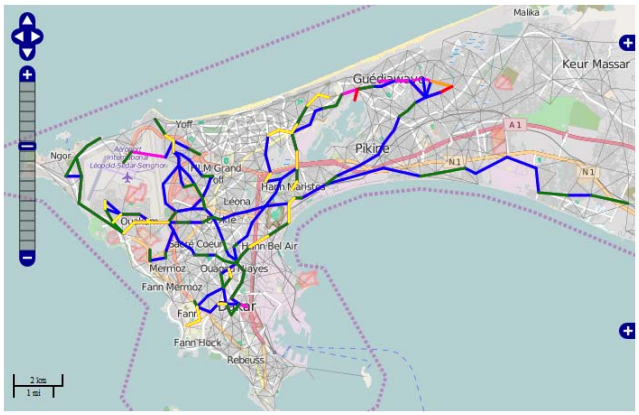
(b) Rutas no satisfechas entre las 21:00 hrs y las 22:00 hrs



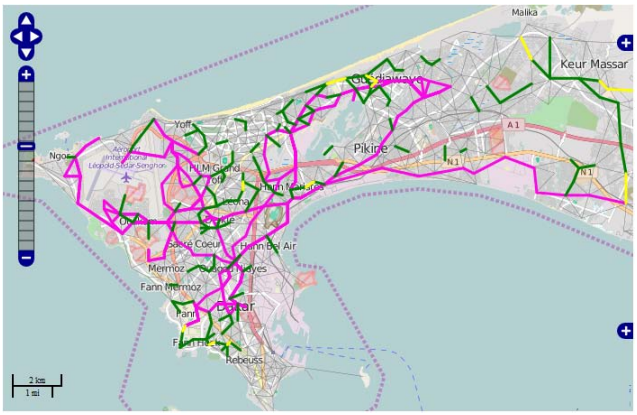
(c) Rutas satisfechas entre las 22:00 hrs y las 23:00 hrs



(d) Rutas no satisfechas entre las 22:00 hrs y las 23:00 hrs



(e) Rutas satisfechas entre las 23:00 hrs y las 24:00 hrs



(f) Rutas no satisfechas entre las 23:00 hrs y las 24:00 hrs

Figura 5.22: Comparación entre las rutas satisfechas y las que no entre las 21:00 hrs y las 23:00 hrs

Capítulo 6

6.1.- Conclusiones

El algoritmo *ACO* propuesto fue diseñado para encontrar la ruta más corta entre dos vértices cualesquiera, y que este mismo diseño sirviera para hallar rápidamente la distancia más corta entre todos los pares de vértices. Se esperaba que compitiera con algoritmos voraces como lo es *Dijkstra* cuya implementación menos óptima es de $O(n^3)$, sin embargo los resultados muestran que la ruta más corta obtenida por la colonia de hormigas tiende a acercarse asintóticamente hacia soluciones que son en promedio un 10% más largos que las rutas más cortas encontrada por el algoritmo de *Dijkstra*. La calidad de estas soluciones vienen dadas por el uso de una cantidad mayor de agentes lo cual conlleva directamente en una cantidad mayor de iteraciones y evidentemente de tiempo.

Además de lo anterior, el algoritmo *ACO* propuesto no es estable debido a los procesos probabilísticos de los que hace uso, por lo que la ruta más corta encontrada podría no ser la misma si se vuelve a ejecutar todo el proceso de búsqueda, mientras que los algoritmos tradicionales siempre llegan al mismo resultado. El algorimo de *Dijkstra* para enconrtar las rutas más cortas entre todos los pares de nodos tiene una complejidad algorítmica de $O(n^3)$, para obtener una ruta tan corta como la que estos algortimos encuentran, habría que ejecutar más de una vez al algoritmo *ACO* que se propuso, lo cual también repercutiría en una complejidad computacional más alta.

A pesar de que se ha optimizado lo más posible el algoritmo *ACO* éste algoritmo no podría ser utilizado como un simulador de multitudes, debido al aumento en el costo computacional que esto acarrearía, además de la saturación de los niveles de feromona que ocurriría en muchas de las aristas de la gráfica. Cabe mencionar que el algoritmo *ACO* que se propuso puede ser útil para conocer la diferentes rutas cortas potenciales para ir de un vértice a otro, esto es algo que no se tiene en el conjunto de soluciones que hallan los algoritmos voraces y que puede ser útil para los estudios de movilidad.

Para los análisis de movilidad es más factible utilizar algoritmos voraces tradicionales para encontrar las rutas más transitadas, pues como ya se expuso, siempre convergen al mismo conjunto de soluciones, y dejar a los algoritmos bio-inspirados la oportunidad de proponer rutas potenciales por ejemplo para el caso de contingencias como incendios u otros, de esta manera se espera que las rutas de transporte circulen por los trayectos más cortos, los cuales por la experiencia sabemos que se saturan más, y dejando para los equipos de emergencia las rutas subóptimas que incluso podrían hacerles arribar más de prisa debido a la falta de saturación en ellas.

Sin embargo, la ventaja del algoritmo *ACO* propuesto radica en que puede ser usado en otro tipo de contextos como las telecomunicaciones. En las redes es importante entregar los paquetes de la forma más eficiente posible, se cree que esto siempre es a través de los trayectos más cortos calculados por los algoritmos de ruteo, sin embargo el camino más corto no siempre es el menos congestionado o el más fiable; para estas circunstancias el algoritmo de optimización por colonia de hormigas que se desarrollo puede brindar más de una alternativa para llegar al mismo

destino, lo que parecía su desventaja se convierte ahora en una mejora frente a algoritmos como *Dijkstra*.

Se puede decir de manera general, que la forma en la que se procedió para el filtrado de datos para la obtención de los puntos origen, destino, y permanentes fue el adecuado puesto que enfatizaron puntos geográficos que siempre son muy visitados como es el caso del aeropuerto, los sitios turísticos que no están apartados de las zonas más urbanizadas, entre otros. Esta situación se esperaba que ocurriera pues nuestra experiencia cotidiana nos lleva a creer que sucede en todas partes. Sin embargo algo que también se esperaba que pasará, y no ocurrió, fue la distribución de las horas pico, estás no aparecieron como estamos acostumbrados a ver, es decir, una hora pico en la mañana y también por la tarde, sino que éstas sólo se concentraron en la tarde y por la noche lo que lleva a suponer que los problemas de movilidad de Dakar no son todos impulsados por el ingreso al trabajo o la escuela o que incluso por la falta de rutas de transporte y transporte en sí mismo, los horarios de entrada son muy diversos. Esta conclusión es preliminar debido a las características mismas de la base de datos proporcionada, pues como se mencionó corresponde a historiales de llamadas, es decir actividad registrada en las antenas que no necesariamente corresponde en su totalidad con movilidad de los usuarios, por lo que esto debería corroborarse con un estudio de campo.

No se puede concluir demasiado sobre la forma en la que se agrupó a los puntos de interés, de origen y destino debido a que la efectividad con la cual se unieron no depende sólo de las estadísticas obtenidas en el presente trabajo sino que también requieren de un análisis de campo o al menos que alguien nativo de Dakar les verifique.

6.2.- Trabajo futuro

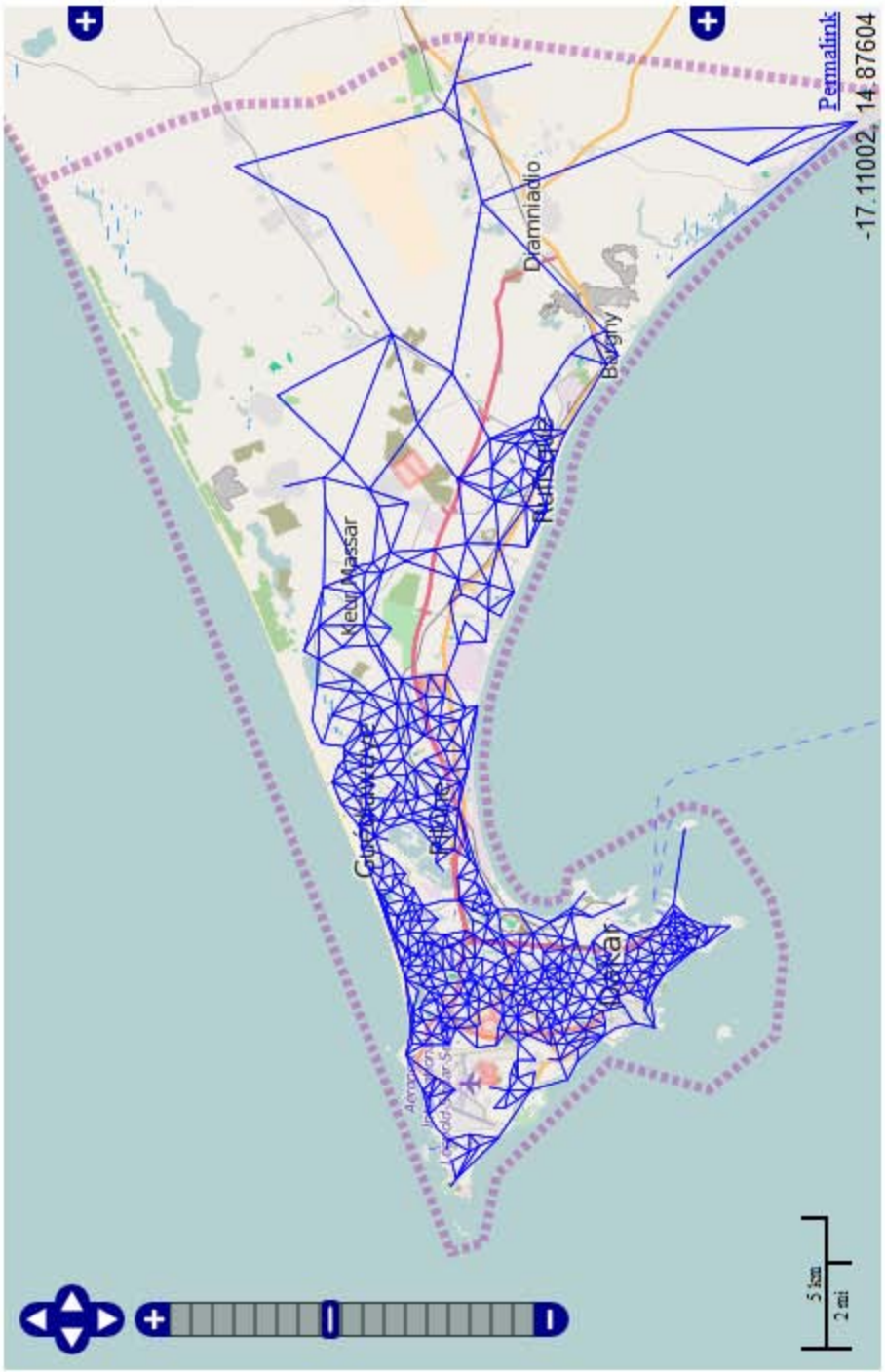
Una forma en la que podría competir el algoritmo *ACO* propuesto, con los algoritmos voraces tradicionales, para encontrar las rutas más cortas entre todos los pares de vértices de una gráfica, sería integrar una fase con un algoritmo evolutivo para encontrar la distancia máxima a la que es perceptible la feromona, el nivel máximo de feromonas y la velocidad a la que deben desplazarse todas las hormigas, al igual que la optimización por colonia de hormigas encontrar la mezcla adecuada de las tres variables mencionadas, es un problema de optimización combinatoria no trivial.

De igual manera sería útil proporcionar otra forma de orientación para las hormigas, o incluso otra forma espacial de distribuir los niveles de feromonas, para que su desempeño en gráficas con cavidades no convexas mejore.

Por otro lado también se sugiere mejorar el algoritmo propuesto para que funcione con gráficos que no sean geométricamente exactos o que las aristas de sus arcos representen una característica diferente a la distancia que separa a los vértices que unen, tal como capacidad, cantidad de tráfico entre otras; y que a su vez sea computacionalmente eficiente.

Se sugiere además para un análisis de movilidad más profundo que se diseñe e implemente una aplicación para teléfonos móviles que periódicamente guarde la posición geográfica del usuario con una estampa de tiempo y de igual manera que se envie posteriormente a una base de datos remota. Lo anterior proporcionaría estadísticas de una calidad mayor a las presentadas en el trabajo, puesto que los datos almacenados no serían los historiales de llamadas esporádicas por parte de los usuarios sino los trazos auténticos de la movilidad de una gran cantidad de personas y que además harían posible inferir el tiempo en el que permanecen en un atasco vehicular, de haberlo y la velocidad promedio a la cual se desplazan, entre otros análisis más. Las etapas propuestas en el capítulo 3 (Congruencia metodológica) pueden ser usados debido a que el constante muestreo de las posiciones de los usuarios llevará también a una alta repetición de datos que no representan movilidad, para el caso en que se busquen las rutas más usadas. Considerando el muestreo masivo constante las etapas y algoritmos utilizados en cada una de ellas deben adaptarse para funcionar en tiempo real y almacenar sólo la información necesaria una vez que se hagan los análisis buscados para obviamente evitar la saturación prematura de los equipos de almacenamiento.

Apéndice


Permalink
-17.11002, 14.87604
Dramniadio
Bargny
Rufisque
Keur Massar
Guédiawaye
Pikine
Dakar
5 km
2 mi

Vertice	Longitud	Latitud	Vertices conectados
1	-17.525142	14.746832	2, 3
2	-17.524360	14.747434	1
3	-17.522576	14.745198	1, 4, 5, 7, 8
4	-17.516398	14.746730	3, 5
5	-17.512870	14.740658	3, 4, 7, 9, 10
6	-17.512103	14.748411	10
7	-17.510958	14.737403	3, 5, 8, 9
8	-17.508395	14.730968	3, 7, 9, 19
9	-17.507036	14.740671	5, 7, 8, 10
10	-17.508766	14.747767	5, 6, 9, 11
11	-17.505067	14.751808	10, 12
12	-17.499875	14.754555	11, 15, 20, 23
13	-17.497565	14.728712	14, 16, 18
14	-17.497759	14.732262	13, 18
15	-17.493970	14.754447	12, 20, 21, 22, 23
16	-17.494845	14.724453	13, 18, 19
17	-17.490356	14.733770	18
18	-17.491645	14.727389	13, 14, 16, 17, 19, 24, 26
19	-17.491293	14.719656	8, 16, 18, 24, 25, 27
20	-17.491396	14.756656	12, 15, 22, 31
21	-17.487934	14.750047	15, 22, 23, 44
22	-17.486997	14.752257	15, 20, 21, 31
23	-17.491255	14.745721	12, 15, 21
24	-17.486972	14.724319	18, 19, 26, 27, 28, 29
25	-17.487569	14.707238	19, 30, 32, 33
26	-17.485364	14.726616	18, 24, 29
27	-17.484902	14.718472	19, 24
28	-17.480833	14.724458	24, 29, 34, 38
29	-17.481918	14.725431	24, 26, 28
30	-17.480514	14.713803	25, 33
31	-17.479050	14.762229	20, 22, 35, 42, 53
32	-17.481245	14.703930	25, 37, 47
33	-17.480766	14.709816	25, 30, 36, 37, 39
34	-17.480860	14.718215	28, 38
35	-17.474797	14.760298	31, 42, 53
36	-17.474843	14.707023	33, 37, 39, 40, 46, 52
37	-17.476595	14.703825	32, 33, 36, 40, 47
38	-17.474466	14.723921	28, 34, 48, 49, 57
39	-17.475090	14.711362	33, 36, 41, 46, 59
40	-17.472775	14.700727	36, 37, 47, 52
41	-17.475263	14.713093	39, 48, 59, 60
42	-17.475318	14.756450	31, 35, 44, 50, 53
43	-17.476416	14.738208	45, 55, 56
44	-17.475218	14.747570	21, 42, 50, 51, 54, 56
45	-17.475376	14.731479	43, 49, 58
46	-17.472078	14.708471	36, 39, 52, 59, 64, 73
47	-17.472331	14.695087	32, 37, 40, 52, 62, 65
48	-17.471791	14.720893	38, 41, 57, 60, 72
49	-17.470591	14.728215	38, 45, 57, 58, 72, 75
50	-17.471946	14.753731	42, 44, 51, 53, 68, 69

Tabla 8.1: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 1).

Vertice	Longitud	Latitud	Vertices conectados
51	-17.471624	14.750367	44, 50, 54, 63, 68
52	-17.470132	14.701305	36, 40, 46, 47, 65, 70, 73
53	-17.470171	14.760457	31, 35, 42, 50, 67, 69, 86, 145
54	-17.469863	14.748479	44, 51, 56, 63
55	-17.468955	14.738509	43, 56, 71, 78
56	-17.469568	14.745296	43, 44, 54, 55, 63, 78, 80
57	-17.470860	14.724240	38, 48, 49, 72
58	-17.469460	14.730911	45, 49, 75
59	-17.468904	14.713397	39, 41, 46, 60, 64, 74, 76
60	-17.470313	14.716990	41, 48, 59, 72, 76, 87
61	-17.469347	14.677531	66, 82, 109
62	-17.469349	14.690587	47, 66
63	-17.468546	14.748886	51, 54, 56, 68, 80
64	-17.467651	14.709607	46, 59, 73, 74, 81, 84
65	-17.468378	14.694284	47, 52, 70, 79, 85
66	-17.467849	14.683629	61, 62, 77, 82
67	-17.467381	14.759610	53, 69, 86
68	-17.466153	14.750553	50, 51, 63, 69, 80, 83, 90, 98
69	-17.466940	14.756418	50, 53, 67, 68, 86, 90
70	-17.465813	14.699413	52, 65, 73, 79, 91
71	-17.467852	14.737170	55, 88
72	-17.465708	14.724483	48, 49, 57, 60, 75, 87, 89, 92, 100
73	-17.465968	14.705823	46, 52, 64, 70, 84, 91, 105
74	-17.466756	14.711590	59, 64, 76, 81
75	-17.465261	14.729225	49, 58, 72, 92, 93, 97
76	-17.465463	14.714412	59, 60, 74, 81, 87
77	-17.466756	14.685505	66, 82, 94
78	-17.466465	14.740890	55, 56, 80, 88, 95
79	-17.463679	14.695285	65, 70, 85, 91
80	-17.466239	14.746694	56, 63, 68, 78, 83, 95
81	-17.463652	14.712400	64, 74, 76, 84, 87, 102, 103
82	-17.463549	14.680952	61, 66, 77, 94, 104, 109
83	-17.462295	14.749303	68, 80, 95, 98, 112, 116
84	-17.461845	14.707758	64, 73, 81, 102, 105, 106, 114
85	-17.462928	14.692848	65, 79, 94, 101, 113
86	-17.463921	14.759949	53, 67, 69, 90, 123, 145
87	-17.463202	14.717809	60, 72, 76, 81, 89, 99, 103
88	-17.463548	14.738111	71, 78, 93, 95, 107
89	-17.461157	14.721505	72, 87, 99, 100, 108
90	-17.462370	14.757709	68, 69, 86, 96, 98
91	-17.462261	14.698695	70, 73, 79, 101, 105
92	-17.462257	14.726853	72, 75, 97, 100, 117
93	-17.461361	14.733214	75, 88, 97, 107, 119
94	-17.461646	14.688180	77, 82, 85, 104, 110, 113
95	-17.460571	14.741360	78, 80, 83, 88, 107, 111, 112, 118
96	-17.459618	14.756228	90, 98, 115
97	-17.460410	14.729414	75, 92, 93, 117, 119, 121
98	-17.459463	14.752255	68, 83, 90, 96, 115, 116
99	-17.460724	14.720153	87, 89, 103, 108, 122
100	-17.461115	14.724244	72, 89, 92, 108, 117

Tabla 8.2: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 2).

Vertice	Longitud	Latitud	Vertices conectados
101	-17.459076	14.696802	85, 91, 105, 113, 128, 131
102	-17.458913	14.712193	81, 84, 103, 114, 127
103	-17.458807	14.715797	81, 87, 99, 102, 122, 127
104	-17.459468	14.680770	82, 94, 109, 110, 120, 124
105	-17.459146	14.701626	73, 84, 91, 101, 106, 125, 131
106	-17.457884	14.706047	84, 105, 114, 125
107	-17.459382	14.737253	88, 93, 95, 118, 119, 138
108	-17.458092	14.721619	89, 99, 100, 117, 122, 126
109	-17.458340	14.679061	61, 82, 104, 124, 130, 150
110	-17.456986	14.686536	94, 104, 113, 120, 132, 140
111	-17.457424	14.741267	95, 112, 118, 136, 138
112	-17.457446	14.744069	83, 95, 111, 116, 133, 136
113	-17.458318	14.690811	85, 94, 101, 110, 128, 132
114	-17.456147	14.707861	84, 102, 106, 125, 127, 141
115	-17.456609	14.754299	96, 98, 116, 129, 134
116	-17.458094	14.750450	83, 98, 112, 115, 133, 134
117	-17.456937	14.726170	92, 97, 100, 108, 121, 126, 135, 137
118	-17.457853	14.740414	95, 107, 111, 138
119	-17.456170	14.734001	93, 97, 107, 121, 138, 152, 154
120	-17.456405	14.682720	104, 110, 124, 140
121	-17.455123	14.730459	97, 117, 119, 137, 154
122	-17.455971	14.718371	99, 103, 108, 126, 127, 142
123	-17.456315	14.758179	86, 145
124	-17.455961	14.680300	104, 109, 120, 130, 140, 147
125	-17.454238	14.705465	105, 106, 114, 131, 141, 157, 159
126	-17.455121	14.721854	108, 117, 122, 135, 142, 146, 160
127	-17.455819	14.712339	102, 103, 114, 122, 141, 142, 144
128	-17.455101	14.693734	101, 113, 131, 132, 139
129	-17.454573	14.756774	115, 134, 145, 151, 153
130	-17.453092	14.676733	109, 124, 147, 150
131	-17.452843	14.700483	101, 105, 125, 128, 139, 148, 157
132	-17.453588	14.688325	110, 113, 128, 139, 140, 155, 161
133	-17.453738	14.747978	112, 116, 134, 136, 151, 167
134	-17.455278	14.752542	115, 116, 129, 133, 151
135	-17.454267	14.725193	117, 126, 137, 146, 162, 166
136	-17.453263	14.742610	111, 112, 133, 138, 143, 156, 164
137	-17.454343	14.729793	117, 121, 135, 154, 166
138	-17.453261	14.738625	107, 111, 118, 119, 136, 143, 152, 164
139	-17.452326	14.693656	128, 131, 132, 148, 161, 170
140	-17.453440	14.683465	110, 120, 124, 132, 147, 155, 163
141	-17.453441	14.709997	114, 125, 127, 144, 149, 158, 159
142	-17.453055	14.716741	122, 126, 127, 144, 160, 168, 174
143	-17.452308	14.740615	136, 138, 164
144	-17.451809	14.714067	127, 141, 142, 149, 168
145	-17.452314	14.762411	53, 86, 123, 129, 153, 169, 186
146	-17.451919	14.722878	126, 135, 160, 162
147	-17.451185	14.678560	124, 130, 140, 150, 163, 165, 179
148	-17.449819	14.696307	131, 139, 170, 191, 192
149	-17.450986	14.711912	141, 144, 158, 168, 172
150	-17.450794	14.674675	109, 130, 147, 175, 179, 208

Tabla 8.3: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 3).

Vertice	Longitud	Latitud	Vertices conectados
151	-17.451205	14.754448	129, 133, 134, 153, 167, 176, 178
152	-17.452184	14.736771	119, 138, 154, 164, 180, 182
153	-17.450758	14.757879	129, 145, 151, 169, 178
154	-17.451419	14.732130	119, 121, 137, 152, 166, 180
155	-17.449332	14.686768	132, 140, 161, 163, 177, 185
156	-17.449631	14.744686	136, 164, 173, 190
157	-17.450096	14.703719	125, 131, 159, 183, 192
158	-17.449222	14.709474	141, 149, 159, 171, 172
159	-17.450545	14.707033	125, 141, 157, 158, 171, 183
160	-17.449695	14.720262	126, 142, 146, 162, 174, 184
161	-17.450404	14.690296	132, 139, 155, 170, 177
162	-17.449841	14.724163	135, 146, 160, 166, 181, 184
163	-17.449755	14.681792	140, 147, 155, 165, 179, 185
164	-17.449850	14.740438	136, 138, 143, 152, 156, 182, 189, 190
165	-17.449754	14.679568	147, 163, 179
166	-17.449089	14.726699	135, 137, 154, 162, 180, 181, 195
167	-17.448685	14.749559	133, 151, 173, 176, 194, 203
168	-17.448846	14.714874	142, 144, 149, 172, 174, 187
169	-17.448004	14.761569	145, 153, 178, 186, 196, 198
170	-17.446510	14.694214	139, 148, 161, 177, 188, 191
171	-17.447830	14.709486	158, 159, 172, 183, 187, 206
172	-17.448241	14.710364	149, 158, 168, 171, 187
173	-17.448722	14.746471	156, 167, 190, 203, 207
174	-17.448159	14.715617	142, 160, 168, 187, 204, 205
175	-17.447423	14.672989	150, 179, 193, 197, 200, 208
176	-17.448362	14.752828	151, 167, 178, 194
177	-17.447022	14.687635	155, 161, 170, 185, 201
178	-17.446935	14.755621	151, 153, 169, 176, 194, 198, 199
179	-17.446367	14.677470	147, 150, 163, 165, 175, 185, 193, 202
180	-17.446842	14.731638	152, 154, 166, 182, 195
181	-17.445823	14.725674	162, 166, 184, 195, 210
182	-17.444960	14.736088	152, 164, 180, 189
183	-17.444736	14.706359	157, 159, 171, 192, 206, 224
184	-17.445925	14.723525	160, 162, 181, 210
185	-17.446062	14.683975	155, 163, 177, 179, 201, 202, 211
186	-17.446843	14.763943	145, 169, 196, 209, 236, 294
187	-17.446144	14.712313	168, 171, 172, 174, 204, 206
188	-17.443712	14.692505	170, 191, 232
189	-17.444715	14.738841	164, 182, 190, 220, 231, 245
190	-17.444959	14.742476	156, 164, 173, 189, 207, 220
191	-17.444697	14.697815	148, 170, 188, 232
192	-17.445921	14.701803	148, 157, 183, 224, 232
193	-17.443503	14.675178	175, 179, 200, 202, 219
194	-17.444333	14.753048	167, 176, 178, 199, 203, 215
195	-17.444145	14.727931	166, 180, 181, 210, 213, 233, 239
196	-17.443766	14.760615	169, 186, 198, 209, 212
197	-17.442134	14.666512	175, 200, 208, 216
198	-17.444046	14.758063	169, 178, 196, 199, 212, 222, 229
199	-17.443511	14.755149	178, 194, 198, 215, 229
200	-17.443365	14.670718	175, 193, 197, 214, 216, 217, 219

Tabla 8.4: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 4).

Vertice	Longitud	Latitud	Vertices conectados
201	-17.443259	14.687435	177, 185, 211, 237
202	-17.443206	14.678117	179, 185, 193, 211, 219, 228
203	-17.441609	14.750080	167, 173, 194, 207, 215, 218, 223, 240
204	-17.442743	14.715371	174, 187, 206, 234, 244
205	-17.443513	14.719337	174, 210, 244
206	-17.441900	14.708396	171, 183, 187, 204, 224, 234
207	-17.441113	14.746749	173, 190, 203, 220, 223
208	-17.441881	14.664208	150, 175, 197, 216, 221, 225, 226
209	-17.442331	14.762564	186, 196, 212, 236
210	-17.442384	14.723569	181, 184, 195, 205, 239, 244, 248
211	-17.440909	14.682723	185, 201, 202, 228, 237
212	-17.441792	14.761555	196, 198, 209, 222, 236, 238
213	-17.440738	14.733186	195, 231, 233
214	-17.440460	14.669595	200, 216, 217, 227
215	-17.439990	14.753479	194, 199, 203, 218, 229
216	-17.440436	14.667928	197, 200, 208, 214, 226, 227
217	-17.439632	14.672184	200, 214, 219, 227, 235
218	-17.439341	14.752508	203, 215, 229, 240, 251
219	-17.440287	14.674553	193, 200, 202, 217, 228, 230, 235
220	-17.439115	14.743563	189, 190, 207, 223, 240, 245, 255
221	-17.440208	14.660389	208, 225, 241, 254, 258
222	-17.439010	14.758103	198, 212, 229, 238, 251
223	-17.440589	14.747783	203, 207, 220, 240
224	-17.439075	14.704144	183, 192, 206, 234, 268
225	-17.437469	14.662898	208, 221, 226, 243, 246, 254
226	-17.437762	14.667043	208, 216, 225, 227, 242, 243
227	-17.437594	14.669627	214, 216, 217, 226, 235, 242, 247
228	-17.438060	14.679810	202, 211, 219, 230, 237, 249
229	-17.439466	14.755914	198, 199, 215, 218, 222, 251
230	-17.436298	14.675781	219, 228, 235, 249, 250
231	-17.439254	14.735788	189, 213, 233, 245, 262
232	-17.438993	14.698692	188, 191, 192, 252
233	-17.438275	14.731385	195, 213, 231, 239, 262
234	-17.437649	14.709367	204, 206, 224, 244, 256, 268
235	-17.436481	14.672241	217, 219, 227, 230, 247, 250
236	-17.437258	14.766124	186, 209, 212, 238, 260, 269, 272, 294
237	-17.436633	14.686404	201, 211, 228, 249, 252
238	-17.435953	14.761432	212, 222, 236, 251, 260, 263, 264
239	-17.437041	14.729788	195, 210, 233, 248, 262, 278
240	-17.436565	14.748868	203, 218, 220, 223, 251, 255
241	-17.433530	14.655467	221, 254, 258
242	-17.436104	14.667438	226, 227, 243, 247, 253
243	-17.436201	14.665928	225, 226, 242, 246, 253
244	-17.435958	14.718992	204, 205, 210, 234, 248, 256, 270
245	-17.436633	14.739359	189, 220, 231, 255, 262, 266
246	-17.434497	14.662749	225, 243, 253, 254, 261
247	-17.434355	14.670721	227, 235, 242, 250, 253, 257
248	-17.436537	14.723338	210, 239, 244
249	-17.433789	14.676848	228, 230, 237, 250, 271, 283
250	-17.432943	14.673258	230, 235, 247, 249, 257, 267, 271

Tabla 8.5: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 5).

Vertice	Longitud	Latitud	Vertices conectados
251	-17.435862	14.756231	218, 222, 229, 238, 240, 259, 264
252	-17.433045	14.692323	232, 237
253	-17.432688	14.667638	242, 243, 246, 247, 257, 261, 267
254	-17.433604	14.659902	221, 225, 241, 246, 261
255	-17.435168	14.741912	220, 240, 245
256	-17.432269	14.713494	234, 244, 268, 270, 286
257	-17.432321	14.670375	247, 250, 253, 267
258	-17.433549	14.651822	221, 241
259	-17.431178	14.751528	251, 264, 277
260	-17.432433	14.763861	236, 238, 263, 272, 276
261	-17.430814	14.666353	246, 253, 254, 267, 271, 283
262	-17.433216	14.734829	231, 233, 239, 245, 266, 273, 278
263	-17.431159	14.760272	238, 260, 264, 275, 276
264	-17.431715	14.757625	238, 251, 259, 263, 275, 277
265	-17.430347	14.699048	268, 282, 286
266	-17.431201	14.739776	245, 262, 287
267	-17.430032	14.670340	250, 253, 257, 261, 271
268	-17.429042	14.706157	224, 234, 256, 265, 286
269	-17.428684	14.768691	236, 272, 279, 294
270	-17.427909	14.724121	244, 256, 278
271	-17.428465	14.670912	249, 250, 261, 267, 283
272	-17.430142	14.765228	236, 260, 269, 276, 279, 288
273	-17.429871	14.734365	262, 274, 278, 281, 285
274	-17.427916	14.739849	273, 281, 284, 287
275	-17.428758	14.758768	263, 264, 276, 277, 280
276	-17.428020	14.761437	260, 263, 272, 275, 280, 288
277	-17.426597	14.756671	259, 264, 275, 280, 292, 293
278	-17.427857	14.730956	239, 262, 270, 273, 285
279	-17.425711	14.768775	269, 272, 288, 290, 294
280	-17.425364	14.760166	275, 276, 277, 288, 293
281	-17.426557	14.736299	273, 274, 284, 285, 291, 295
282	-17.425157	14.687782	265
283	-17.425784	14.671096	249, 261, 271, 316
284	-17.425205	14.739142	274, 281, 287, 295
285	-17.425537	14.733195	273, 278, 281, 291
286	-17.427027	14.705377	256, 265, 268, 289
287	-17.424548	14.744221	266, 274, 284, 292, 295, 297, 302
288	-17.424340	14.762116	272, 276, 279, 280, 293, 296, 298
289	-17.421231	14.701275	286
290	-17.422637	14.769997	279, 294, 296
291	-17.423018	14.733418	281, 285, 295, 297, 301
292	-17.424649	14.749654	277, 287
293	-17.420338	14.759644	277, 280, 288, 298
294	-17.420754	14.772338	186, 236, 269, 279, 290, 296, 299, 307
295	-17.421336	14.737062	281, 284, 287, 291, 297
296	-17.421033	14.767416	288, 290, 294, 298, 299, 303
297	-17.417973	14.738869	287, 291, 295, 301, 302
298	-17.417958	14.764623	288, 293, 296, 300, 303
299	-17.416721	14.770730	294, 296, 303, 304, 307
300	-17.415131	14.762452	298, 303, 308

Tabla 8.6: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 6).

Vertice	Longitud	Latitud	Vertices conectados
301	-17.413631	14.738052	291, 297, 302, 309, 333
302	-17.416400	14.744657	287, 297, 301, 309
303	-17.415323	14.766134	296, 298, 299, 300, 308
304	-17.412967	14.770074	299, 307, 308, 310
305	-17.414196	14.749365	306
306	-17.411237	14.752171	305, 311, 313
307	-17.412965	14.775014	294, 299, 304, 312
308	-17.409955	14.767178	300, 303, 304, 310
309	-17.410001	14.740563	301, 302, 311, 319, 333
310	-17.407699	14.770926	304, 308, 312, 315
311	-17.406508	14.745924	306, 309, 313, 319
312	-17.406459	14.777187	307, 310, 315, 318, 322
313	-17.406378	14.751601	306, 311, 317, 319, 321, 327
314	-17.403287	14.768607	315
315	-17.401944	14.772847	310, 312, 314, 318, 323, 324, 325
316	-17.399363	14.667383	283
317	-17.399162	14.753349	313, 320, 321, 327
318	-17.398950	14.777574	312, 315, 322, 323
319	-17.400597	14.745270	309, 311, 313, 321, 328, 333
320	-17.395879	14.754732	317, 321, 327, 328, 329, 332
321	-17.398003	14.749345	313, 317, 319, 320, 328
322	-17.398166	14.778446	312, 318, 323, 330, 336
323	-17.396113	14.774319	315, 318, 322, 325, 330, 331
324	-17.398737	14.767750	315, 325, 326, 327
325	-17.396639	14.771096	315, 323, 324, 326, 331, 338, 339
326	-17.394876	14.764435	324, 325, 327, 329, 335, 338
327	-17.396561	14.759777	313, 317, 320, 324, 326, 329
328	-17.393928	14.748613	319, 320, 321, 332, 333, 334
329	-17.393672	14.760057	320, 326, 327, 332, 335, 340
330	-17.394190	14.775865	322, 323, 331, 336, 337, 341
331	-17.393315	14.774390	323, 325, 330, 339, 341
332	-17.391730	14.752540	320, 328, 329, 334, 340, 342
333	-17.391927	14.744436	301, 309, 319, 328, 334, 346, 356
334	-17.391303	14.747953	328, 332, 333, 342, 346
335	-17.389477	14.761888	326, 329, 338, 340, 343
336	-17.390437	14.784365	322, 330, 337, 349
337	-17.387440	14.779987	330, 336, 341, 345, 349
338	-17.389612	14.766942	325, 326, 335, 339, 343, 344
339	-17.388912	14.768881	325, 331, 338, 341, 344
340	-17.388276	14.758121	329, 332, 335, 342, 343, 350, 354
341	-17.388106	14.776314	330, 331, 337, 339, 345, 348, 351
342	-17.386638	14.752153	332, 334, 340, 346, 347, 354
343	-17.385586	14.762934	335, 338, 340, 344, 350
344	-17.383317	14.764955	338, 339, 343, 352
345	-17.384452	14.779569	337, 341, 349, 351, 353
346	-17.382946	14.743604	333, 334, 342, 347, 355, 356
347	-17.381514	14.750327	342, 346, 354, 355, 361
348	-17.384027	14.773106	341, 351
349	-17.382916	14.787288	336, 337, 345, 353, 357
350	-17.380425	14.761916	340, 343, 352, 354, 362

Tabla 8.7: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 7).

Vertice	Longitud	Latitud	Vertices conectados
351	-17.382058	14.775739	341, 345, 348, 352, 353, 358, 360
352	-17.380054	14.768207	344, 350, 351, 360, 362
353	-17.379190	14.782621	345, 349, 351, 357, 358, 359
354	-17.378316	14.759396	340, 342, 347, 350, 361, 362
355	-17.378751	14.746689	346, 347, 356, 361, 364, 365
356	-17.377913	14.741290	333, 346, 355, 364, 390
357	-17.377095	14.785350	349, 353, 359
358	-17.376922	14.776504	351, 353, 359, 360, 363, 366
359	-17.375197	14.783466	353, 357, 358, 363, 370
360	-17.375774	14.770187	351, 352, 358, 362, 366, 367, 368
361	-17.375165	14.756141	347, 354, 355, 365
362	-17.374784	14.763198	350, 352, 354, 360, 368, 369
363	-17.372658	14.782240	358, 359, 366, 370, 375
364	-17.371919	14.742478	355, 356, 365, 371, 382, 390
365	-17.371847	14.749116	355, 361, 364, 371, 373
366	-17.373589	14.775756	358, 360, 363, 375, 377
367	-17.370399	14.771740	360, 368, 377, 378
368	-17.368889	14.764656	360, 362, 367, 369, 378, 380
369	-17.370421	14.760550	362, 368, 374, 380
370	-17.370338	14.783532	359, 363, 372, 375, 381
371	-17.367944	14.746567	364, 365, 373, 379, 382
372	-17.369331	14.790920	370, 376, 383
373	-17.367982	14.751639	365, 371, 374, 379
374	-17.366416	14.756247	369, 373, 379, 380, 384, 385
375	-17.365284	14.777193	363, 366, 370, 377, 381
376	-17.360767	14.794309	372, 398
377	-17.365803	14.773671	366, 367, 375, 378, 381, 389
378	-17.363901	14.765330	367, 368, 377, 380, 389, 392
379	-17.362311	14.750437	371, 373, 374, 382, 385, 386
380	-17.362786	14.761854	368, 369, 374, 378, 384, 391, 392
381	-17.361811	14.778213	370, 375, 377, 383, 388, 389
382	-17.361154	14.743747	364, 371, 379, 386, 387, 390
383	-17.359223	14.787370	372, 381, 388, 395
384	-17.361364	14.759368	374, 380, 385, 391
385	-17.359270	14.754807	374, 379, 384, 386, 391, 393, 394
386	-17.359867	14.750764	379, 382, 385, 387, 394
387	-17.356759	14.742023	382, 386, 390, 394, 402
388	-17.354386	14.776883	381, 383, 389, 395, 396
389	-17.357797	14.770236	377, 378, 381, 388, 392, 396
390	-17.355964	14.738071	356, 364, 382, 387
391	-17.355212	14.761077	380, 384, 385, 392, 393, 397
392	-17.355092	14.765380	378, 380, 389, 391, 396, 397, 401
393	-17.352510	14.755504	385, 391, 394, 397, 399
394	-17.352732	14.749454	385, 386, 387, 393, 399, 402
395	-17.350808	14.781414	383, 388, 396, 398, 400
396	-17.351521	14.774416	388, 389, 392, 395, 400, 401
397	-17.346801	14.763930	391, 392, 393, 399, 401, 405
398	-17.348632	14.792699	376, 395, 400, 406
399	-17.345369	14.754224	393, 394, 397, 402
400	-17.346760	14.780162	395, 396, 398, 406

Tabla 8.8: Lista de adyacencia de los vértices de la triangulación Delaunay restringida ()Parte 8.

Vertice	Longitud	Latitud	Vertices conectados
401	-17.346545	14.767762	392, 396, 397, 403, 405
402	-17.345050	14.748365	387, 394, 399, 408
403	-17.334653	14.774639	401, 405, 407, 412
404	-17.336544	14.797036	406, 413, 421, 445
405	-17.335022	14.771625	397, 401, 403, 410, 412
406	-17.339373	14.792272	398, 400, 404, 407, 413
407	-17.335275	14.783797	403, 406, 412, 413
408	-17.333992	14.744456	402, 409, 411, 416
409	-17.333304	14.734990	408, 411, 414
410	-17.328629	14.767369	405, 412, 417
411	-17.326125	14.742183	408, 409
412	-17.327504	14.776569	403, 405, 407, 410, 413, 417, 420
413	-17.327621	14.791916	404, 406, 407, 412, 420, 421
414	-17.322137	14.734151	409, 415, 418, 422, 424
415	-17.324326	14.730056	414, 422
416	-17.319936	14.747263	408, 419, 423, 430
417	-17.322975	14.774418	410, 412, 420, 426
418	-17.318592	14.738754	414, 419, 424
419	-17.316681	14.743604	416, 418, 423
420	-17.316665	14.776986	412, 413, 417, 421, 425, 426, 427
421	-17.314130	14.790477	404, 413, 420, 425, 428, 436, 445
422	-17.315868	14.725569	414, 415, 424, 433
423	-17.311180	14.744648	416, 419, 424, 429, 430, 435
424	-17.313185	14.733987	414, 418, 422, 423, 433, 435
425	-17.310757	14.782069	420, 421, 427, 428
426	-17.309017	14.770456	417, 420, 427, 430, 431, 432
427	-17.308942	14.776337	420, 425, 426, 428, 432
428	-17.306781	14.779394	421, 425, 427, 432, 436, 439
429	-17.300799	14.752824	423, 430, 431, 435
430	-17.304989	14.762330	416, 423, 426, 429, 431
431	-17.301805	14.767697	426, 429, 430, 432, 439, 447
432	-17.305544	14.773463	426, 427, 428, 431, 439
433	-17.298860	14.730770	422, 424, 434, 435, 440, 441
434	-17.299874	14.720930	433, 437, 438, 440
435	-17.298678	14.741757	423, 424, 429, 433, 441, 446
436	-17.285133	14.789759	421, 428, 439, 445, 450, 451
437	-17.286711	14.714413	434, 438, 442, 457, 464, 474
438	-17.288601	14.717067	434, 437, 440, 442
439	-17.285786	14.771577	428, 431, 432, 436, 447, 450
440	-17.285503	14.721419	433, 434, 438, 441, 442, 443, 448
441	-17.283862	14.731798	433, 435, 440, 443, 444, 446, 449
442	-17.285611	14.714662	437, 438, 440, 448, 457
443	-17.280255	14.723505	440, 441, 448, 449, 452
444	-17.279759	14.733409	441, 446, 449, 455
445	-17.278805	14.803825	404, 421, 436, 451, 469
446	-17.276177	14.744689	435, 441, 444, 455, 460, 466
447	-17.284472	14.761459	431, 439, 450
448	-17.276849	14.717804	440, 442, 443, 452, 453, 454, 457
449	-17.276510	14.726831	441, 443, 444, 452, 455
450	-17.279840	14.774764	436, 439, 447, 451, 456, 466

Tabla 8.9: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 9).

Vertice	Longitud	Latitud	Vertices conectados
451	-17.275614	14.790427	436, 445, 450, 456, 469
452	-17.271845	14.722300	443, 448, 449, 453, 454, 455, 458, 459
453	-17.273310	14.717857	448, 452, 454
454	-17.270324	14.715575	448, 452, 453, 457, 458, 461, 462
455	-17.272012	14.733086	444, 446, 449, 452, 459, 460, 463
456	-17.268864	14.783741	450, 451, 466, 469, 477
457	-17.269163	14.712002	437, 442, 448, 454, 461, 464
458	-17.266627	14.721345	452, 454, 459, 462
459	-17.263785	14.725115	452, 455, 458, 462, 463, 468
460	-17.261962	14.733729	446, 455, 463, 466, 467, 468, 470, 471
461	-17.262507	14.714573	454, 457, 462, 464, 465, 468
462	-17.262710	14.717958	454, 458, 459, 461, 468
463	-17.262989	14.728585	455, 459, 460, 468
464	-17.259207	14.707808	437, 457, 461, 465, 472, 474
465	-17.258484	14.712489	461, 464, 467, 468, 470
466	-17.256660	14.758244	446, 450, 456, 460, 477
467	-17.253959	14.717348	460, 465, 468, 470
468	-17.258770	14.720176	459, 460, 461, 462, 463, 465, 467
469	-17.246589	14.806147	445, 451, 456, 477
470	-17.247327	14.716256	460, 465, 467
471	-17.238663	14.746541	460, 477, 480
472	-17.235778	14.693928	464, 473, 474, 475
473	-17.231794	14.706243	472, 475
474	-17.232490	14.690125	437, 464, 472, 475, 476
475	-17.225915	14.699081	472, 473, 474, 476
476	-17.224150	14.694309	474, 475, 480
477	-17.224978	14.767129	456, 466, 469, 471, 479
478	-17.204877	14.673411	485
479	-17.184373	14.789131	477, 481
480	-17.178288	14.736500	471, 476, 488
481	-17.165920	14.820360	479, 484
482	-17.165143	14.645918	483, 485, 488
483	-17.152019	14.625859	482, 485, 488
484	-17.145683	14.749488	481, 486, 487
485	-17.150009	14.608968	478, 482, 483
486	-17.150519	14.743846	484, 487
487	-17.136879	14.745807	484, 486, 489, 490
488	-17.153215	14.673134	480, 482, 483
489	-17.120247	14.741666	487
490	-17.129907	14.719436	487

Tabla 8.10: Lista de adyacencia de los vértices de la triangulación Delaunay restringida (Parte 10).

Bibliografía

[1] A. Noulas, S. Scellato, R. Lambiotte, M. Pontil, and C. Mascolo, "A tale of many cities: Universal patterns in human urban mobility," *PLoS ONE*, vol. 7, no. 5, p. e37027, 2012. [Online]. Available: http://goo.gl/rfisQi

[2] F. Giannotti, M. Nanni, D. Pedreschi, F. Pinelli, C. Renso, S. Rinzivillo, and R. Trasarti, "Unveiling the complexity of human mobility by querying and mining massive trajectory data," *The VLDB Journal*, vol. 20, no. 5, pp. 695–719, 2011. [Online]. Available: http://goo.gl/EtlJK6

[3] E. W. Dijkstra, "A note of two problems in connection with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://goo.gl/mo5cZ0

[4] M. C. González, C. A. Hidalgo, and A.-L. Barabási, "Understanding individual human mobility patterns," *Nature*, vol. 458, no. 7235, pp. 238–238, 2009. [Online]. Available: http://goo.gl/8OJq1d

[5] S. Hasan, X. Zhan, and S. V. Ukkusuri, "Understanding urban human activity and mobility patterns using large-scale location-based data from online social media," *Proceedings of the 2Nd ACM SIGKDD International Workshop on Urban Computing*, 2013. [Online]. Available: http://doi.acm.org/10.1145/2505821.2505823

[6] F. Calabrese, M. Diao, G. Di Lorenzo, J. Ferreira, and C. Ratti, "Understanding individual mobility patterns from urban sensing data: A mobile phone trace example," *Transportation Research Part C: Emerging Technologies*, vol. 26, pp. 301–313, 2013.

[7] G. Di Lorenzo, M. Sbodio, F. Calabrese, M. Berlingerio, F. Pinelli, and R. Nair, "All aboard: Visual exploration of cellphone mobility data to optimise public transport," *IEEE Trans. Visual. Comput. Graphics*, vol. 22, no. 2, pp. 1036–1050, 2016.

[8] J. Aguilar and M. A. Labrador, "Un algoritmo de enrutamiento distribuido para redes de comunicacion basado en sistemas de hormigas," *IEEE LATIN AMERICA TRANSACTIONS*, vol. 5, no. 8, pp. 616–624, 2007. [Online]. Available: http://goo.gl/uMdqse

[9] E. Mancera-Galván, B. A. Garro-Licón, and K. Rodríguez-Vázquez, "Optimización mediante algoritmo de hormigas aplicado a la recolección de residuos sólidos en unam-cu," *Research in computing science*, vol. 94, pp. 163–177, 2015. [Online]. Available: http://goo.gl/80CEYK

[10] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: An autocatalytic optimizing process," *IEEE Transactions on Systems, Man, and Cybernetics*, 1991. [Online]. Available: http://goo.gl/WzmHmN

[11] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization-artificial ants as a computational intelligence technique," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006. [Online]. Available: https://goo.gl/5VqmsE

[12] T. Stützle and H. H. Hoos, "Max min ant system," *Future Generation Computer Systems*, vol. 16, no. 9, pp. 889–914, 2000. [Online]. Available: https://goo.gl/bgSXC8

[13] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *Biosystems*, vol. 43, no. 2, pp. 73–81, 1997. [Online]. Available: http://goo.gl/eTnjyc

[14] M. Dorigo and L. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997. [Online]. Available: http://goo.gl/7mLbqZ

[15] L. M. Gambardella and M. Dorigo, "Solving symmetric and asymmetric tsps by ant colonies," *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 622–627, 1996. [Online]. Available: http://goo.gl/HH6mJL

[16] D. Gaertner and K. L. Clark, "On optimal parameters for ant colony optimization algorithms," in *Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI 2005, Las Vegas, Nevada, USA, June 27-30, 2005, Volume 1*, 2005, pp. 83–89. [Online]. Available: http://goo.gl/0EftBc

[17] K. Socha, "The influence of run-time limits on choosing ant system parameters," in *Genetic and Evolutionary Computation – GECCO-2003*, ser. LNCS, E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, Eds., vol. 2723. Chicago: Springer-Verlag, 12-16 July 2003, pp. 49–60. [Online]. Available: http://goo.gl/jfohuV

[18] M. A. Weiss, *Data structures and algorithm analysis in C++.* Addison Wesley, 1999.

[19] R. Sedgewick, *Algorithms.* Addison-Wesley, 1988.

[20] F. Rivero Mendoza, *Geometría computacional.* Universidad de los Andes, 2006. [Online]. Available: http://goo.gl/6HeFbs

[21] S. Sen and A. Kumar, *Notes in computational geometry: Voronoi diagrams.* Indian Institute of Technology, 2006. [Online]. Available: http://goo.gl/aZIiGS

[22] M. De Berg, *Computational geometry.* Springer, 2000.