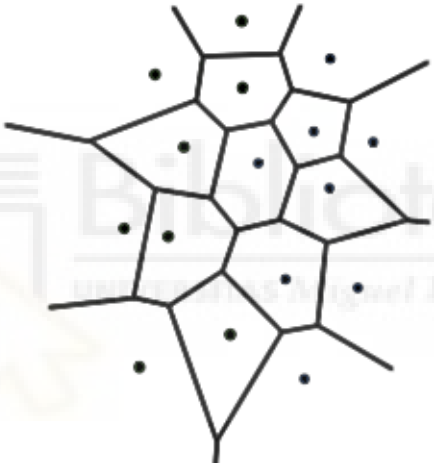
Geometría Computacional: Diagramas de Voronoi



Realizado por

María José Hernández Ferrández

Dirigido por

Mercedes Ruiz Landete

Universidad Miguel Hernández
Facultad de Ciencias Sociales y Jurídicas de Elche
Grado en Estadística Empresarial (2015 - 2019)

Índice

1 Resumen

Los diagramas de Voronoi realizan divisiones del plano euclídeo en regiones a través de unos puntos establecidos y, además, se encuentran internamente relacionados con los problemas de proximidad.

En este trabajo se exponen varios supuestos de aplicaciones del diagrama de Voronoi, así como problemas clásicos concominantes, para poder observar que dicha representación gráfica se puede adaptar a distintas situaciones.
El objetivo es investigar y comprender la funcionalidad de los diagramas de Voronoi. Para ello, se presentan varios algoritmos para su construcción y, una vez seleccionado uno de ellos, pasamos a implantarlo en una aplicación nacida de una idea personal pero comúnmente útil. La localización para llevar a cabo la aplicación ha sido la ciudad de Elche, Alicante.

La memoria se ha realizado mediante RSweave, incluyendo la configuración del sistema LaTeX.

2 Preliminares

2.1 ¿Qué es la Minería de Datos?

La **Minería de Datos**, también conocida como **Data Mining** , es el proceso de descubrir patrones, tendencias o relaciones interesantes que normalmente no se detectan con los métodos tradicionales en los grandes conjuntos de datos. Esta metodología nace de la necesidad de tratar esos grandes volúmenes de datos para extraer conocimiento útil y se basa fundamentalmente en agrupar los datos y obtener reglas de asociación pudiendo, posteriormente, crear un modelo que permita clasificarlos.
Es importante destacar que la Minería de Datos o Data Mining hace referencia a una etapa de la metodología **KDD (Knowledge Discovery in Databases)** que consiste en una secuencia iterativa de procesos que explota el contenido de los datos para extraer información útil, la cual es previamente desconocida.

2.2 Knowledge Discovery in Databases

El **Descubrimiento de Conocimiento de Bases de Datos o KDD** implica la evaluación e interpretación de patrones y modelos para tomar decisiones con respecto a lo que constituye conocimiento.

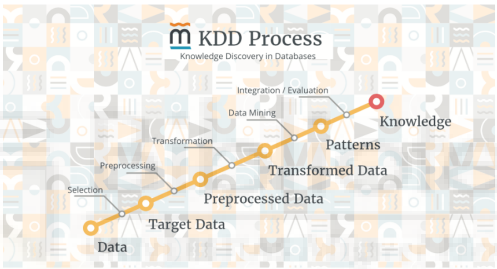


Figure 1: Proceso KDD

Para aclarar estos conceptos, procedemos a explicar brevemente las fases del KDD:

1. **Selección** de los datos relevantes para el análisis, definición de las fuentes de datos y el tipo de información a utilizar.
2. **Preprocesamiento** de los datos consistente en la preparación y limpieza de los mismos para su posterior transformación.
3. En la etapa de **transformación** se realiza un primer tratamiento de los datos a través de operaciones de normalización.
4. **Data Mining**: proceso de extracción de patrones previamente desconocidos que aplica herramientas de diversas áreas tales como Sistemas de Bases de datos, Aprendizaje Automático, Estadística, Visualización de la Información y Computación de Alto Desempeño para descubrir así información que está implícita en los datos, la cual puede ser usada en la toma de decisiones del usuario.
5. El proceso KDD concluye con la **interpretación y evaluación** de los resultados obtenidos en la etapa anterior.

Nuestro proyecto se basa en un problema de proximidad en el que es necesario implementar Data Mining con el objetivo de particionar o segmentar un conjunto de datos en distintos grupos empleando para ello la **Geometría Computacional**.

3 Los Diagramas de Voronoi

El trabajo está dedicado a un problema de **Geometría Computacional**, internamente relacionado con el concepto de proximidad, muy importante e influyente llamado **Diagrama de Voronoi o Teselación de Voronoi**, cuyos orígenes datan del siglo XVII. Varios nombres han sido dados a estos diagramas dependiendo del área de estudio. Su creador fue el matemático ruso Gueorgui Feodósievich Voronói, aunque éste no fue el primero en estudiarlos de una manera más compleja, sino que fue el meteorólogo americano Alfred H.Thiessen el que implementó la construcción geométrica a través de particiones del plano euclídeo; a esa construcción la denominó **Polígonos de Thiessen**.

En primer lugar, un diagrama es una representación gráfica de las relaciones entre los elementos que constituyen un conjunto o sistema. Este concepto relacionado con la Geometría Computacional crea los ya nombrados **Diagramas de Voronoi**; esto es, una descomposición de un espacio métrico en regiones, de manera que a cada objeto se le asigna una región de dicho espacio métrico formada por los puntos que están más cerca de él que de ninguno de los otros objetos. Es decir, a cada punto u objeto se le asigna la región formada por todo lo que está más cerca de él que de ningún otro.

Con ello, cuando nos planteamos problemas de proximidad surgen varios casos reales a los que asignarles este método, cada uno caracterizados con diferentes propiedades. Muchos matemáticos se han dedicado a su estudio, incluso antes de Voronoi, como es el caso del médico inglés John Snow, precursor de la epidemiología, quien utilizó la estructura de las regiones de Voronoi en 1854 cuando se produjo el **brote de cólera en Londres**. John demostró que la enfermedad era causada por el consumo de aguas contaminadas por heces mediante la relación que había entre la distribución de muertes por cólera y las fuentes de agua potable, delimitando las regiones de Voronoi. De esta manera, calculó la distancia entre la residencia de cada difunto y la fuente de agua más cercana localizando como culpable la bomba de Broad Street al ser la zona más afectada por la enfermedad según las regiones establecidas. Después de retirar dicha bomba, los brotes de cólera disminuyeron notablemente hasta extinguirse.

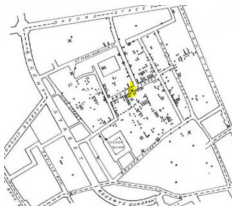


Figure 2: Diagrama de Voronoi aplicado al brote de cólera en Londres

Tras su creación, el Diagrama de Voronoi ha tenido cada vez más uso y se ha ido extendiendo. Su **utilidad**, por tanto, ha estado presente en campos muy diversos como la robótica, los sistemas de información geográfica, los fenómenos naturales o la arquitectura. Existe mucha bibliografía al respecto e incluso una web online donde poder consultar sus diversas aplicaciones [1].

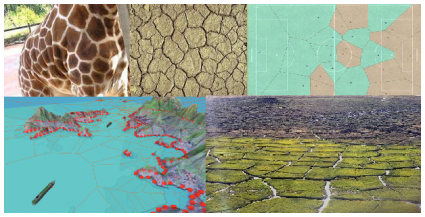


Figure 3: Aplicaciones del Diagrama del Voronoi

En definitiva, en este proyecto abordaremos el estudio de los Diagramas de Voronoi, una de las estructuras fundamentales dentro de la Geometría Computacional.

3.1 La Geometría Computacional

La Geometría Computacional es una rama de la ciencia de la computación que estudia el diseño y análisis de algoritmos multi-dimensionales, enfocado principalmente en problemas 2D y 3D, para resolver problemas que tienen como entrada y salida objetos geométricos. Estos algoritmos operan sobre objetos geométricos tales como puntos, definidos mediante coordenadas cartesianas, segmentos o polígonos. Los algoritmos en dos dimensiones son representados como un conjunto de puntos p_i donde p_i=(x_i, y_i) con x_i, $y_i \in \mathbb{R}$.

El origen de la utilización del término Geometría Computacional lo introduce el matemático estadounidense **Michael Ian Shamos** a finales de los años 70; el autor de *Computational Geometry* (1985), junto con Franco P. Preparata [2]. Desde entonces la investigación en esta área se ha expandido considerablemente encontrando así diversas aplicaciones entre las que se incluyen la robótica (planificación de movimientos y problemas de visualización), el diseño gráfico, sistemas de información geográfica (SIG) o el reconocimiento de patrones, entre muchas otras.

3.1.1 Algunos problemas clásicos

- **La envolvente convexa**

Antes de pasar a definir la envolvente convexa, también conocida como cierre convexo o convex hull, es necesario añadir el significado de **convexo**.

Definición 1 *Un conjunto de puntos en el plano es convexo si su intersección con cualquier recta del plano es conexa. Ver Figura 4.*

Por lo tanto, un conjunto de puntos en el plano es convexo si el segmento que une dos cualquiera de sus puntos está totalmente contenido en el conjunto. Ahora bien, la envolvente convexa de un conjunto de puntos $S = \{p_1, \ldots, p_n\}$ es la intersección de todos los semiplanos que contienen a S.

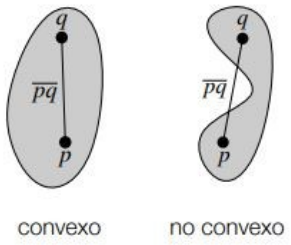


Figure 4: Concepto convexidad

El objetivo es, para cualquier subconjunto del plano S, hallar su envolvente convexa como el menor conjunto convexo que pueda contener a S; podemos ver un ejemplo en la *Figura 5.* Para calcular la envolvente convexa existen varios algoritmos como el algoritmo de Graham o el algoritmo de Quick-Hull.

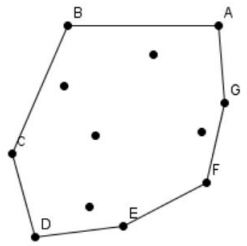


Figure 5: Envolvente convexa

Entre sus múltiples aplicaciones existentes se encuentran el cálculo del diámetro y la anchura de un conjunto. El uso de estos cálculos se implementa tanto en medidas de dispersión como en la trayectoria de robots.

- **Triangulación de un polígono**

La triangulación es la descomposición de un polígono en un conjunto de triángulos por un conjunto máximo de diagonales que no se cruzan. Con ello, ¿existe siempre una triangulación para un polígono?

Teorema 1 *Todo polígono simple*[1] *admite una triangulación, y toda triangulación de un polígono simple con n vértices consiste exactamente de n-2 triángulos.*

Demostración: Sea P un polígono con n vértices.

– Cuando $n = 3$, el polígono es un triángulo por lo que el teorema se cumple.
– Cuando $n > 3$, asumimos que el teorema se cumple para toda $m_i < n$, siendo m_i el número de vértices de cada subpolígono. Veamos un ejemplo.

Cualquier diagonal corta a P en dos subpolígonos simples P_1 y P_2, siendo m_1 el número de vértices de P_1, m_2 el número de vértices de P_2 y n el número de vértices de P. (*Ver Figura 6*)

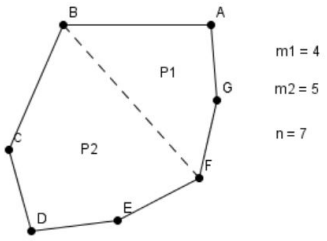


Figure 6: Demostración triangulación

Se cumple entonces que tanto m_1 como m_2 son menores que n por lo que podemos implementar el *Teorema 1* y, por inducción, P_1 y P_2 pueden ser triangulados.

Con ello tenemos que $m_1 + m_2 = n + 2$; esto implica que cualquier triangulación de P_i consiste de m_i - 2 triángulos, lo que implica que P está formado de (m_1 - 2) + (m_2 - 2) = n - 2 triángulos.

[1] Un polígono simple es aquel en el que dos de sus aristas no consecutivas no se instersectan.

Si implementamos nuestro ejemplo tenemos que:

$m_1 + m_2 = n + 2$ --→ $4 + 5 = 7 + 2$ --→ $9 = 9$ ✓

$(m_1 - 2) + (m_2 - 2) = n - 2$ --→ $(4 - 2) + (5 - 2) = 7 - 2$ --→ $2 + 3 = 5$ ✓

Con ello podemos ver que nuestro polígono P debe ser triangulado en 5 triángulos, dos pertenecientes a P_1 y 3 pertenecientes a P_2, como podemos ver en la *Figura 7*.

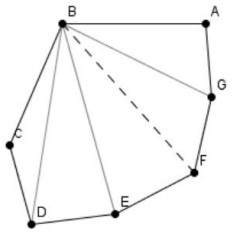


Figure 7: Demostración n - 2 triángulos

Por otro lado, existen varias formas de triangularizar los polígonos. Algunas triangulaciones, las cuales presentan propiedades especiales, son la triangulación voraz, la triangulación de peso mínimo, la triangulación en abanico y la triangulación de Delaunay; esta última nos será de vital importancia en nuestro proyecto.

Además, existen numerosas aplicaciones que utilizan la triangulación de un polígono como el cálculo de áreas de terreno.

- **Problema del mayor círculo vacío**

Este problema de geometría computacional es definido de la siguiente manera:

Definición 2 *Dados n puntos en un conjunto P, se pide hallar el mayor círculo que no contenga ningún punto del conjunto P y cuyo centro esté dentro del polígono convexo que engloba todos los puntos, es decir, dentro del cierre o envolvente convexo.*

Este problema puede solucionarse mediante la construcción del Diagrama de Voronoi o la triangulación de Delaunay (Ver Figura 8).

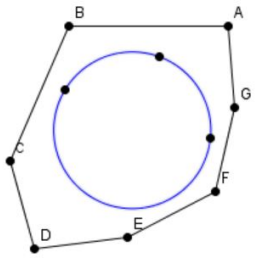


Figure 8: Mayor círculo vacío (en azul) con centro interior al cierre convexo

Sus aplicaciones se ejecutan en procesos de planificación de recursos donde debamos elegir una ubicación en el interior de un área que esté lo más alejada posible de una serie de puntos. Por ejemplo, la localización de una central nuclear lo más alejada posible de la zona urbana.

- **Búsqueda del vecino más cercano**

La **búsqueda por similitud** es un problema fundamental para un gran número de aplicaciones informáticas. Lo que se pretende es generar inferencias y establecer categorías de objetos ya que los objetos similares tienden a tener comportamientos similares.
La búsqueda del vecino más cercano o **k-vecinos más cercanos** consta de una muestra objeto y de un valor k que indica el número máximo de puntos u objetos más similares a la muestra. Para medir la similitud entre puntos u objetos es necesario hablar de distancia. *(Ver Figura 9).*

Definición 3 *Dada una muestra x, su vecino más cercano será aquel punto u objeto, nn, que se encuentra a menor distancia de x.*

$$d(x, p_2) = mind(x, p_i), \forall p \in S \tag{1}$$

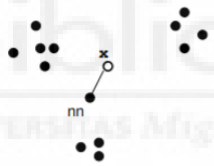


Figure 9: La búsqueda del vecino más cercano en el caso de $k = 1$

El concepto de distancia nos permite interpretar geométricamente el conjunto de objetos al posibilitar la representación de estos mismos como puntos en un espacio métrico. Ahora bien, nosotros vamos a trabajar con la denominada distancia euclídea, definida más adelante en el apartado 4.

En algunas aplicaciones la distancia euclídea es una elección adecuada aunque ésta no es la más eficiente ya que un factor crucial en el algoritmo es el tiempo de ejecución necesario para resolverlo.

La búsqueda del vecino más cercano se utiliza actualmente en una multitud de áreas como en las bases de datos multimedia, sistemas de información geográfica o el reconocimiento de formas.

3.1.2 Caso particular: La triangulación de Delaunay

La triangulación de Delaunay es una de las triangulaciones más importantes debido a que es capaz de resolver una multitud de problemas sin aparente relación y, además, cuenta con algoritmos bastante eficientes gracias a sus propiedades geométricas.

Para ser una triangulación de Delaunay debe cumplir con la **condición de Delaunay**:

La circunferencia circunscrita de cada triángulo de la red no debe contener ningún vértice de otro triángulo en su interior.

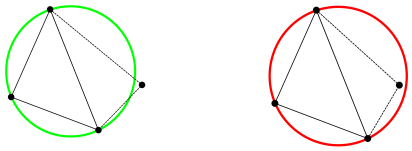


Figure 10: Se cumple la condición No se cumple la condición

Las **propiedades** básicas de esta triangulación son:

1. Todos los puntos están conectados entre sí y forman el mayor número de triángulos posibles sin que se crucen sus aristas.
2. Los puntos más próximos de los triángulos están conectados entre sí mediante aristas. La frontera externa forma la envolvente convexa, anteriormente descrita, del conjunto de puntos.
3. Con ello, se pretende maximizar sus ángulos y minimizar la longitud de sus lados, formando triángulos regulares o equiláteros.

La triangulación de Delaunay puede calcularse a través de varios métodos; a continuación, se explica uno de ellos:

Propiedades respecto a circunferencias.

Definición 4 *Dada una nube de puntos, tres de ellos son vértices de un mismo triángulo de la triangulación si y solo si puede trazarse un círculo cuyo contorno contenga esos tres puntos y no otros puntos de la nube en su interior.*

Además, dos puntos de dicha nube definen una arista si y solo si es posible trazar un círculo cuyo contorno contenga a esos dos puntos pero en su interior no contenga ningún otro punto de la nube.

Es interesante ver el contraste creado con el problema básico de triángulación *(Figura 7)*; podemos ver que el mismo polígono ahora establece una triangulación distinta pero igual de factible.

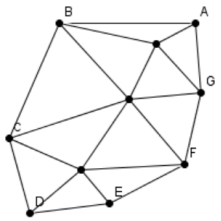


Figure 11: Triangulación de Delaunay

Por último, cabe destacar que la triangulación de Delaunay está directamente relacionada con dos de los problemas básicos explicados en el apartado 3.1.1 y también con los Diagramas de Voronoi,

como más tarde comprobaremos.

Por un lado, la propiedad 2 nos está diciendo que la frontera externa forma la envolvente convexa del conjunto de puntos mientras que la búsqueda del vecino más cercano se encuentra en cada una de las aristas de la triangulación de Delaunay, descritas como distancias en el problema de los k-vecinos más cercanos.

Por otro lado, el método de triangulación enlaza de manera natural con el Diagrama de Voronoi por lo que la construcción de uno es trivial a partir de otro.

La triangulación de Delaunay es el dual del Diagrama de Voronoi con todos sus circuncentros[2]. La circunferencia circunscrita de un triángulo es la circunferencia que contiene los tres vértices del triángulo. Con ello, los circuncentros de cada uno de los triángulos que aparecen en la Triangulación de Delaunay son los vértices de los segmentos del Diagrama de Voronoi.

A continuación vamos a poder comprobar lo dicho en el párrafo anterior observando, paso a paso, la construcción del Diagrama de Voronoi partiendo de la Triangulación de Delaunay a través de un ejemplo sencillo.

1. Se crean las cricunferencias circunscritas de cada uno de los triángulos contenidos en el polígono.

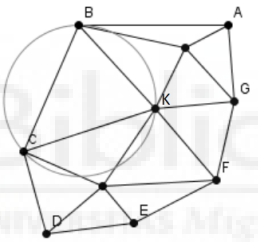


Figure 12: Circunsferencia circunscrita del triángulo formado por los puntos B,C y K

2. A partir de estas circunsferencias, se obtiene el circuncentro.

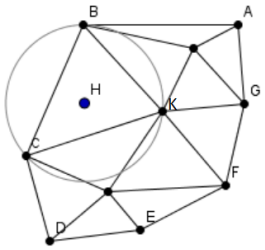


Figure 13: Obtenemos el punto H, circuncentro conseguido mediante la circunferencia obtenida en la figura anterior

Realizamos los pasos 1 y 2 para todos los triángulos contenidos en el polígono. Con ello, se nos

[2]Centro de la circunferencia circunscrita a un triángulo.

queda la siguiente figura:

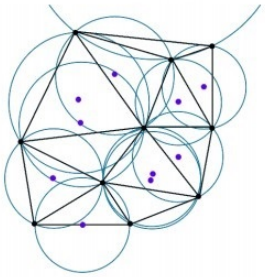


Figure 14: La triangulación de Delaunay es el dual del Diagrama de Voronoi con todos sus circuncentros

La unión de estos circuncentros produce el Diagrama de Voronoi, comprobando, de esta manera, que la triangulación de Delaunay enlaza con el Diagrama de Voronoi.

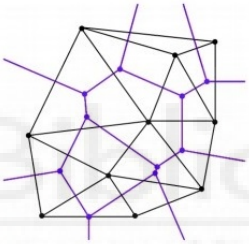


Figure 15: La triangulación de Delaunay es el dual del Diagrama de Voronoi con todos sus circuncentros

3.2 Planteamiento del problema

Nos basamos en varias razones para abordar este proyecto. En primer lugar, cabe destacar la variada implementación que conllevan los Diagramas de Voronoi, lo cual acarrea las infinitas aplicaciones a promulgar sobre este tema.

Una vez investigadas varias ideas, nos centramos en la rama de geolocalización a través de mapas en 2D. Existen varias app móviles que ofrecen servicios de transporte a través de vehículos con conductor, como puede ser Uber, que se basan en la proximidad y la disponibilidad. Por el momento, estas empresas se encuentran ubicadas en las grandes ciudades por lo que, ¿por qué no implementarlo en Elche?

Otra duda que se plantea es la fiabilidad de que realmente estas app se rigen por la cercanía. El problema recae en una correcta conexión conductor-cliente, refiriéndonos a correcta como aquel servicio más económico, es decir, una conexión lo más próxima posible.

Una vez definidos dichos puntos, lo que se ha pretendido realizar en este proyecto es una app que conlleve la solución de los dos problemas anteriores más una inventiva de negocio. Esta inventiva de negocio está basada en facilitar a los clientes la elección del destino y, además, se podría catalogar, no como un problema, sino como una mejora para los presentes y futuros servicios de transporte.

3.3 Herramientas utilizadas

Para la realización del trabajo ha sido necesario el uso de las herramientas LaTeX, dos extensiones de *R*, *RSweave* y *RShiny*, y *Cinderella*.

En primera instancia, la redacción del trabajo se ha desarrollado a través de LaTeX, un sistema de composición de documentos escritos con una alta calidad tipográfica de software libre. De esta manera, con ayuda de la herramienta *RSweave*, perteneciente a *RStudio*, poseemos un marco para mezclar texto y código *R* con el objetivo de generar documentos de manera automatizada.

Por otra parte, *RStudio* es un entorno de desarrollo integrado (IDE) para el lenguaje de programación *R*, dedicado a la computación estadística y a la visualización gráfica. *R* es uno de los lenguajes de programación más utilizados en investigación científica, siendo muy popular en el campo de la minería de datos. Además, *RStudio* nos proporciona diferentes librerías o paquetes con una gran variedad de funciones. En nuestro caso, a la hora de realizar el algoritmo para la ejecución de los diagramas de Voronoi se ha utilizado el paquete *deldir*, cuyo nombre proviene de 'Delaunay Triangulation and Dirichlet (Voronoi) Tellesation'.

Los diferentes cuerpos geométricos se han desarrollado mediante el software de geometría interactivo conocido como *Cinderella*, escrito en lenguaje de programación Java. Este programa te permite realizar diferentes ejercicios de construcción geométrica (ver punto medio, segmentos, circuncentro, tangentes exteriores).

Finalmente, para la ejecución de la App hemos utilizado *RShiny*, un paquete que facilita el desarrollo de aplicaciones web interactivas desde el mismo *R*. Es importante destacar que cada app está formada por los dos archivos *ui.R* y *server.R*. Por un lado, *ui.R* proporciona una interfaz (UI) de la app, es decir, contiene una secuencia de comandos que controla el diseño y aspecto de la aplicación. Por otro lado, *server.R* constituye los componentes de R de tu app.

4 Propiedades

Los Diagramas de Voronoi son uno de los métodos de interpolación más simples, basados en la distancia euclídea. Estos se crean al unir los puntos entre sí, trazando las mediatrices de los segmentos de unión, determinando así una serie de regiones poligonales en un espacio bidimensional.

4.1 Conceptos previos

En esta sección conoceremos las propiedades básicas de los Diagramas de Voronoi introduciendo nomenclatura más compleja para poder trabajar con ella a lo largo de todo el proyecto.

Definición 5 ***Distancia euclídea*** *Sean p y q dos puntos cualesquiera en el plano con sus coordenadas respectivas (p_x, p_y) y (q_x,q_y), la distancia euclídea entre dichos puntos está dada por:*

$$d(p,q) = ||p-q|| = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2} \tag{2}$$

- Las condiciones necesarias para ser catalogada como distancia euclídea son:

1. d(p,q) >= 0
2. d(p,q) = d(q,p) *Propiedad simétrica*
3. d(p,p) = 0
4. d(p,q) <= d(p,z) + d(z,q) *Propiedad de la desigualdad triangular*
5. Si d(p,q) = 0, entonces p=q

En un espacio n-dimensional, la distancia euclídea entre P= (p_1, ...p_n) y Q= (q_1, ...,q_n) se ajusta a:

$$d(P,Q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} \tag{3}$$

Definición 6 *Dado un conjunto P= {p_1,...,p_n} de n puntos distintos en el plano definimos el* ***Diagrama de Voronoi*** *de P como la subdivisión del plano en n regiones, tantas como puntos hayan, cumpliendo la propiedad de proximidad y lo denotaremos como Vor(P). Dichas regiones se denominan* ***regiones de Voronoi****.*

Un punto x pertenece a una región de P si y solo si,

$$d(p, s_i) < d(p, s_j), \forall s_i \in Sconj \neq i \tag{4}$$

Por lo tanto, la región de Voronoi de un punto p_i, Vor(p_i), consiste en todos los puntos x pertenecientes a $\mathbb{R}^2$ para los que p_i es el vecino más cercano.

$$Vor(p_i) = \left\{x \in \mathbb{R}^2 | d(x, p_i) \leqslant d(x, p_j), \forall j \neq i\right\} \tag{5}$$

La región *Vor(p_i)* es poligonal y convexa, y es acotada si y solo si p_i es interior a la envolvente convexa de P. Además, la región de p_i está compuesta por la intersección de $n-1$ semiplanos que conforman la región poligonal y está acotada como máximo por $n-1$ vértices y $n-1$ aristas.

Definición 7 *Se les denomina* ***aristas de Voronoi*** *a aquellos segmentos que contienen más de un punto y que son comunes para dos regiones de Voronoi. Los puntos finales de las aristas son los* ***vértices de Voronoi****. De esta manera, el Diagrama de Voronoi de P, Vor(P), está definido como la unión de todas las aristas a partir de los vértices de Voronoi.*

4.2 Construcción

Existen numerosos algoritmos que nos permiten crear las regiones de Voronoi. A continuación, vamos a explicar cómo se forman estas regiones de una manera clara y sencilla en honor a Euclides y aquello de que *el camino más corto entre dos puntos es la línea recta.*

Imaginemos que el ayuntamiento de Madrid ha presentado un proyecto en el que nos pide que realicemos un análisis sobre la proximidad de los hospitales de Madrid y su área de influencia, es decir, si hay un accidente en una ubicación determinada, ¿cuál es el hospital más cercano al que acudir? Pues bien, para ello empleamos el Diagrama de Voronoi.

En un barrio de Madrid, por ejemplo el barrio Salamanca, existen dos hospitales. En este caso, el barrio quedaría dividido en aquellas ubicaciones que están más próximas del hospital p, las que están más cerca de q y aquellas que se encuentran a la misma distancia de ambos hospitales. Estas divisiones formarían las regiones de Voronoi.

Para crear las regiones tenemos que, dados dos puntos, p y q, en un plano P, la mediatriz[3] al segmento pq divide el plano en dos regiones, $Vor(p)$ y $Vor(q)$; la región de p contiene todos los puntos

[3]Recta perpendicular a un segmento que se traza en su punto medio.

cuya distancia a p es menor que a q, mientras que la región de p contiene el resto.(Ver Figura 16).

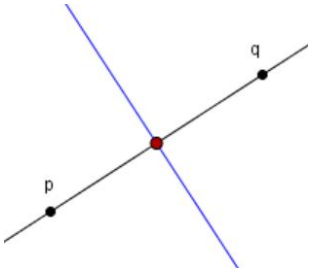


Figure 16: La línea azul (mediatriz) divide al barrio según la proximidad de las posibles ubicaciones de los accidentes a cada uno de los hospitales.

Ahora resulta que abren un nuevo hospital, d, por lo que, razonando de forma similar, nos quedaría una división de 3 regiones, cada una de ellas representa la región de Voronoi de cada hospital. Obtendríamos el Diagrama de Voronoi siguiendo el mismo método:

(1) Se calculan los segmentos correspondientes a cada uno de los puntos.
(2) Se colocan los puntos medios para posteriormente realizar las mediatrices de cada uno de los segmentos, lo que nos da como resultado el punto G, es decir, el circuncentro.
(3) Señalamos las mediatrices desde el circuncentro, que son las que realmente interesan.
(4) Finalmente, tenemos construido el Diagrama de Voronoi con cada una de las correspondientes regiones de Voronoi para los 3 hospitales del barrio (p, q, d).

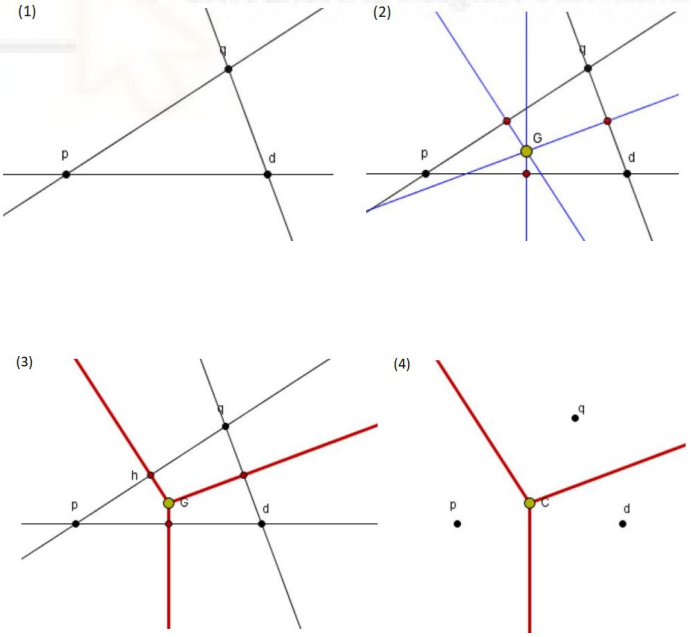


Figure 17: Formación del Diagrama de Voronoi con 3 puntos

4.2.1 Diagrama de Voronoi vs Triangulación de Delaunay

Como pudimos observar en el apartado 3.1.2 sobre el caso particular de la Triangulación de Delaunay, a través del método "Propiedades respecto a circunferencias" se construía la triangulación y, a continuación, el Diagrama de Voronoi. En este apartado vamos a demostrar cómo el circuncentro de la circunferencia de un triángulo, calculada en dicho apartado, coincide con el circuncentro creado mediante las mediatrices de los segmentos del triángulo.

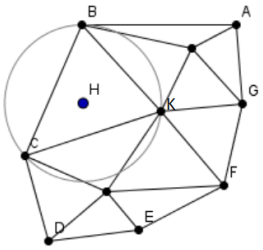


Figure 18: Circunferencia circunscrita y su circuncentro

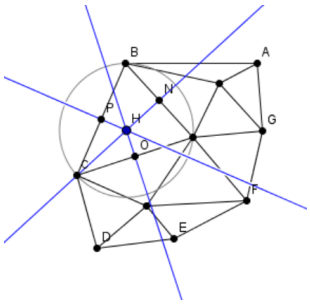


Figure 19: Circuncentro formado a partir de las mediatrices de los segmentos del triángulo

4.2.2 Algoritmos para la construcción del Diagrama de Voronoi

Existen diversos algoritmos debido a la gran cantidad de aplicaciones que éste posee y la influencia que han recibido de ello varios investigadores. Nosotros, para el proyecto que nos han pedido, decidimos implementar el Algoritmo Divide y Vencerás, aunque existen muchos otros como es el caso del Algoritmo de Fortune o el Algoritmo Incremental.

Divide y vencerás hace referencia a un refrán que interpreta la mejor resolución de un problema a través de la división de éste en partes más simples, tantas como sean necesarias, hasta que la solución de los subproblemas suponga la solución del problema original.
Este término es uno de los más importantes en el área de la computación y se le asigna a aquellos algoritmos que reducen cada problema a un único subproblema, como es nuestro caso.

Ahora el proyecto aborda también uno de los barrios contiguos al de Salamanca, el barrio de Tetuán. En este caso, tenemos más puntos por lo que es necesario implementar un algoritmo para generar el diagrama de forma óptima.

A continuación exponemos los pasos a seguir del algoritmo Divide y Vencerás para la construcción pertinente:

(1) Seleccionamos las coordenadas (latitud,longitud) de los seis hospitales y definimos dichos puntos en el plano. Los hospitales del barrio de Salamanca son los que teníamos anteriormente, p_1, q_1, d_1,

mientras que los del barrio de Tetuán son p_2, q_2 y d_2.

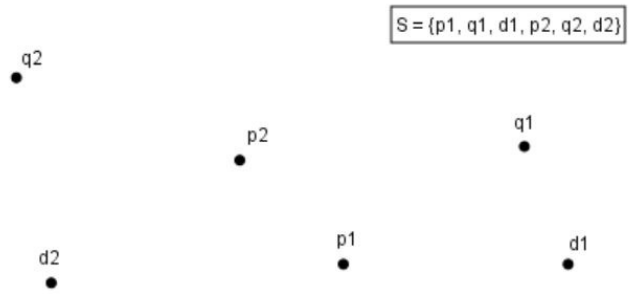


Figure 20: Área de Madrid en el que se encuentran los 6 hospitales

(2) Dividimos dichos puntos en dos conjuntos S_1,S_2 de aproximadamente el mismo tamaño a través de una línea vertical.

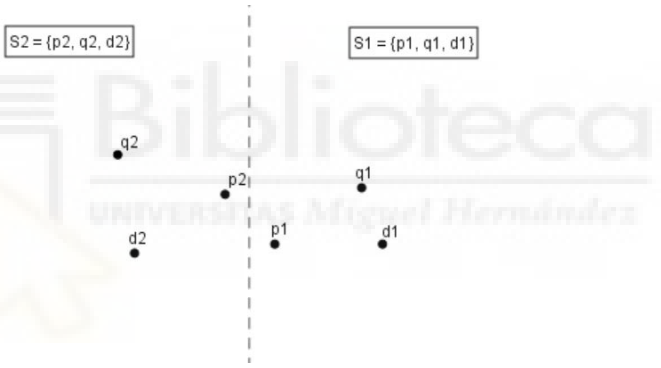


Figure 21: El conjunto S_1 pertenece al barrio de Salamanca mientras que S_2 pertenece al barrio de Tetuán

(3) Calculamos el diagrama de Voronoi de ambos conjuntos recursivamente, es decir, $Vor(S_1)$ y

$Vor(S_2)$, a través del método explicado anteriormente.

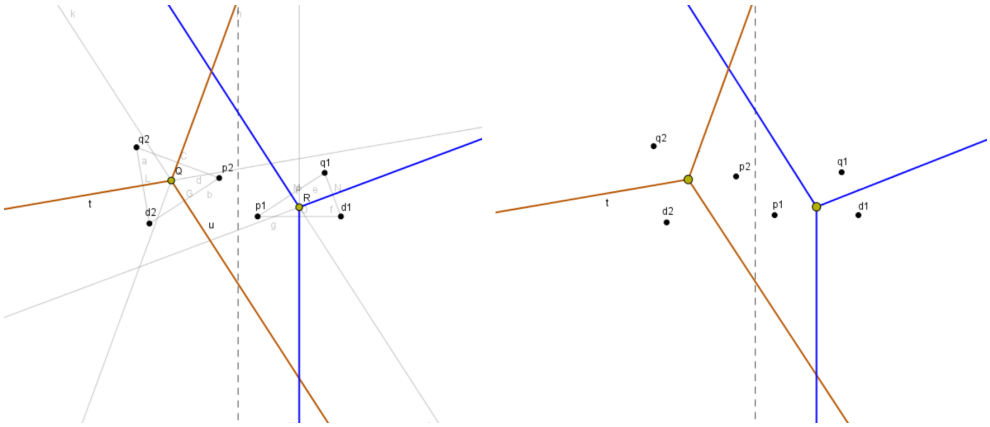


Figure 22: En cada barrio quedan definidas las regiones de cada uno de los hospitales

(4) Realizamos la envolvente convexa de ambos conjuntos y unimos, a través de segmentos, los puntos superiores e inferiores de sus respectivos cascos convexos. A continuación, realizamos la mediatriz del segmento que une los puntos superiores para comenzar así la cadena divisoria.

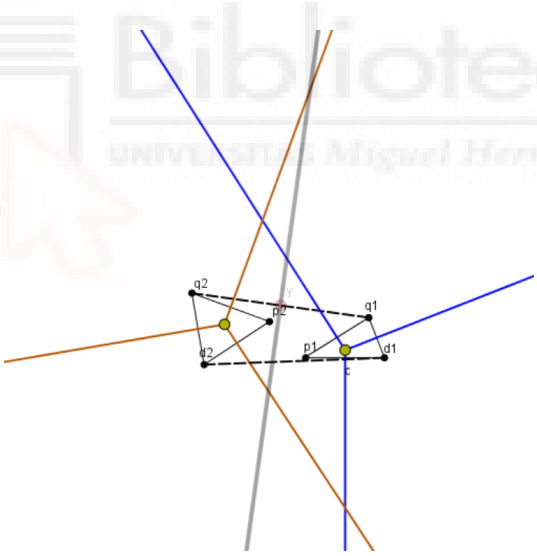


Figure 23: El segmento superior es q_2q_1 y el segmento inferior es d_2d_1. Realizamos la mediatriz del segmento superior q_2q_1.

(5) Caminamos por la mediatriz hasta intersectar con alguna mediatriz de $Vor(S_1)$ o $Vor(S_2)$, seleccionando dicho punto de corte como un nuevo nodo. A continuación, vemos a qué segmento

pertenece dicha mediatriz y actualizamos el punto del segmento superior.

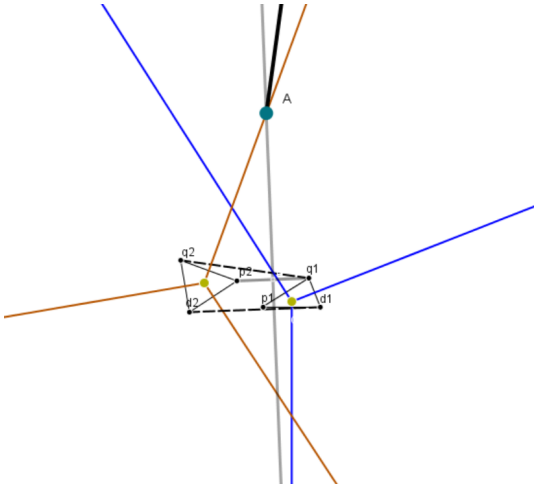


Figure 24: Intersección con la línea marrón, perteneciente a la mediatriz del segmento q_2p_2 por lo que seleccionamos dicho corte como el nodo A y, posteriormente, cambiamos q_2 por p_2 y realizamos la mediatriz del nuevo segmento p_2q_1

(6) Volvemos al paso 5 una vez actualizado el segmento.

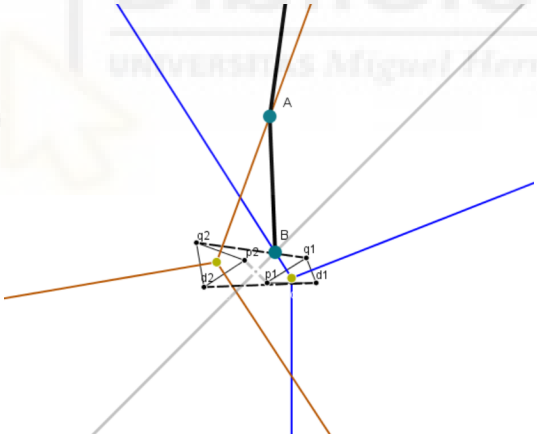


Figure 25: Intersección con la línea azul, perteneciente a la mediatriz del segmento q_1p_1 por lo que colocamos el nuevo nodo B y cambiamos q_1 por p_1 y realizamos la mediatriz del nuevo segmento p_2p_1

(7) Volvemos al paso 5 una vez actualizado el segmento.

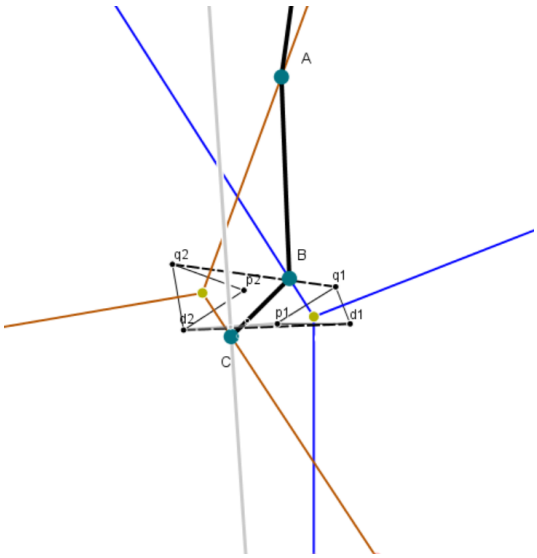


Figure 26: Intersección con la línea marrón de nuevo, perteneciente a la mediatriz del segmento p_2d_2 por lo que colocamos el nuevo nodo C y cambiamos p_2 por d_2 y realizamos la mediatriz del nuevo segmento d_2p_1

(8) Volvemos al paso 5 una vez actualizado el segmento.

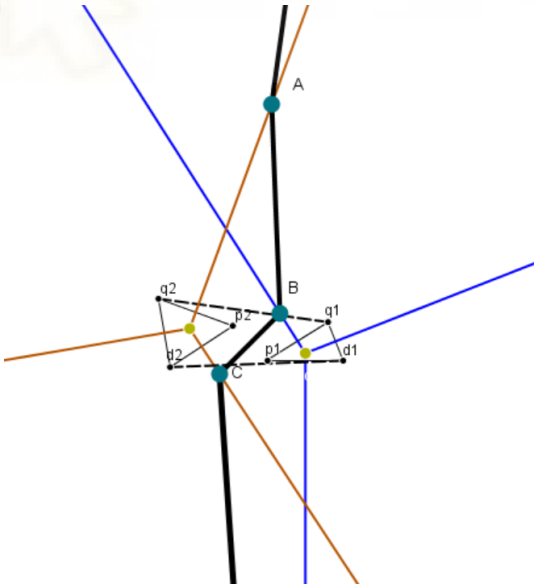


Figure 27: Intersección con la línea azul de nuevo, perteneciente a la mediatriz del segmento q_1p_1 por lo que colocamos el nuevo nodo D, que no se visualiza en la Figura debido a su lejanía, y cambiamos p_1 por d_1 y realizamos la mediatriz del nuevo y último segmento d_2d_1

(9) Terminado el Diagrama de Voronoi para los 6 hospitales, pasamos a ver el resultado final paso

por paso.

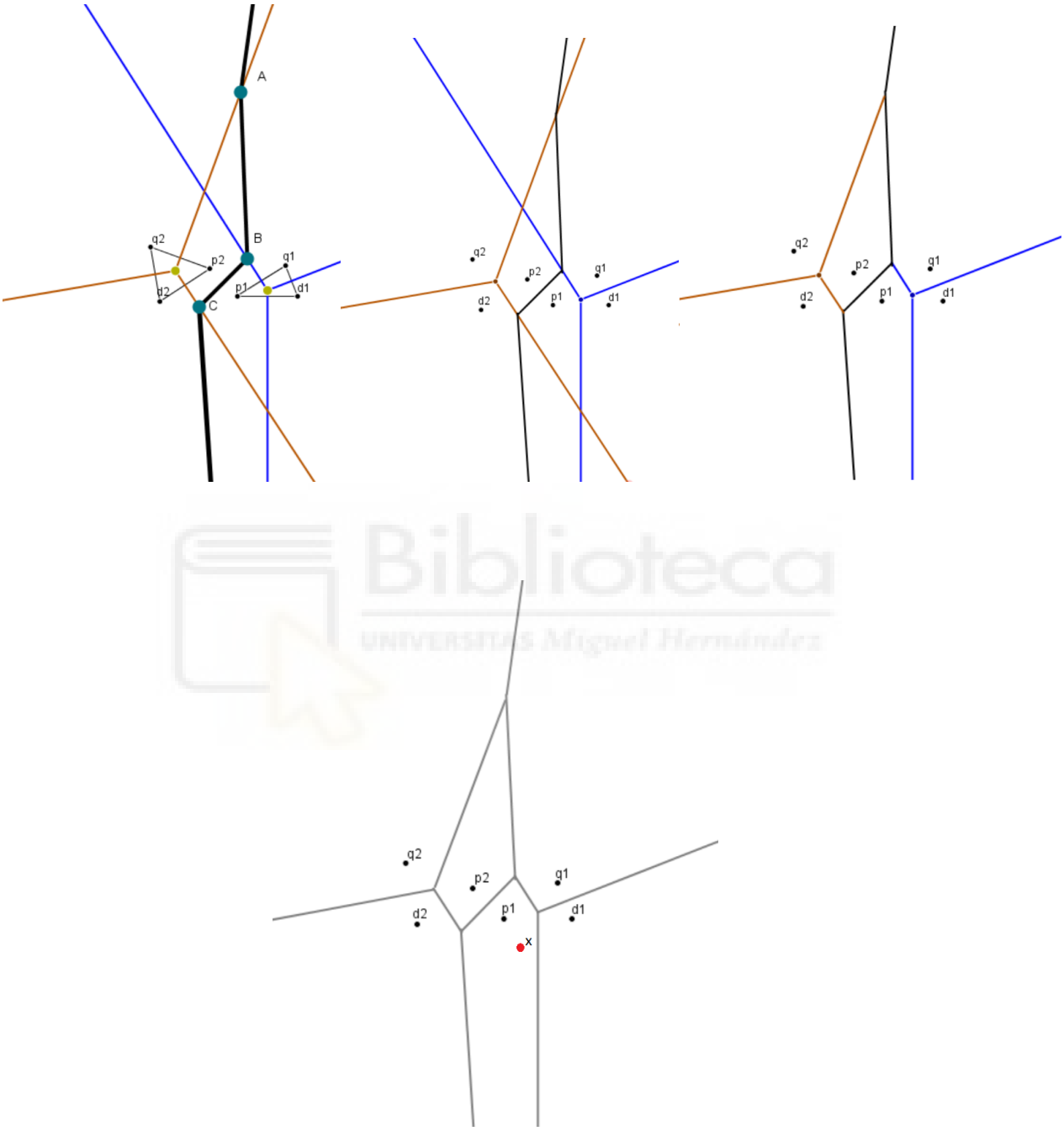


Figure 28: Construcción final del Diagrama de Voronoi en el que podemos ver las distintas regiones para cada hospital. Además, observamos que en el supuesto de que ocurra un accidente de tráfico en una ubicación del barrio de Salamanca, punto x, éste directamente será localizado en la región del hospital p_1 y el herido será trasladado a dicho hospital ya que es el más cercano.

Ésta es una de las muchas implementaciones que se pueden realizar del Diagrama de Voronoi y también, uno de los numerosos algoritmos existentes.

5 Aplicación a la disposición de los taxis en Elche

5.1 Introducción

El planteamiento del problema, como ya vimos en la *sección 3.2*, se basa en la disposición de los taxis en la ciudad de Elche, Alicante. La implementación de la app en Elche se ha efectuado ya que resulta casi imprescindible poder transladarte de un lugar a otro sin esperas para localizar el centro urbano, un museo de Elche, un parque cualquiera o a una calle de la ciudad. Esto es algo que se está implementando ya en las grandes provincias españolas pero no en las urbes, aunque la necesidad sea la misma.

Existen varias empresas que proporcionan vehículos de transporte con conductor a sus clientes para así conectar a los pasajeros con dichos conductores que están prestando el servicio. Con todo ello, lo que se pretende es proporcionar un servicio de transporte a particulares a través de un software de aplicación móvil. Dentro de este rango de estudio se encuentran diversas empresas como puede ser Mytaxi, Uber o Cabify, encontrando entre ellas una gran competencia.

En el caso de Uber, empresa internacional, debes descargarte la aplicación en primer lugar e ingresar tus datos personales así como la ubicación del móvil. Con ello, el cliente que quiere trasladarse a cualquier lugar lo que debe hacer es añadir la dirección exacta a la que ir; la aplicación instantáneamente realiza el recorrido del uber que esté disponible y sea el más cercano hasta tu punto de recogida, que es sabida de antemano gracias a que, en un principio, el cliente proporcionó su ubicación.

Como curiosidad, hoy en día la competencia con Mytaxi es intensa debido, en parte, a que Uber te da un precio exacto una vez que tiene calculado el recorrido mientras que Mytaxi te da un rango de precio estimado, el cual puede aumentar al final del recorrido.

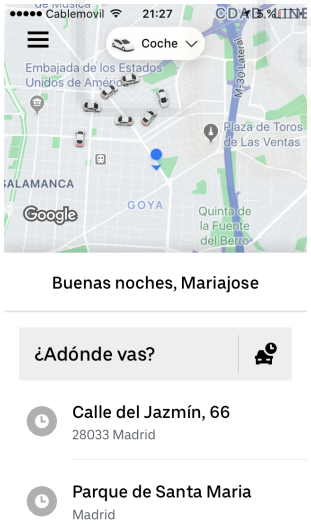


Figure 29: Aplicación de Uber donde aparecen los vehículos en el área donde estés establecido

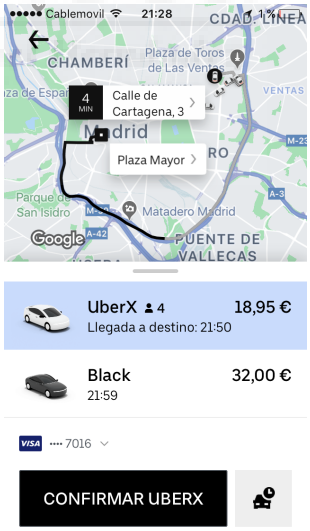


Figure 30: Desde tu ubicación (*C/Cartagena,3*) hasta el punto de llegada, *Plaza Mayor*, indica un total de 18,95€

5.2 Objetivo

Todas estas empresas de transporte trabajan según la disponibilidad y la proximidad. Ahora bien, cuando te proporcionan el vehículo, ¿Cómo podemos cerciorarnos de que realmente te recoge el más cercano, y, con ello, el más económico para ti como cliente?

Uno de los objetivos de nuestro proyecto va a ser implementar el Diagrama de Voronoi en la distribución de los taxis en la ciudad de Elche, en la cual no existen este tipo de empresas, suponiendo distancias euclídeas entre los puntos, para así poder establecer una correcta conexión conductor-pasajero en función de la proximidad de los vehículos a través del desarrollo de una aplicación.

Otro de los objetivos es: "minimizar el tiempo, maximizando el servicio". Con esta frase queremos decir que la gran mayoría de los clientes prefieren la facilidad y comodidad ante todo, y, con ello, el tiempo invertido es un componente que les afecta directamente. De esta manera, hemos querido desarrollar la app, no solo como un problema de proximidad, sino también como una posible mejora del servicio. Véamoslo con un ejemplo.

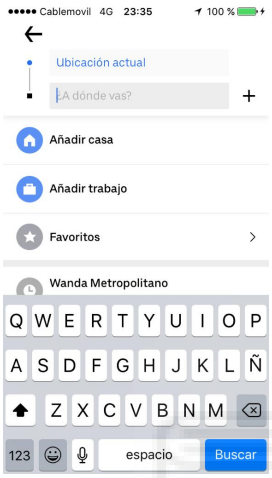


Figure 31: Insertamos el destino.

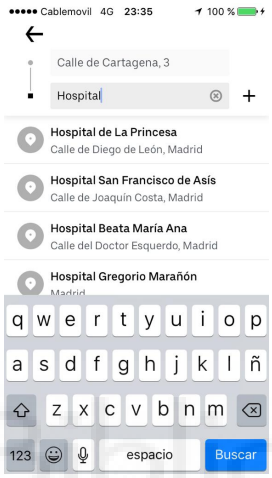


Figure 32: Te indican varios hospitales que se encuentran próximos a tu ubicación.

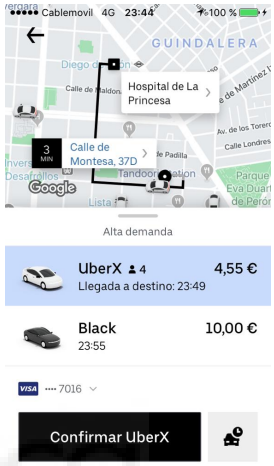


Figure 33: Selección del *Hospital de La Princesa* como punto de llegada y realización inmediata del recorrido.

En nuestro caso, lo que pretendemos es facilitar este método de localización mediane listados de gastronomía, ocio, entretenimiento, sanidad, educación y demás categorías para así obtener en el mapa de la ciudad de Elche los distintos hospitales, restaurantes, colegios, guarderías, etc... organizados de una manera clara y visible para poder seleccionar el destino de manera automática mediante un mapa interacivo. Todo ello lo podremos ver en los siguientes apartados.

5.3 Desarrollo del proyecto

Una vez definido el objetivo del proyecto, pasamos a la fase de desarrollo. En dicha fase, en primer lugar, lo que se ha realizado es un algoritmo que obtenga el Diagrama de Voronoi para unos puntos generados en un tiempo de ejecución óptimo. Para ello se ha empleado el lenguaje de programación R junto con la extensión RShiny, lo que nos ha permitido crear y visualizar el diagrama según unos puntos dados.

Los puntos generados para la ejecución del Diagrama de Voronoi se han obtenido mediante simulaciones de n números aleatorios , $Vor(P) = \{p_1,p_2,...,p_n\}$. Estos puntos aleatorios se encuentran limitados a través de un máximo y un mínimo de coordenadas que coinciden con la periferia de Elche.

El sistema de coordenadas geográficas es un sistema que referencia cualquier punto de la superficie terrestre y que utiliza para ello dos coordenadas angulares, latitud[4] y longitud[5], para determinar los

[4]La latitud proporciona la localización de un lugar, en dirección norte o sur desde el ecuador y se expresa en medidas angulares que varían desde los 0° del Ecuador hasta los 90°N (+90°) del polo Norte o los 90°S (-90°) del polo Sur.

[5]La longitud proporciona la localización de un lugar, en dirección Este u Oeste desde el meridiano de referencia 0°, o meridiano de Greenwich, expresándose en medidas angulares comprendidas desde los 0° hasta 180°E (+180°) y 180°W

ángulos laterales de la superficie terrestre con respecto al centro de la Tierra y alineadas con su eje de rotación.

Cabe destacar que, cuando se indican las coordenadas, se coloca primero la latitud y luego la longitud. Por ejemplo, Elche posee unas coordenadas de (38.269830, -0.710711), es decir, tiene una latitud de 38.269830 grados (norte) y una longitud de -0.710711 grados (oeste). Esto quiere decir que Elche se encuentra aproximadamente a 38° al norte del ecuador y a 71° al oeste del primer meridiano.

Véase en la siguiente Figura el mapa de Elche con las restricciones de la periferia a través de las longitudes y latitudes máximas y mínimas localizadas mediante Google Maps e implementadas en RShiny, cuyo código se encuentra en el apartado de *Anexos* referenciando la *Figura 34*.

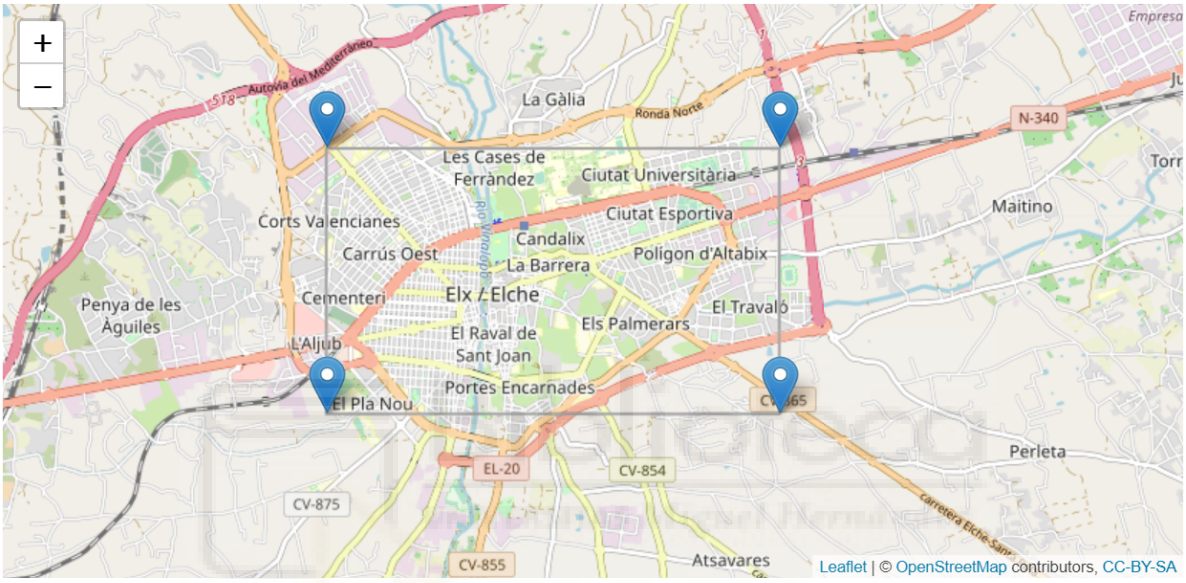


Figure 34: Área de Elche con la que vamos a trabajar

Una vez tenemos el área establecida en la que poder trabajar, vamos a exponer el algoritmo creado a partir del paquete de RStudio denominado **Deldir** basado en la triangulación de Delaunay y la Teselación de Dirichlet, nombradas anteriormente.

5.3.1 Librería Deldir

Lo que se pretende con esta librería es calcular la triangulación de Delaunay y de Dirichlet; recordemos que los polígonos de Thiessen también son conocidos como polígonos de Voronoi o teselación de Dirichlet, de un conjunto de puntos planares.

En el *apartado 4.2.1* veíamos que la triangulación de Delaunay es el dual del Diagrama de Voronoi, es decir, que a partir de uno se podía construir el otro. Por lo tanto, con esta función se calcula la triangulación de Delaunay y, con ello, el diagrama de Voronoi basándose en el segundo algoritmo (iterativo) de Lee and Schacter [3].

(-180°).

Uso

```
deldir (x, y, dpl = NULL, rw = NULL, eps = 1e-09, sort = TRUE, plotit = FALSE,
digits = 6, z = NULL, zdum = NULL, suppressMsge = FALSE, ...)
```

Los parámetros más importantes para luego poder entender el algoritmo creado son:

x, y --→ Especifican las coordenadas del conjunto de puntos, es decir, las longitudes y latitudes de cada uno de ellos.

rw --→ Las coordenadas de las esquinas de la ventana rectangular convexa que encierra el diagrama así como cualquier punto fuera de estos se descarta.

sort --→ Es un argumento lógico; si es "verdadero" los datos son ordenados antes de la visualización, lo que hace que el algoritmo sea un poco más eficiente.

plotit --→ Lo que realiza este parámetro lógico es que si es "verdadero", se dibuja la triangulación. Otra manera de dibujarlo es, posteriormente, con la función *plot.deldir*.

digits --→ El número de posiciones decimales de los valores de la lista debe ser redondeado. El valor predeterminado es 6.

wlines --→ Existe la opción de dibujar la triangulación, el diagrama de Voronoi o ambas. En nuestro objeto de estudio, seleccionaremos wlines = "tess", que proviene de teselación.

A continuación, vamos a ilustrar la función **deldir** mediante el ejemplo del *apartado 4.2.2* en el que construimos el diagrama mediante el método de Divide y Vencerás en un total de 9 pasos. Con ello, lo primero que hacemos es generarnos una base de datos con las latitudes y longitudes aproximadas de los puntos generados en la *Figura 20*.

La función en R es muy sencilla de ejecutar una vez tenemos claros los conceptos. En primer lugar, creamos las latitudes y longitudes de los 6 puntos y construimos la base de datos, *data*:

```
> latitudes<-c(38.263,38.262,38.273,38.271,38.261,38.275)
> longitudes<-c(-0.691,-0.667,-0.679,-0.699,-0.720,-0.724)
> data<-data.frame(latitudes,longitudes)
```

latitudes	**longitudes**
38.263	-0.691
38.262	-0.667
38.273	-0.679
38.271	-0.699
38.261	-0.720
38.275	-0.724

Table 1: Base de datos. Valores de las coordenadas generadas

Posteriormente, ejecutamos la librería en cuestión y su función. Cabe destacar que no hemos puesto el parámetro "rw" ya que, en este caso, no queremos acotar la región de Voronoi.

```
> library(deldir)
> voronoi<-deldir(x = data$longitudes, y = data$latitudes,  sort=TRUE, plotit=TRUE, digits=3,
+                 wlines='tess')
```

Lo que nos da como resultado es la representación de cada uno de los puntos en el plano, con sus

respectivas regiones de Voronoi como vemos en la siguiente Figura:

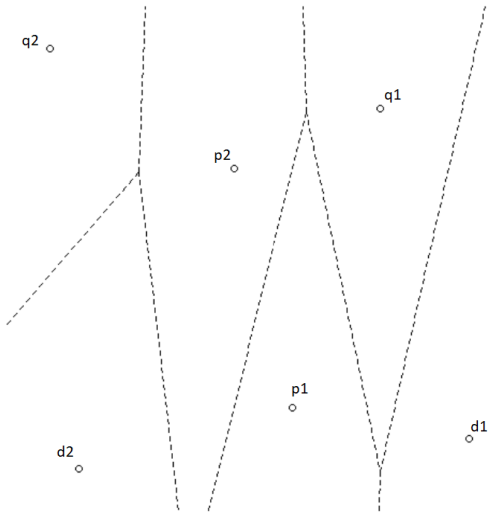


Figure 35: Ilustración de la función Deldir

```
> library(deldir)
> voronoi<-deldir(x = data$longitudes, y = data$latitudes,  sort=TRUE, plotit=TRUE, digits=3,
+                 wlines='tess')
```

Es interesante destacar que dicha representación del diagrama de Voronoi se puede ver de manera más clara y visual con ayuda del paquete de R denominado **ggplot2**, el cual permite crear elegantes visualizaciones de datos utilizando la gramática de los mismos.

```
> library(ggplot2)
> ggplot(data=data, aes(x=longitudes,y=latitudes)) +
+   #Plot the voronoi lines
+   geom_segment(
+     aes(x = x1, y = y1, xend = x2, yend = y2),
+     size = 2,
+     data = voronoi$dirsgs,
+     linetype = 1,
+     color= "grey") +
+   #Plot the points
+   geom_point(
+     fill=rgb(70,130,180,255,maxColorValue=255),
+     pch=21,
+     size = 2,
+     color="black")
```

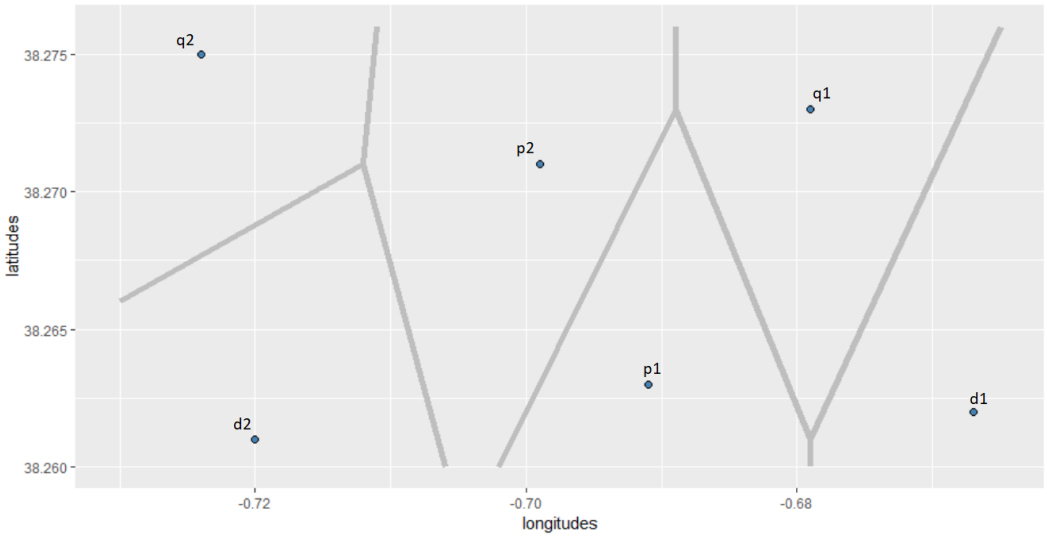


Figure 36: Ilustración de la función deldir entrelazada con ggplot

Con todo ello, lo que queremos abstraer es la idea de que, computacionalmente, se opera en un tiempo de ejecución óptimo, por lo que siempre se ha de llegar a un algoritmo adecuado para la compilación de cualquier problema. En nuestro caso, hemos llegado a la programación de dicho algoritmo aunque podríamos haber utilizado otro que nos proporcionase una solución similar.

5.4 App: *PortePlace*

Para realización de la aplicación hemos decidio otorgarle un nombre para nombrarla a lo largo de la explicación. Como curiosidad, **PortePlace** es una palabra inventada que proviene en primer lugar de "Porte", acción de transportar una mercancía de un lugar a otro, y "Place" que, en español, significa lugar. Así, al unir estas dos palabras nos queda un apelativo bastante concorde con el propósito de nuestro proyecto.

Para la correcta realización de la aplicación vamos a proveernos de diversos datos.

En primer lugar, poseemos una base de datos creada en Excel, sacada a mano, con las direcciones reales que nos ofrece Google Maps. Dicha base de datos se presenta de la siguiente manera:

	Category_Product	Sub_product	Product	Latitud	Longitud
2	Sanidad	Hospitales	Hospital de Elche	38,259565	-0,687113
3	Sanidad	Hospitales	Hospital General Universitario de Elche	38,260239	-0,68471
4	Sanidad	Hospitales	Psicotécnico Elche	38,270583	-0,682305
5	Sanidad	Residencias	Residencia Tercera Edad De Elche	38,27208	-0,688904
6	Sanidad	Residencias	Centro Residencial Elche Seniors	38,267768	-0,670195
7	Entretenimiento	Centros comerciales	CC Aljub	38,263731	-0,7213
8	Entretenimiento	Parques	Parque municipal	38,270615	-0,698067
9	Entretenimiento	Parques	Parque de los Peces	38,272821	-0,698345
10	Entretenimiento	Parques	Multiaventura Elche	38,263119	-0,668908
11	Entretenimiento	Museos	Museo del Belén	38,265657	-0,698783
12	Entretenimiento	Museos	Museo Arqueológico y de Historia de Elche	38,268285	-0,697602
13	Gastronomía	Restaurantes	Bar Restaurante Lincoln 7	38,265707	-0,696808
14	Gastronomía	Restaurantes	Restaurante Mesón El Granaino	38,267291	-0,70477
15	Gastronomía	Restaurantes	Restaurante Frisone	38,262624	-0,696809
16	Gastronomía	Supermercados	Hiperber	38,265286	-0,703353
17	Gastronomía	Supermercados	Carrefour Express	38,265741	-0,695027
18	Gastronomía	Supermercados	Dialprix	38,267207	-0,70213
19	Educación	Universidades	Universidad Miguel Hernández	38,276919	-0,689478
20	Educación	Universidades	UNED Centro Asociado Elche	38,270703	-0,693261
21	Educación	Colegios	Colegio Público Virgen de la Asunción	38,265202	-0,707841
22	Educación	Colegios	Colegio Público Ausías March	38,276119	-0,702434
23	Educación	Colegios	Escola Eugeni d'Ors	38,278747	-0,711919
24	Educación	Colegios	Colegio Público Ramón Llull	38,264427	-0,713506
25	Educación	Academias	Academia Eliot	38,269109	-0,695139
26	Educación	Academias	Academia Mega	38,26766	-0,691877
27	Educación	Academias	Centro de Estudios 10	38,268621	-0,685351

Figure 37: Coordenadas de los lugares seleccionados para la base de datos

Como podemos observar en la *Figura 37*, tenemos un total de 5 variables que pasamos a resumir a continuación:

1. **Category_Product**: diferentes categorías en las que se engloban los lugares de destino.
2. **Sub_Product**: establecimientos según categoría.

Sanidad { - Hospitales; - Residencias }

Entretenimiento { - Centros comerciales; - Parques; - Museos }

Gastronomía { - Restaurantes; - Supermercados }

Educación { - Universidades; - Colegios; - Academias }

3. **Product**: destinos finales según categoría y subcategoría.

   En **PortePlace** aparece el objeto de la siguiente figura en el cual podemos seleccionar el destino final o los distintos lugares que hay en la base de datos a través de las listas desplegables.

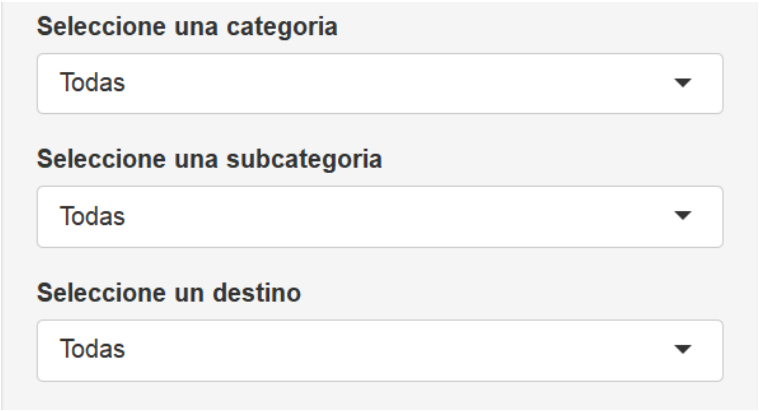


Figure 38: Selección del punto de destino a través de filtros

4. **Latitud y Longitud**: coordenadas geográficas para cada uno de los lugares.

En segundo lugar, es necesario establecer un origen, es decir, la ubicación de la persona que contrata el servicio a través de un input formado por la latitud y la longitud; la cual se modificará en función de donde se establezca la ubicación real del cliente. Cabe destacar que, en un principio, es decir, cuando arranca la app, el punto predeterminado de origen será aquel que determine las coordenadas geográficas de la ciudad de Elche, (38.26656,-0.7010700); esto ya lo hemos visto en el apartado *5.3* , cuando hablábamos del área de Elche en la que vamos a trabajar. Esto lo podríamos comprobar tanto en la *Figura 39* como si nos vamos al apartado de *Anexos*, donde se referencia la app *PortePlace* en el archivo *ui.R*.

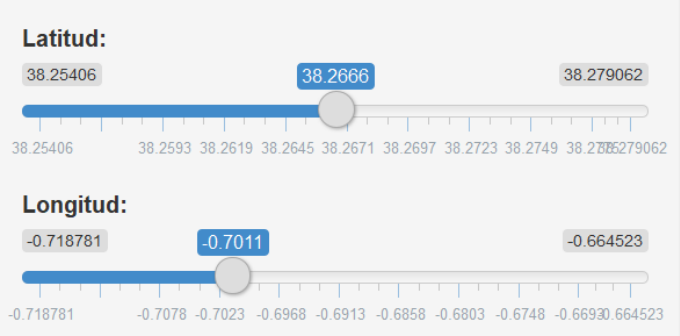


Figure 39: Coordenadas geográficas que representan el origen

Por último, va a ser necesario simular una serie de puntos, los taxis de Elche, a través de números aleatorios limitados por el área de trabajo que establecimos ya al principio del apartado. A continuación aparece el código implementado en el que vemos la función *runif*, utilizada para la ejecución de los números aleatorios. Un ejemplo de los puntos aleatorios que podrían surgir en el mapa es el siguiente:

```
> #Latitudes y longitudes máximas y mínimas para limitar el área de Elche
> maxLat<- 38.279062
> minLng<- -0.718781
> minLat<- 38.254060
> maxLng<- -0.664523
> #Construcción de las coordenadas aleatorias de los 10 taxis,que se encuentran
> #entre los valores anteriores
> latitudes<-runif(10,max = maxLat, min = minLat);latitudes

 [1] 38.25467 38.27057 38.26775 38.27580 38.25929 38.26235 38.27546 38.26565
 [9] 38.27109 38.26604

> longitudes<-runif(10,max = maxLng, min = minLng);longitudes

 [1] -0.6707658 -0.7163981 -0.7151995 -0.6975717 -0.7076466 -0.6935384
 [7] -0.7120829 -0.6704664 -0.7054551 -0.7157903
```

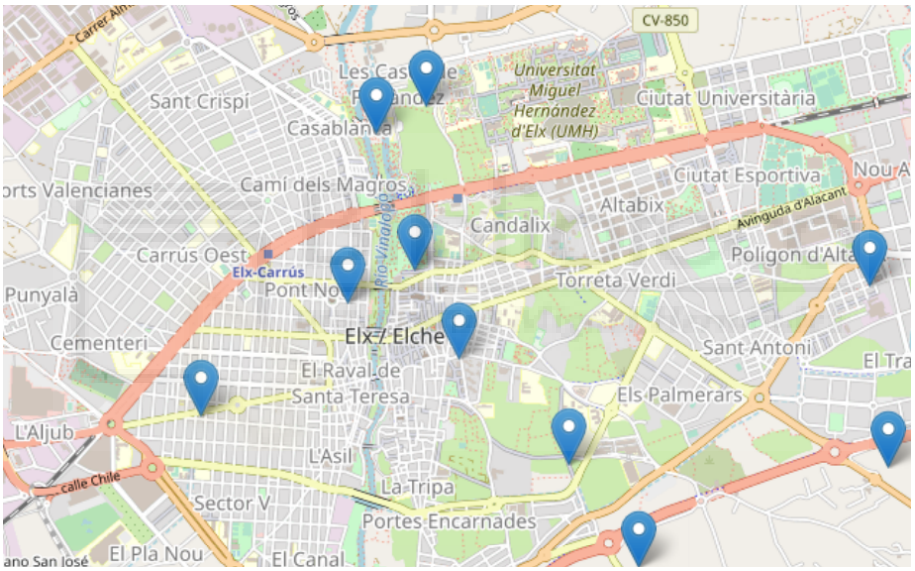


Figure 40: Simulación de los taxis distribuidos en la ciudad de Elche

5.4.1 Funcionalidad de la aplicación

La visualización de la app queda dividida en cada uno de las figuras anteriores que, conjuntamente estructuradas, forman una aplicación interactiva y fácil de interpretar.

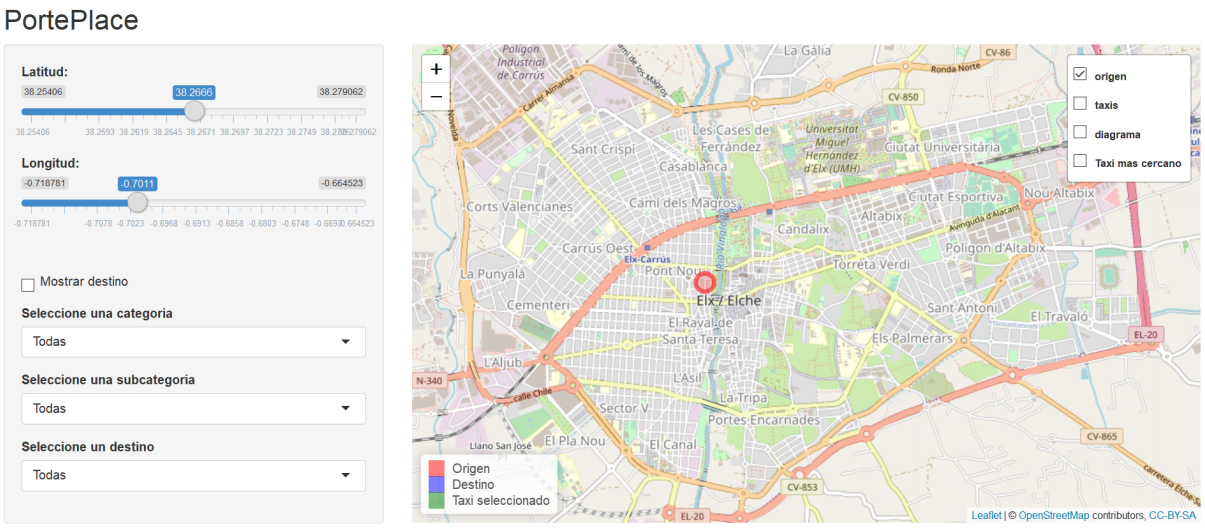


Figure 41: Aplicación PortePlace

Al ejecutar **PortePlace** nos aparece, como podemos ver en la *Figura 41*, el origen correspondiente a las coordenadas de Elche, que quedan predeterminadas, aunque el cliente, en su caso, podrá determinar cual es su verdadera localización o punto de partida.

En el ejemplo, hemos seleccionado como origen las coordenadas (38.2763,-0.6893), correspondientes a la ubicación donde se encuentra la universidad Miguel Hernández de Elche.

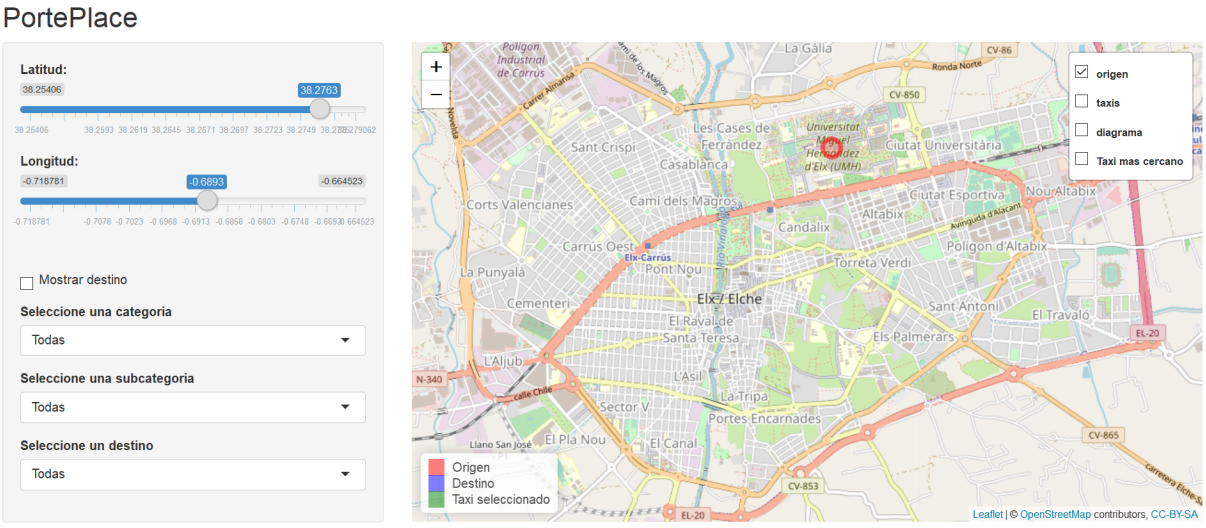


Figure 42: Punto de partida u origen: Universidad Miguel Hernández

Después de establecer el origen, el cliente pasaría a determinar su punto de llegada. Para ello solo tendrá que filtrar por categoría, subcategoría y, finalmente, elegir su destino final. Además, existe una casilla de verificación que, cuando se selecciona, permite visualizar cada uno de los posibles lugares filtrados. En la figura podemos ver que, en un primer momento, nos aparecen todos los hospitales que se encuentran disponibles y en la siguiente figura aparece seleccionado el Hospital Universitario de Elche como destino final.

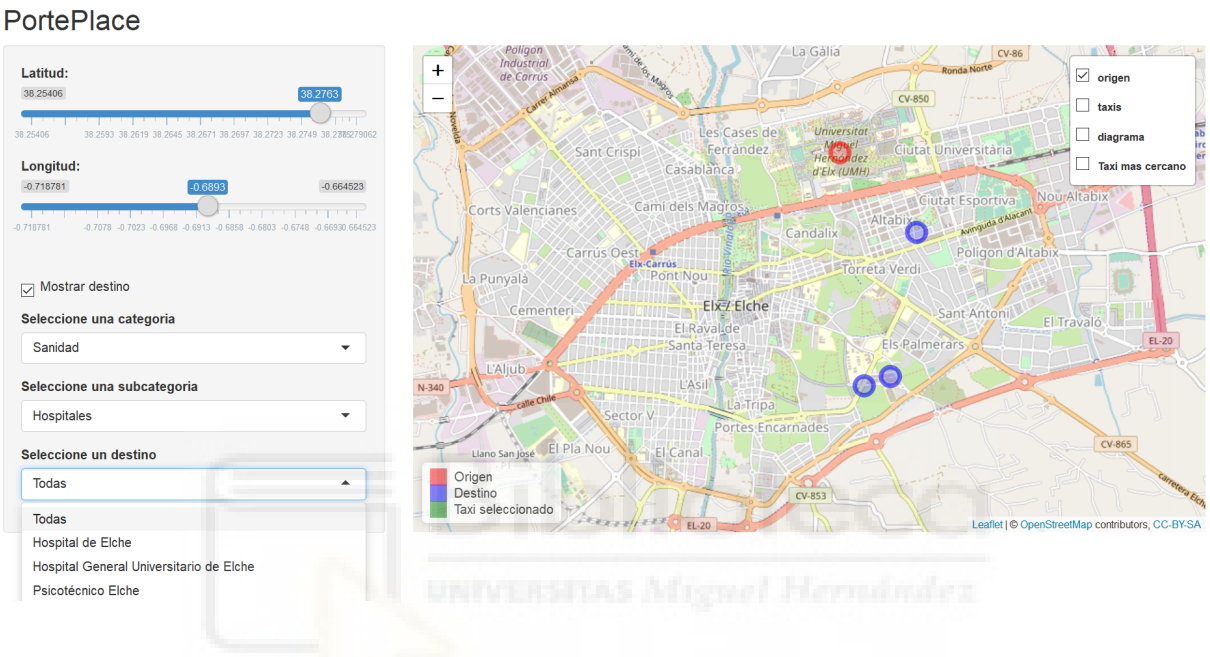


Figure 43: Lugares filtrados y visualizados en el mapa

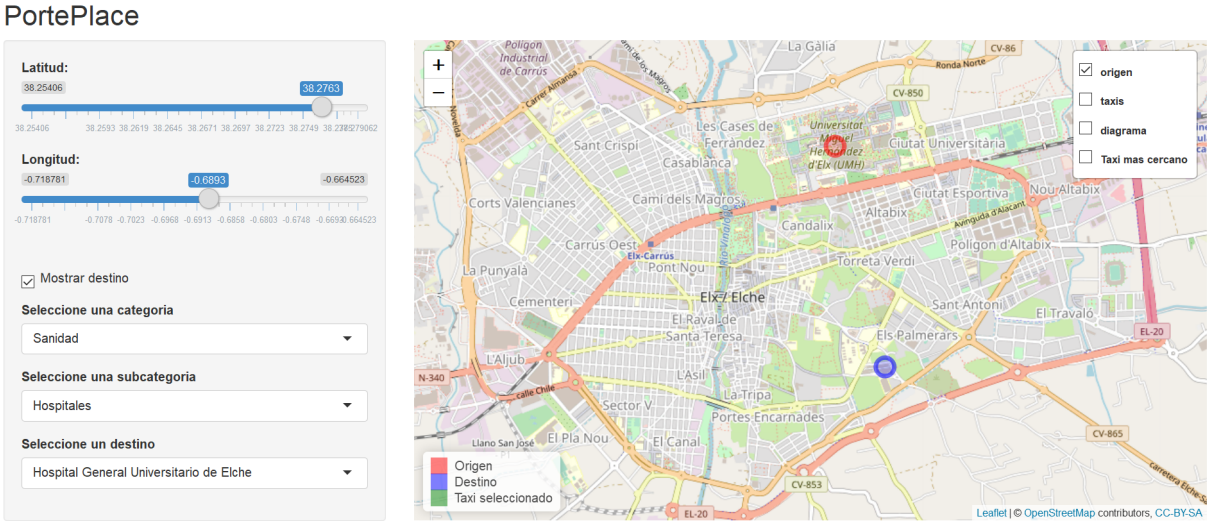


Figure 44: Destino final seleccionado por el cliente

Una vez seleccionados el origen y el desino del cliente, procedemos a insertar la ubicación de los taxis que se encuentran en dicho momento en servicio. Como ya hemos dicho anteriormente, estos taxis son simulados y van modificando su ubicación a medida que cambiamos los filtros citados. De esta manera, entendemos que, a medida que el cliente cambie las opciones del destino final, los taxis estarán en continuo movimiento y su posición irá variando.

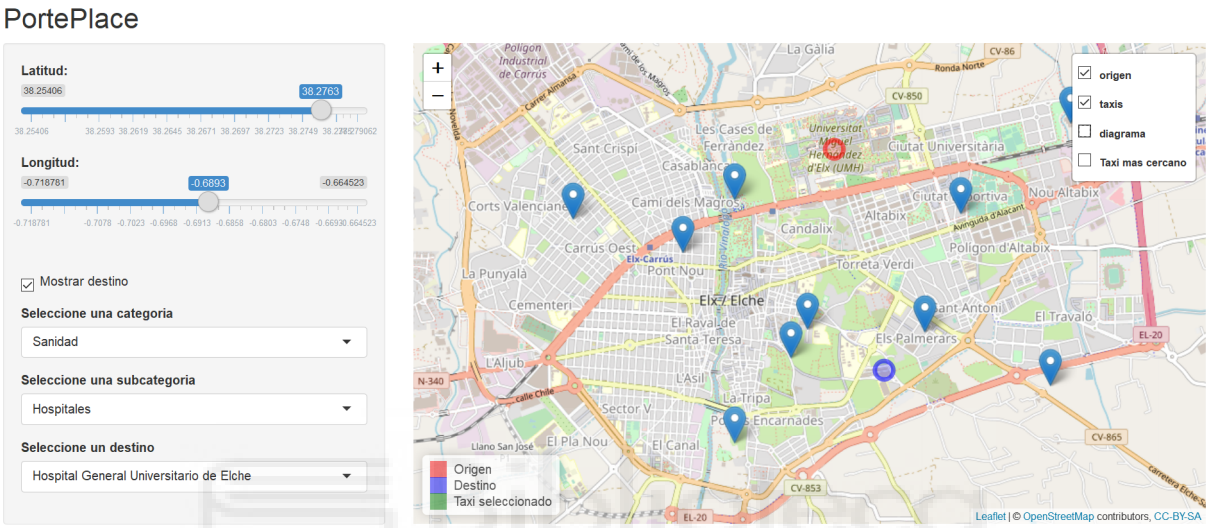


Figure 45: Ubicación de los taxis en el momento de la elección del destino final

Ya tenemos la disposición de los taxis a la hora de pedir el servicio, pero... ¿cómo podemos saber cuál es el taxi más cercano al origen? Para responder a esta pregunta recurrimos a los Diagramas de Voronoi, los cuales me proporcionan una descomposición de la ciudad en regiones, de manera que a cada taxi se le asigna una región formada por las ubicaciones que están más cerca de él que de ninguno de los otros taxis. Cabe destacar que suponemos distancias euclídeas para el manejo de los diagramas de Voronoi, la función aparece en el apartado de *Anexos*, en la archivo *load.R* de la App PortePlace.

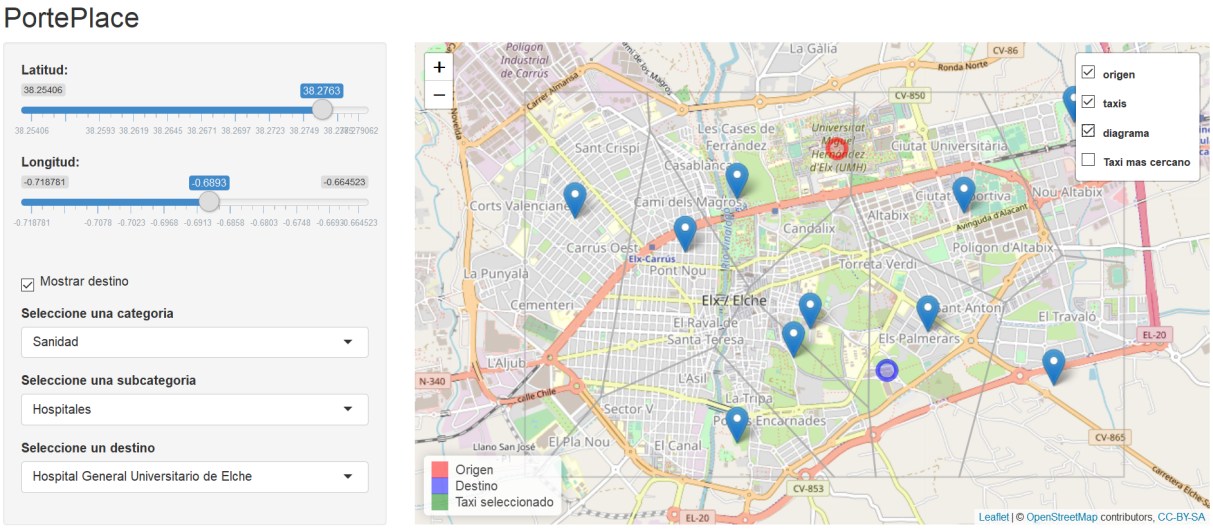


Figure 46: Diagrama de Voronoi en el espacio métrico

Ahora bien, visualmente podemos comprobar cuál sería el taxi asignado al cliente para realizar el servicio ya que su origen se encuentra dentro de la región de Voronoi de dicho taxi. En la *Figura 47* podemos ver el taxi asignado, y, lo que es lo mismo, el taxi más cercano a la ubicación del cliente, con un punto verde.

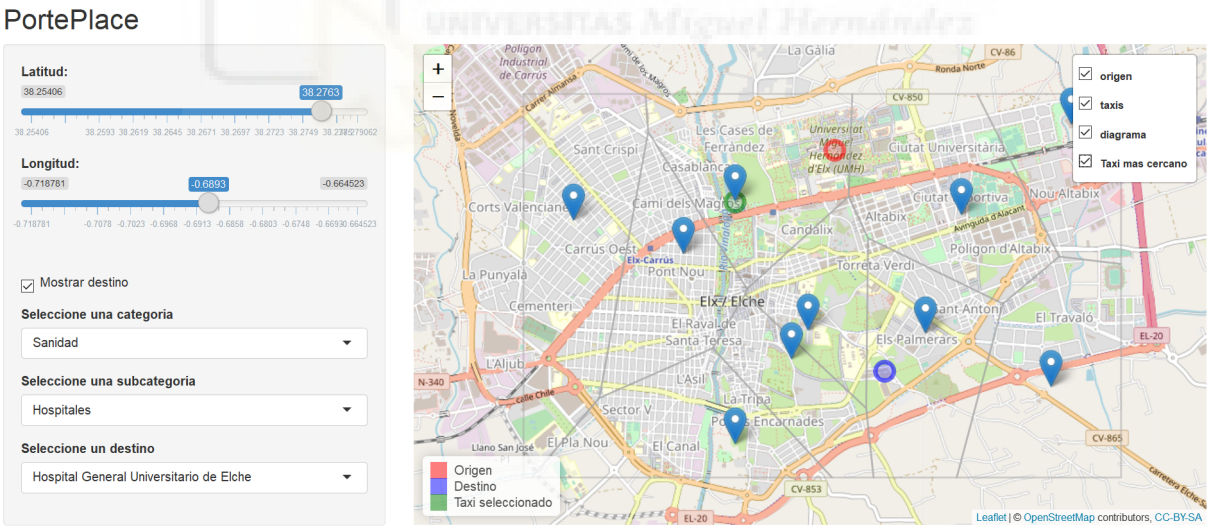


Figure 47: Asignación final del taxi

En definitiva, con esta herramienta podemos confirmar que se han desarrollado correctamente los objetivos establecidos en un principio. Además, esta app aparece como una idea novedosa para aquellos desarrolladores que se integran en el sector de la geometría computacional.

6 Conclusiones

Al concluir esta memoria, podemos observar que se ha llevado a cabo una investigación profunda sobre los Diagramas de Voronoi, además de incluir demostraciones y relacionar los métodos de la Geometría Computacional con dichos diagramas.

Por otro lado, se ha creado un algoritmo que permite realizar la construcción del diagrama de Voronoi suponiendo distancias euclídeas para unos puntos generados con ayuda de la librería *deldir* de R. En dicha función podemos ver que la Triangulación de Delaunay es el dual del Diagrama de Voronoi.

Por último, se ha conseguido implementar los diagramas mediante una aplicación interactiva realizada mediante *RShiny*. En esta aplicación, denominada PortePlace, se ha logrado adaptar los diagramas de Voronoi a un caso real, iniciar una propuesta para facilitar a los ilicitanos y demás personas que se establezcan en Elche los desplazamientos a zonas concretas de la ciudad y, también, es una manera de solucionar la incertidumbre sobre si las empresas que prestan este tipo de servicio están siendo realmente justas con los clientes.

En definitiva, vemos que durante todo el trabajo se pretende informar al receptor sobre los métodos más próximos a los Diagramas de Voronoi, además de disponer de aclaraciones visuales.

7 Investigaciones adicionales

¿Es realmente adaptable la app descrita en el trabajo con la vida real? ¿Se podría construir un diagrama de Voronoi que se basara en medir distancias entre puntos acarreando interferencias como puede ser una variable que te indique el tráfico aproximado en la zona afectada o los kilómetros a efectuar?

Estas son preguntas que nos surgen una vez terminado el proyecto sobres los Diagramas de Voronoi y su implantación en el área de Elche como una app turística conectada con el transporte; en este caso, los vehículos que sirven servicios de transporte urbano. Con ello, se podría mejorar en gran medida la estimación de los diagramas de Voronoi a escala y tiempo real.

Para consultar información sobre dicha idea podemos entrar en la Plataforma de Google Maps, en el apartado de "APIS de geolocalización", mediante la cual se puede obtener una gran cantidad de información válida. [4] También se pueden calcular las distancias entre puntos a través del propio Google Maps [5]; a continuación vemos un ejemplo sencillo:

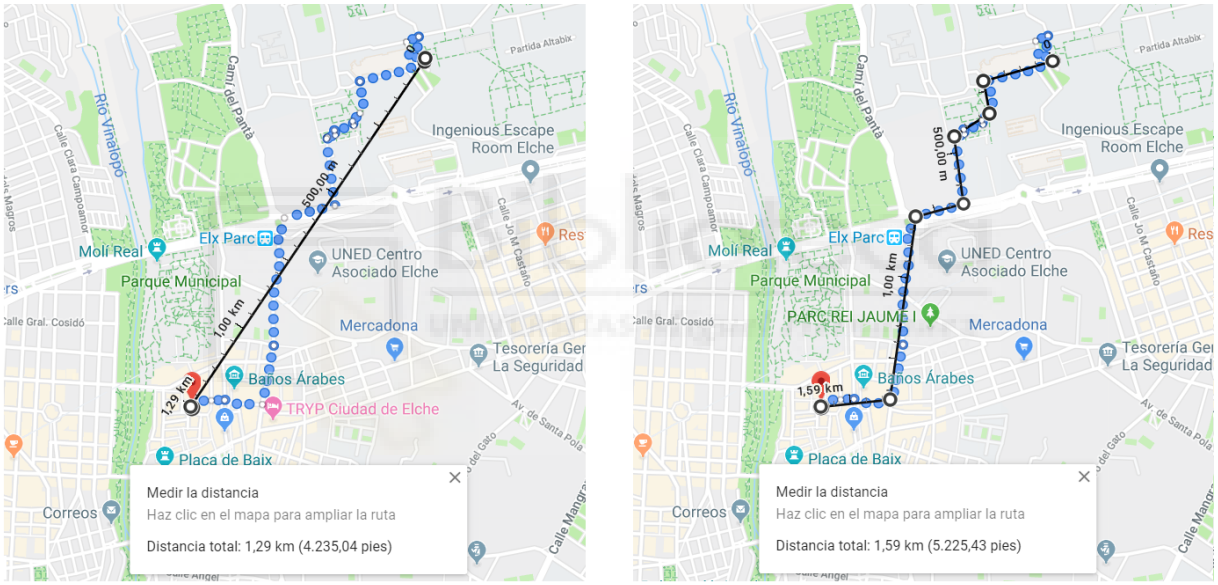


Figure 48: Medir distancias entre puntos (Km)

Referencias

[1] The Voronoi Web Site, *http://www.voronoi.com*

[2] Computational Geometry: An Introduction, *Preparata, F.P. and Shamos, M.I.*, 1985

[3] Two Algorithms for Constructing Delaunay Triangulations. *D. T. Lee, B. Schachter*, 1980

[4] APIs de geolocalización/Google Maps Platform/Google Cloud, *https://cloud.google.com/maps-platform/?hl=es*

[5] Google Maps, *https://www.google.com/maps*

[6] Computational Geometry: Algorithms and Applications, *M.de Berg, M. Van Kreveld, M. Overmars, O.Schwarzkopf*

[7] Voronoi Diagrams and Delaunay Triangulations, *Franz Aurenhammer, Rolf Klein, Der-Tsai Lee*

[8] Euclidean Distance, *https://www.cut-the-knot.org/pythagoras/DistanceFormula.shtml*

[9] Convex hull, *http://geomalgorithms.com/a10-$_h$ull − 1.html*

[10] Método Divide y Vencerás, *https://www.youtube.com/watch?v=sqsDN7aql1Et=1304s*

[11] Data mining, *https://mnrva.io/kdd-platform.html*

[12] LaTeX, *http://metodos.fam.cie.uva.es/ latex/curso-2015/apuntes3.pdf*

[13] RShiny, *https://rpubs.com/moviedo/shiny*

8 Anexos

Figura 31. **Código del mapa de Elche en RShiny**

1. Archivo *ui.R*

```
> library(shiny)
> library(sp)
> library(deldir)
> library(leaflet)
> shinyUI(fluidPage(
+
+   titlePanel("Voronoi Map"),
+   leafletOutput("ElcheMap")
+
+ ))
```

2. Archivo *server.R*

```
> library(shiny)
> #Mapa de Elche
> shinyServer(function(input, output) {
+
+   output$ElcheMap <- renderLeaflet({
+
+     #Longitud y latitud superior izquierda
+     LatSupIzq <- 38.279062
+     LongSupIzq <- -0.718781
+
+     #Longitud y latitud superior derecha
+     LatSupDer <- 38.279062
+     LongSupDer <- -0.664523
+
+     #Longitud y latitud inferior izquierda
+     LatInfIzq <- 38.254060
+     LongInfIzq <- -0.718781
+
+     #Longitud y latitud superior derecha
+     LatInfDer <- 38.254060
+     LongInfDer <- -0.664523
+
+     latView<-mean(c(LatSupIzq,LatInfIzq))
+     lngView<-mean(c(LongSupIzq,LongInfDer))
+
+     latitudes<-c(LatSupIzq,LatSupDer,LatInfIzq,LatInfDer)
+     longitudes<-c(LongSupIzq,LongSupDer,LongInfIzq,LongInfDer)
+
+     datos<-data.frame(cbind(latitudes,longitudes))
+
+     #Obtener las coodernadas en el mapa
+     vor_pts <- SpatialPointsDataFrame(cbind(datos$longitudes,
+                                             datos$latitudes),
+                                       datos, match.ID=TRUE)
+
+     leaflet() %>% setView(lngView,latView, zoom = 13)%>%addTiles()%>%
```

```
+       addMarkers(data = datos,lng=~longitudes, lat=~latitudes)%>%
+       addRectangles(lng1 = LongSupIzq, lat1 = LatSupIzq,lng2 = LongSupDer,lat2 = LatSupDer,
+                     color = "#a5a5a5",weight=2,opacity=1)%>%
+       addRectangles(lng1 = LongSupDer, lat1 = LatSupDer,lng2 = LongInfDer,lat2 = LatInfDer,
+                     color = "#a5a5a5",weight=2,opacity=1)%>%
+       addRectangles(lng1 = LongInfIzq, lat1 = LatInfIzq,lng2 = LongSupIzq,lat2 = LatSupIzq,
+                     color = "#a5a5a5",weight=2,opacity=1)%>%
+       addRectangles(lng1 = LongInfIzq, lat1 = LatInfIzq,lng2 = LongInfDer,lat2 = LatInfDer,
+                     color = "#a5a5a5",weight=2,opacity=1)
+   })
+ })
```

App PortePlace

1. Archivo *load.R*

```
> # library(readxl)
> # library(data.table)
> # product_list<-read_excel("C:/Users/Jesus Carrion/Desktop/DiagramasVoronoi/datos_TFG.xlsx")
> # setDT(product_list)
> #
> # maxLat<- 38.279062
> # minLng<- -0.718781
> # minLat<- 38.254060
> # maxLng<- -0.664523
> #
> # latView<-mean(c(maxLat,minLat))
> # lngView<-mean(c(maxLng,minLng))
> #
> # distanciaEuclidea<-function(x,lat,lng){
> #   setDT(x)
> #   x[,Distancia:=sqrt((lat-latitudes)^2+(lng-longitudes)^2)]
> #   x<-x[Distancia==min(Distancia)]
> #   return(x)
> # }
```

2. Archivo *ui.R*

```
> # library(data.table)
> # library(readxl)
> # library(shiny)
> # library(sp)
> # library(deldir)
> # library(leaflet)
> # source("load.R")
> #
> #
> # shinyUI(fluidPage(
> #
> #   titlePanel("PortePlace"),
> #
> #   sidebarLayout(
> #     sidebarPanel(
> #       verticalLayout( sliderInput("lat",
> #                                   "Latitud:",
```

```
> #                                   min = 38.254060,
> #                                   max = 38.279062,
> #                                   value = 38.26656),
> #                       sliderInput("lng",
> #                                   "Longitud:",
> #                                   min = -0.718781,
> #
> #                                         max = -0.664523,
> #                                   value = -0.7010700),
> #                       br(),
> #                       checkboxInput("checkbox",label="Mostrar destino",value = F),
> #                       selectInput("mainproduct","Seleccione una categoria",
> #                                   choices = c("Todas",unique(product_list$Category_Product))),
> #                       selectInput("subproduct","Seleccione una subcategoria",
> #                                   choices = NULL),
> #                       selectInput("product","Seleccione un destino",
> #                                   choices = NULL),
> #                       tags$head(
> #                         tags$style(HTML('#run{background-color:#79a6d2}'))
> #                       )
> #       )
> #       ),
> #
> #
> #     # Show a plot of the generated distribution
> #     mainPanel(
> #        leafletOutput("voronoiMap",height = 520)
> #     )
> #   )
> # ))
```

3. Archivo *server.R*

```
> # library(shiny)
> # library(dplyr)
> # # library(readxl)
> # # product_list<-read_excel("pruebashiny_Excel.xlsx")
> #
> # shinyServer(function(session,input, output) {
> #   data<-reactiveValues(product_list=product_list)
> #   observeEvent(input$mainproduct,{
> #     if(input$mainproduct=="Todas"){
> #       x<-product_list$Sub_product
> #     }else{
> #       x<-product_list[Category_Product == input$mainproduct]$Sub_product
> #     }
> #     updateSelectInput(session,"subproduct","Seleccione una subcategoria",
> #     choices = c("Todas",unique(x)))
> #   })
> #
> #   observeEvent({input$subproduct
> #     input$mainproduct},{
> #     if(input$mainproduct=="Todas"){
> #       aux2<-product_list
> #     }else{
```

```
> #       aux2<-product_list %>% filter(Category_Product == input$mainproduct)
> #     }
> #     if(input$subproduct=="Todas"){
> #       productdata<-aux2$Product
> #     }else{
> #       productdata<-aux2$Product[aux2$Sub_product == input$subproduct]
> #     }
> #     updateSelectInput(session,"product","Seleccione un destino",
> #     choices = c("Todas",productdata))
> #   })
> #
> #   observeEvent({input$mainproduct
> #                 input$subproduct
> #                 input$product},{
> #     aux<-product_list
> #     if(input$mainproduct=="Todas"){
> #       aux<-product_list
> #     }else{
> #       aux<-product_list[Category_Product == input$mainproduct,]
> #     }
> #     if(input$subproduct=="Todas"){
> #       aux<-aux
> #     }else{
> #       aux<-aux[Sub_product == input$subproduct,]
> #     }
> #     if(input$product=="Todas"){
> #       aux<-aux
> #     }else{
> #       aux<-aux[aux$Product == input$product,]
> #     }
> #
> #     data$product_list<-aux
> #   })
> #   output$voronoiMap <- renderLeaflet({
> #
> #     latitudes<-runif(10,max = maxLat, min = minLat)
> #     longitudes<-runif(10,max = maxLng, min = minLng)
> #
> #     datos<-data.frame(cbind(latitudes,longitudes))
> #
> #     vor_pts <- SpatialPointsDataFrame(cbind(datos$longitudes,
> #                                             datos$latitudes),
> #                                       datos, match.ID=TRUE)
> #
> #     SPointsDF_to_voronoi_SPolysDF <- function(sp) {
> #
> #       vor_desc <- tile.list(deldir(sp@coords[,1], sp@coords[,2]))
> #
> #       lapply(1:(length(vor_desc)), function(i) {
> #
> #         tmp <- cbind(vor_desc[[i]]$x, vor_desc[[i]]$y)
> #         tmp <- rbind(tmp, tmp[1,])
> #
> #         Polygons(list(Polygon(tmp)), ID=i)
```

```
> #
> #       }) -> vor_polygons
> #
> #       sp_dat <- sp@data
> #
> #       rownames(sp_dat) <- sapply(slot(SpatialPolygons(vor_polygons),
> #                                       'polygons'),
> #                                  slot, 'ID')
> #
> #       SpatialPolygonsDataFrame(SpatialPolygons(vor_polygons),
> #                                data=sp_dat)
> #
> #     }
> #
> #     vor <- SPointsDF_to_voronoi_SPolysDF(vor_pts)
> #
> #     distancia<-distanciaEuclidea(datos,input$lat,input$lng)
> #     if(input$checkbox){
> #       leaflet() %>% setView(lngView,latView, zoom = 14)%>%addTiles()%>%
> #         addMarkers(data = datos,lng=~longitudes, lat=~latitudes,group = "taxis")%>%
> #         addPolygons(data=vor,
> #                     stroke=TRUE, color="#a5a5a5", weight=2,
> #                     fill=TRUE, fillOpacity = 0.0,
> #                     smoothFactor=0.5,
> #                     popup=sprintf("Total In/Out: %s",
> #                                   as.character(vor@data$tot)),group = "diagrama")%>%
> #         addCircleMarkers(lng=input$lng,lat=input$lat,color="red",opacity=0.55,
> #         group = "origen") %>%
> #         addCircleMarkers(data=data$product_list,lng=~Longitud,lat=~Latitud,color="blue",
> #         opacity=0.55,group="destino")%>%
> #         addCircleMarkers(data=distancia,lng=~longitudes,lat=~latitudes,color="green",
> #         opacity=0.55,group="Taxi mas cercano")%>%
> #         addLayersControl(overlayGroups = c("origen","taxis","diagrama","Taxi mas cercano"),
> #         options = layersControlOptions(collapsed = FALSE))%>%
> #         hideGroup(group = "diagrama")%>%
> #         hideGroup(group = "Taxi mas cercano")%>%
> #         hideGroup(group = "taxis")%>%
> #         addLegend(position="bottomleft",colors = c("red","blue","green"),
> #         labels = c("Origen","Destino","Taxi seleccionado"))
> #     }else{
> #       leaflet() %>% setView(lngView,latView, zoom = 14)%>%addTiles()%>%
> #         addMarkers(data = datos,lng=~longitudes, lat=~latitudes,group = "taxis")%>%
> #         addPolygons(data=vor,
> #                     stroke=TRUE, color="#a5a5a5", weight=2,
> #                     fill=TRUE, fillOpacity = 0.0,
> #                     smoothFactor=0.5,
> #                     popup=sprintf("Total In/Out: %s",
> #                                   as.character(vor@data$tot)),group = "diagrama")%>%
> #         addCircleMarkers(lng=input$lng,lat=input$lat,color="red",opacity=0.55,
> #         group = "origen") %>%
> #         addCircleMarkers(data=distancia,lng=~longitudes,lat=~latitudes,color="green",
> #         opacity=0.55,group="Taxi mas cercano")%>%
> #         addLayersControl(overlayGroups = c("origen","taxis","diagrama","Taxi mas cercano"),
> #         options = layersControlOptions(collapsed = FALSE))%>%
```

```
> #         hideGroup(group = "diagrama")%>%
> #         hideGroup(group = "Taxi mas cercano")%>%
> #         hideGroup(group = "taxis")%>%
> #         addLegend(position="bottomleft",colors = c("red","blue","green"),
> #         labels = c("Origen","Destino","Taxi seleccionado"))
> #     }
> #   })
> #
> # })
```