Índice

[1. Resumen 1](#_Toc503870701)

[2. Librerias 2](#_Toc503870702)

[2.1 GGMap 2](#_Toc503870703)

[2.2 Leaflet 3](#_Toc503870704)

[2.3 GGally 4](#_Toc503870705)

[2.4 Xlsx 5](#_Toc503870706)

[3. Datos de muestra 6](#_Toc503870707)

[4. Solución planteada 9](#_Toc503870708)

[4.1 Sistema inicial 9](#_Toc503870709)

[4.2 Obtener distancia total. 9](#_Toc503870710)

[4.3 Algoritmo 9](#_Toc503870711)

[4.3.1 Realizamos el torneo 9](#_Toc503870712)

[4.3.2 Algoritmo de cruce 9](#_Toc503870713)

[4.3.3 Algoritmo de mutación 10](#_Toc503870714)

[4.3.4 Comprobar mejor recorrido 10](#_Toc503870715)

[5. Resultados obtenidos 11](#_Toc503870716)

[5.1 Unicamente utilizando la distancia 11](#_Toc503870717)

[5.2 Unicamente Utilizando el tiempo 12](#_Toc503870718)

[5.3 Uso de datos tiempo y Distancia 13](#_Toc503870719)

[5.4 Uso de datos tiempo o distancia 15](#_Toc503870720)

[6. Conclusión 17](#_Toc503870721)

# Resumen

El objetivo de este proyecto es generar un programa el cual pueda obtener la ruta más óptima para recorrer las 52 ciudades de España. Para conseguir este objetivo se hará uso de algoritmos Bioinspirados asimismo de diferentes librerías que facilitaran la obtención de datos reales. Para que coger los datos se hará uso de la librería ggmap para obtener el tiempo y las distancias correctas entre las ciudades además se usara la librería leaflet para poder ver el recorrido que hemos obtenido representado en un mapa.

En los puntos posteriores de este documento se explicará brevemente cómo funcionan las diferentes librerías que hemos usado en el proyecto, además, de explicar el algoritmo que hemos utilizado para este proyecto y los resultados obtenidos por medio de las 2 variables que hemos utilizado para sacar el recorrido más óptimo. Las cuales son tiempo y distancia.

# Librerias

En este apartado se explicarán las diferentes librerías que se han usado en el proyecto además de explicar para que y como las hemos usado.

## GGMap

Es una colección de funciones para visualizar datos espaciales y modelos encima de mapas estáticos en diferentes fuentes en línea como podría ser Google Maps. En este caso sobre usamos esta librería para obtener la distancia que hay entre las diferentes ciudades por medio de la función *mapdist*.

En la cual le tenemos que enviar la ciudad de origen y la ciudad de destino e indicar como queremos hacer el recorrido de las cuales hay 3 opciones (driving,Walking, Ciling) en nuestro caso ponemos la opción de driving. Esta función te devuelve todos los datos que se muestran en la figura 2.1

Figura 2.1 Ejemplo de sacar la distancia y el tiempo que se tarda entre 2 ciudades

Los datos que usamos de la figura 2.1 son la distancia en Km y el tiempo en minutos el resto de datos los desechamos y no se utilizan.

Hacemos uso de la función *geocode* para obtener la latitud y altitud de donde se encuentra cada una de las ciudades de España. Como se muestra en la figura 2.2 la cual sacamos la latitud y altitud de Bilbao y Madrid.

Figura 2.2 Latitud y Altitud de Bilbao Madrid

Estos datos los usamos para poder colocar las ciudades en el mapa por medio del paquete leaflet() que se explicara posteriormente.

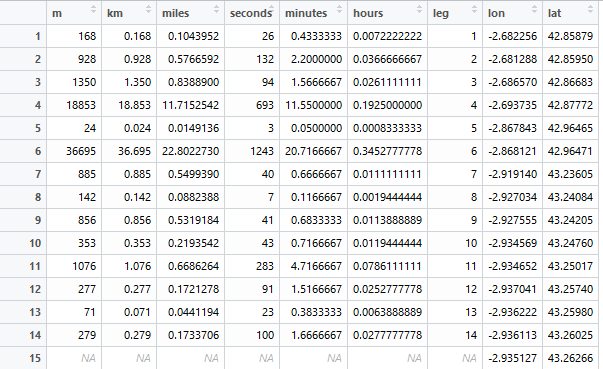
Por último, la función *route* la cual funciona igual que la función *mapdist* pero esta función devuelve los recorridos que hay que hacer y donde debemos girar para llegar a nuestro destino como se muestra en la figura 2.3 la cual se ha usado la distancia que hay que recorrer desde Vitoria hasta Bilbao.

Figura 2.3 Distancia por medio de una ruta desde Vitoria hasta Bilbao

## Leaflet

La función *leaflet()*devuelve un mapa, en el cual almacena una lista de objetos que se pueden modificar más adelante. La mayoría de las funciones de este paquete tienen un mapa de argumentos como su primer argumento lo que facilita el uso concatenado de funciones por medio de %>% el cual pertenece al paquete magrittr.

En nuestro caso usaremos esta librería mara mostrar todas las ciudades de España. Por medio de marcadores. Como se muestra en el ejemplo de código:

leaflet(Ciudades) %>% addTiles() %>%

addAwesomeMarkers(Ciudades$lon, Ciudades$lat,label =Ciudades$Ciudades)

También usamos esta librería para poder mostrar el recorrido en orden en el mapa, así como mostrar también las distancias que hay entre las diferentes ciudades. Como se muestra en el siguiente código.

leaflet(Ciudades) %>% addTiles() %>%

addAwesomeMarkers(result$lon, result$lat,label =result$Ciudades, popup = result$dist) %>%

addPolylines(result$lon, result$lat)

Como se muestra en la siguiente figura 2.4 si pasamos por el encima el cursor en el mapa saldra el nombre de la ciudad y el numero en el cual se pasa por esa ciudad.

Figura 2.4 Mostrar nombre de la ciudad y posición en la cual se pasa por esa ciudad

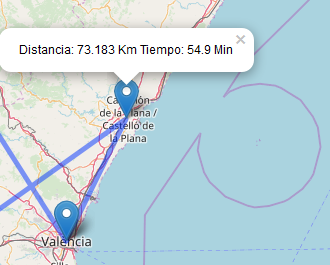
Si pinchamos en los marcadores podremos ver el tiempo y la distancia que hay desde la anterior ciudad hasta la ciudad la cual hemo pinchado. como se muerta en la figura 2.5 se puede ver la distancia y el tiempo que hay desde valencia hasta castilla la plana.

Figura 2.5 distancia y tiempo que hay desde valencia hasta catilla la plana

También, mostramos las diferentes rutas que hay, pero del problema recae que de tener únicamente 53 líneas rectas entre las diferentes ciudades pasan a ser 1000 por ello el mapa se termina colgando.

## GGally

GGally es un sistema de trazado para que se pueda reducir la complejidad de combinar objetos geométricos con datos transformados.

## Xlsx

Mediante esta Api Rstudio tiene la posibilidad de leer archivos (.xlsx) de Excel además de poder generar dichos tipos de archivos lo que facilita en gran medida el proyecto ya que podremos guardar los datos obtenidos del tiempo, la distancia y la altitud y latitud de las diferentes ciudades en un Excel. De esa manera evitamos superar los 2500 intentos que tiene diariamente el paquete ggmap.

# Datos de muestra

En este capítulo se mostrarán las diferentes ciudades que hemos utilizado en este proyecto por medio de la tabla las cuales son las 52 ciudad españolas.

|  | **Ciudades** | **lon** | **lat** |
| --- | --- | --- | --- |
| 1 | Vitoria, spain | -2.6818614 | 42.85913 |
| 2 | Albacete, spain | -1.8585424 | 38.99435 |
| 3 | Alicante, spain | -0.4906855 | 38.34600 |
| 4 | Almeria, spain | -2.4637136 | 36.83405 |
| 5 | Avila, spain | -4.6812086 | 40.65669 |
| 6 | Badajoz, spain | -6.9706535 | 38.87945 |
| 7 | mallorca, spain | 2.6501603 | 39.56960 |
| 8 | Barcelona, spain | 2.1799207 | 41.37870 |
| 9 | burgos, spain | -3.6969060 | 42.34399 |
| 10 | Caceres, spain | -6.3724247 | 39.47528 |
| 11 | Cadiz, spain | -6.2885962 | 36.52706 |
| 12 | Castello de la plana, spain | -0.0513246 | 39.98636 |
| 13 | Ciudad Real, spain | -3.9273778 | 38.98483 |
| 14 | Cordoba, spain | -4.7793835 | 37.88818 |
| 15 | coruna, spain | -8.4115401 | 43.36234 |
| 16 | Cuenca, spain | -2.1338114 | 40.07452 |
| 17 | Girona, spain | 2.8214264 | 41.97940 |
| 18 | Granada, spain | -3.5985571 | 37.17734 |
| 19 | Guadalajara, spain | -3.1601700 | 40.63249 |
| 20 | Donostia, spain | -1.9812313 | 43.31833 |
| 21 | huelva, spain | -6.9447224 | 37.26142 |
| 22 | huesca, spain | -0.4078058 | 42.13184 |
| 23 | Jaen, spain | -3.7849057 | 37.77959 |
| 24 | Leon, spain | -5.5666706 | 42.59970 |
| 25 | lleida, spain | 0.6200146 | 41.61759 |
| 26 | Logrono, spain | -2.4449852 | 42.46272 |
| 27 | lugo, spain | -7.5567582 | 43.00974 |
| 28 | madrid, spain | -3.7037902 | 40.41678 |
| 29 | malaga, spain | -4.4213988 | 36.72127 |
| 30 | Murcia, spain | -1.1306544 | 37.99224 |
| 31 | pamplona, spain | -1.6457745 | 42.81253 |
| 32 | ourense, spain | -7.8638810 | 42.33579 |
| 33 | Oviedo, spain | -5.8493887 | 43.36191 |
| 34 | Palencia, spain | -4.5288016 | 42.00969 |
| 35 | Gran Canaria, spain | -15.5474373 | 27.92022 |
| 36 | pontevedra, spain | -8.6446202 | 42.42988 |
| 37 | Salamanca, spain | -5.6635397 | 40.97010 |
| 38 | Santander, spain | -3.8074547 | 43.46069 |
| 39 | segovia, spain | -4.1088069 | 40.94290 |
| 40 | Sevilla, spain | -5.9918484 | 37.39324 |
| 41 | Soria, spain | -2.4790306 | 41.76660 |
| 42 | tarragona, spain | 1.2444909 | 41.11888 |
| 43 | Teruel, spain | -1.1064345 | 40.34569 |
| 44 | Toledo, spain | -4.0213162 | 39.85960 |
| 45 | valencia, spain | -0.3762881 | 39.46991 |
| 46 | Valladolid, spain | -4.7245321 | 41.65225 |
| 47 | Bilbao, spain | -2.9349852 | 43.26301 |
| 48 | zamora, spain | -5.7467879 | 41.50347 |
| 49 | Zaragoza, spain | -0.8890853 | 41.64882 |
| 50 | Ceuta, spain | -5.3213455 | 35.88939 |
| 51 | Melilla, spain | -2.9380973 | 35.29228 |
| 52 | Tenerife, spain | -16.6291304 | 28.29156 |

Tabla 3.1 Ciudades españolas con su latitud y altitud

# Solución planteada

En este apartado se explicará el algoritmo que se ha utilizado para poder conseguir el camino más óptimo.

## Sistema inicial

El programa genera tantas poblaciones como popsize hallamos puesto en nuestro caso hemos puesto 100. Estas poblaciones se generan de manera aleatoria. Posteriormente comprobamos si estas poblaciones están repetidas y en caso de que estén repetidas generamos otra nueva población de manera aleatoria.

## Obtener distancia total.

Posteriormente obtenemos la distancia como se muestra en el paso 15a o el tiempo total como se muestra en el paso 15b que tarda en realizar ese recorrido con esa población. En el paso 15c se han utilizado ambos datos. para sacar el mejor recorrido.

## Algoritmo

### Realizamos el torneo

Generamos un indexparents aleatorio con el tamaño de un popsize para realizar que las diferentes poblaciones que hay generen un algoritmo de cruce.

### Algoritmo de cruce

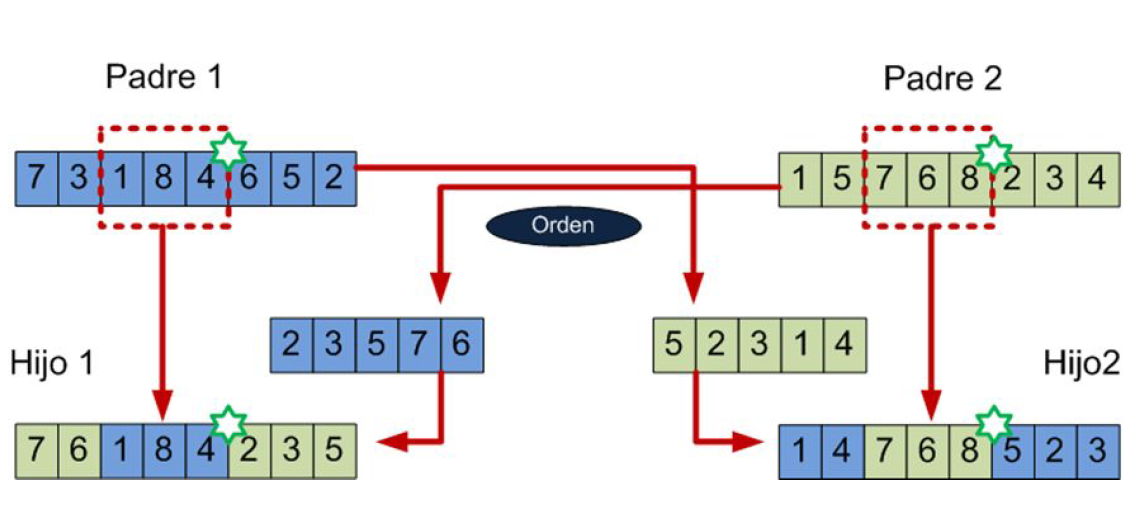
Para plantear una solución más factible se ha generado un operador de cruce por orden el cual consiste en cruzar dos poblaciones seleccionadas por el indexparents generado en el paso anterior. El cruze se realiza mediante la generación de dos números aleatorios los cuales indicaran donde debemos cortar la población para realizar el cruce. El cruce se realiza al igual que se muestra en la figura 4.1.

Figura 4.1 Algoritmo de cruce

Después de generar el algoritmo de cruce también se comprobará si las poblaciones generadas por el algoritmo de cruce están repetidas y en caso de que lo sean generar una nueva población de manera aleatoria.

### Algoritmo de mutación

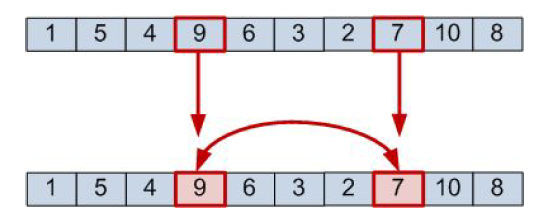
El algoritmo de mutación se cambiará la posición de un dato para cada una de las poblaciones que hemos generado después de hacer el algoritmo de cruce. Esta posición se seleccionará de manera aleatoria para generar él cambio. La mutación se hará al igual que se muestra en la figura 4.2

Figura 4.2 Algoritmo de mutación

### Comprobar mejor recorrido

En este caso se han tenido en cuenta 4 diferentes probabilidades para sacar el mejor recorrido los cuales son:

1. Paso 16 a Se selecciona el mejor recorrido únicamente utilizando la distancia
2. Paso 16b Se selecciona el mejor recorrido únicamente utilizando el tiempo
3. Paso 16c1 Se selecciona el mejor recorrido por medio de la distancia y el tiempo (AND)
4. Paso 16c2 Se selecciona el mejor recorrido por medio de la distancia o el tiempo(OR)

# Resultados obtenidos

En este punto se mostrarán los mejores resultados obtenidos por medio de las pruebas que se han realizado con las 4 diferentes tipos de dato. Para sacar los mejores recorridos en todas las generaciones se han utilizado la misma los mismos datos tanto para el popsize como para la cantidad de generaciones. Estos datos son:

* generations = 10000
* tournamentsize = 2
* probcrossover = 0.45
* probmutation = 0.3
* popsize = 100

## Unicamente utilizando la distancia

El recorrido más optimo que se ha obtenido utilizando únicamente la distancia ha sido con una distancia total de 12809.53Km. El recorrido de las ciudades es el siguiente:

37 6 52 35 21 14 10 28 19 39 34 20 31 26 49 22 25 45 12 42 17 8 7 3 30 2 44 5 46 41 43 16 13 23 4 51 18 29 50 11 40 9 47 1 38 27 32 36 15 33 24 4

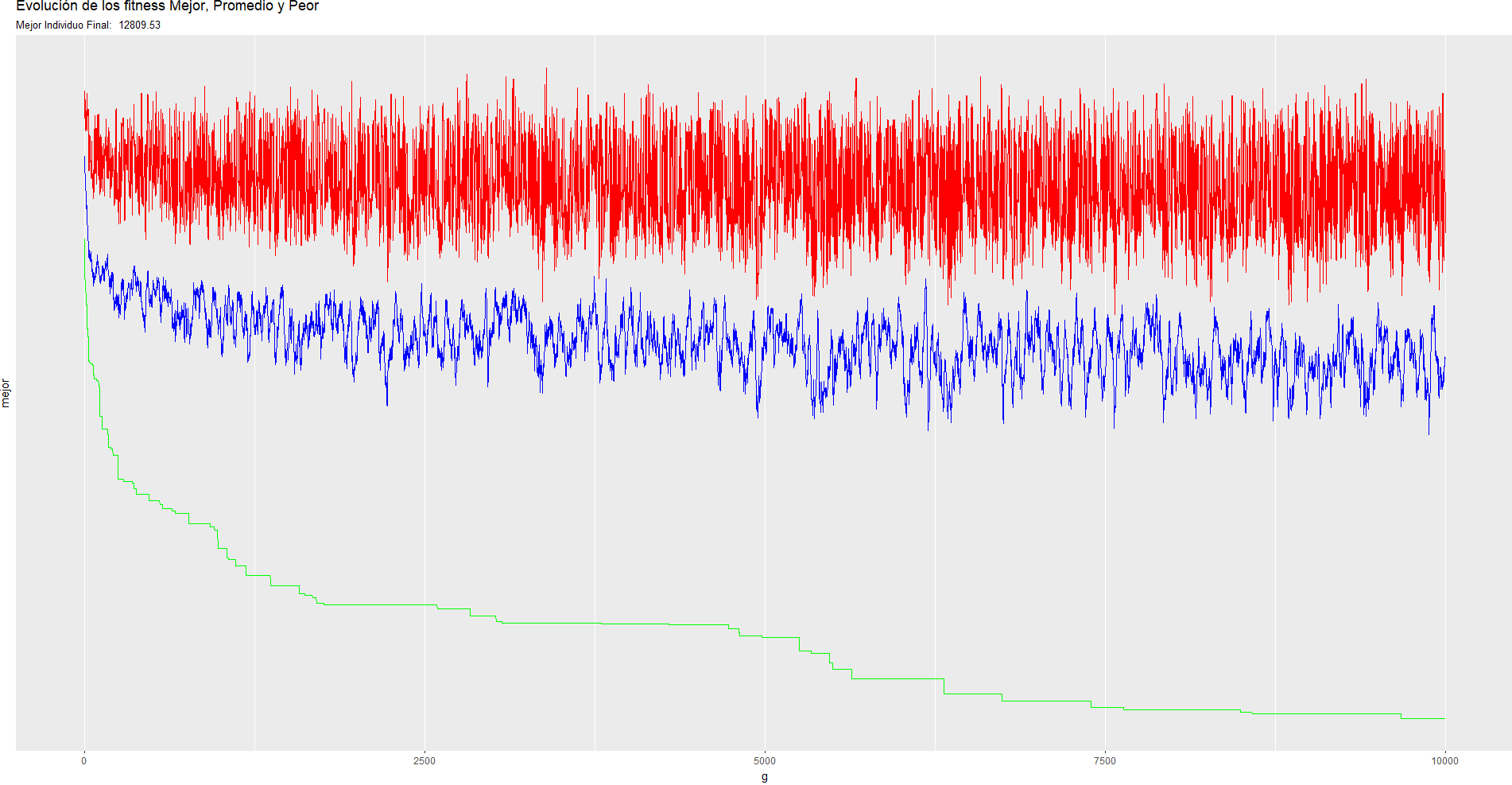
El recorrido que se ha obtenido tiene un índice de mejora igual al que se muestra en la figura 5.1.

Figura 5.1 Índice de mejora únicamente con distancia

Como se puede observar en la figura 5.1 el algoritmo a obteniendo una mejora significativa hasta la generación 5000. Posteriormente las siguientes mejoras suelen ser significativas esto se suele deber a que la mejora sustancial se da cuando las islas canarias están unidas con, Hueva, Jerez o Badajoz. Además, que ambas islas canarias siempre tienen que estar unidas es decir de Gran canaria a Tenerife esa condición ha de cumplirse siempre. Hasta que no se obtenga esa mejora el recorrido suele superar los 20000 kilómetros.

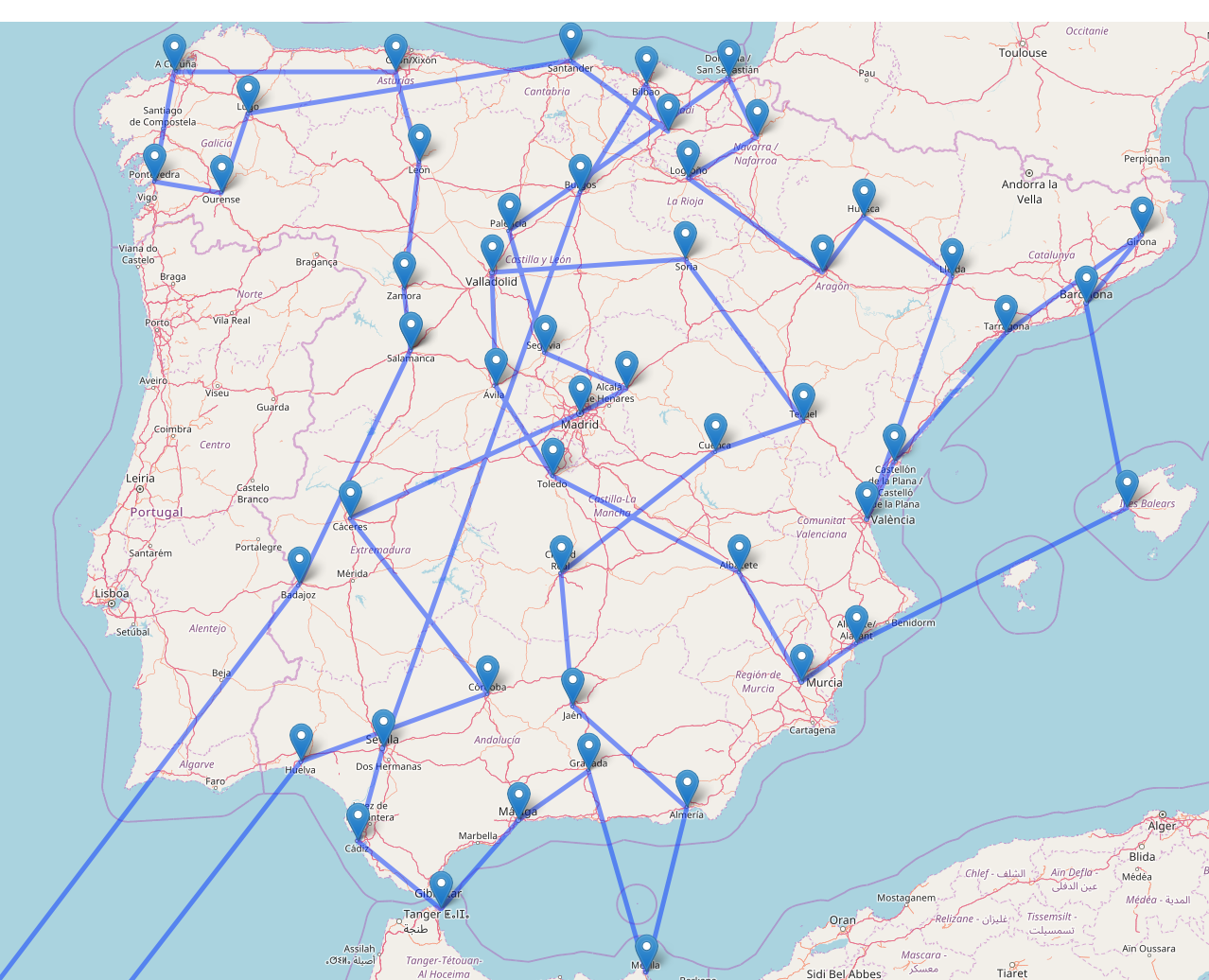
El recorrido que se ha obtenido utilizando únicamente la distancia se muestra en la figura 5.2.

Figura 5.2 Mejor recorrido utilizando únicamente distancia

## Unicamente Utilizando el tiempo

El recorrido más optimo que se ha obtenido utilizando únicamente el tiempo ha sido con un tiempo total de 12760.21" Minutos. El recorrido que se ha obtenido de las ciudades ha sido.

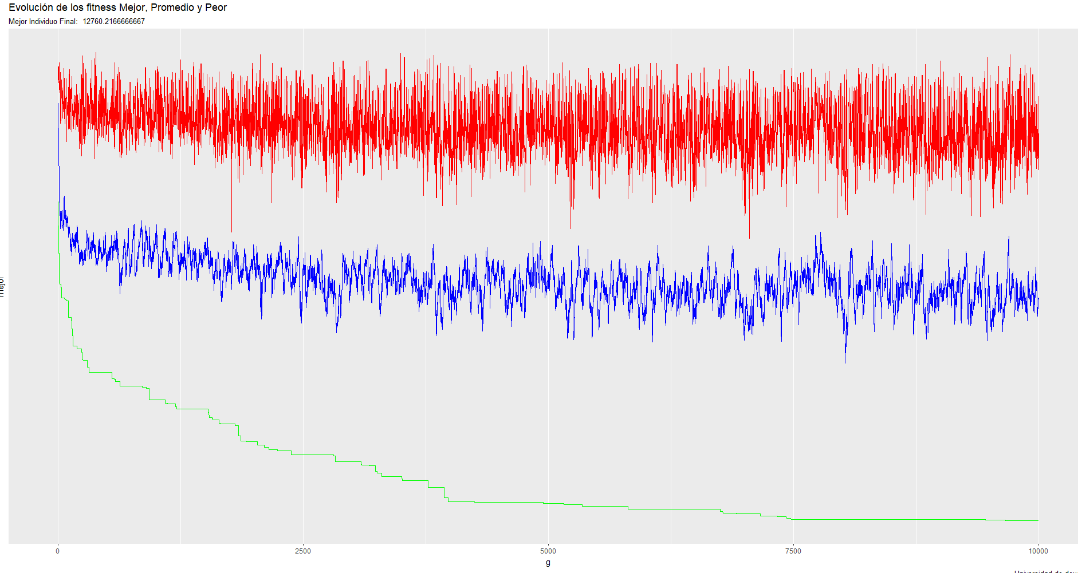
El recorrido que se ha obtenido tiene un índice de mejora igual al que se muestra en la figura 5.3. Por medio de un gráfico.

Figura 5.3 Índice de mejora únicamente con tiempo

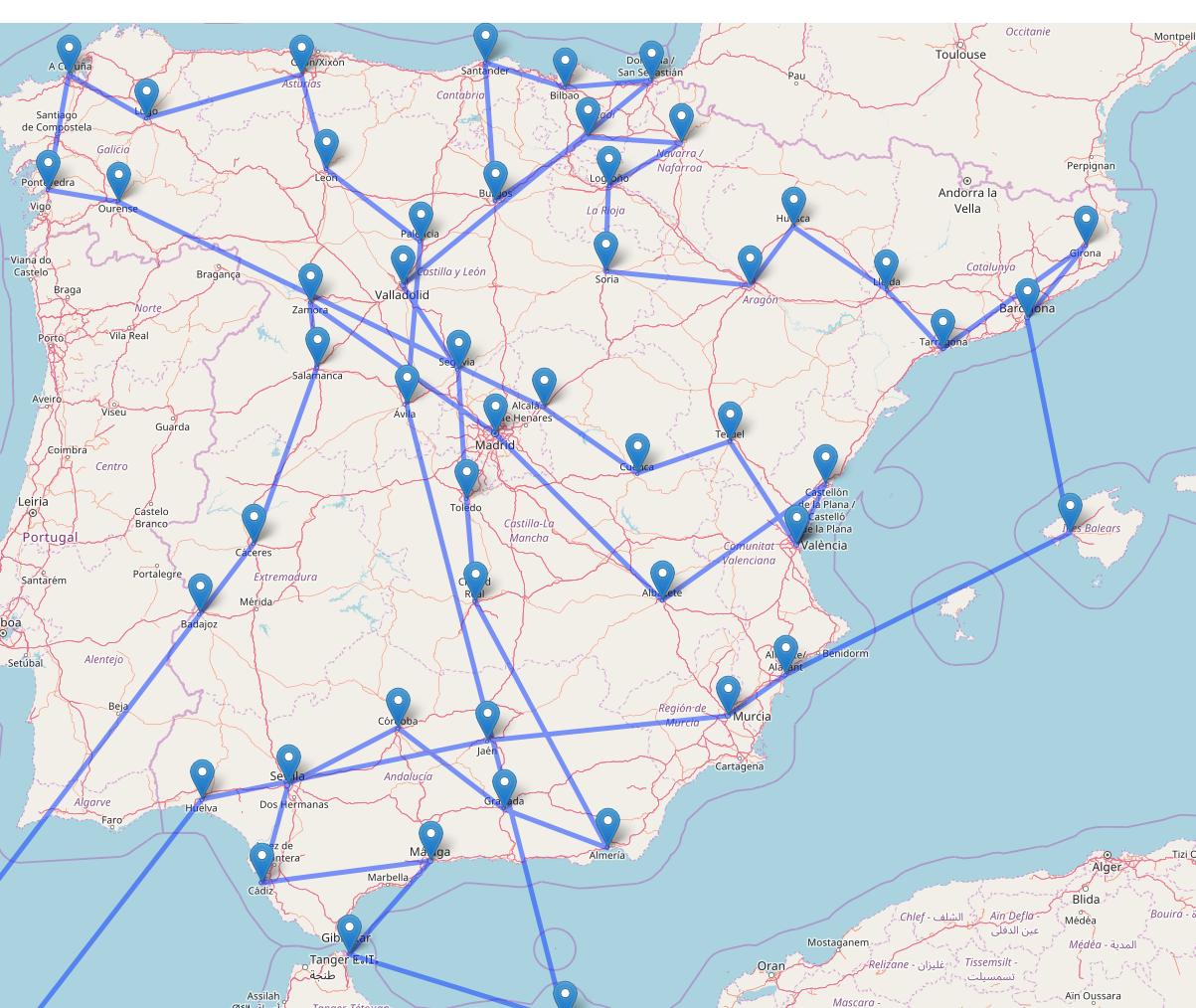
El recorrido más optimo que se obtiene utilizando únicamente tiempo se muestra en la figura 5.4.

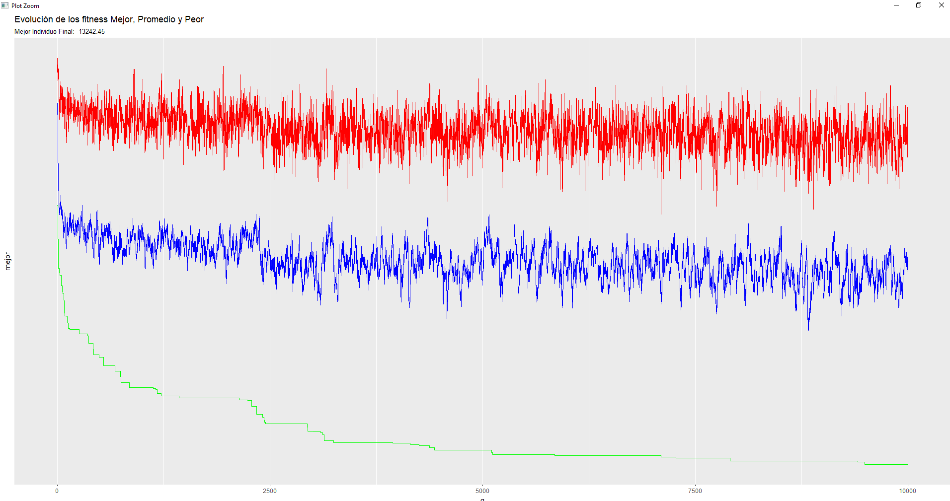
Figura 5.4 Recorrido más optimo entre las ciudades utilizando únicamente el tiempo

## Uso de datos tiempo y Distancia

El recorrido más optimo que se ha obtenido usando los datos de la distancia y el tiempo se ha obtenido un mejor Tiempo - 13242.45min mejor distancia es - 13810.18Km". El recorrido que se ha obtenido es:

37 48 24 13 14 23 18 51 50 40 35 52 21 11 29 6 10 46 34 25 22 49 19 28 5 39 41 42 8 17 7 30 2 16 27 15 36 32 33 47 1 31 20 38 9 26 43 12 45 3 4 44

El índice de mejora se muestra en la figura 5.5.

Figura 5.5 Grafico de mejora utilizando tiempo y distancia

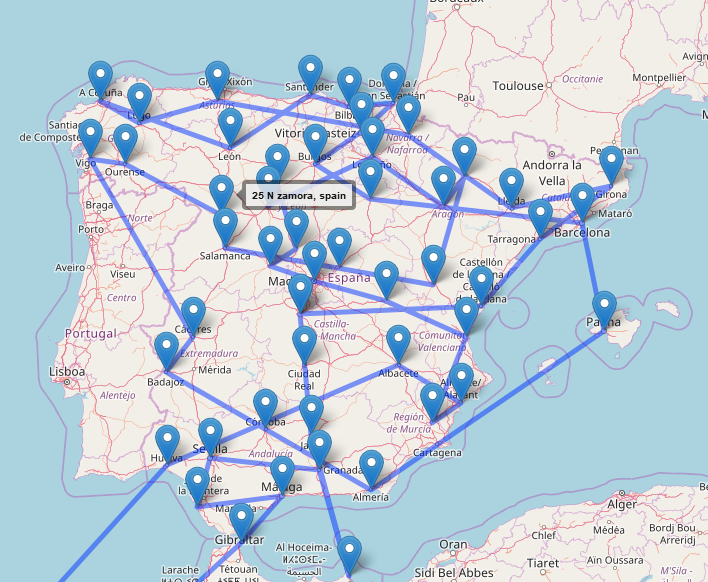
El recorrido que se ha obtenido mostrado en el mapa se muestra en la figura 5.6.

Figura 5.6 Recorrido más optimo obtenido usando tiempo y distancia

## Uso de datos tiempo o distancia

El recorrido más optimo que se ha obtenido usándolos datos de la distancia o el tiempo se ha obtenido un mejor Tiempo - 13531.85 mejor distancia es - 14259.591. El recorrido que se ha obtenido es:

14 13 44 16 2 30 3 45 17 8 7 43 12 4 18 51 50 11 48 5 41 26 1 20 31 22 25 42 49 19 10 21 52 35 23 28 39 27 15 24 34 46 37 9 47 38 33 32 36 6 40 29

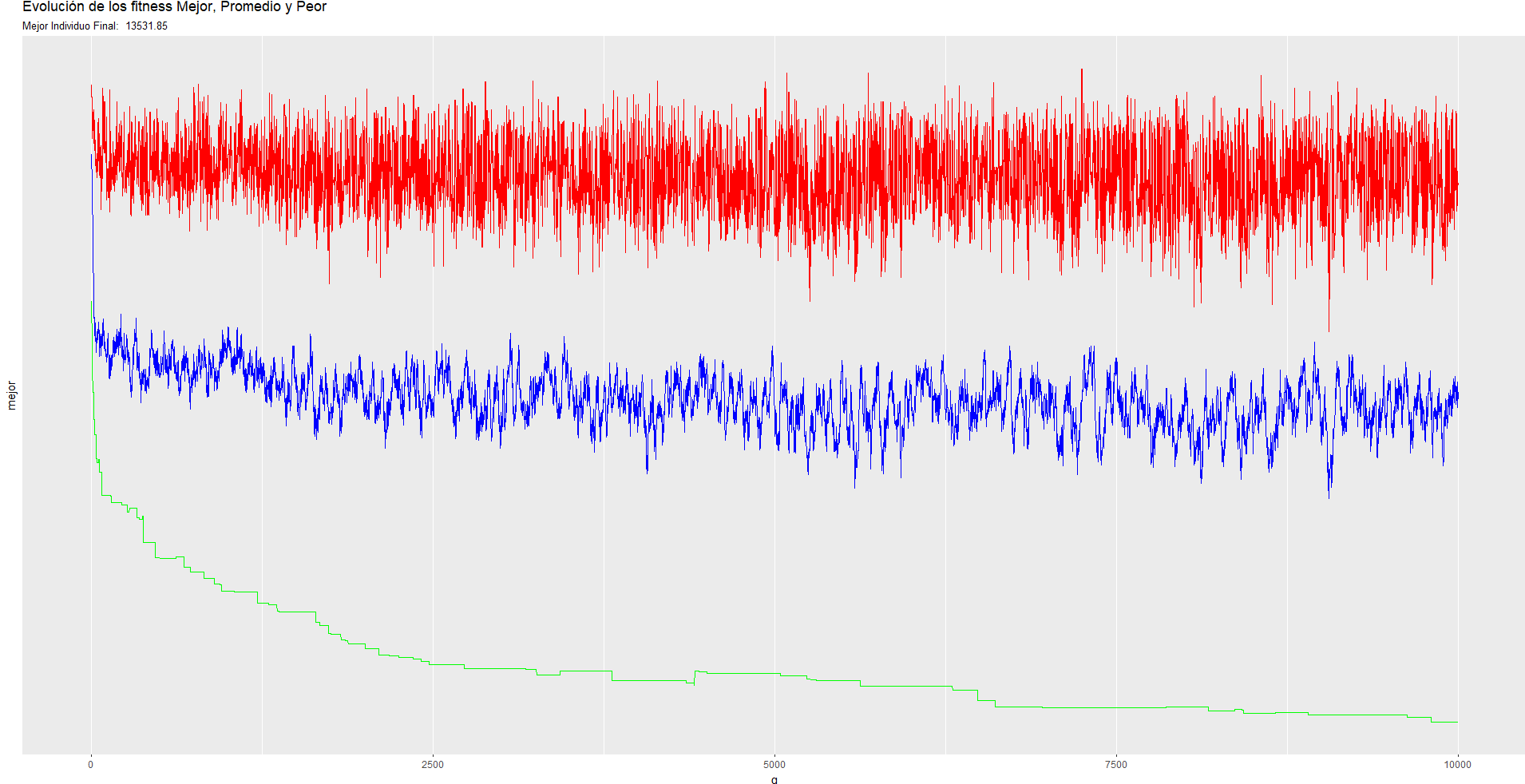
El índice de mejora se muestra en la figura 5.7.

Figura 5.7 Índice de mejora usando tiempo o distancia

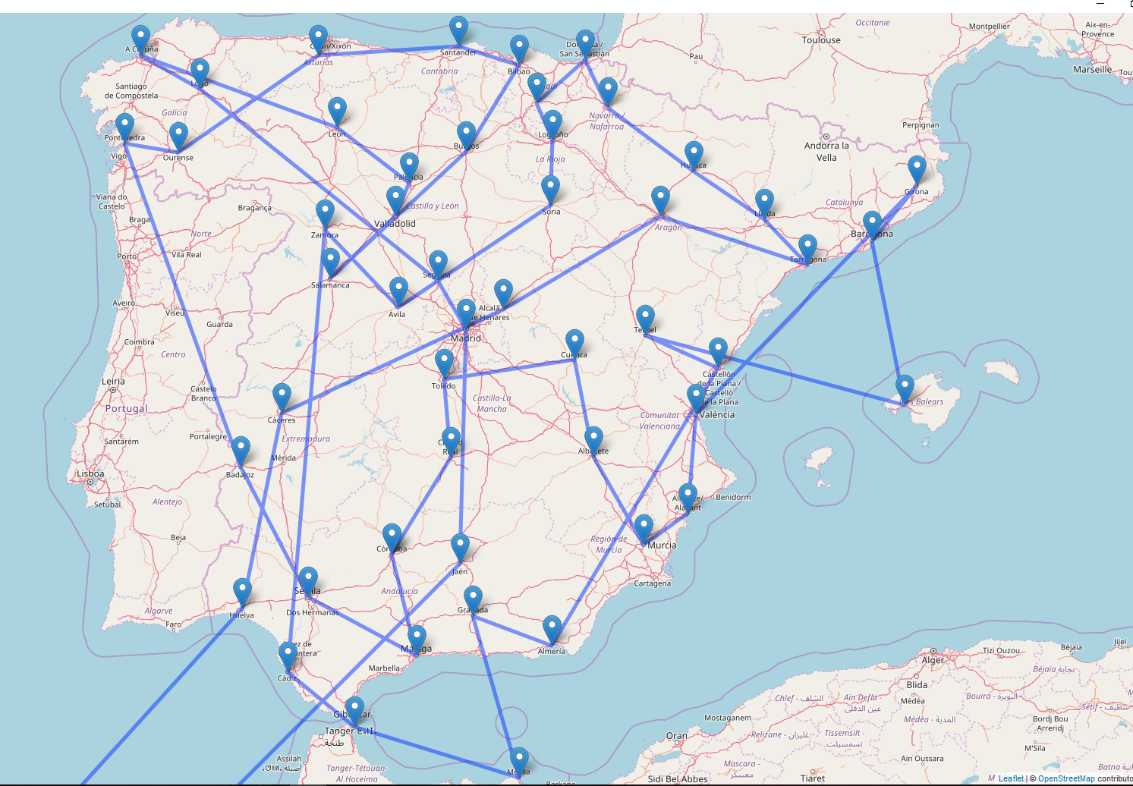
El recorrido que se ha obtenido mostrado en el mapa se muestra en la figura 5.8

Figura 5.8 Recorrido más optimo obtenido usando tiempo o distancia

En este caso el problema es que al realizar una OR puede que se mejore el tiempo, pero la distancia aumente y el caso contrario también podría suceder por ello tiene menos probabilidades de obtener un recorrido más optimo que mediante los otros tres problemas que hemos utilizado

# Conclusión