Universidad Zaragoza
1542

Trabajo Fin de Grado

Optimización de red de distribución de alimentos

Food distribution network optimization

Autor

Raquel Escalera Lapuerta

Director

Luis Mariano Esteban Escaño

Escuela Universitaria Politécnica La Almunia

2020

Escuela Universitaria Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

MEMORIA

Optimización de red de distribución de alimentos

Food distribution network optimization

425.20.44

Autor: Raquel Escalera Lapuerta

Director: Luis Mariano Esteban

Fecha: 22/06/2020

INDICE DE CONTENIDO

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

1. RESUMEN

En la actualidad el hambre es uno de los principales problemas que afecta a la humanidad. Por esta razón, el segundo objetivo de desarrollo sostenible es: hambre cero.

En Navarra existen dos organizaciones: Banco de alimentos de Navarra y Cruz Roja Navarra que tratan de paliar dicho problema colaborando en un programa de ayuda alimentaria nacional. Ambas organizaciones, llamadas organizaciones asociadas de distribución, se encargan de realizar la distribución de alimentos básicos a numerosas organizaciones asociadas de reparto, las cuales llegan a los destinatarios finales.

Este trabajo fin de grado pretende optimizar la red de distribución desde una de las organizaciones asociadas de distribución Banco de Alimentos de Navarra, hasta las organizaciones asociadas de reparto, que se encuentran en diferentes municipios de la Comunidad, con las que colabora.

El objetivo del trabajo es crear una aplicación que permita seleccionar uno de los almacenes del Banco de Alimentos de Navarra como origen y los municipios donde estén situadas las organizaciones asociadas de reparto, y sea necesario repartir en esa fase, como destinos de la ruta. La aplicación calcula y muestra la ruta más óptima, la longitud de ésta y la representación del trayecto en el mapa.

1.1. PALABRAS CLAVE

Distribución de alimentos, matriz de distancias, TSP, optimización de red, aplicación

2. ABSTRACT

Nowadays, hunger is one of the main problems that affects to humanity. Due to this reason, the second target of sustainable development is: zero hunger.

In Navarra there are two organizations which try to alleviate this problem by collaborating in a national food aid program: Banco de Alimentos de Navarra and Cruz Roja Navarra. Both organizations, called distribution partner organizations, are responsible for distributing primary food to division partner organizations, which distribute to final receivers.

This final degree project aims to improve the distribution network from one of the distribution partner organizations: Banco de Alimentos de Navarra, to the division partner organizations, which are located in different municipalities of the Community and counts with its collaboration.

The target of this final degree project is to develop an application that allows selecting the origin of the route, between the two municipalities where the warehouses of Banco de Alimentos de Navarra association are and the towns where the division partner organizations are located and the distribution is necessary at the given stage. The application will return the most optimal route, the length of the route and the representation of it on the map.

3. INTRODUCCIÓN

Mil millones de personas en el mundo no pueden acceder a los alimentos básicos necesarios para vivir dignamente. El hambre es fundamentalmente una violación de derechos humanos. En la comunidad foral de Navarra, el Banco de alimentos de Navarra es una de las dos organizaciones asociadas de distribución que lleva a cabo el Programa de ayuda alimentaria para intentar paliar esta situación distribuyendo alimentos a las organizaciones asociadas de reparto asentadas en diferentes municipios.

El Banco de alimentos de Navarra lleva a cabo varias fases de distribución al año. En cada una de las fases de distribución se produce el correspondiente movimiento de alimentos. Todos los alimentos llegan al almacén de Tudela en trasporte gestionado por los proveedores. Tras la preparación de los lotes para cada entidad, los que se van a distribuir desde Berrioplano se transportan al almacén de Berrioplano en transportes que contrata el Banco de alimentos de Navarra. El resto permanece en el almacén situado en Tudela.

Actualmente, las organizaciones asociadas de reparto, es decir, las entidades que se encargan de distribuir a las personas desfavorecidas que se encuentran asentadas en los diferentes municipios de la comunidad, acuden al almacén correspondiente y retiran los alimentos asignados.

En este trabajo fin de grado se pretende plantear mejoras en este programa mediante la optimización de la red de distribución.

Para ello, se plantea un escenario diferente en el que el Banco de alimentos de Navarra contrate los medios de transporte y se encargue de llevar los alimentos hasta los municipios que se encuentran en la Comunidad Foral donde están las organizaciones asociadas de reparto. Consideramos que el destino de la distribución desde cualquiera de los dos almacenes del Banco de alimentos de Navarra es, por tanto, los diferentes municipios en los que se encuentran las organizaciones asociadas de reparto.

El proyecto se centra en la optimización de esta red de reparto y el desarrollo de una aplicación que permita seleccionar el origen de la ruta y los destinos a los que es necesario distribuir alimentos en es esa fase, para que la aplicación calcule y muestre la ruta más óptima. La aplicación permite seleccionar los destinos para poder obtener

la ruta óptima cuando no es necesario distribuir a absolutamente todos los municipios asignados a un almacén en cada fase y se pueda ayudar a los municipios más necesitados con más frecuencia y de una manera óptima.

Abordamos, por tanto, un problema clásico de optimización: el problema del viajante, o más conocido como TSP (traveling salesman problem).

De esta forma, se conseguiría reducir el coste logístico, el tiempo de distribución y la contaminación emitida por los medios de transporte y por consiguiente, se contribuye a alcanzar las metas del Banco de alimentos de Navarra que se exponen posteriormente (Conclusiones y mejoras del Programa de ayuda alimentaria).

Los alimentos se distribuyen desde dos almacenes situados en Berrioplano y en Tudela respectivamente. Cada almacén tiene asignados unos municipios a los que distribuir alimentos en cada fase según las necesidades existentes. Concretamente desde Tudela se distribuye a un máximo de 16 municipios:

Tabla 1. Origen Tudela y destinos

ORIGEN	DESTINOS
Tudela	Ablitas
	Arguedas
	Cadreita
	Caparroso
	Cascante
	Cintruénigo
	Cortes
	Falces
	Fitero
	Fustiñana
	Marcilla
	Murchante

Autor: **Raquel Escalera Lapuerta**
425.20.44

ORIGEN	DESTINOS
	Murillo el cuende
	San adrián
	Valtierra
	Villafranca

Por otro lado, el almacén de Berrioplano tiene asignados 12 municipios a los que puede distribuir:

Tabla 2. Origen Berrioplano y destinos

ORIGEN	DESTINOS
Berrioplano	Alsasua
	Barañain
	Berriozar
	Erro
	Etxarri Aranatz
	Irurtzun
	Lekunberri
	Noáin
	Pamplona
	Tafalla
	Uharte Arakil
	Viana

La metodología que se sigue para alcanzar los objetivos del trabajo fin de grado es la expuesta a continuación:

1º. Trabajo de investigación sobre el programa de distribución de alimentos en Navarra para ayudar a personas desfavorecidas y sobre el problema del viajante (TSP).

2º. Creación de la matriz de distancias entre todos los municipios donde están asentadas las organizaciones que forman parte de la red de distribución.

3º. Implementación y ejecución del algoritmo que calcule una ruta óptima.

4º. Creación de la aplicación que permita seleccionar el origen y los destinos de la ruta para mostrar la ruta óptima, su longitud y representarla en el mapa.

5º. Obtención de conclusiones.

6º. Propuestas de mejora/cambios (si procede).

4. DESARROLLO

4.1. DESCRIPCIÓN DEL PROGRAMA DE AYUDA ALIMENTARIA

El programa de ayuda alimentaria es un programa nacional que distribuye alimentos a las personas más desfavorecidas. Está cofinanciado por el Fondo de ayuda Europea para los Más Desfavorecidos (FEAD) en un 85 % y en un 15% por el presupuesto de la Administración General del Estado. El destino de la financiación comprende la compra de alimentos en el mercado, su suministro a los centros de almacenamiento y su distribución de las organizaciones asociadas de distribución (OAD) a las organizaciones asociadas de reparto autorizadas (OAR), para que lo entreguen gratuitamente a las personas más desfavorecidas.

Tiene como objetivos: promover la cohesión social, reforzar la inclusión social y contribuir a alcanzar el objetivo de hambre cero y de erradicar la pobreza.

Los alimentos que se incluyen en cada uno de los lotes son: arroz blanco, garbanzo cocido, conserva de atún, conserva de sardina, conserva de carne (magro de cerdo), pasta alimenticia, tomate frito en conserva, galletas, macedonia de verduras en conserva, fruta en conserva en almíbar ligero, cacao soluble, tarritos infantiles de fruta y de pollo, leche entera UHT, batidos de chocolate y aceite de oliva.

Los alimentos se distribuyen a las personas más desfavorecidas. Son aquellos individuos, familias, hogares o grupos que se encuentren en situación de pobreza económica, así como las personas sin hogar y otras personas en situación de especial vulnerabilidad social. Esta circunstancia es determinada por los servicios sociales públicos.

4.1.1. Conclusiones y mejoras del Programa de ayuda alimentaria

Las principales conclusiones del estudio confirman que el programa de ayuda alimentaria está muy bien valorado por las personas beneficiarias y se configura como

una herramienta que contribuye a paliar la pobreza. El 36% de los beneficiarios del programa lleva entre 3 y 5 años o más en el programa, lo que apunta el riesgo de cronificación de las situaciones de pobreza.

Entre las principales metas para mejorar están:

1. Aumentar la frecuencia de reparto.
2. Incluir mayor variedad de productos (como alimentos frescos, productos de limpieza e higiene personal).
3. Considerar necesidades familiares y de salud, así como aspectos culturales y regionales en relación al tipo de alimentos.
4. Revisar las modalidades de entrega.
5. Agilizar y flexibilizar los trámites.

El presente trabajo que se centra en optimizar la red de reparto y desarrollar una aplicación que permita seleccionar la ruta deseada y devuelva la ruta más óptima, pretende colaborar en la consecución de los objetivos haciendo posible el aumento de números de reparto, es decir, de fases a lo largo del año y haciéndolo de manera más continuada en aquellos municipios en los que sea más necesario.

4.2. DESCRIPCIÓN DEL TSP

En este trabajo abordamos un problema clásico de optimización combinatoria famoso para obtener la ruta más óptima en cada fase de distribución. Es un problema que se caracteriza por ser sencillo de enunciar pero complejo de resolver. Se denomina el problema del viajante, o más conocido como TSP (traveling salesman problem).

El enunciado del problema TSP es el siguiente:

"Un viajante quiere visitar n ciudades una y sólo una vez cada una, empezando por una cualquiera de ellas y regresando al mismo lugar del que partió. Supongamos que conoce la distancia entre cualquier par de ciudades. ¿De qué forma debe hacer el recorrido si pretende minimizar la distancia total".

4.2.1. NP-HARD

El problema del viajante, dentro de los problemas de optimización combinatoria, pertenece al tipo NP-Hard (NP-Difícil). Esto significa que es un problema que no se puede resolver en tiempo polinómico y cuya solución no puede ser verificada en tiempo polinómico.

El hecho de que sea un problema NP-Hard hace que para altas dimensiones encontrar la solución óptima no sea posible, sino que se realiza una aproximación a esta solución óptima. Por esta razón, cuantas más ciudades comprende el problema, menos exacta es la solución.

4.2.2. Principales tipos de TSP

- TSP simétrico: la matriz de distancias es simétrica, es decir, la distancia entre dos ciudades es la misma en ambos sentidos. Este es el tipo de TSP que se ha empleado para desarrollar el presente trabajo fin de grado.
- TSP asimétrico: las distancias no son iguales para todos los pares de ciudades.

4.2.3. Algoritmo voraz

La solución del TSP puede seguir un algoritmo voraz que trata de elegir la opción óptima en cada paso local, con la esperanza de llegar a una solución general óptima.

Se dice que es “voraz” porque en cada paso el algoritmo escoge la mejor opción sin tener en cuenta los pasos que faltan hasta encontrar la solución final. Después, repite el procedimiento una y otra vez hasta completar la ruta.

Este tipo de algoritmos no deshace una decisión ya tomada. Por ello cuando incorpora un nodo a la solución, ese nodo formará parte del trayecto. Seguramente no se obtendrá la solución más óptima, pero se encuentra un resultado bastante bueno en una corta fracción de tiempo.

Este tipo de algoritmo son los de menor dificultad para los investigadores que deben diseñar y comprobar el funcionamiento de diferentes procesos o estrategias.

4.2.4. Heurísticas para el TSP

Un método heurístico es un algortimo que trata de resolver un problema de manera eficiente, aunque puede suponer que no nos proporcione la solución óptima. Hay muchos algoritmos heurísticos que a lo largo de los años se han ido desarrollando para intentar simplificar la resolución del TSP.

A continuación mostraremos alguno de ellos.

4.2.4.1. Heurístico del vecino más próximo

El algoritmo heurístico del vecino más cercano (nearest neighbor algorithm) es uno de los más conocidos. Este algoritmo añade a la última ciudad de la ruta, la ciudad más cercana que todavía no ha sido visitada. La idea es moverse de una ciudad a la siguiente, de manera que la ciudad elegida sea la más cercana a donde se encuentra el viajero. Se repite el proceso hasta que todas las ciudades forman parte de la ruta.

Trata de construir un ciclo hamiltoniano de bajo coste basándose en el vértice cercano a uno dado. Este algoritmo es debido a Rosenkrantz, Stearns y Lewis.

Es una heurística que escoge la mejor opción que tiene disponible en cada paso local. Esto puede obligarle a tomar malas decisiones posteriormente, ya que al final del proceso puede que queden nodos cuya unión obligue a introducir aristas de coste elevado, lo que significa que serán ciudades alejadas del punto donde se encuentre el viajero.

4.2.4.2. Heurísticos de inserción

Los algoritmos heurísticos de inserción consisten en construir primero ciclos que visiten únicamente unos cuantos vértices o nodos, para posteriormente extenderlos insertando los vértices que faltan según diferentes reglas o criterios. En cada paso se inserta un nuevo vértice en el ciclo hasta obtener un ciclo hamiltoniano. Este algoritmo también es debido a Rosenkrantz, Stearns y Lewis.

Los criterios más utilizados son:

- Inserción más cercana: se comienza con un subgrafo y en el paso de selección, se añade el vértice más cercano al ciclo que no esté incluido en el subgrafo.
- Inserción más barata: este procedimiento difiere del de inserción más cercana en que el nodo que se inserte al subgrafo sea el de menor incremento de coste.
- Inserción más lejana: dado el subgrafo como en los casos anteriores, se añade el nodo a insertar seleccionando el vértice más lejano al ciclo.

- Inserción arbitraria: difiere de la inserción más cercana en que el nodo que se inserte se elige arbitrariamente entre todos los nodos que no estén en la subgrafo.

La siguiente figura muestra la diferencia entre dos de los criterios descritos en un caso dado. El ciclo actual está formado por 4 vértices y hay que determinar el próximo nodo a insertar. La inserción más cercana escogerá el vértice "i" y la más lejana el "s".

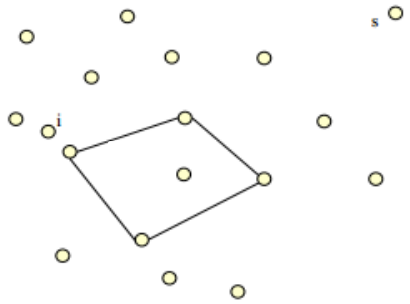

Ilustración 1. Heurística de inserción

4.2.5. Ciclo hamiltoniano

Hallar la ruta de longitud mínima que visita todas las ciudades, es decir, el ciclo hamiltoniano de longitud mínima, es un problema cuya solución directa es intentar todas las posibilidades y ver cuál es la mejor, pero no es operativo ya que conllevaría mucho tiempo. Es importante conocer qué es un ciclo hamiltoniano.

Un camino hamiltoniano en un grafo es un camino que contiene a todos los vértices del grafo exactamente una vez. Un ciclo hamiltoniano es el camino que visita todos los vértices sin repetir ninguno, excepto el vértice inicial que coincide con el vértice final. Es decir, empieza en el mismo vértice en el que termina y no repite ningún otro vértice en el camino.

Un grafo hamiltoniano es aquel grafo que contiene un ciclo hamiltoniano. El recorrido del viajante es un ciclo hamiltoniano de peso mínimo.

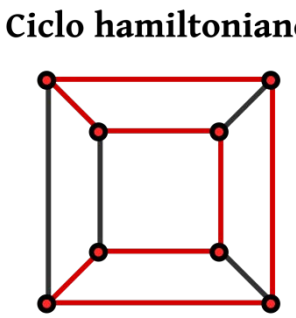

Ilustración 2. Ciclo hamiltoniano

4.2.6. Evolución TSP óptimo

A lo largo de la historia, diferentes autores han resuelto el problema del viajante con un número diferente de ciudades.

A continuación se muestra un resumen en la siguiente tabla:

Tabla 3. Evolución del TSP óptimo

AÑO	AUTORES	Nº CIUDADES
1954	G.Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R.M.Karp	64
1975	P.M. Carmerini, L.Fratta, F. Maffioli	67
1977	M.Grotschel	120
1980	H. Crowder, M.W. Padberg	318
1987	M. Padberg, G. Rinaldi	532
1987	M.Grotschel, O. Holland	666
1991	M. Padberg, G. Rinaldi	1.002
1991	M. Padberg, G. Rinaldi	2.392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7.397

AÑO	AUTORES	Nº CIUDADES
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13.509

4.3. HERRAMIENTAS

4.3.1. Programa estadístico R-project statistical

Para llevar a cabo la realización del presente trabajo fin de grado se ha elegido el programa estadístico R-Project Statistical Computing versión 3.6.3.

El programa R es un software libre que permite programar en el lenguaje de programación R. Permite la manipulación de datos, la realización de cálculos y la obtención de gráficos con enfoque al análisis estadístico. Cuenta con un conjunto de operadores para cálculo.

El lenguaje de programación es simple y eficaz; incluye condicionales, bucles, funciones recursivas y posibilidad de entradas y salidas. El almacenamiento, así como el manejo de los datos, también son eficaces y contienen una colección de herramientas amplia y variada.

Cabe mencionar que es un programa que evoluciona constantemente, ya que es un proyecto colaborativo y abierto en que los usuarios pueden publicar paquetes que complementan su configuración básica. En la actualidad cuenta con multitud de paquetes estadísticos y cada uno realiza diferentes tareas específicas del mundo de la estadística: modelos lineales y no lineales, tests estadísticos, análisis de series temporales, algoritmos de clasificación y agrupamiento…

4.3.1.1. Instalación de R

La página principal desde la que se puede acceder a los archivos necesarios para su instalación es: http://www.r-project.org.

Una vez concluida la instalación, podemos ejecutar el programa desde cualquiera de los iconos que nos genera. Lo primero que nos aparece es una ventana, llamada consola, donde podemos manejar R mediante la introducción de código.

Sin embargo, ésta no es la manera más eficiente de trabajar en R. Si se está realizando un trabajo de mediana o alta complejidad como es el caso de este presente trabajo fin de grado, es más útil manejar todas las entradas de código que

solicitemos a R en un entorno donde se pueda corregir, retocar, repetir o guardar para continuar el trabajo en otro momento. Para ello existe el editor de R (script). De esta forma se pueden modificar las líneas de código con comodidad y guardarlas para el futuro.

4.3.1.2. RStudio y su instalación

El presente trabajo fin de grado se está realizando concretamente en un entorno de desarrollo integrado (IDE) para R que funciona con la versión estándar de R disponible en CRAN. Se llama Rstudio y al igual que R, RStudio es un software libre. Se descarga desde la página: https://rstudio.com/products/rstudio/download/

RStudio nos presenta una interfaz clara que, además de la consola, nos permite monitorear y acceder a varias estructuras del sistema.

En la configuración que se utiliza para este proyecto se obtiene, además del acceso a la consola (panel inferior izquierdo), acceso al listado de paquetes instalados, con la posibilidad de cargarlos en memoria y consultar su ayuda o visualización de mapas o grafos (panel inferior derecho), un detalle de todos los objetos cargados en memoria (panel superior derecho) y un editor de script (panel superior izquierdo).

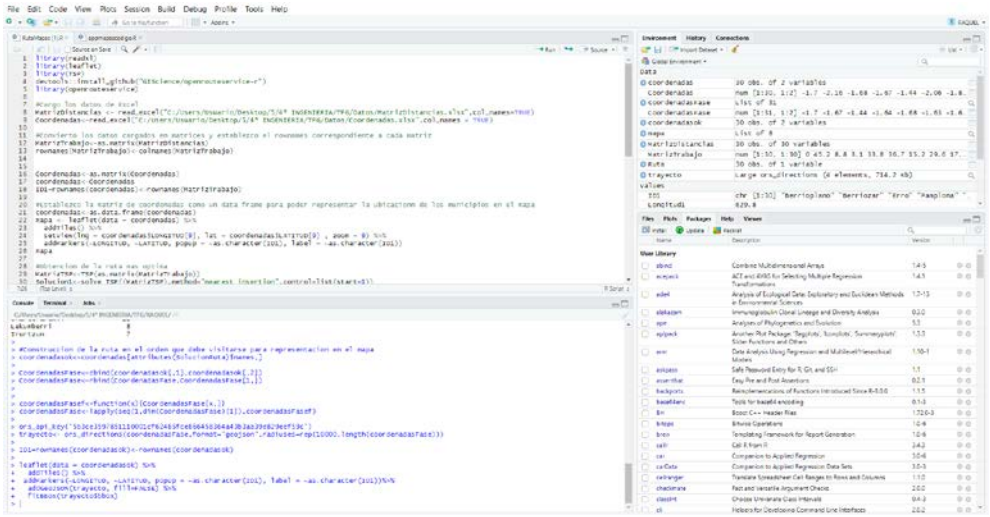

Ilustración 3. Interfaz de RStudio

Se trata de una herramienta muy útil para la creación de aplicaciones html trabajando junto al paquete Shiny. Se pretende que sea sencillo e intuitivo de manejar. De esta forma será útil tanto para los usuarios más experimentados como para los nuevos usuarios.

El proyecto fin de grado, se basa en la programación específica y se apoya en los paquetes que se describen a continuación:

- Readxl:

Readxl se encuentra en la categoría de paquetes de importación de datos. Existen otros paquetes con una funcionalidad similar: xlsx, gdata, o xlsReadWrite. Readxl es un paquete fácil de instalar, es compatible con todos los sistemas operativos y trabaja los datos a partir de tabulaciones de ellos.

Readxl soporta las versiones en formato .xls y .xlsx. Gracias a esta librería se realizará la lectura de datos de las tablas de Excel que se han definido previamente, las cuales son la matriz de distancias y las coordenadas de los municipios que forman la red de distribución.

- Leaflet:

El paquete de R llamado Leaflet se utiliza para la creación y personalización de mapas interactivos. Permite visualizar dichos mapas de una manera sencilla y dinámica. También permite hacer zoom para visualizar mejor zonas en concreto.

Este paquete permitirá ver la ubicación de los municipios en el mapa.

- TSP:

El paquete TSP comprende algunos de los algoritmos que permiten optimizar rutas. Contiene, por tanto, los algoritmos para la resolución del problema del viajante. Posee varios métodos para encontrar buenas soluciones.

- ORS Open Route Service:

Open Route Service es un paquete que permite generar rutas entre puntos previamente definidos. En la actualidad, este paquete no se encuentra en el CRAN (Comprehensive R Archive Network), pero se ha desarrollado un paquete para facilitar la consulta de la API de openrouteservice de R. Para ello se introduce la siguiente línea en el código antes de cargar la librería: devtools::install_github("GIScience/openrouteservice-r").

- Shiny:

Es un paquete de R que facilita la creación de aplicaciones web interactivas directamente desde R. Estas webs interactivas son aplicaciones que permiten a los usuarios interactuar con datos sin tener que manipular el código. En nuestro caso, el usuario sólo deberá seleccionar el almacén de origen y los municipios a los que es necesario hacer la distribución de alimentos en esa fase.

4.3.2. Microsoft Office: Excel

Además del software R, también se ha utilizado en el trabajo fin de grado el software Excel. Es un software que está integrado en el paquete Microsoft Office 97-2003. Excel se ha utilizado principalmente para la elaboración de la matriz de distancias entre los diferentes municipios y para la elaboración de una tabla con las coordenadas de dichos municipios (longitud y latitud), para su posterior tratamiento y representación gráfica en R.

4.4. RED DE DISTRIBUCIÓN: MATRIZ DE DISTANCIAS

Como se ha indicado, en este trabajo fin de grado se pretende optimizar la red de distribución que lleva a cabo Banco de alimentos de Navarra hasta los diferentes municipios donde se alojan las organizaciones asociadas de reparto.

Para ello se ha realizado un estudio sobre la red de distribución y se han tenido en cuenta todas las posibles combinaciones entre los municipios, también llamados nodos. Se han obtenido las distancias mínimas por carretera entre los municipios que forman la red de distribución gracias a Google Maps y se ha construido una matriz de distancias en la que encontramos los nombres de los municipios, es decir, todos los nodos que forman la red y los valores que se corresponden con las distancias mínimas que unen los nodos en kilómetros. Dichas distancias mínimas son llamadas también pesos de los arcos.

En el anexo III se muestra la matriz de distancias que contiene las conexiones.

En la matriz que se representa en el anexo III es importante tener en cuenta:

- Lo valores 0 que están situados en la diagonal muestran que la distancia entre un nodo y el mismo nodo es considerada nula.
- El resto de valores numéricos corresponden a los pesos de los arcos que unen los dos nodos, siendo este peso como se menciona con anterioridad, el valor de la distancia mínima entre ambos nodos en kilómetros.
- El desplazamiento de vehículos en ambos sentidos está permitido entre todos los nodos y el peso del arco de un nodo a otro es el mismo en ambos sentidos, por lo que la matriz es simétrica.

4.5. UBICACIÓN DE LOS MUNICIPIOS QUE FORMAN LA RED

Además de la matriz de distancias, en RStudio se procede a la lectura de los datos de otra tabla realizada en Excel. Dicha tabla contiene las coordenadas de cada uno de los municipios que forman la red de distribución. Se indica la ubicación de cada uno de los municipios mediante los valores de latitud y longitud. La longitud y la latitud son ángulos medidos desde el centro de la Tierra hasta el punto de la superficie de la Tierra. La medida que se utiliza para esta medición son los grados.

La tabla descrita que muestra el listado de municipios con sus coordenadas se encuentra en el anexo IV.

4.6. OBTENCIÓN DE LA SOLUCIÓN

Comenzamos a realizar los pasos necesarios para conseguir el objetivo que se plantea al inicio de este trabajo fin de grado.

El primer paso es instalar el software libre R-project. Está disponible en www.r-project.org/. A continuación instalamos RStudio, descargándolo desde https://rstudio.com/products/rstudio/download/.

Seguidamente instalaremos y cargamos los siguientes paquetes, cuyas funciones se han descrito con anterioridad en el apartado RStudio y su instalación (4.3.1.2):

library(readxl)

library(leaflet)

library(TSP)

library(openrouteservice)

Una vez realizados estos pasos previos, se debe realizar el volcado de los datos en el programa. Para ello y gracias a la librería ReadXl ya cargada, importamos la matriz de distancias. De igual forma, se procede a la lectura de las coordenadas correspondientes a cada uno de los municipios que forman parte de la red de distribución mostrada en el apartado anterior (4.4). En primer lugar fijamos el directorio de trabajo donde se encuentran los ficheros Excel y luego cargamos los datos:

setwd("C:/Directorio de trabajo")

MatrizDistancias<-read_excel("MatrizDistancias.xlsx",col_names=TRUE)

Coordenadas<-read_excel("Coordenadas.xlsx",col_names = TRUE)

Mediante la función as.matrix() se convierten los datos cargados en matrices. De esta forma se obtienen los objetos de clase matrix que se necesitan posteriormente.

Además, se especifica cuáles son los nombres que corresponden a las filas en ambas matrices, por tanto la primera columna y la primera fila se reconocen como nombres, ajenos a las matrices. En el caso de la matriz de distancias, el paquete reconoce los ceros (0) como imposibilidad de enlace.

MatrizTrabajo<-as.matrix(MatrizDistancias)

rownames(MatrizTrabajo)<-colnames(MatrizTrabajo)

Coordenadas<-as.matrix(Coordenadas)

coordenadas<-Coordenadas

ID1=rownames(coordenadas)<-rownames(MatrizTrabajo)

Utilizando la librería leaflet representamos la ubicación de los municipios. Esta función tiene varios parámetros. Hacemos uso de los que se describen a continuación:

- addTiles(): permite añadir título.
- setView(): sirve para definir el punto central del mapa. Además de indicar el zoom que deseamos.
- addMarkers(): este parámetro permite poner etiquetas al mapa y añadir pop-ups.

Para la representación de la ubicación de todos los municipios que forman la red con la librería Leaflet es necesario convertir la matriz en un objeto de clase data frame mediante la función as.data.frame().

coordenadas<-as.data.frame(coordenadas)

mapa <- leaflet(data = coordenadas) %>%

addTiles() %>%

setView(lng = coordenadas$LONGITUD[9], lat = coordenadas$LATITUD[9] , zoom = 9) %>%

addMarkers(~LONGITUD, ~LATITUD, popup = ~as.character(ID1), label = ~as.character(ID1))

La representación que se obtiene es la de un mapa real con los puntos marcados correspondientes a las ubicaciones de los municipios que forman parte de la red de distribución y permite hacer zoom en las zonas deseadas.

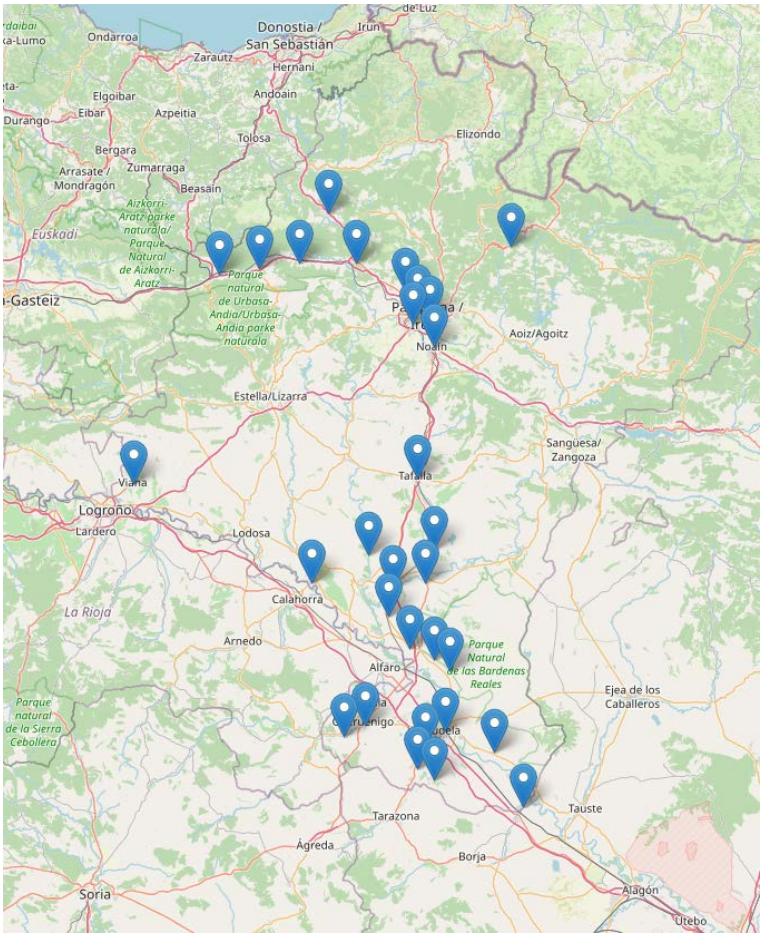

Ilustración 4. Ubicación de los municipios de la red en el mapa.

Para realizar el cálculo de la ruta más óptima, es necesario conocer cómo trabaja la función solve_TSP incluida en la librería TSP que nos permite atacar el problema del viajante:

solve_TSP(x, method, control)

dónde:

- x: corresponde a la matriz de distancias entre los municipios que se incluyen en la ruta dada como un objeto de clase TSP.

- method: esta variable permite indicar el algoritmo que va a emplear la función para calcular la ruta. En caso de no introducir un método específico, por defecto el algoritmo escogido es "nearest_insertion".
- control: permite controlar la forma en que las soluciones son calculadas, en este caso particular es la herramienta que se utiliza para imponer el nodo que debe aparecer como inicio de la ruta.

La solución será un ciclo hamiltoniano que incluya todos los nodos de la ruta y que pase por cada uno de los nodos una sola vez. La matriz de los municipios que se incluyen en la ruta debe ser un objeto de clase TSP tal y como se especifica en la forma de trabajar de la función solve_TSP(). Para transformar la matriz en dicha clase de objeto se utiliza la función TSP().

```
MatrizTSP<-TSP(as.matrix(MatrizTrabajo))
```

Los algoritmos que están disponibles son: "nearest_insertion", "farthest_insertion", "cheapest_insertion" y "arbitrary_insertion". En el apartado de Heurísticas para el TSP (4.2.4) se especifica cómo trabaja cada uno de los algoritmos que se pueden elegir.

Respecto a la opción de control que nos permite seleccionar el municipio de inicio de la ruta, debe escogerse el municipio deseado e insertarlo mediante el comando start. La inclusión de dicho municipio debe realizarse como una lista.

Una vez definidos los parámetros que deben introducirse en la función que proporciona la solución, se incluyen en la misma para que la consola de R devuelva la solución.

```
Solucion1<-
solve_TSP((MatrizTSP),method="nearest_insertion",control=list(start=1))
```

```
Solucion2<-
solve_TSP((MatrizTSP),method="farthest_insertion",control=list(start=1))
```

Los algoritmos utilizados en el presente trabajo, son: "nearest_insertion", "farthest_insertion".

La justificación de esta elección se debe a que son los dos únicos algoritmos que permiten respetar la función de control. Es importante que se respete puesto que la ruta no puede comenzar en cualquier municipio, únicamente en aquellos dos en los que se sitúan los almacenes del Banco de Alimentos de Navarra: Berrioplano y Tudela.

Se obtiene la solución por ambos métodos (Solucion1 y Solucion2) para poder escoger la ruta más óptima de las dos comparando las longitudes (Longitud1 y

Longitud2) de las rutas. Dichas longitudes se definen a partir de la función tour_length().

```
Longitud1<-tour_length(Solucion1)

Longitud2<-tour_length(Solucion2)

if(Longitud1<Longitud2){

  SolucionRuta=Solucion1

}else{

  SolucionRuta=Solucion2}
```

El objeto de clase tour que es devuelto y definido como "SolucionRuta" tras la comparación de la longitud y la elección de la menor, indica la distancia asociada a la ruta, el número de ciudades introducidas en la ruta, el nombre de las mismas y el algoritmo usado. Para obtener el nombre de las ciudades en el orden que deben visitarse se emplea la función labels().

```
Ruta<-data.frame(labels(SolucionRuta))
```

El orden obtenido es el siguiente:

```
> Ruta
   labels.SolucionRuta.
1            Berrioplano
2               Irurtzun
3              Lekumberri
4           Uharte arakil
5                 Alsasua
6         Etxarri Aranarz
7                  Viana
8              San Adrian
9                  Fitero
10           Cintruenigo
11             Murchante
12              Cascante
13               Ablitas
14                Cortes
15             Fustiñana
16                Tudela
17              Arguedas
18             Valtierra
19              Cadreita
20           Villafranca
21              Marcilla
22                Falces
23             Caparroso
24      Murilo el cuende
25                Tafalla
26                  Noáin
27              Barañain
28              Pamplona
29                  Erro
30             Berriozar
```

Ilustración 5. Listado de los municipios en el orden que deben visitarse.

Para representar la ruta en el mapa, se asocia al listado en orden de los municipios sus coordenadas. Para ello, se construye una matriz con el listado de las coordenadas correspondientes a cada municipio en el orden que deben visitarse. El orden que se debe seguir es el obtenido en el objeto denominado como "SolucionRuta" y para formar la matriz utilizamos la función attributes que accede a los atributos de un objeto, es este caso en concreto se hace referencia a los nombres de los municipios.

```
coordenadasok<-coordenadas[attributes(SolucionRuta)$names,]
```

A continuación, añadimos a la matriz creada con las coordenadas, una última fila que corresponde al último municipio que se debe visitar y que debe coincidir con el primero. De esta forma obtendremos el objeto "CoordenadasFase" que es una matriz con las coordenadas de la ruta en el orden en que deben visitarse, incluyendo el regreso al primer municipio.

```
CoordenadasFase<-cbind(coordenadasok[,1],coordenadasok[,2])

CoordenadasFase<-rbind(CoordenadasFase,CoordenadasFase[1,])
```

Utilizamos la función lapply para generar una lista con las coordenadas en orden. Obtenemos el objeto llamado "coordenadasFase" y de tipo lista, en el que se encuentran las coordenadas de los municipios en el orden que deben visitarse para realizar la distribución de forma óptima según la solución obtenida anteriormente.

```
coordenadasFasef<-function(x){CoordenadasFase[x,]}

coordenadasFase<-lapply(seq(1,dim(CoordenadasFase)[1]),coordenadasFasef)
```

A continuación, se realiza la representación del trayecto en el mapa, en el que se ven las ubicaciones de los municipios que deben visitarse. Como se puede ver en las líneas de código siguientes, es necesario registrarse en Openrouteservice. De este modo, al registrarte te proporcionan un token que se incluye con la orden ors_api_key.

```
ors_api_key('5b3ce3597851110001cf62485fce866458364a43b3aa39e829eef59c'
)
```

En la creación del trayecto, utilizamos la función ors_directon que tiene varios parámetros. Primero indicamos las coordenadas, es decir, la lista de las longitudes y latitudes en orden. Después hemos indicado el formato, que por defecto es "geojson", y por último, indicamos un radio máximo de 10.000 kilómetros entre municipios.

```
trayecto<-
ors_directions(coordenadasFase,format="geojson",radiuses=rep(10000,length(coorde
nadasFase)))
```

Antes de representar el trayecto en el mapa, creamos un nuevo identificador. En este caso, se asocia el nombre de los municipios en el orden que deben visitarse según lo obtenido en la ruta óptima.

```
ID1=rownames(coordenadasok)<-rownames(coordenadasok)
```

Finalmente, de nuevo mediante el paquete Leaflet obtenemos la representación gráfica que se muestra tras las líneas de código. En este caso hacemos uso de dos nuevos parámetros correspondientes a la función Leaflet:

-addGeoJSON(): acepta datos GeoJSON en formato analizado (listas anidadas) o en forma de cadena (vector de caracteres de elemento único).

-fitBBOx(): es una función auxiliar para establecer los límites de un widget del mapa.

```
leaflet(data = coordenadasok) %>%
  addTiles() %>%
  addMarkers(~LONGITUD, ~LATITUD, popup = ~as.character(ID1), label =
~as.character(ID1))%>%
  addGeoJSON(trayecto, fill=FALSE) %>%
  fitBBox(trayecto$bbox)
```

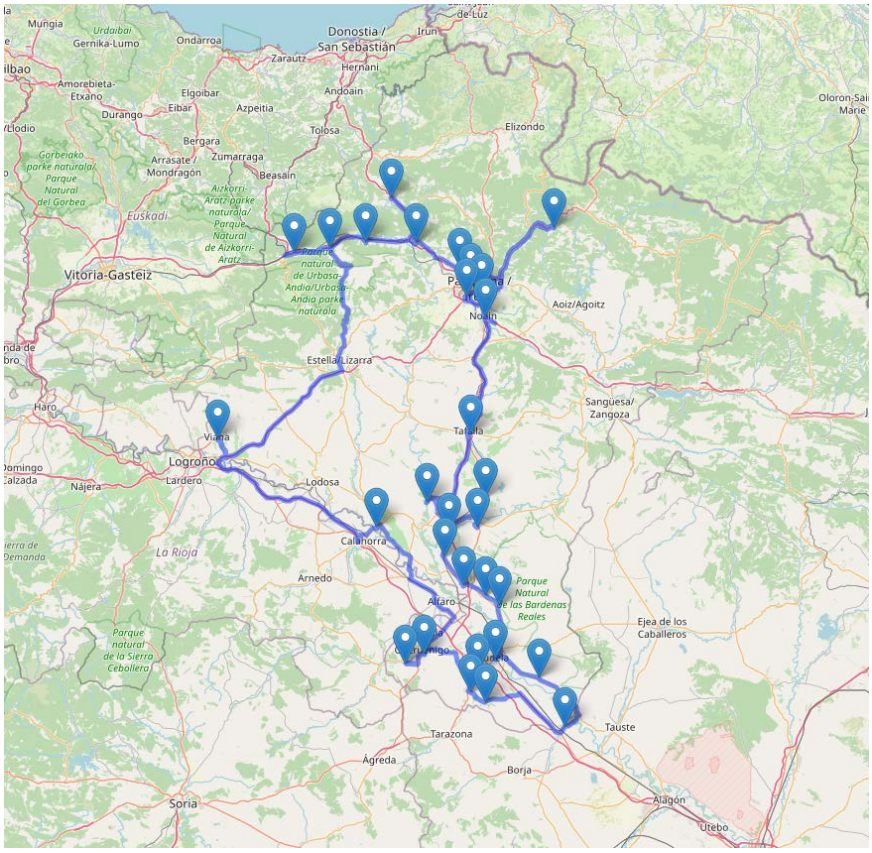

Ilustración 6. Trayecto que debe seguirse marcado en el mapa.

4.7. GENERACIÓN DE LA APLICACIÓN

El paquete escogido para la generación de la aplicación nos da la posibilidad de realizar una programación reactiva. Además tiene una colección básica de elementos de control y una clara diferencia entre la interfaz del usuario y el tratamiento de los datos, también llamado controlador, lo cual simplifica en gran medida la programación.

La programación reactiva permite que la interfaz responda de inmediato cuando el usuario realiza algún cambio en dicha interfaz.

La presente aplicación está contenida en un solo script en el que se diferencian tres componentes:

- objeto de interfaz de usuario
- función de servidor
- llamada a la shinyApp función

El objeto de interfaz (ui) de usuario controla el diseño y la apariencia de la aplicación que se desarrolla, es decir, lo que el usuario puede ver. La función del servidor (server) contiene las instrucciones que el programa necesita para construir la aplicación, es decir, que contiene el código necesario para el tratamiento y control de los datos. Finalmente, la función shinyApp crea objetos de aplicación Shiny a partir de un par explícito de ui / server.

En un script instalamos y cargamos las librerías que son necesarias, al igual que hemos hecho en el apartado de Obtención de la solución 4.5, y se instala y carga el paquete Shiny. También se cargan los datos necesarios, es decir, se realiza la lectura de la matriz de distancias y de la tabla de las coordenadas de todos los municipios. Ambas tablas se convierten a matrices mediante la función as.matrix() y se indican los nombres de la primera columna.

Una vez cargado el paquete Shiny, ya se puede comenzar a construir la aplicación. El código que determina la interfaz debe introducirse dentro de la función fluidPage().

Se desea incluir un título en la aplicación. Para ello, introducimos la siguiente línea de código:

titlePanel("ALIMENTOS PARA TODOS")

Seguidamente se desea introducir los datos respectivos a los dos posibles orígenes.

Para ello, se crea una barra desplegable en la que se introducen los dos posibles municipios que pueden ser elegidos como nodo inicial de la ruta por el usuario, ya que son los dos municipios dónde se encuentran los almacenes del Banco de alimentos. La variable de entrada se llama Origen. Se selecciona Berrioplano como almacén por defecto para la aplicación porque es el almacén con mayor superficie de almacenamiento. Las siguientes líneas de código corresponden a lo anteriormente descrito:

```
sidebarLayout(
   sidebarPanel(
    selectInput("Origen",label="Almacen",
                choices=c("Berrioplano","Tudela"),selected=c("Berrioplano")),
```

Se procede a la introducción del resto de inputs que son necesarios para la generación de la aplicación. Para poder incluir los municipios en la ruta de distribución, se crea una tabla en la que están todos los municipios que pueden formar parte de la ruta y que por tanto, forman parte de la matriz de distancias, a través de un checkbox. Se seleccionan 10 municipios por defecto. Se han escogido estos 10 municipios porque son los municipios dónde con más frecuencia es necesario repartir alimentos ya que cuentan con una mayor cantidad de habitantes. De esta forma, se evita que la aplicación se inicie mostrando un error relacionado con la falta de municipios seleccionados, ya que el paquete TSP genera rutas a partir de una cantidad mínima de municipios seleccionados.

checkboxGroupInput("variable","Seleccionar las ciudades que se desea incluir en la ruta",
choices=c("Berrioplano"="Berrioplano","Alsasua"="Alsasua","Barañain"="Barañain","Berriozar"="Berriozar","Erro"="Erro","Etxarri Aranarz"="Etxarri Aranarz","Irurtzun"="Irurtzun","Lekumberri"="Lekumberri","Noáin"="Noain","Pamplona"="Pamplona","Tafalla"="Tafalla","Uharte arakil"="Uharte arakil","Viana"="Viana","Tudela"="Tudela","Ablitas"="Ablitas","Arguedas"="Arguedas","Cadreita"="Cadreita","Caparroso"="Caparroso","Cascante"="Cascante","Cintruenigo"="Cintruenigo","Cortes"="Cortes","Falces"="Falces","Fitero"="Fitero","Fustiñana"="Fustiñana","Marcilla"="Marcilla","Murchante"="Murchante","Murilo el cuende"="Murilo el cuende", "San Adrian"="San drian","Valtierra"="Valtierra","Villafranca"="Villafranca"),
selected=c("Berrioplano"="Berrioplano","Alsasua"="Alsasua","Barañain"="Barañain","Berriozar"="Berriozar","Noáin"="Noain","Pamplona"="Pamplona","Tafalla"="Tafalla","Tudela"="Tudela","Cintruenigo"="Cintruenigo",
"San Adrian"="San Adrian"), inline=TRUE)

Se ha incluido al final de estas líneas de código el comando inline=True ya que por defecto se incluye la tabla de los municipios en vertical y de esta forma las variables están colocadas horizontalmente.

Una vez introducidas todas las variables de entrada para la generación de la aplicación que debe ver el usuario, ajustamos estas variables de entrada al ancho común de una ventana html, equivalente al valor 12 de anchura.

width=12)

Las líneas de código relacionadas con los inputs quedan así finalizadas. Tras crear la parte de la interfaz destinada a las variables de entrada, se procede a programar las salidas que se desea que la aplicación retorne en la interfaz. Para ello, se incluyen los outputs en la función mainPanel().

El primer output que devuelve la interfaz es la distancia que se debe recorrer para realizar la distribución de los alimentos en los municipios de esa fase. Empleamos la función textOutput() que nos permite incluir un texto en la interfaz y la función verbatimTextOutput() que tras captar el texto que devuelve cualquier función incluida en paquetes de R, la imprime en el panel que corresponde a la salida.

El segundo output que se desea que la aplicación retorne es el orden en el que los municipios incluidos en la ruta deben ser visitados. La función que se utiliza en este caso es TableOutput que imprime tablas en la interfaz de forma que el orden se verá en una tabla vertical y en sentido descendente. La ruta que debe seguirse es la que recorre la tabla comenzando por el primer municipio de la tabla. Y finalmente, el último output es el mapa en el que queda marcado el recorrido que debe realizarse.

```
mainPanel(textOutput("Distancia"),
          verbatimTextOutput("valuess"),
          tableOutput("values"),
          leafletOutput("mapa"))))
```

La programación destinada a la obtención de los resultados y a la definición de los objetos a los que se llama en las funciones descritas anteriormente se encuentran en la función server que se analiza a continuación.

En la función server se introducen los comandos que generan los objetos que se ofrecen como solución a partir de los inputs que el usuario ha seleccionado en la interfaz.

```
server(function(input,output) {
```

El municipio escogido como Origen debe estar incluido en la ruta. Sin embargo, no debe estar incluido dos veces si también se escoge en el panel de selección de los municipios que deben formar parte de la ruta en la fase. Para evitar que en este último caso el municipio aparezca dos veces en la ruta, se incluye en el código una función que coge los municipios seleccionados como variable, seguidamente distingue si el municipio seleccionado como Origen se encuentra entre las variables o no, y en el caso de que no esté, lo incluye en la ruta.

```
if(input$Origen%in%input$variable==TRUE)
    MatrizTrabajo<-Distancias[c(input$variable),c(input$variable)]
  else  MatrizTrabajo<-Distancias[c(input$Origen,input$variable),
                                  c(input$Origen,input$variable)]
```

Las líneas de código anteriormente descritas son necesarias en todos los objetos que se van a crear a continuación. De esta forma, se trabaja con la matriz correcta en todas las ocasiones.

Los cálculos necesarios se van a realizar con las ventajas que supone la programación reactiva que ofrece shiny. La función reactive() permite que cuando el usuario modifica las variables de entrada los resultados son calculados de nuevo inmediatamente.

El primer valor que se muestra como output es la longitud de la ruta. El objeto que se crea toma por valor el último comando que se ejecuta en su interior. En este caso, al ser la longitud de la ruta, corresponderá a la función tour_lenght().

El objeto se nombra como slidervalues1 y contiene la información relativa a la distancia de la ruta. Las líneas de código para la creación de este objeto son análogas a los explicados anteriormente en el apartado Obtención de la solución 4.5.

La librería TSP requiere de un mínimo de municipios seleccionados para ofrecer una solución, por ello, se ha programado el error que debe retornar la aplicación cuando el número de municipios seleccionados sea menor de 2. En ese caso se detiene el cálculo y se devuelve el error: “Elegir mínimo dos ciudades para obtener la ruta”. En caso contrario, se escoge la ruta de menor longitud.

```
if(length(input$variable)<2)
    stop("Elegir minimo dos ciudades para obtener ruta")
else {
    MatrizTSP<-TSP(as.matrix(MatrizTrabajo), labels = NULL)
    initialtour<-as.integer(which(labels(MatrizTrabajo)[[1]]==input$Origen[1]))
    solucion1<-
solve_TSP((MatrizTSP),method="farthest_insertion",control=list(start=initialtour))
    solucion2<-
solve_TSP((MatrizTSP),method="nearest_insertion",control=list(start=initialtour))
```

```
Longitud1<-tour_length(solucion1)

Longitud2<-tour_length(solucion2)

if(Longitud1<Longitud2){

  solucionRuta=solucion1

}else{

  solucionRuta=solucion2

}

V<-as.integer(solucionRuta)

tour_length(MatrizTSP, V)}})
```

El objeto slidervalues1 contiene la información que se desea que se imprima en la interfaz gracias a la definición de las salidas: Distancia y valuess. Se usa la función para textos renderText() que designa un valor al objeto Distancia y que es empleado por la función ui del archivo.

```
Out$Distancia<-rendertext(
```

Al igual que hemos creado el objeto slidervalues1 para obtener la longitud de la ruta, se realizan pasos similares para la obtención de los nombres de los municipios en el orden en el que se han de visitar. También en este caso se mantienen el carácter reactivo y se fija el error con el mismo texto cuando los municipios seleccionados no sean suficientes para la resolución del algoritmo. El nuevo objeto se llama slidervalues. La función que ofrece el nombre de los municipios es labels() y contiene las siguientes líneas de código:

```
else {

  MatrizTSP<-TSP(as.matrix(MatrizTrabajo), labels = NULL)

  initialtour<-as.integer(which(labels(MatrizTrabajo)[[1]]==input$Origen[1]))

  solucion1<-
solve_TSP((MatrizTSP),method="farthest_insertion",control=list(start=initialtour))

  solucion2<-
solve_TSP((MatrizTSP),method="nearest_insertion",control=list(start=initialtour))

  Longitud1<-tour_length(solucion1)

  Longitud2<-tour_length(solucion2)
```

```
if(Longitud1<Longitud2){
  solucionRuta=solucion1
}else{
  solucionRuta=solucion2
}
labels(solucionRuta)}})
```

La salida nombrada values en la función ui que está asociado a esta solución debe tener formato de tabla. Se ha titulado “Ruta de reparto” y se utiliza la función renderTable().

```
output$values<-renderTable({
   Tabla<-data.frame(slidervalues())
   names(Tabla)<-"Ruta de reparto"
   Tabla
 })
```

Finalmente creamos el objeto denominado slidervalues2 que corresponde a la salida definida en el mainpanel como “mapa”. También en este objeto se incluyen las líneas de código referentes al error de la selección mínima de municipios. En este objeto se construye una matriz que contenga la lista de municipios seleccionados en el orden en que deben visitarse y se asocian a sus coordenadas correspondientes para representar el trayecto en el mapa.

```
else {
   MatrizTSP<-TSP(as.matrix(MatrizTrabajo), labels = NULL)
   initialtour<-as.integer(which(labels(MatrizTrabajo)[[1]]==input$Origen[1]))
   solucion1<-
solve_TSP((MatrizTSP),method="farthest_insertion",control=list(start=initialtour))
   solucion2<-
solve_TSP((MatrizTSP),method="nearest_insertion",control=list(start=initialtour))
   Longitud1<-tour_length(solucion1)
   Longitud2<-tour_length(solucion2)
```

```
        if(Longitud1<Longitud2){
          solucionRuta=solucion1
        }else{
          solucionRuta=solucion2 }
        coordenadasok<-coordenadas[attributes(solucionRuta)$names,]
        CoordenadasFase<-cbind(coordenadasok[,1],coordenadasok[,2])
        CoordenadasFase<-rbind(CoordenadasFase,CoordenadasFase[1,])
        coordenadasFasef<-function(x){CoordenadasFase[x,]}
        coordenadasFase<-
lapply(seq(1,dim(CoordenadasFase)[1]),coordenadasFasef)

ors_api_key('5b3ce3597851110001cf62485fce866458364a43b3aa39e829eef59c')
        trayecto<-
ors_directions(coordenadasFase,format="geojson",radiuses=rep(10000,length(coordenadasFase)))
        ID1=rownames(coordenadasok)<-rownames(coordenadasok)
        leaflet(data = coordenadasok) %>%
          addTiles() %>%
          addMarkers(~LONGITUD, ~LATITUD, popup = ~as.character(ID1), label =
~as.character(ID1))%>%
          addGeoJSON(trayecto, fill=FALSE) %>%
          fitBBox(trayecto$bbox)})
```

Las líneas de código que indican que la salida debe ser el mapa representado son las siguientes:

```
      output$mapa<-renderLeaflet({
        slidervalues2()
      })
    }
```

4.8. INTERFAZ DE LA APLICACIÓN APP "ALIMENTOS PARA TODOS"

En este apartado se muestra el aspecto de la aplicación html de Shiny que se ha creado en este trabajo fin de grado, donde diferenciamos entre las entradas y las salidas. Servirá también cómo manual de uso de la misma.

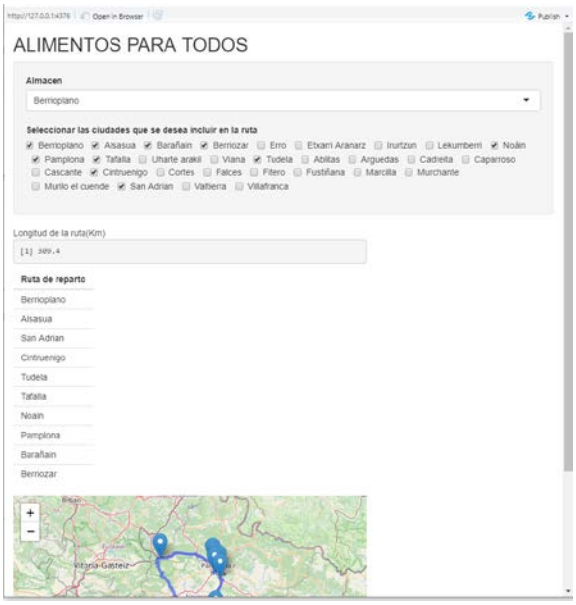

Ilustración 7. Interfaz de la aplicación

Primero se presenta en la interfaz la selección del Almacén. Se debe seleccionar el municipio en el que se encuentra el almacén desde el que se va a iniciar la ruta y que, por tanto, se considera el municipio de origen. En este desplegable se encuentran los municipios de Berrioplano y Tudela.

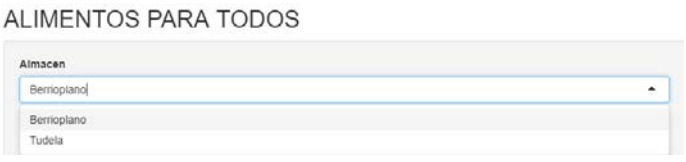

Ilustración 8. Desplegable para elegir el origen de la ruta

A continuación se deben seleccionar los municipios que se desean incluir en la ruta. Se muestran en un panel de selección todos los municipios que forman parte de la red de distribución y que están incluidos en la matriz de distancias.

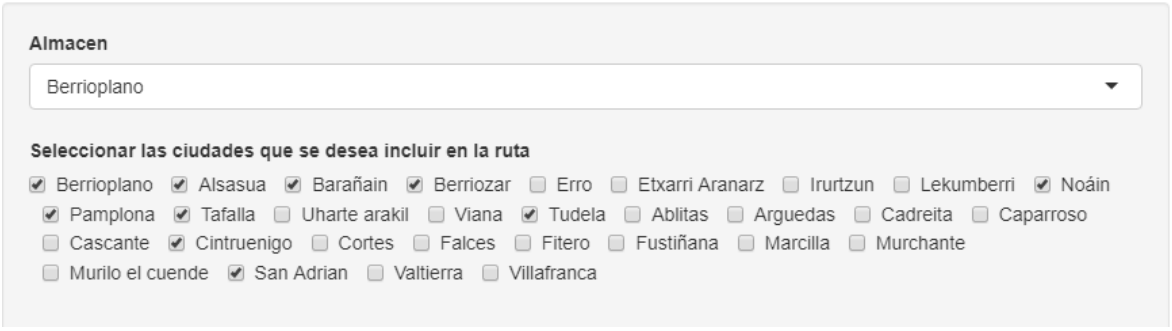

Ilustración 9. Desplegable y panel de selección de la APP

Tras ver las entradas que muestra la interfaz, pasamos a ver las salidas de la aplicación.

La primera salida es la longitud de la ruta. La medida está en kilómetros y se muestra de la siguiente forma:

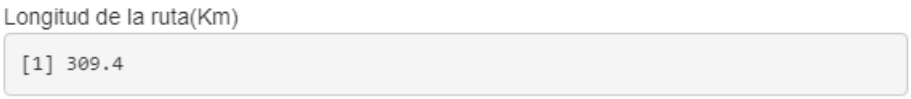

Ilustración 10. Longitud de la ruta en la APP

Seguidamente se muestra una tabla titulada “Ruta de reparto” con el orden de los municipios en el que deben ser visitados, como la siguiente:

Ruta de reparto
Berrioplano
Alsasua
San Adrian
Cintruenigo
Tudela
Tafalla
Noain
Pamplona
Barañain
Berriozar

Ilustración 11. Ruta de reparto en la APP

Para finalizar, se puede observar un mapa con la ubicación de los municipios a los que hay que distribuir alimentos. En él está marcado el trayecto que debe

realizarse para visitar todos los municipios que conforman esa fase siguiendo la ruta más óptima obtenida. El mapa permite hacer zoom.

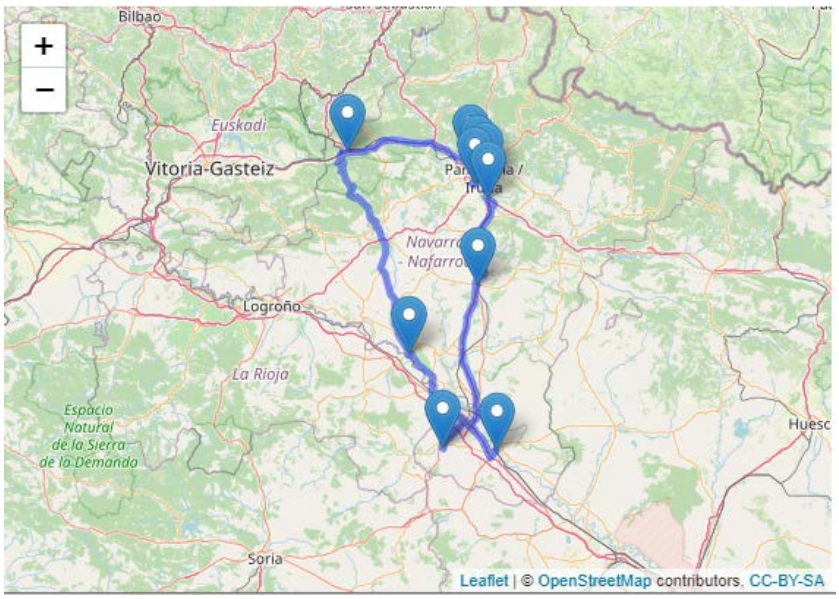

Ilustración 12. Trayecto en mapa en la APP

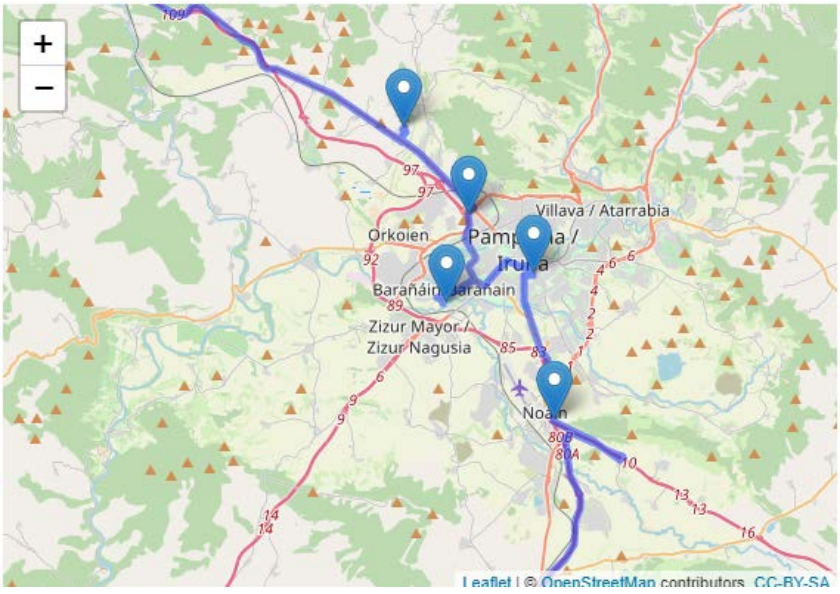

Ilustración 13. Zoom de una parte del trayecto en el mapa

5. CONCLUSIONES

Con todo lo expuesto en este proyecto fin de grado y tras el esfuerzo invertido en su elaboración, considero que se ha cumplido el principal objetivo de optimizar la red de distribución que lleva a cabo el Banco de Alimentos de Navarra en la comunidad Foral y el resto de objetivos que nos llevan a la consecución del principal y que se describen a continuación:

- Investigación sobre el problema del viajante (Travelling salesman problem), conociendo así las diferentes heurísticas que nos han permitido alcanzar una solución óptima en una breve fracción de tiempo para disminuir el coste logístico y el tiempo de distribución. Así como conocer la razón del uso de dichas heurísticas teniendo en cuenta lo necesario que es en el mundo de hoy obtener soluciones óptimas a tiempo, cumpliendo con plazos de entrega.

- Investigación sobre la red de distribución que lleva a cabo el Banco de Alimentos de Navarra, conociendo así la ubicación de sus almacenes y de los municipios a dónde reparten. Gracias a este estudio, se conoce también que lo hacen en diferentes fases y sus pretensiones principales de aumentar la frecuencia de reparto en los municipios de mayor necesidad, lo que guía el trabajo para colaborar en la consecución de este objetivo de mejora.

- Creación de la aplicación “Alimentos para todos” que se basa en las líneas de código escritas para la obtención de la solución genérica para la distribución a todos los municipios desde un solo almacén. La aplicación nos permite acercarnos a la situación real, eligiendo el almacén desde el que se van a distribuir los alimentos. La representación de la ubicación en el mapa de los municipios a los que se reparte deja ver las dos zonas marcadas que se forman en el mapa y la cercanía de cada zona a uno de los almacenes del Banco de Alimentos de Navarra. La lógica hace que sea frecuente que desde cada almacén se distribuya a la zona que le corresponde por cercanía según el mapa, pero en muchas ocasiones los alimentos salen sólo desde un almacén o se entremezclan municipios de ambas zonas en una fase, por lo que la posibilidad se selección de la aplicación es válida y útil en todas las ocasiones.

- Visualización de la ruta óptima en el mapa creando así una herramienta visual que completa la aplicación. De esta forma, se facilita y agiliza el trabajo de los usuarios ya que permite ver en el mismo momento en el que se seleccionan los

municipios que formarán parte de la ruta, y de un solo vistazo, el trayecto que se debe realizar a lo largo de la Comunidad Foral.

Además la aplicación nos permite una mejor adaptación de cara al futuro puesto que se pueden añadir municipios sin problema y con facilidad en el caso de fuera necesario por motivos circunstanciales, como en el que se ha producido en la actualidad con la pandemia global que ha hecho aumentar la pobreza y por tanto, la necesidad de distribución de alimentos.

Considero que uno de los trabajos a futuro que podría llevarse a cabo es la ampliación de este trabajo de forma que en la propia aplicación, y junto al listado de municipios en el orden en que deben visitarse, se indique el número de cestas de alimentos que se deben distribuir en cada uno de los municipios. De esta forma el usuario tendrá toda la información sobre la distribución en la misma aplicación y agilizará el trabajo, reduciendo aún más tanto el coste de distribución como el tiempo.

6. BIBLIOGRAFÍA

Amnistia internacional Española. (2019). *El Hambre en el mundo, una cuestión de Derechos Humanos*. http://grupos.es.amnesty.org/es/navarra/grupos/pamplona/paginas/noticia/articulo/el-hambre-en-el-mundo-una-cuestion-de-derechos-humanos/?pk_kwd=fbk&pk_campaign=comp

ArcGIS for Desktop. (2020, abril 15). *Qué son los sistemas de coordenadas geográficas*. https://desktop.arcgis.com/es/arcmap/10.3/guide-books/map-projections/about-geographic-coordinate-systems.htm

Cruz Roja Sala de Prensa. (2018, octubre 23). *El 90.2% de las personas beneficiarias del programa Europeo de Alimentos FEAD se encuentra en pobreza extrema*. https://www.cruzroja.es/principal/web/sala-de-prensa/-/not-presentacion-boletin-alimentos-fead-cruz-roja-fesbal-banco-de-alimentos

FEGA. (2020, febrero 22). *Ayuda alimentaria | fega.es*. https://www.fega.es/es/PwfGcp/es/accesos_directos/plan2010_ayudas/index.jsp

García, S. P. (2019). *El problema del viajante. Métodos de resolución y un enfoque hacia la Teoría de la Computación*. 64.

Héctor Carlos Perez Cisneros. (2018). *Optimización de una red de recogida de residuos*. 183.

Hutcheson, G.D. (2013). *Rcmdr.com*. https://www.rcommander.com/

IDElab Universidad de Valladolid. (2013). *El problema del viajante | IDELab*. http://idelab.uva.es/el-problema-del-viajante

Jens Elistrup Rasmussen. (2005, febrero 8). *de Villafranca a Valtierra—Google Maps*. https://www.google.es/maps/dir/Villafranca/Valtierra/@42.0973036,-1.5327236,10z/data=!3m1!4b1!4m14!4m13!1m5!1m1!1s0xd5a66c8c51f7c1b:0xdd40eb599b0f1abc!2m2!1d-1.7453642!2d42.2784669!1m5!1m1!1s0xd5a424bf2870a5b:0x8a877a78b5e3821c!2m2!1d-1.6348557!2d42.1967999!3e0?hl=es

Michael Hahsler. (2020, abril 17). *CRAN - Package TSP*. https://cran.r-project.org/web/packages/TSP/index.html

ONG con implantación estatal. (2014, mayo 28). *Programa de Ayuda Alimentaria || Solidaridad Intergeneracional*. https://solidaridadintergeneracional.es/ayuda/9262

Ordas, D. J. N. (2018). *Optimization of a delivery route*. 33.

Pamplona Actual. (2020, febrero 3). *El Banco de Alimentos de Navarra y Cruz Roja distribuirán 340.000 kilos procedentes del reparto de ayuda alimentaria de la Unión Europea*. https://pamplonaactual.com/el-banco-de-alimentos-de-navarra-y-cruz-roja-distribuiran-340-000-kilos-procedentes-del-reparto-de-ayuda-alimentaria-de-la-union-europea/

RStudio. (2020, marzo 10). *Tutorial Shiny*. https://shiny.rstudio.com/

Sistema geodésico. (2020, marzo 25). *Coordenadas Geograficas, GPS*. https://www.coordenadas-gps.com/

Soler, F. A. (2016). *Optimización de una Red de Abastecimiento*. 84.

Universidad de Cádiz. (2015, julio 7). *TSP: Algoritmos de resolución*. http://knuth.uca.es/moodle/mod/page/view.php?id=3417

Relación de documentos

(X) Memoria ..... 45 páginas

(_) Anexos ..... NN páginas

La Almunia, a 22 de Junio de 2020

Firmado: Raquel Escalera Lapuerta