El Dilema del Vendedor Viajante - Algoritmos Geneticos en R

Martín Vedani

12-OCT-2020

```
if(! "TSP" %in% installed.packages()) install.packages("TSP", depend = TRUE)
if(! "doParallel" %in% installed.packages()) install.packages("doParallel", depend = TRUE)
```

El Problema es un clásico y dice así:

Dada una lista de ciudades y las distancias entre cada par de ciudades, ¿cuál es la ruta más corta posible que visita cada ciudad al menos una vez? Se puede parametrizar también si se regresa a la ciudad o punto de origen o no.

Pues veamos como resolverlo:

Cargar los paquetes y los datos a utilizar en la sesión actual de R

```
library("TSP")
data("USCA50")
```

Revisar el objeto de datos con el que trabajaremos al principio

```
USCA50
```

```
## object of class 'TSP'
## 50 cities (distance 'euclidean')
```

Calculamos giras del viajante (viajes) con diferentes heurísticas y almacenamos los resultados en la lista de recorridos "viajes". Como ejemplo, mostramos la primera gira que muestra el método empleado, el número de ciudades y la distancia total. Todas las longitudes de camino se comparan utilizando la tabla de puntos en la figura 1. Para el gráfico, agregamos un punto para la solución óptima que tiene una longitud de recorrido de 14497 millas. La solución óptima se puede encontrar utilizando Concorde (method = "concorde"). Se omite aquí ya que Concorde tiene que ser instalado por separado.

Actualmente, los siguiente métodos están disponibles

```
metodos <- c("nearest_insertion", "cheapest_insertion", "farthest_insertion",
             "arbitrary_insertion", "nn", "repetitive_nn", "two_opt")
```

Empezaremos siempre con la misma semilla del randomizador para poder duplicar los mismos resultados, aunque entre diferentes sistemas operativos esto no es siempre exactamente posible

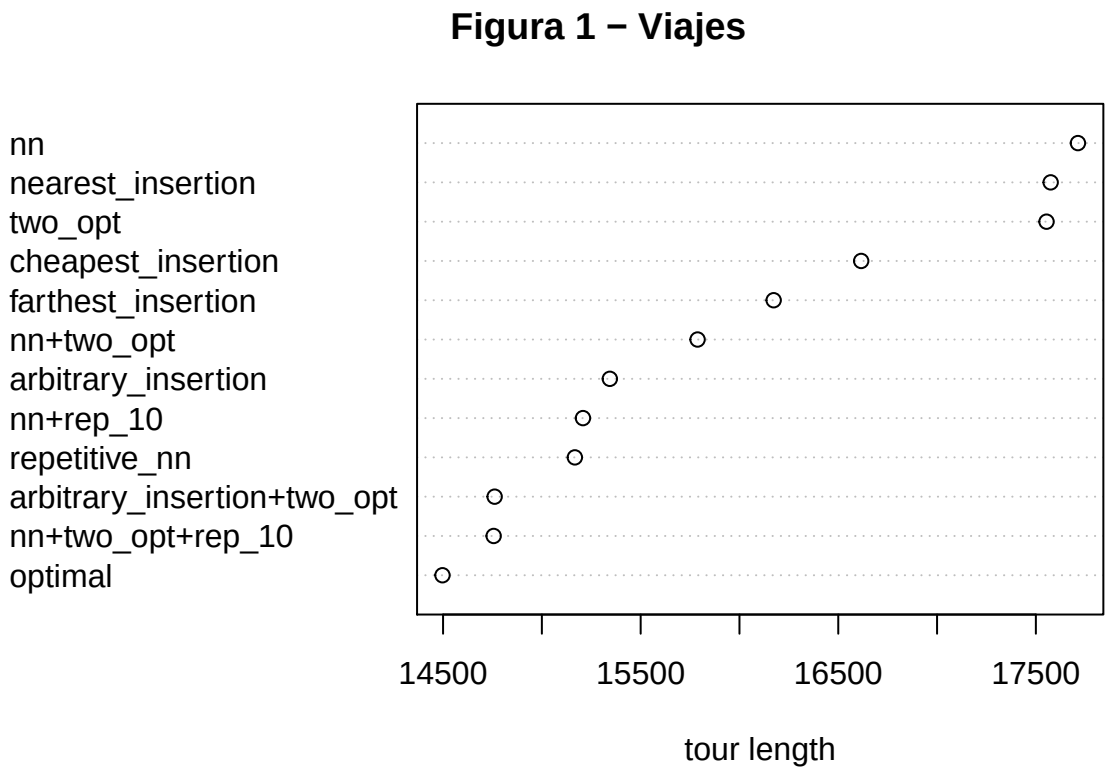
```
set.seed(123)
viajes <- sapply(metodos, FUN = function(m) solve_TSP(USCA50, method = m), simplify = FALSE)
```

Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones

```
viajes$'nn+two_opt' <- solve_TSP(USCA50, method="nn", two_opt=TRUE)
viajes$'nn+rep_10' <- solve_TSP(USCA50, method="nn", rep=10)
viajes$'nn+two_opt+rep_10' <- solve_TSP(USCA50, method="nn", two_opt=TRUE, rep=10)
viajes$'arbitrary_insertion+two_opt' <- solve_TSP(USCA50)
```

Visualizar resultados de las distancias generadas por cada método

```
dotchart(sort(c(sapply(viajes, tour_length), optimal = 14497)),xlab = "tour length")
title("Figura 1 - Viajes")
```




```
viajes[["nn+two_opt+rep_10"]]
```

```
## object of class 'TOUR'
## result of method 'nn+two_opt_rep_10' for 50 cities
## tour length: 14756
```

```
labels(viajes[["nn+two_opt+rep_10"]])
```

```
##  [1] "Belleville, ON"    "Buffalo, NY"       "Burlington, ONT"
##  [4] "Brantford, ON"     "Canton, OH"        "Akron, OH"
##  [7] "Ann Arbor, MI"     "Bay City, MI"      "Battle Creek, MI"
## [10] "Cedar Rapids, IA"  "Bloomington, IL"   "Bowling Green, KY"
## [13] "Ashland, KY"       "Asheville, NC"     "Augusta, GA"
```

```
## [16] "Atlanta, GA"       "Birmingham, AL"    "Biloxi, MS"
## [19] "Baton Rouge, LA"   "Beaumont, TX"      "Austin, TX"
## [22] "Abilene, TX"       "Amarillo, TX"      "Albuquerque, NM"
## [25] "Bakersfield, CA"   "Berkeley, CA"      "Carson City, NV"
## [28] "Boise, ID"         "Butte, MT"         "Billings, MT"
## [31] "Bismarck, ND"      "Brandon, MB"       "Calgary, AB"
## [34] "Bellingham, WA"    "Anchorage, AK"     "Alert, NT"
## [37] "Bangor, ME"        "Augusta, ME"       "Cambridge, MA"
## [40] "Boston, MA"        "Brockton, MA"      "Brattleboro, VT"
## [43] "Albany, NY"        "Bridgeport, CT"    "Central Islip, NY"
## [46] "Atlantic City, NJ" "Baltimore, MD"     "Allentown, PA"
## [49] "Binghamtom, NY"    "Burlington, VT"
```

El camino de Hamilton, utilizando 312 ciudades reales y 1 ficticia

```
data("USCA312")

tsp <- insert_dummy(USCA312, label = "cut")
tsp
```

```
## object of class 'TSP'
## 313 cities (distance 'euclidean')
```

El TSP contiene ahora una ciudad ficticia adicional y podemos tratar de resolver este TSP

```
set.seed(123)
gira <- sapply(metodos, FUN = function(m) solve_TSP(tsp, method = m), simplify = FALSE)
```

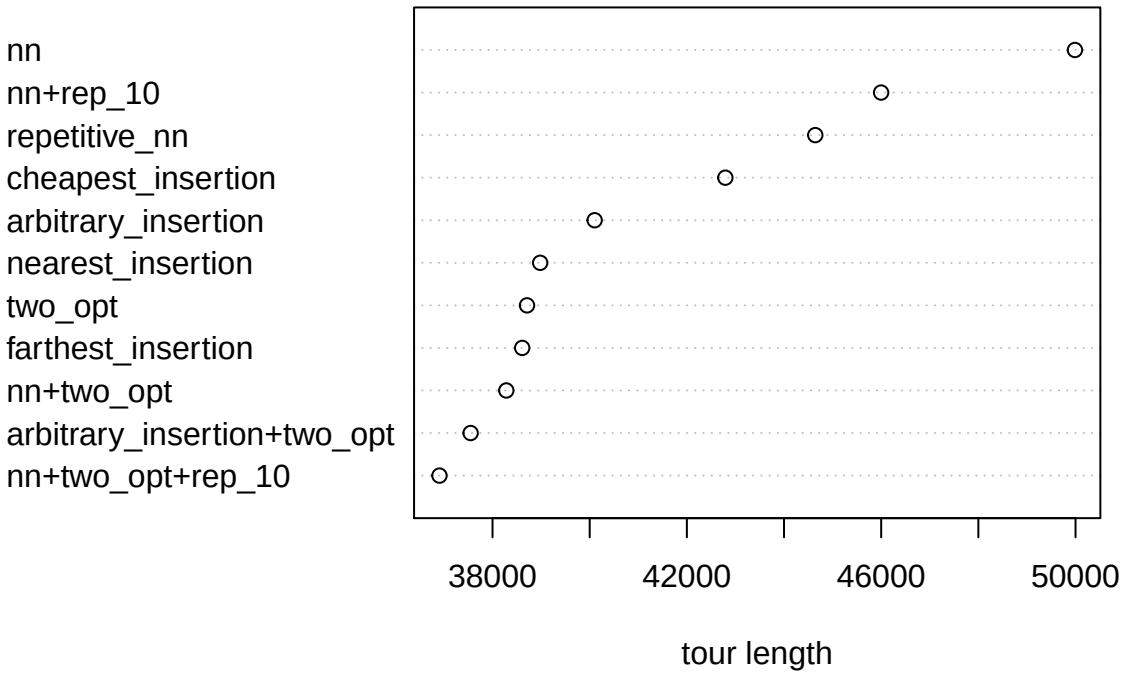
Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones

```
gira$'nn+two_opt' <- solve_TSP(tsp, method="nn", two_opt=TRUE)
gira$'nn+rep_10' <- solve_TSP(tsp, method="nn", rep=10)
gira$'nn+two_opt+rep_10' <- solve_TSP(tsp, method="nn", two_opt=TRUE, rep=10)
gira$'arbitrary_insertion+two_opt' <- solve_TSP(tsp)
```

Visualizar resultados de las distancias generadas por cada método

```
dotchart(sort(c(sapply(gira, tour_length))), xlab = "tour length")
title("Figura 2 - Gira")
```

Figura 2 − Gira



Como la ciudad ficticia ("dummy") tiene distancia cero a todas las demás ciudades,la longitud del camino es igual a la longitud de la gira que nos informó R anteriormente. La ruta se inicia con la primera ciudad en la lista después de la ciudad ficticia y termina con la ciudad justo antes de ella.

```
gira[["nn+two_opt+rep_10"]]
```

```
## object of class 'TOUR'
## result of method 'nn+two_opt_rep_10' for 313 cities
## tour length: 36908
```

Utilizamos cut_tour() para crear un camino y mostramos las primeras y últimas 6 ciudades a continuación.

```
gira <- gira[["nn+two_opt+rep_10"]]
camino <- cut_tour(gira, "cut")
head(labels(camino))
```

```
## [1] "Hilo, HI"          "Honolulu, HI"      "Lihue, HI"
## [4] "Prince Rupert, BC" "Juneau, AK"        "Whitehorse, YK"
```

```
tail(labels(camino))
```

```
## [1] "Saint Petersburg, FL" "Sarasota, FL"         "West Palm Beach, FL"
## [4] "Miami, FL"            "Key West, FL"         "San Juan, PR"
```

La gira que se encuentra en este ejemplo nos da un camino desde Hilo, Hawaii a San Juan de Puerto Rico. Un camino de este tipo también se puede visualizar mediante el uso los paquetes sp, mapas y MapTools (Pebesma y Bivand 2005).

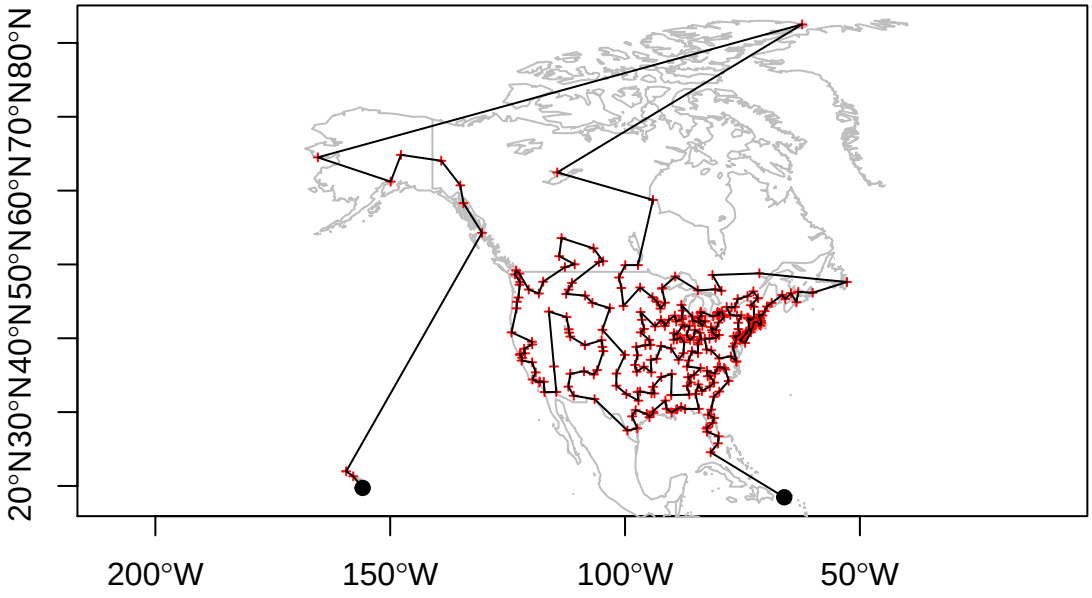
```
library("maps")
library("sp")
library("maptools")
data("USCA312_GPS")
```

Busca todo el path en tu máquina local al archivo USCA312_map.rda provisto.Solía ser parte del paquete TSP pero fue actualizada a nuevo formato bajo el nombre USCA312_GPS. Puedes observar las diferencias por tú mismo ya que hemos cargado ambos formatos como variables globales y están disponibles en RStudio.

Creamos un gráfico especial que utilizaremos varias veces.

```
plot_path <- function(path){
  plot(as(USCA312_coords, "Spatial"), axes = TRUE)
  plot(USCA312_basemap, add = TRUE, col = "gray")
  points(USCA312_coords, pch = 3, cex = 0.4, col = "red")
  path_line <- SpatialLines(list(Lines(list(Line(USCA312_coords[path,])), ID="1")))
  plot(path_line, add=TRUE, col = "black")
  points(USCA312_coords[c(head(path,1), tail(path,1)),], pch = 19, col = "black")
}

plot_path(camino)
title("Figura 3")
```

Figura 3



Como próximo paso, vamos a elegir Nueva York como la ciudad de partida. Transformamos los datos en un objeto ATSP ("Asymetric" Travelling Sales Person) y establecemos la columna correspondiente a Nueva York como punto cero antes de resolverlo. Por lo tanto, la distancia para volver desde la última ciudad hasta Nueva York, no contribuye a la longitud de la trayectoria o camino. Nosotros usamos la heurística del vecino más cercano ("nn" por sus siglas en ingles del nearest neighbor) para calcular un recorrido inicial.

```
atsp <- as.ATSP(USCA312)
ny <- which(labels(USCA312) == "New York, NY")
atsp[, ny] <- 0

set.seed(123)
tour_inicial <- sapply(metodos, FUN = function(m) solve_TSP(atsp, method = m), simplify = FALSE)
```

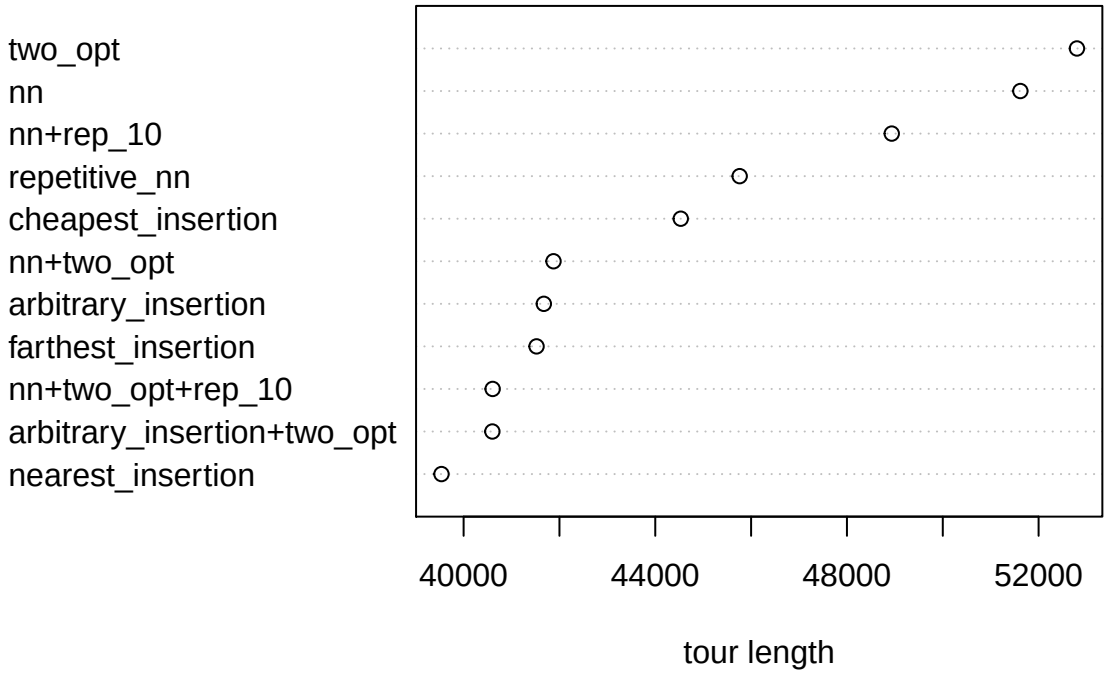
Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones

```
tour_inicial$'nn+two_opt' <- solve_TSP(atsp, method="nn", two_opt=TRUE)
tour_inicial$'nn+rep_10' <- solve_TSP(atsp, method="nn", rep=10)
tour_inicial$'nn+two_opt+rep_10' <- solve_TSP(atsp, method="nn", two_opt=TRUE, rep=10)
tour_inicial$'arbitrary_insertion+two_opt' <- solve_TSP(atsp)
```

Visualizar resultados de las distancias generadas por cada método

```
dotchart(sort(c(sapply(tour_inicial, tour_length))), xlab = "tour length")
title("Figura 4 - Tour Inicial NY")
```

Figura 4 − Tour Inicial NY



Como la ciudad ficticia ("dummy") tiene distancia cero a todas las demás ciudades, la longitud del camino es igual a la longitud de la gira que nos informó R anteriormente. La ruta se inicia con la primera ciudad en la lista después de la ciudad ficticia y termina con la ciudad justo antes de ella.

```
tour_inicial[["nearest_insertion"]]
```

```
## object of class 'TOUR'
## result of method 'nearest_insertion' for 312 cities
## tour length: 39538
```

Utilizamos cut_tour() para crear un camino y mostramos las primeras y últimas 6 ciudades a continuación.

```
tour_inicial <- tour_inicial[["nearest_insertion"]]

camino2 <- cut_tour(tour_inicial, cut = "New York, NY",
                    exclude_cut = FALSE)

head(labels(camino2))
```

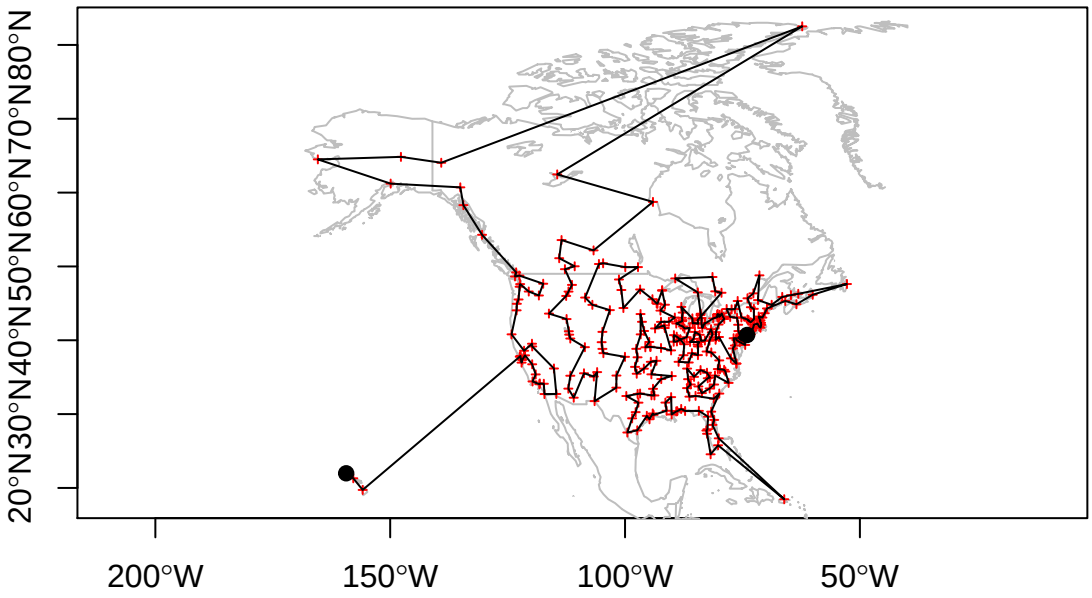
```
## [1] "New York, NY"     "Jersey City, NJ"  "Elizabeth, NJ"    "Newark, NJ"
## [5] "Paterson, NJ"     "White Plains, NY"
```

```
tail(labels(camino2))
```

```
## [1] "Oakland, CA"       "Berkeley, CA"      "San Francisco, CA"
## [4] "Hilo, HI"          "Honolulu, HI"      "Lihue, HI"
```

```
plot_path(camino2)
title("Figura 5 - Tour Inicial NY")
```

Figura 5 − Tour Inicial NY



Para encontrar el camino más corto de Hamilton también podemos restringir los dos puntos extremos, el de partida y el final. Este problema puede ser transformado a un TSP mediante la sustitución de las dos ciudades por una sola ciudad que contiene las distancias desde el punto de inicio en sus columnas y las distancias al punto final en sus filas.

Para el siguiente ejemplo, sólo estamos interesados en caminos que empiezan en Nueva York y que terminan en Los Ángeles. Por lo tanto, eliminamos las dos ciudades de la matriz de distancias, creamos un TSP asimétrico e insertamos una ciudad ficticia llamada "LA / NY". Las distancias de "DESDE" esta ciudad ficticia se sustituyen por las distancias desde Nueva York y las distancias "HACIA" se sustituyen por las distancias hacia Los Ángeles.

```
m <- as.matrix(USCA312)
ny <- which(labels(USCA312) == "New York, NY")
la <- which(labels(USCA312) == "Los Angeles, CA")
atsp <- ATSP(m[-c(ny,la), -c(ny,la)])
atsp <- insert_dummy(atsp, label = "LA/NY")
la_ny <- which(labels(atsp) == "LA/NY")
atsp[la_ny, ] <- c(m[-c(ny,la), ny], 0)
atsp[, la_ny] <- c(m[la, -c(ny,la)], 0)
```

Utilizamos la heurística de inserción más cercana

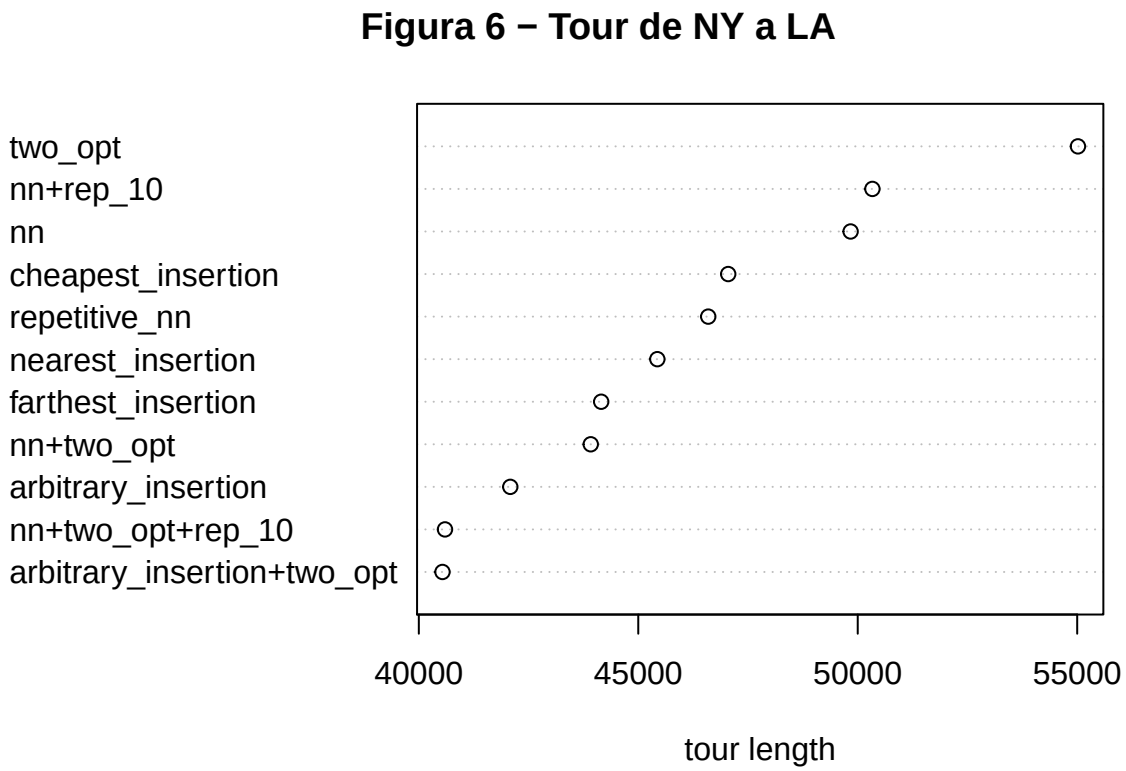
```
set.seed(123)
tour_final <- sapply(metodos, FUN = function(m) solve_TSP(atsp, method = m), simplify = FALSE)
```

Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones

```
tour_final$'nn+two_opt' <- solve_TSP(atsp, method="nn", two_opt=TRUE)
tour_final$'nn+rep_10' <- solve_TSP(atsp, method="nn", rep=10)
tour_final$'nn+two_opt+rep_10' <- solve_TSP(atsp, method="nn", two_opt=TRUE, rep=10)
tour_final$'arbitrary_insertion+two_opt' <- solve_TSP(atsp)
```

Visualizar resultados de las distancias generadas por cada método

```
dotchart(sort(c(sapply(tour_final, tour_length))), xlab = "tour length")
title("Figura 6 - Tour de NY a LA")
```




```
tour_final[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 311 cities
## tour length: 40539
```

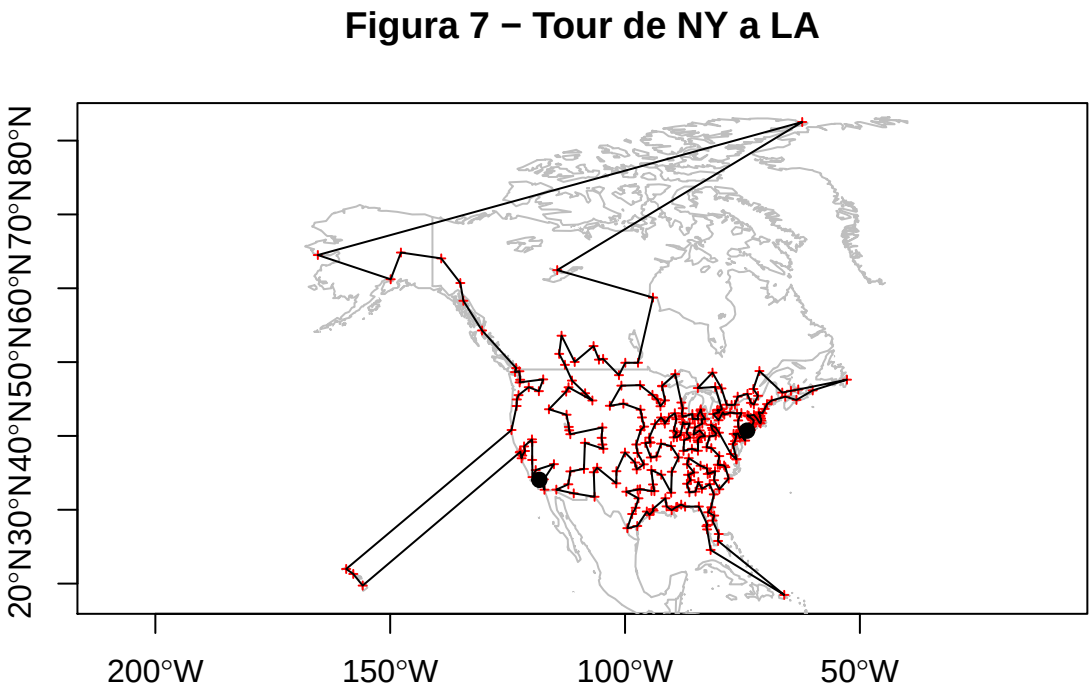
```
tour_final <- tour_final[['arbitrary_insertion+two_opt']]
path_labels <- c("New York, NY", labels(cut_tour(tour_final, la_ny)), "Los Angeles, CA")
path_ids <- match(path_labels, labels(USCA312))
head(path_labels)
```

```
## [1] "New York, NY"     "Jersey City, NJ"  "Elizabeth, NJ"    "Newark, NJ"
## [5] "Paterson, NJ"     "White Plains, NY"
```

```
tail(path_labels)
```

```
## [1] "Bakersfield, CA"    "Las Vegas, NV"      "San Bernardino, CA"
## [4] "San Diego, CA"      "Pasadena, CA"       "Los Angeles, CA"
```

```
plot_path(path_ids)
title("Figura 7 - Tour de NY a LA")
```



La ruta que se muestra en la Figura 7 puede contener algunos cruces que indican que la solución es sub-óptima. La solución óptima generada por reformular el problema como un TSP y el uso de Concorde sólo tiene una longitud de recorrido de 38.489 millas aproximadamente.

La solución final sera MUCHO más corta, pero nos va a requerir bastante análisis.

Comencemos!

Reordenamiento por agrupación (CLUSTERING)

La idea es que los objetos en un grupo son visitados en orden consecutivo y para pasar de un grupo al siguiente más grande es necesario hacer “saltos”.

Este tipo de reordenamiento por agrupación (si recordamos la Unidad 2.1 - Agrupamiento o Clustering) sugiere encontrar automáticamente los límites de cluster o numero k de grupos mediante la adición de un numero k de ciudades ficticias que tienen distancia constante c a un centro. En la solución óptima del TSP, las ciudades ficticias deben separar las ciudades más distantes y por lo tanto representan los límites óptimos para clusters k.

PAUSA, cambiamos de datos y nos vamos a las ciencias naturales un rato

Para ilustrar esta estrategia de agrupar antes de medir distancias, y verla más claramente, vamos utilizar un conocido conjunto de datos de R llamado iris, dejando de lado las ciudades por un rato.

El conjunto de datos Iris contiene tres clases o tipos diferentes de especies de flores identificadas bajo la variable "Species" y se miden las dimensiones/tamaños de sus hojas (para nuestra ilustración, podemos perfectamente suponer que estas medidas de tamaño son equivalentes a las mediciones de distancia entre las ciudades).

```
data("iris")
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
tsp <- TSP(dist(iris[-5]), labels = iris[, "Species"])
unique(iris$Species)
```

```
## [1] setosa     versicolor virginica
## Levels: setosa versicolor virginica
```

Como hay 3 especies en el conjunto iris, es una buena razón para empezar con una agrupación de n = 3 clusters. Por más que tenemos una buena razón, estadisticamente hablando, no deja de ser una decisión inicial y arbitraria hasta que se demuestre lo contrario.

```
tsp_dummy <- insert_dummy(tsp, n = 3, label = "boundary")

set.seed(123)
tour_iris <- sapply(metodos, FUN = function(m) solve_TSP(tsp_dummy, method = m), simplify = FALSE)
```

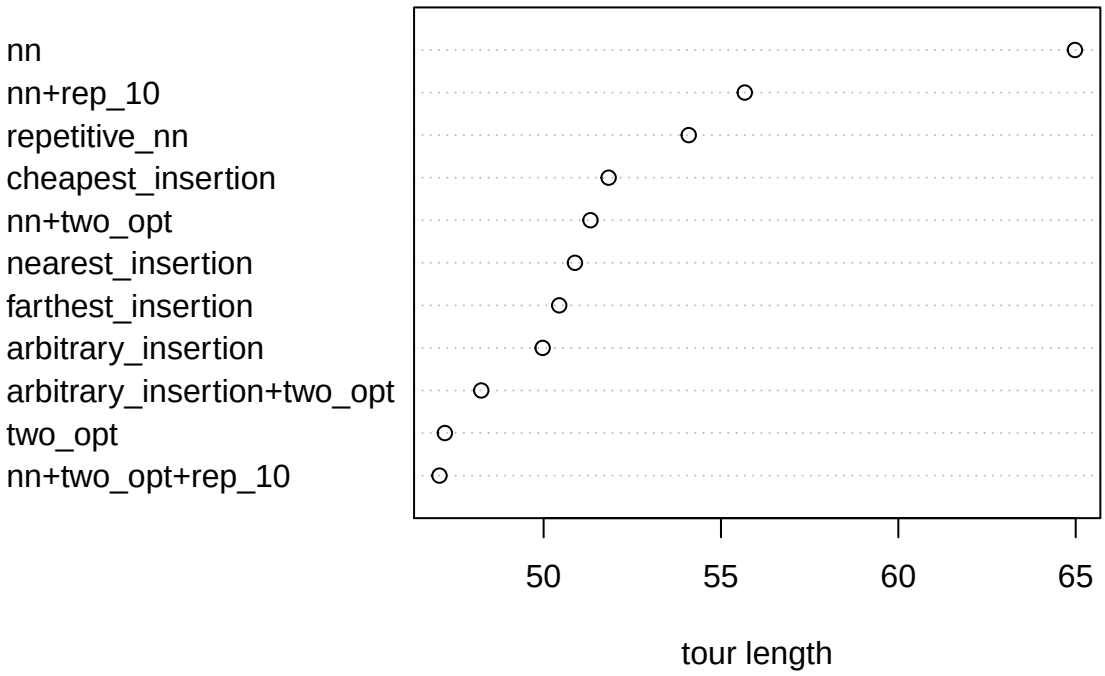
Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones:

```
tour_iris$'nn+two_opt' <- solve_TSP(tsp_dummy, method="nn", two_opt=TRUE)
tour_iris$'nn+rep_10' <- solve_TSP(tsp_dummy, method="nn", rep=10)
tour_iris$'nn+two_opt+rep_10' <- solve_TSP(tsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_iris$'arbitrary_insertion+two_opt' <- solve_TSP(tsp_dummy)
```

Visualizar resultados de las distancias generadas por cada método

```
dotchart(sort(c(sapply(tour_iris, tour_length))), xlab = "tour length")
title("Figura 8 - Tour Iris")
```

Figura 8 − Tour Iris




```
tour_iris[['nn+two_opt+rep_10']]
```

```
## object of class 'TOUR'
## result of method 'nn+two_opt_rep_10' for 153 cities
## tour length: 47.0642
```

Recuerda que por ahora no estamos agrupando ciudades por su cercania en millas, sino que estamos agrupando flores por tamaños mucho mas pequeños. Es un ejemplo que nos permitirá entender el siguiente punto más fácilmente.

A continuación, trazamos la matriz de distancia permutada del TSP utilizando sombreado de colores para representar distancias numéricas.La técnica de ilustrar distancias en colores es mucho más fácil de visualizar para el ojo humano que leer cientos de números en una tabla. El resultado que se muestra en la Figura 9 se interpreta de la siguiente forma:

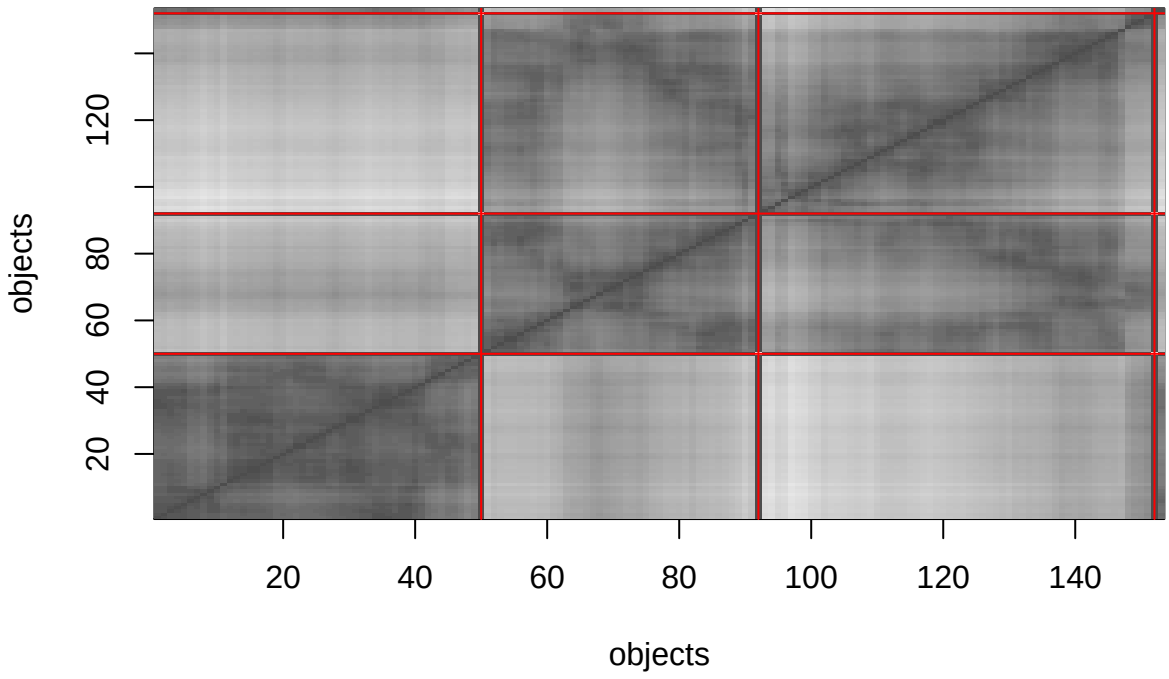
Las áreas más claras representan distancias grandes. Las líneas rojas representan las posiciones de los 3 "boundaries", que, CREEMOS, arbitrariamente, deberían marcan los límites de los clusters obtenidos (los límites entre 3 especies diferentes).

Matriz de distancia

```
tour_iris <- tour_iris[['nn+two_opt+rep_10']]

image(tsp_dummy, tour_iris, xlab = "objects", ylab ="objects")
title("Figura 9")
```

```
## dibujar líneas donde se ubican los limites de cluster "boundaries"

abline(h = which(labels(tour_iris)=="boundary"), col = "red")
abline(v = which(labels(tour_iris)=="boundary"), col = "red")
```

Figura 9



Tres líneas rojas horizontales y tres verticales separan exactamente la areas más oscuras de las más claras.

Podemos ver lo bien que la partición obtenida se ajusta a la estructura de los datos dados por el campo de las especies en el conjunto iris. Dado que usamos a la especie como etiquetas para reemplazar a las "ciudades" en el TSP, las etiquetas en la solución "tour_iris" representan la partición con las "ciudades ficticias" llamadas "boundary" (limites que separan a los diferentes clusters).

El resultado se puede resumir mirando la longitud obtenida bajo la etiqueta del cluster para cada tour obtenido:

```
out <- rle(labels(tour_iris))
data.frame(Species = out$values, Lenghts = out$lengths, Pos = cumsum(out$lengths))
```

```
##       Species Lenghts Pos
## 1      setosa      49  49
## 2    boundary       1  50
## 3   virginica       6  56
## 4  versicolor       1  57
## 5   virginica       1  58
## 6  versicolor       1  59
## 7   virginica       1  60
```

```
## 8  versicolor      23  83
## 9   virginica       8  91
## 10   boundary       1  92
## 11  virginica      33 125
## 12 versicolor      21 146
## 13  virginica       1 147
## 14 versicolor       4 151
## 15   boundary       1 152
## 16     setosa       1 153
```

2 boundary 1 50 <- boundary o “ciudad ficticia” número 1

. . .

10 boundary 1 92 <- boundary o “ciudad ficticia” número 2

. . .

15 boundary 1 152 <- boundary o “ciudad ficticia” número 3

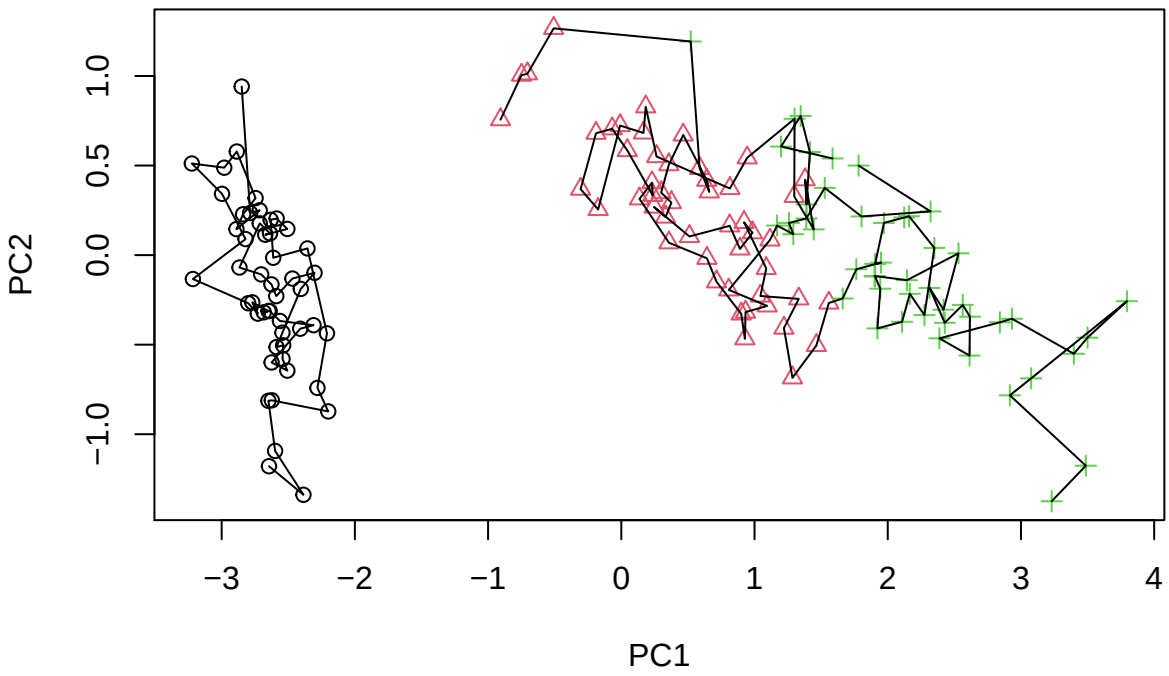
. . .

La especie setosa esta perfectamente separada de las otros dos clusters.

La razón por la que la agrupación por reordenamiento falla para dividir los datos en tres grupos perfectos es la cercanía entre las especies Virginica y Versicolor. Para inspeccionar este problema aún más, podemos proyectar los puntos de datos componentes principales del conjunto de datos y agregar los segmentos de trazado que resultaron del resolver de TSP.

```
prc <- prcomp(iris[1:4])
plot(prc$x, pch = as.numeric(iris[,5]), col = as.numeric(iris[,5]))
indices <- c(tour_iris, tour_iris[1])
indices[indices > 150] <- NA
lines(prc$x[indices,])
title("Figura 10")
```

Figura 10



El resultado se muestra en la Figura 10. Las tres especies se identifican por diferentes marcadores y todos los puntos conectados por una sola ruta representan una agrupación o cluster encontrado. Claramente, los dos grupos a la derecha están demasiado cerca para ser separados correctamente utilizando sólo las distancias entre puntos individuales. Este problema es similar al efecto de encadenamiento conocido en la agrupación jerárquica utilizando el método de un solo vínculo.

HABIENDO DEMOSTRADO LO QUE HAREMOS, volvemos al dilema del vendedor viajero, nuevamente a medir distancias entre ciudades en millas:

Para aplicar clustering a las ciudades, pensemos en iniciar con un n = 4 clusters en base a las principales zonas horarias principales de USA (Este, Centro, Montaña, Pacifico). Esto simplemente se me ocurre de forma completamente arbitraria.

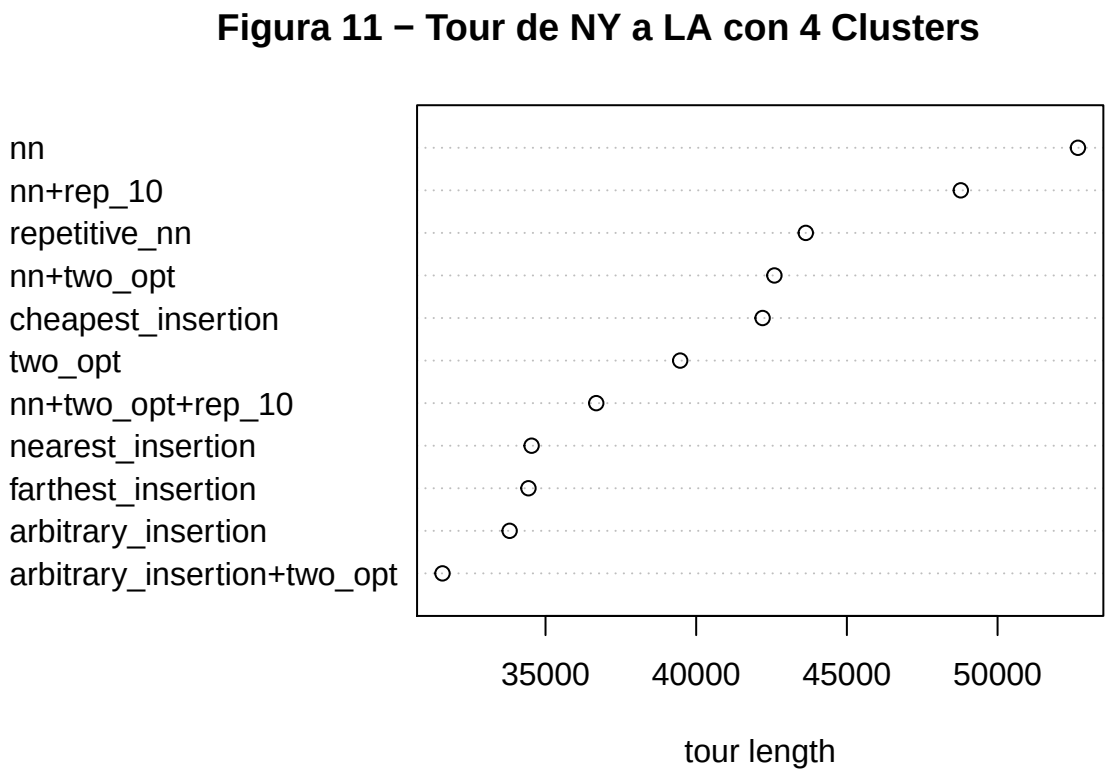
```
atsp_dummy <- insert_dummy(atsp, n = 4, label = "boundary")

set.seed(123)
tour_clusters4 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE)
```

Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones

```
tour_clusters4$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters4$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters4$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters4$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)
```

Visualizar resultados de las distancias generadas por cada método

```
dotchart(sort(c(sapply(tour_clusters4, tour_length))), xlab = "tour length")
title("Figura 11 - Tour de NY a LA con 4 Clusters")
```




```
tour_clusters4[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 315 cities
## tour length: 31581
```

Bastante mejor que la solución óptima de concorde sin clusters que daba un tour length de 38.489 millas.

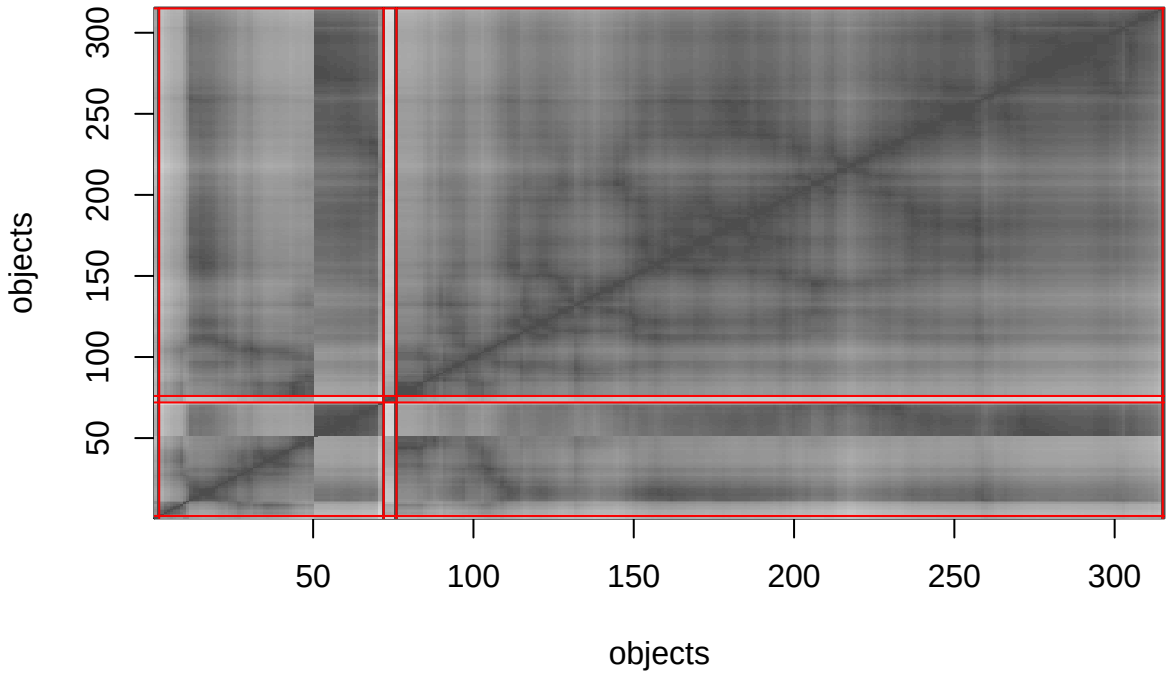
¿Podemos mejorar aún más?

A continuación, trazamos la matriz de distancia permutada del TSP utilizando el sombreado para representar distancias.El resultado se muestra en la Figura 12. Las áreas más claras representan distancias grandes. Las líneas rojas representan las posiciones de las ciudades ficticias en la gira, que marcan los límites de los clusters obtenidos.

Matrix de distancia

```
tour_clusters4 <- tour_clusters4[['arbitrary_insertion+two_opt']]

image(atsp_dummy, tour_clusters4, xlab = "objects", ylab ="objects")
title("Figura 12")

# creados con líneas rojas que es donde se ubican las ciudades ficticias

abline(h = which(labels(tour_clusters4)=="boundary"), col = "red")
abline(v = which(labels(tour_clusters4)=="boundary"), col = "red")
```

Figura 12



Parecen quedar al menos 1 línea vertical y horizontal claramente sin resaltar en rojo.

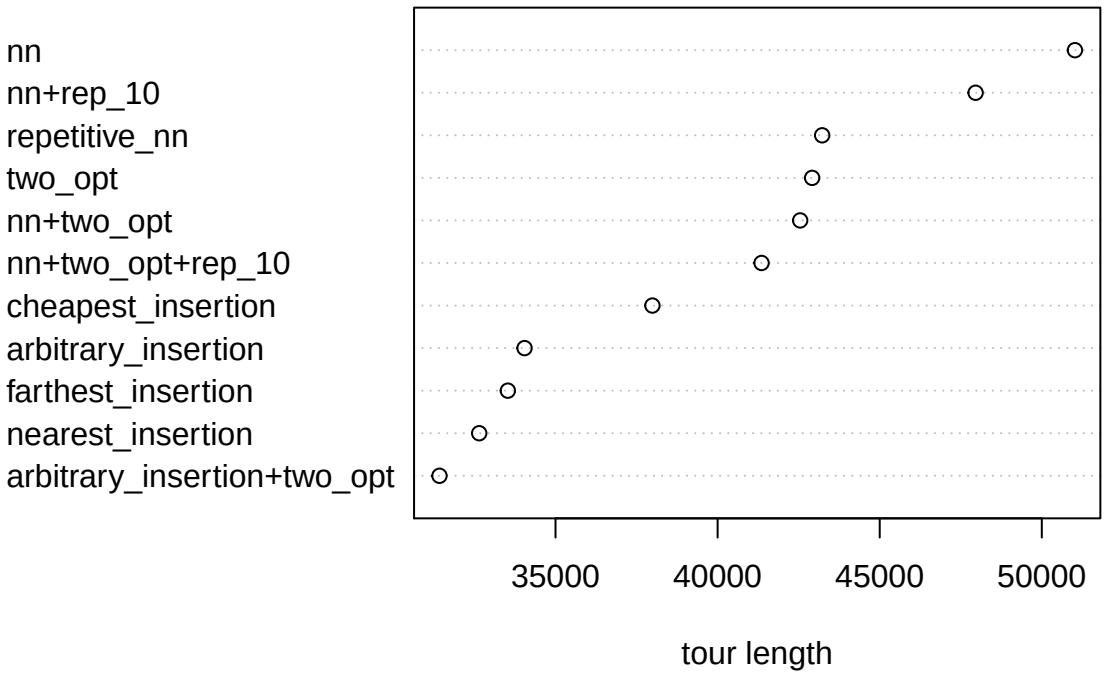
Sumemoslas: n = 4 + 1 y usemos 5 boundaries entonces.

```
atsp_dummy <- insert_dummy(atsp, n = 5, label = "boundary")

set.seed(123)
tour_clusters5 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE
```

Agregamos algunos recorridos usando repeticiones y refinamientos de dos opciones

```
tour_clusters5$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters5$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters5$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters5$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)

#Visualizar resultados de las distancias generadas por cada método
dotchart(sort(c(sapply(tour_clusters5, tour_length))), xlab = "tour length")
title("Figura 13 - Tour de NY a LA con 5 Clusters")
```

Figura 13 − Tour de NY a LA con 5 Clusters



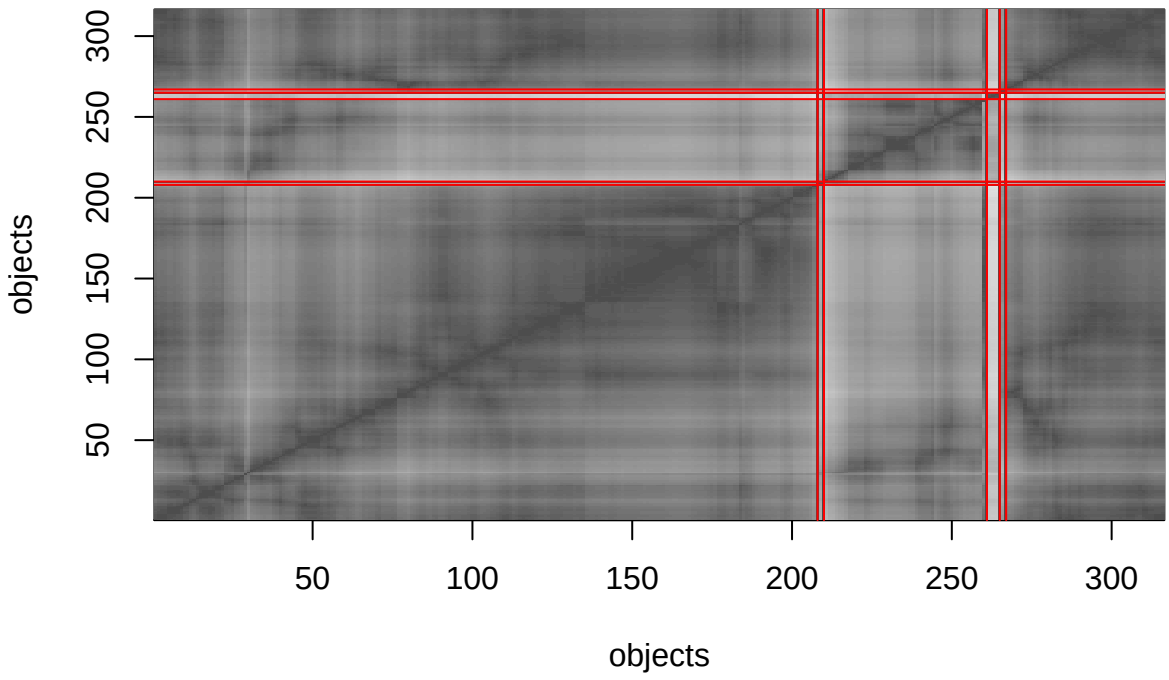

```
tour_clusters5[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 316 cities
## tour length: 31421
```

Reducción de 100 millas, nos ayudó ir de 4 clusters a 5. Sigamos comprobando:

```
tour_clusters5 <- tour_clusters5[['arbitrary_insertion+two_opt']]

image(atsp_dummy, tour_clusters5, xlab = "objects", ylab ="objects")
title("Figura 14")

abline(h = which(labels(tour_clusters5)=="boundary"), col = "red")
abline(v = which(labels(tour_clusters5)=="boundary"), col = "red")
```

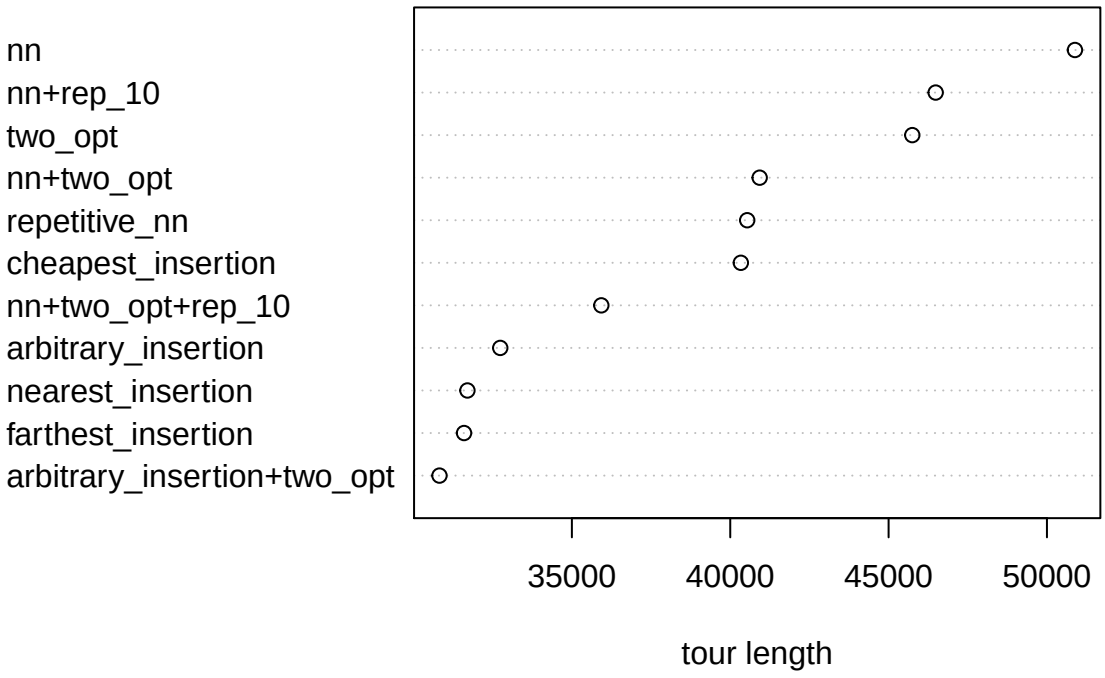
Figura 14



Aun podemos ver 1 línea horizontal y 1 vertical (tal vez con un leve o claro contraste formando 1 cruz) sin resaltar en rojo. Hagamos un intento con n = 6 boundaries.

```
atsp_dummy <- insert_dummy(atsp, n = 6, label = "boundary")

set.seed(123)
tour_clusters6 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE)

tour_clusters6$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters6$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters6$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters6$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)

#Visualizar resultados de las distancias generadas por cada método
dotchart(sort(c(sapply(tour_clusters6, tour_length))), xlab = "tour length")
title("Figura 15 - Tour de NY a LA con 6 Clusters")
```

Figura 15 − Tour de NY a LA con 6 Clusters



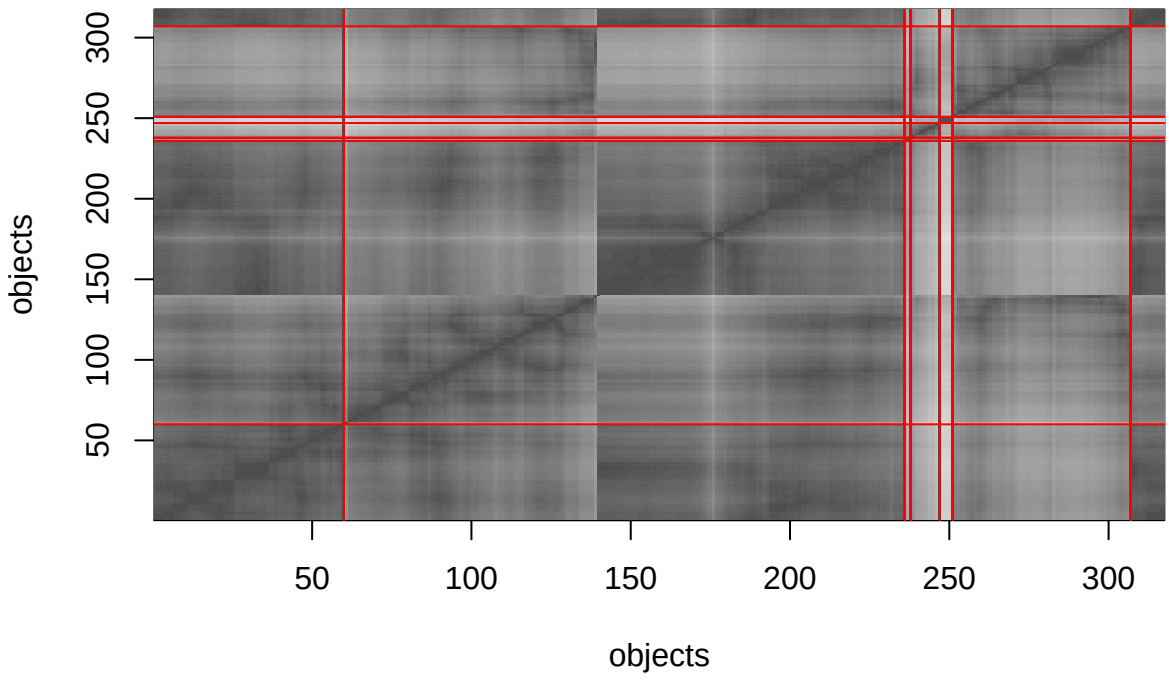

```
tour_clusters6[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 317 cities
## tour length: 30819
```

Reducción de 600 millas aproximadamente, sigamos comprobando con más límites o boundaries para nuestros clusters:

```
tour_clusters6 <- tour_clusters6[['arbitrary_insertion+two_opt']]

image(atsp_dummy, tour_clusters6, xlab = "objects", ylab ="objects")
title("Figura 16")

abline(h = which(labels(tour_clusters6)=="boundary"), col = "red")
abline(v = which(labels(tour_clusters6)=="boundary"), col = "red")
```

Figura 16

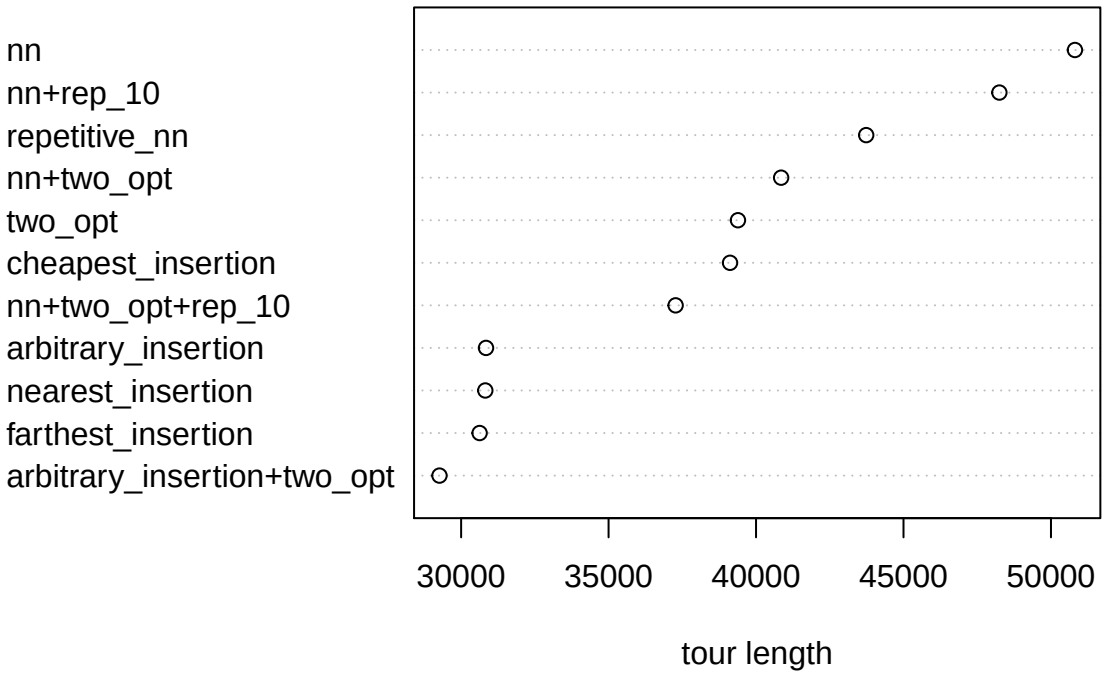


Reducción de Todavía nos queda al menos 1 cruz claramente SIN resaltar en rojo, hagamos un intento de mejora adicional con n = 7.

(Yo veo 2 cruces en realidad, la segunda muy clarita, que por ahora voy a ignorar).

```
atsp_dummy <- insert_dummy(atsp, n = 7, label = "boundary")

set.seed(123)
tour_clusters7 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE)

tour_clusters7$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters7$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters7$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters7$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)

#Visualizar resultados de las distancias generadas por cada método
dotchart(sort(c(sapply(tour_clusters7, tour_length))), xlab = "tour length")
title("Figura 17 - Tour de NY a LA con 7 Clusters")
```

Figura 17 − Tour de NY a LA con 7 Clusters



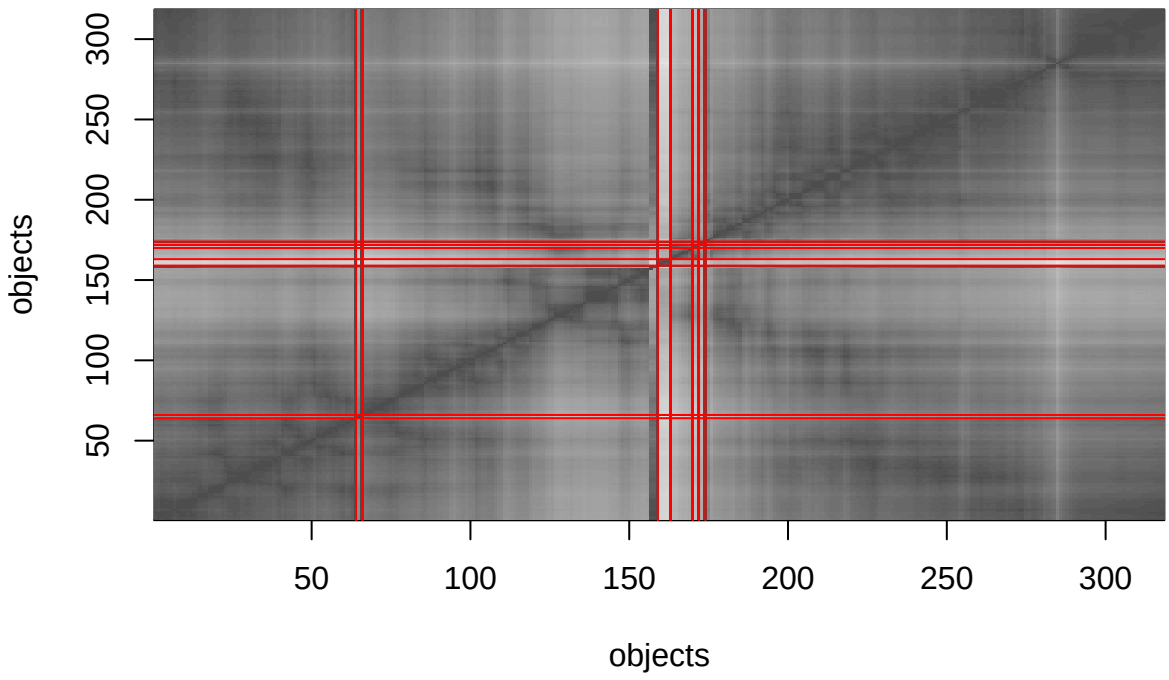

```
tour_clusters7[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 318 cities
## tour length: 29266
```

Rompimos por debajo de 30 mil millas, hemos logrado notables mejoras pasando de 4 a 7 agrupaciones o clusters de forma "supervisada" (o sea, manual, no estamos utilizando machine learning ni algoritmos autónomo como hemos visto con arboles de decisión).

```
tour_clusters7 <- tour_clusters7[['arbitrary_insertion+two_opt']]

image(atsp_dummy, tour_clusters7, xlab = "objects", ylab ="objects")
title("Figura 18")

abline(h = which(labels(tour_clusters7)=="boundary"), col = "red")
abline(v = which(labels(tour_clusters7)=="boundary"), col = "red")
```

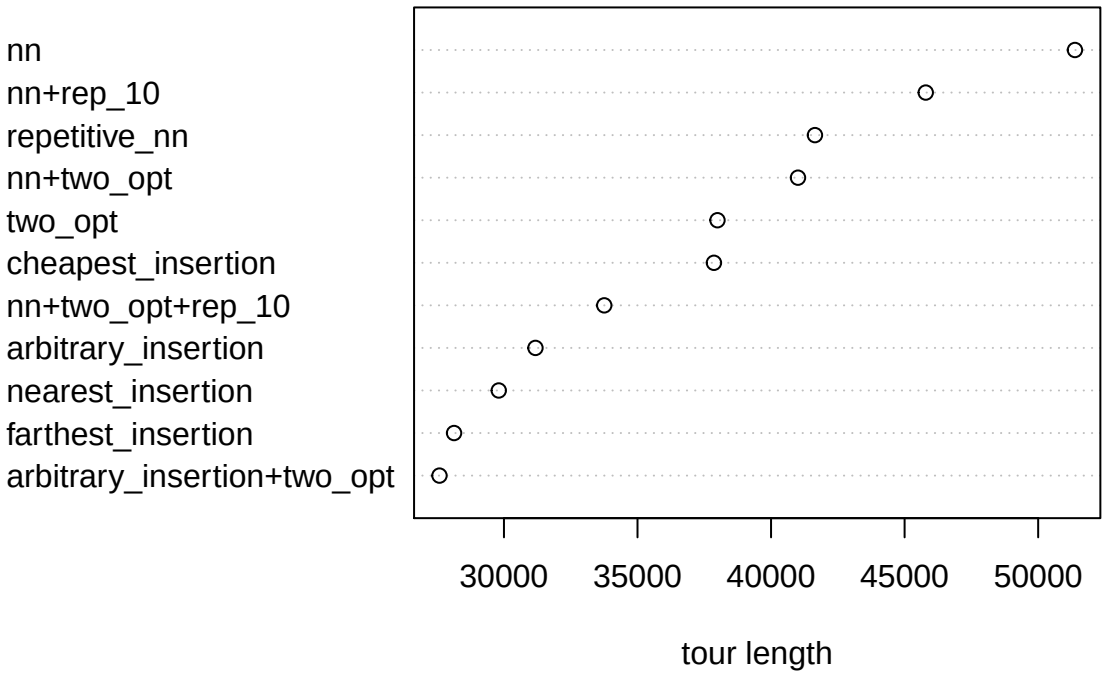
Figura 18



Aún queda 1 cruz clara SIN resaltar en rojo, con una mejora de más de 1000 millas entre 6 y 7 clusters, vale la pena intentar con n = 8.

```
atsp_dummy <- insert_dummy(atsp, n = 8, label = "boundary")

set.seed(123)
tour_clusters8 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE)

tour_clusters8$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters8$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters8$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters8$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)

#Visualizar resultados de las distancias generadas por cada método
dotchart(sort(c(sapply(tour_clusters8, tour_length))), xlab = "tour length")
title("Figura 19 - Tour de NY a LA con 8 Clusters")
```

Figura 19 − Tour de NY a LA con 8 Clusters



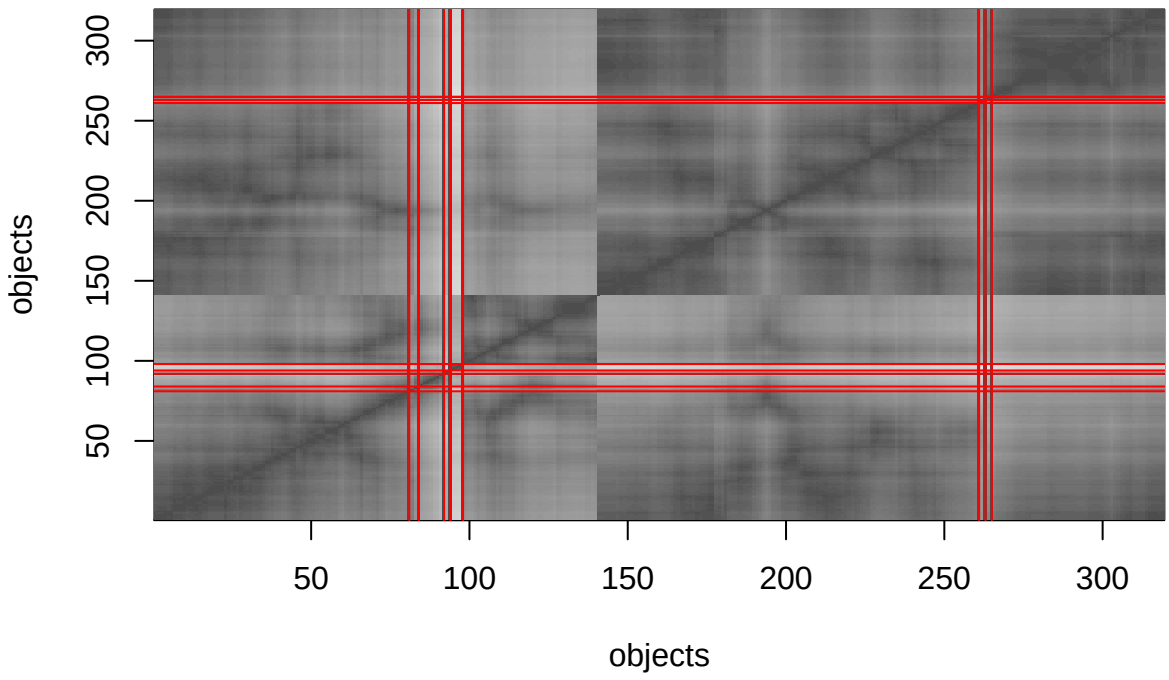

```
tour_clusters8[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 319 cities
## tour length: 27586
```

Otra mejora de más de 1000 millas, veamos la matriz de distancias para ver si podemos mejorar más.

```
tour_clusters8 <- tour_clusters8[['arbitrary_insertion+two_opt']]

image(atsp_dummy, tour_clusters8, xlab = "objects", ylab ="objects")
title("Figura 20")

abline(h = which(labels(tour_clusters8)=="boundary"), col = "red")
abline(v = which(labels(tour_clusters8)=="boundary"), col = "red")
```

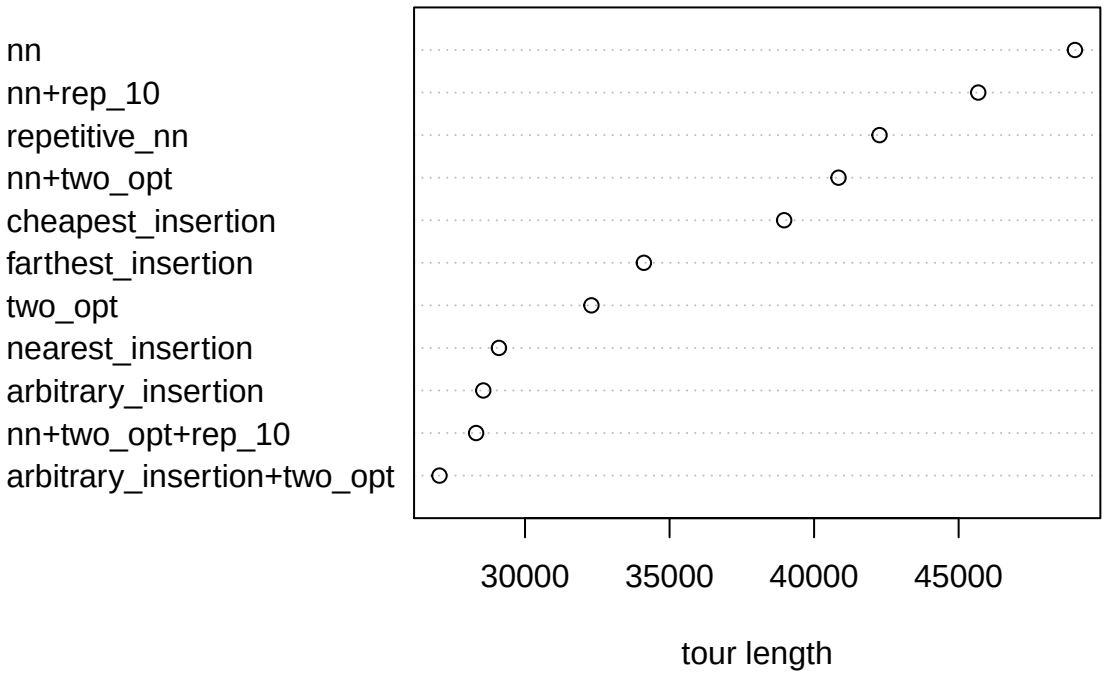
Figura 20



Sí, hay una clara cruz todavía SIN resaltar, por lo que hay que intentar con n = 9 entonces.

```
atsp_dummy <- insert_dummy(atsp, n = 9, label = "boundary")

set.seed(123)
tour_clusters9 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE)

tour_clusters9$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters9$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters9$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters9$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)

#Visualizar resultados de las distancias generadas por cada método
dotchart(sort(c(sapply(tour_clusters9, tour_length))), xlab = "tour length")
title("Figura 21 - Tour de NY a LA con 9 Clusters")
```

Figura 21 − Tour de NY a LA con 9 Clusters



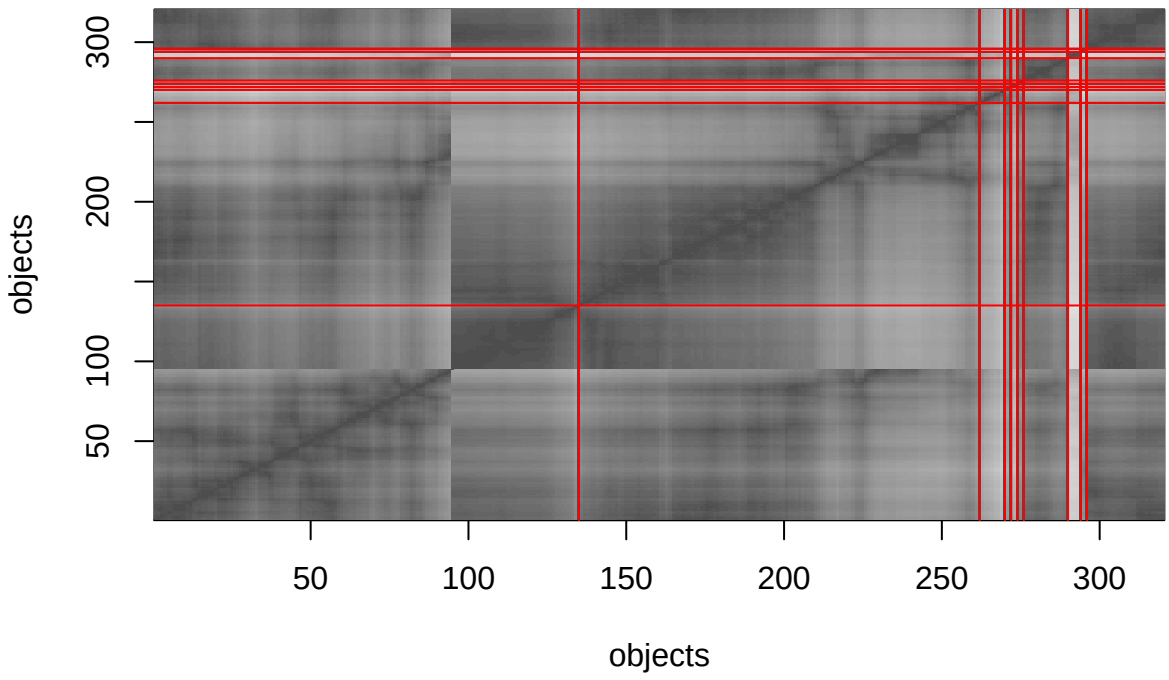

```
tour_clusters9[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 320 cities
## tour length: 27040
```

La mejora ahora es cerca de 500 millas, puede ser que ya tengamos el óptimo número de cluster, comprobemos con la matriz de distancias representadas con sombras.

```
tour_clusters9 <- tour_clusters9[['arbitrary_insertion+two_opt']]

image(atsp_dummy, tour_clusters9, xlab = "objects", ylab ="objects")
title("Figura 22")

abline(h = which(labels(tour_clusters9)=="boundary"), col = "red")
abline(v = which(labels(tour_clusters9)=="boundary"), col = "red")
```

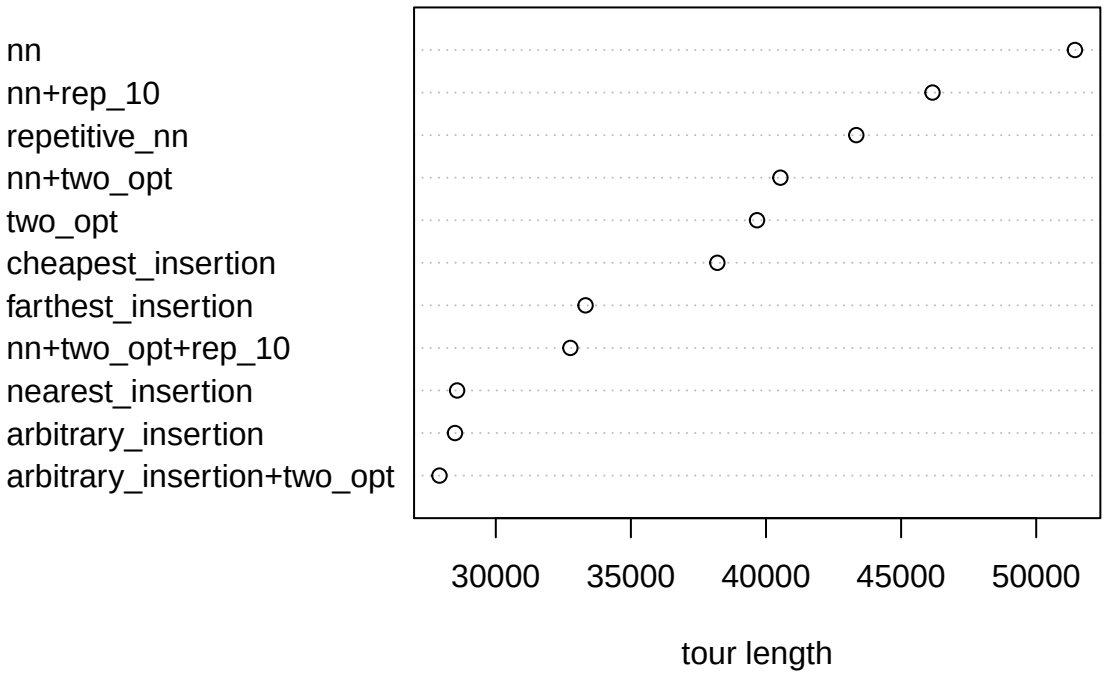
Figura 22



Aún hay una cruz, la mejora no será mucha, pero veamos que nos da TSP con n = 10 de todas formas.

```
atsp_dummy <- insert_dummy(atsp, n = 10, label = "boundary")

set.seed(123)
tour_clusters10 <- sapply(metodos, FUN = function(m) solve_TSP(atsp_dummy, method = m), simplify = FALSE

tour_clusters10$'nn+two_opt' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE)
tour_clusters10$'nn+rep_10' <- solve_TSP(atsp_dummy, method="nn", rep=10)
tour_clusters10$'nn+two_opt+rep_10' <- solve_TSP(atsp_dummy, method="nn", two_opt=TRUE, rep=10)
tour_clusters10$'arbitrary_insertion+two_opt' <- solve_TSP(atsp_dummy)

#Visualizar resultados de las distancias generadas por cada método
dotchart(sort(c(sapply(tour_clusters10, tour_length))), xlab = "tour length")
title("Figura 23 - Tour de NY a LA con 10 Boundaries")
```

Figura 23 − Tour de NY a LA con 10 Boundaries




```
tour_clusters10[['arbitrary_insertion+two_opt']]
```

```
## object of class 'TOUR'
## result of method 'arbitrary_insertion+two_opt' for 321 cities
## tour length: 27915
```

Con un n = 10, hemos empeorado tour_length, por lo tanto el numero de clusters óptimo es n = 9.

```
path_labels <- c("New York, NY", labels(cut_tour(tour_clusters9, la_ny)),"Los Angeles, CA")
path_ids <- match(path_labels, labels(USCA312))
head(path_labels)
```

```
## [1] "New York, NY"     "Jersey City, NJ"  "Elizabeth, NJ"    "Newark, NJ"
## [5] "Paterson, NJ"     "White Plains, NY"
```

```
tail(path_labels)
```

```
## [1] "El Paso, TX"     "Tucson, AZ"      "Phoenix, AZ"     "Yuma, AZ"
## [5] "San Diego, CA"   "Los Angeles, CA"
```

El recorrido completo es el siguiente:

```
path_labels
```

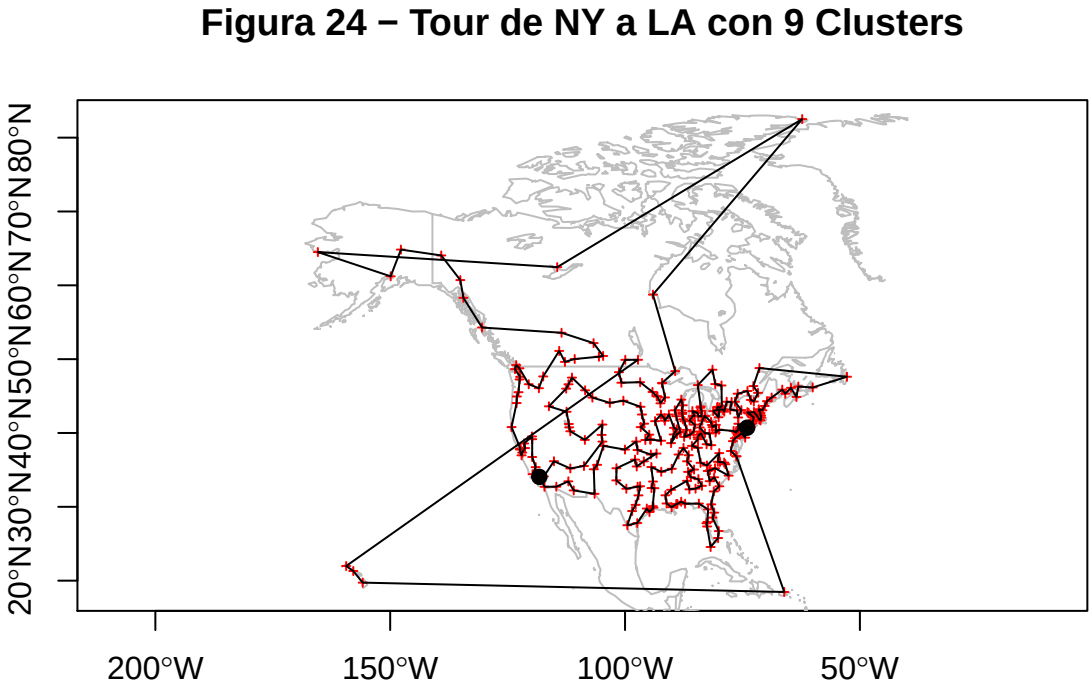
```
##   [1] "New York, NY"          "Jersey City, NJ"      "Elizabeth, NJ"
##   [4] "Newark, NJ"            "Paterson, NJ"         "White Plains, NY"
##   [7] "Stamford, CT"          "Central Islip, NY"    "Bridgeport, CT"
##  [10] "New Haven, CT"         "Meriden, CT"          "New Britain,CT"
##  [13] "Hartford, CT"          "Springfield, MA"      "Pittsfield, MA"
##  [16] "Albany, NY"            "Schenectady, NY"      "Troy, NY"
##  [19] "Brattleboro, VT"       "Worcester, MA"        "Providence, RI"
##  [22] "Fall River, MA"        "Brockton, MA"         "Boston, MA"
##  [25] "Cambridge, MA"         "Lowell, MA"           "Lawrence, MA"
##  [28] "Manchester, NH"        "Concord, NH"          "Portsmouth, NH"
##  [31] "Portland, ME"          "Augusta, ME"          "Bangor, ME"
##  [34] "Fredericton, NB"       "Saint John, NB"       "Moncton, NB"
##  [37] "Halifax, NS"           "Charlottetown, PE"    "Syndey, NS"
##  [40] "Saint John's, NF"      "boundary"             "Quebec City, QC"
##  [43] "Trois-Rivieres, QC"    "Sherbrooke, QC"       "Montpelier, VT"
##  [46] "Burlington, VT"        "Montreal, QC"         "Ottawa, ON"
##  [49] "Kingston, ON"          "Utica, NY"            "Binghamtom, NY"
##  [52] "Syracuse, NY"          "Rochester, NY"        "Belleville, ON"
##  [55] "Peterborough, ON"      "Buffalo, NY"          "Niagara Falls, ON"
##  [58] "Saint Catherines, ON" "Hamilton, ON"         "Erie, PA"
##  [61] "London, ON"            "Brantford, ON"        "Kitchener, ON"
##  [64] "Guelph, ON"            "Burlington, ONT"      "Toronto, ON"
##  [67] "North Bay, ON"         "Sudbury, ON"          "Timmins, ON"
##  [70] "Sault Ste Marie, ON"   "Bay City, MI"         "Saginaw, MI"
##  [73] "Flint, MI"             "Ann Arbor, MI"        "Jackson, MI"
##  [76] "Lansing, MI"           "Grand Rapids, MI"     "Battle Creek, MI"
##  [79] "Kalamazoo, MI"         "South Bend, IN"       "Ft Wayne, IN"
##  [82] "Muncie, IN"            "Indianapolis, IN"     "Terre Haute, IN"
##  [85] "Lafayette, IN"         "Gary, IN"             "Chicago, IL"
##  [88] "Kenosha, WI"           "Racine, WI"           "Milwaukee, WI"
##  [91] "Sheboygan, WI"         "Green Bay, WI"        "Madison, WI"
##  [94] "Rockford, IL"          "Champaign, IL"        "Urbana, IL"
##  [97] "Decatur, IL"           "Saint Louis, MO"      "Springfield, IL"
## [100] "Bloomington, IL"       "Peoria, IL"           "Dubuque, IA"
## [103] "Iowa City, IA"         "Cedar Rapids, IA"     "Waterloo, IA"
## [106] "Des Moines, IA"        "Columbia, MO"         "Kansas City, MO"
## [109] "Kansas City, KS"       "Topeka, KS"           "Saint Joseph, MO"
## [112] "Lincoln, NE"           "Omaha, NE"            "Sioux City, IA"
## [115] "Sioux Falls, SD"       "Pierre, SD"           "Rapid City, SD"
## [118] "Sheridan, WY"          "Billings, MT"         "Great Falls, MT"
## [121] "Helena, MT"            "Butte, MT"            "Boise, ID"
## [124] "Pocatello, ID"         "Ogden, UT"            "Salt Lake City, UT"
## [127] "Provo, UT"             "Grand Junction, CO"   "Cheyenne, WY"
## [130] "Denver, CO"            "Colorado Springs, CO" "Gallup, NM"
## [133] "Flagstaff, AZ"         "Las Vegas, NV"        "San Bernardino, CA"
## [136] "Pasadena, CA"          "Santa Barbara, CA"    "Bakersfield, CA"
## [139] "Fresno, CA"            "Carson City, NV"      "Reno, NV"
## [142] "Sacramento, CA"        "Stockton, CA"         "San Jose, CA"
## [145] "Santa Cruz, CA"        "San Francisco, CA"    "Oakland, CA"
## [148] "Berkeley, CA"          "Eureka, CA"           "Eugene, OR"
## [151] "Salem, OR"             "Portland, OR"         "Tacoma, WA"
```

```
## [154] "Seattle, WA"          "Victoria, BC"         "Vancouver, BC"
## [157] "Bellingham, WA"       "Yakima, WA"           "Walla Walla, WA"
## [160] "Spokane, WA"          "Calgary, AB"          "Lethbridge, AB"
## [163] "Medicine Hat, AB"     "Moose Jaw, SK"        "Regina, SK"
## [166] "Saskatoon, SK"        "Edmonton, AB"         "boundary"
## [169] "Prince Rupert, BC"    "Juneau, AK"           "Whitehorse, YK"
## [172] "Dawson, YT"           "Fairbanks, AK"        "Anchorage, AK"
## [175] "Nome, AK"             "boundary"             "Yellowknife, NT"
## [178] "boundary"             "Alert, NT"            "boundary"
## [181] "Churchill, MB"        "boundary"             "Thunder Bay, ON"
## [184] "Duluth, MN"           "Superior, WI"         "Eau Claire, WI"
## [187] "Rochester, MN"        "Saint Paul, MN"       "Minneapolis, MN"
## [190] "Saint Cloud, MN"      "Fargo, ND"            "Bismarck, ND"
## [193] "Minot, ND"            "Brandon, MB"          "Winnipeg, MB"
## [196] "boundary"             "Lihue, HI"            "Honolulu, HI"
## [199] "Hilo, HI"             "boundary"             "San Juan, PR"
## [202] "boundary"             "Norfolk, VA"          "Portsmouth, VA"
## [205] "Richmond, VA"         "Washington, DC"       "Baltimore, MD"
## [208] "Wilmington, DE"       "Philadelphia, PA"     "Atlantic City, NJ"
## [211] "Trenton, NJ"          "Allentown, PA"        "Wilkes-Barre, PA"
## [214] "Reading, PA"          "Johnstown, PA"        "Lancaster, PA"
## [217] "Harrisburg, PA"       "Pittsburgh, PA"       "Wheeling, WV"
## [220] "Steubenville, OH"     "Youngstown, OH"       "Canton, OH"
## [223] "Akron, OH"            "Cleveland, OH"        "Windsor, ON"
## [226] "Detroit, MI"          "Toledo, OH"           "Lima, OH"
## [229] "Columbus, OH"         "Zanesville, OH"       "Charleston, WV"
## [232] "Ashland, KY"          "Springfield, OH"      "Dayton, OH"
## [235] "Hamilton, OH"         "Cincinnati, OH"       "Louisville, KY"
## [238] "Lexington, KY"        "Knoxville, TN"        "Asheville, NC"
## [241] "Roanoke, VA"          "Winston-Salem, NC"    "Greensboro, NC"
## [244] "Durham, NC"           "Raleigh, NC"          "Wilmington, NC"
## [247] "Charlotte, NC"        "Spartanburg, NC"      "Greenville, SC"
## [250] "Augusta, GA"          "Columbia, SC"         "Charleston, SC"
## [253] "Savannah, GA"         "Jacksonville, FL"     "Daytona Beach, FL"
## [256] "Orlando, FL"          "West Palm Beach, FL"  "Miami, FL"
## [259] "Key West, FL"         "Sarasota, FL"         "Saint Petersburg, FL"
## [262] "Tampa, FL"            "Gainesville, FL"      "Tallahassee, FL"
## [265] "Pensacola, FL"        "Mobile, AL"           "Biloxi, MS"
## [268] "Gulfport, MS"         "New Orleans, LA"      "Baton Rouge, LA"
## [271] "Natchez, MS"          "Jackson, MS"          "Birmingham, AL"
## [274] "Montgomery, AL"       "Columbus, GA"         "Macon, GA"
## [277] "Atlanta, GA"          "Gadsden, AL"          "Huntsville, AL"
## [280] "Chattanooga, TN"      "Nashville, TN"        "Bowling Green, KY"
## [283] "Evansville, IN"       "Paducah, KY"          "Memphis, TN"
## [286] "Little Rock, AR"      "Ft Smith, AR"         "Texarkana, TX"
## [289] "Marshall, TX"         "Shreveport, LA"       "Beaumont, TX"
## [292] "Port Arthur, TX"      "Galveston, TX"        "Houston, TX"
## [295] "Corpus Christi, TX"   "Laredo, TX"           "San Antonio, TX"
## [298] "Austin, TX"           "Waco, TX"             "Dallas, TX"
## [301] "Ft Worth, TX"         "Abilene, TX"          "Lubbock, TX"
## [304] "Amarillo, TX"         "Enid, OK"             "Oklahoma City, OK"
## [307] "Tulsa, OK"            "Springfield, MO"      "Joplin, MO"
## [310] "Wichita, KS"          "Salina, KS"           "Dodge City, KS"
## [313] "Pueblo, CO"           "Santa Fe, NM"         "Albuquerque, NM"
```

```
## [316] "El Paso, TX"          "Tucson, AZ"          "Phoenix, AZ"
## [319] "Yuma, AZ"             "San Diego, CA"       "Los Angeles, CA"
```

Removemos los boundaries para poder ilustrar la ruta sobre el mapa

```
x <- path_labels
x <- x[x != "boundary"]

path_ids_no_NA <- match(x, labels(USCA312))

plot_path(path_ids_no_NA)
title("Figura 24 - Tour de NY a LA con 9 Clusters")
```



No parece nada simple, lo viajes de un cluster a otro no se cuenta, veamos entonce de graficar los clusters por separado.

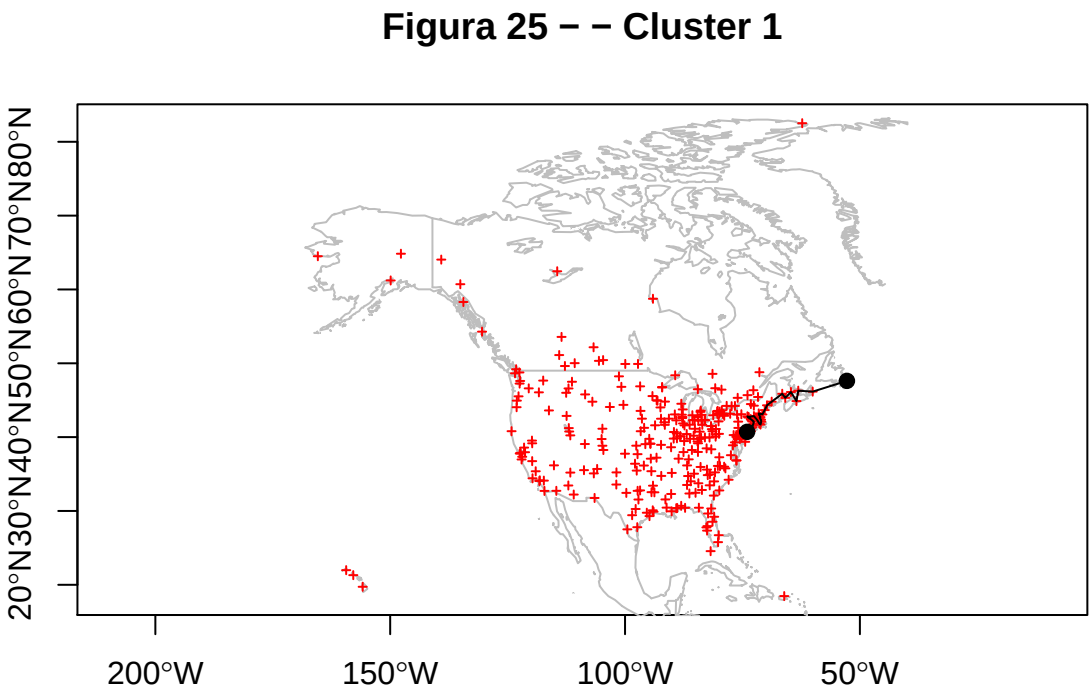
Mirando la ruta completa, vemos que los "boundary" están en los indices de posiciones [41], [168], [176], [178], [180], [182], [196], [200], y [202]. Por ende, 9 boundaries no dan 10 clusters:

```
cluster1 <- path_labels[1:40]
cluster2 <- path_labels[42:167]
cluster3 <- path_labels[169:175]
cluster4 <- path_labels[177]
cluster5 <- path_labels[179]
cluster6 <- path_labels[181]
cluster7 <- path_labels[183:195]
```

```
cluster8 <- path_labels[197:199]
cluster9 <- path_labels[201]
cluster10 <- path_labels[203:321]
```

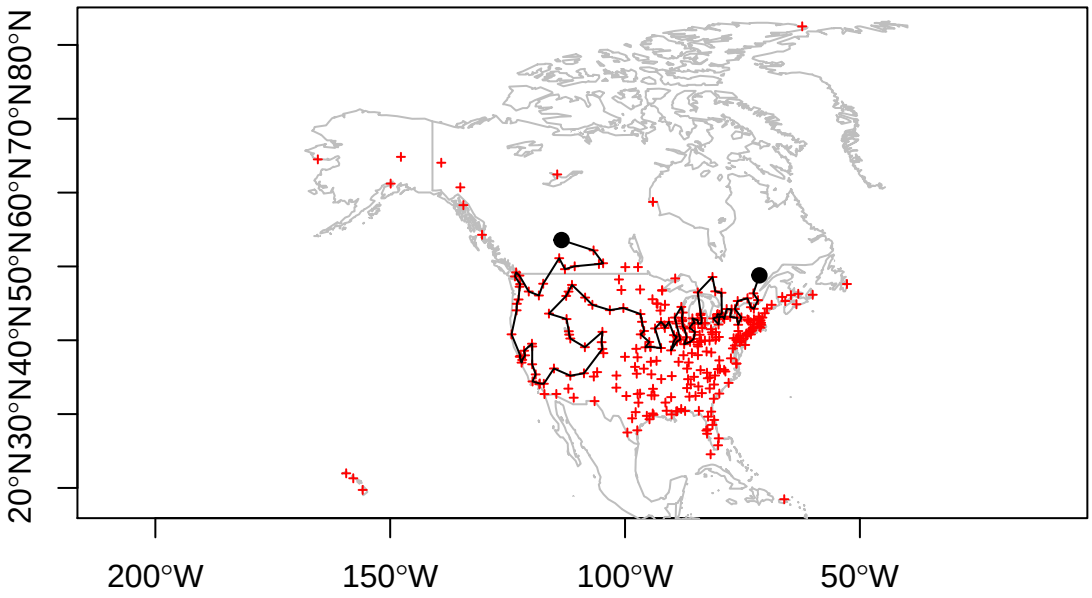
¿Nos sobra un cluster? Tenemos 9 boundaries, sigamos investigando los resultados de TSP.

```
cluster1_ids <- match(cluster1, labels(USCA312))
plot_path(cluster1_ids)
title("Figura 25 - - Cluster 1")
```



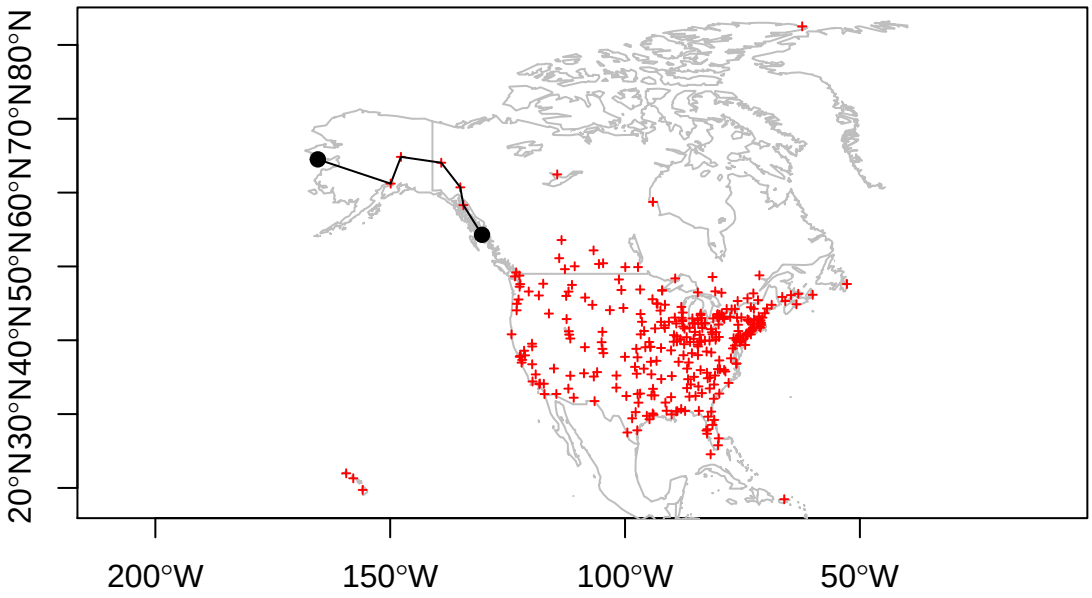

```
cluster2_ids <- match(cluster2, labels(USCA312))
plot_path(cluster2_ids)
title("Figura 26 - Cluster 2")
```

Figura 26 − Cluster 2



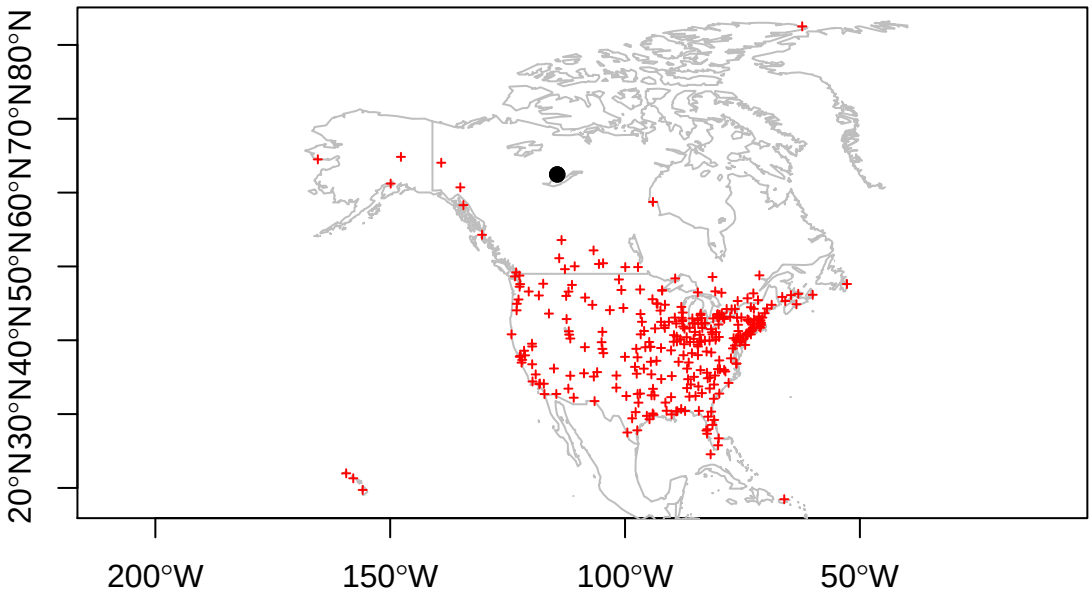

```
cluster3_ids <- match(cluster3, labels(USCA312))
plot_path(cluster3_ids)
title("Figura 27 - Cluster 3")
```

Figura 27 − Cluster 3



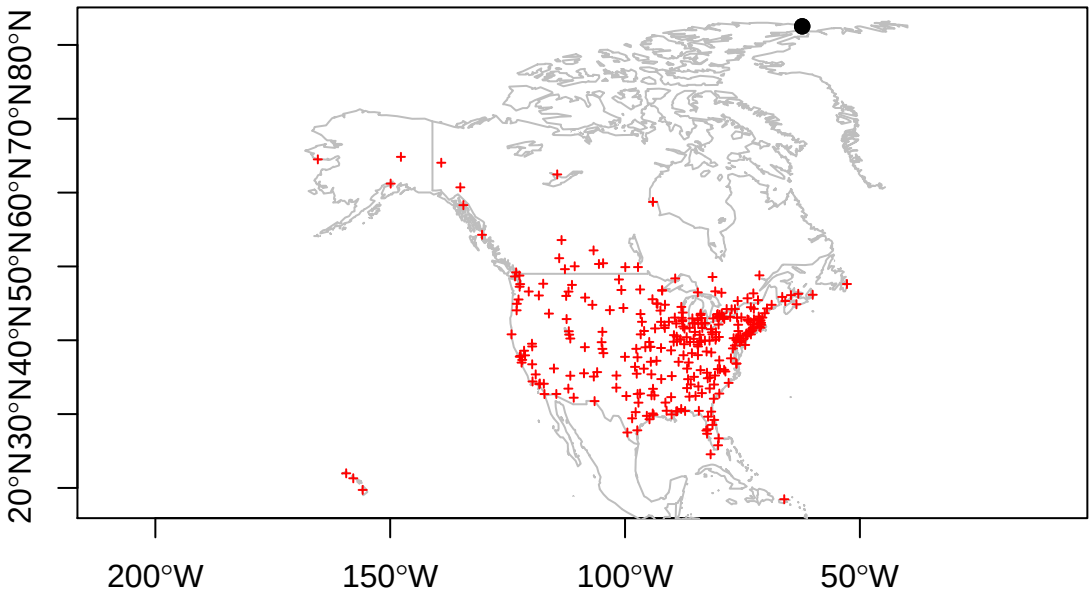

```
cluster4_ids <- match(cluster4, labels(USCA312))
plot_path(cluster4_ids)
title("Figura 28 - Cluster 4")
```

Figura 28 − Cluster 4



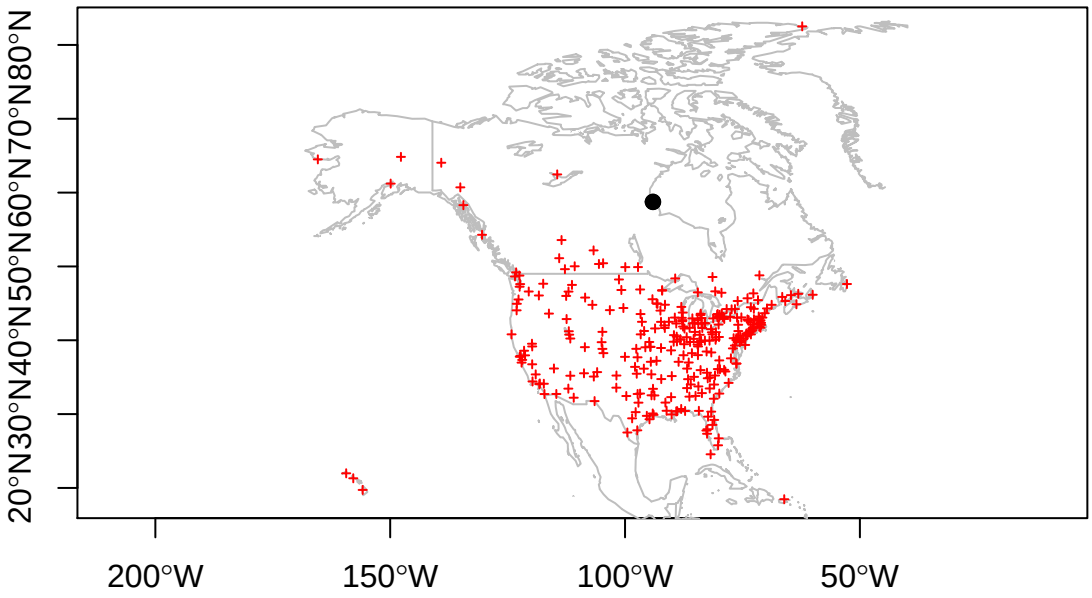

```
cluster5_ids <- match(cluster5, labels(USCA312))
plot_path(cluster5_ids)
title("Figura 29 - Cluster 5")
```

Figura 29 − Cluster 5



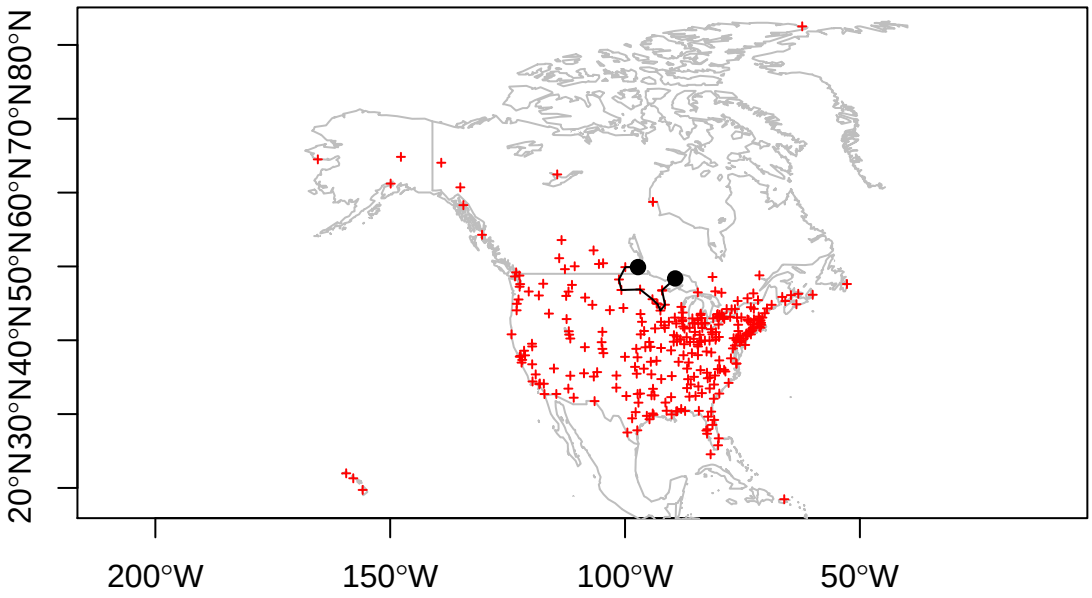

```
cluster6_ids <- match(cluster6, labels(USCA312))
plot_path(cluster6_ids)
title("Figura 30 - Cluster 6")
```

Figura 30 − Cluster 6



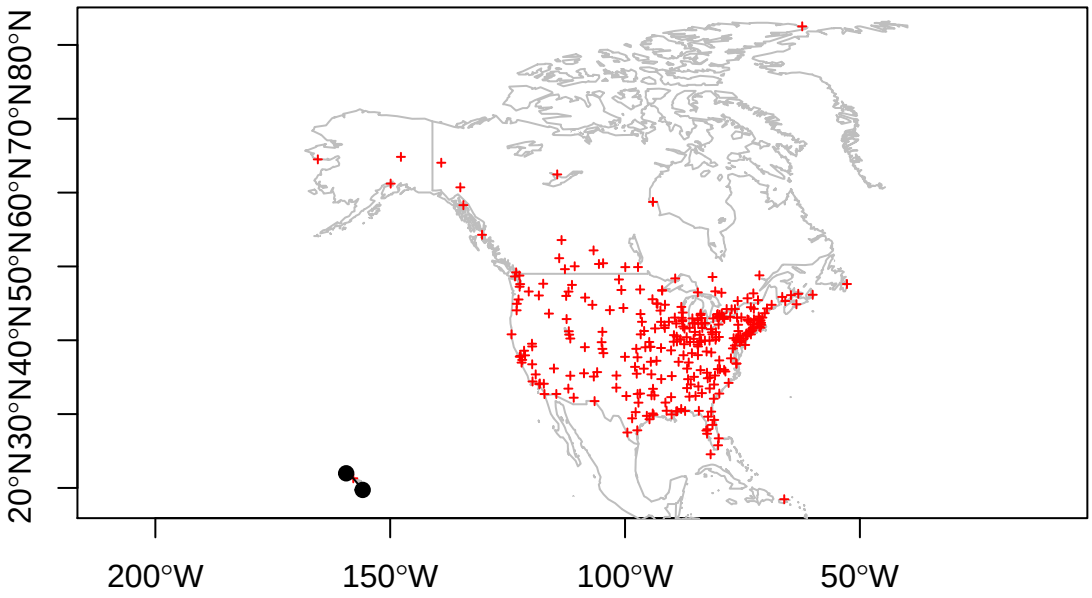

```
cluster7_ids <- match(cluster7, labels(USCA312))
plot_path(cluster7_ids)
title("Figura 31 - Cluster 7")
```

Figura 31 − Cluster 7



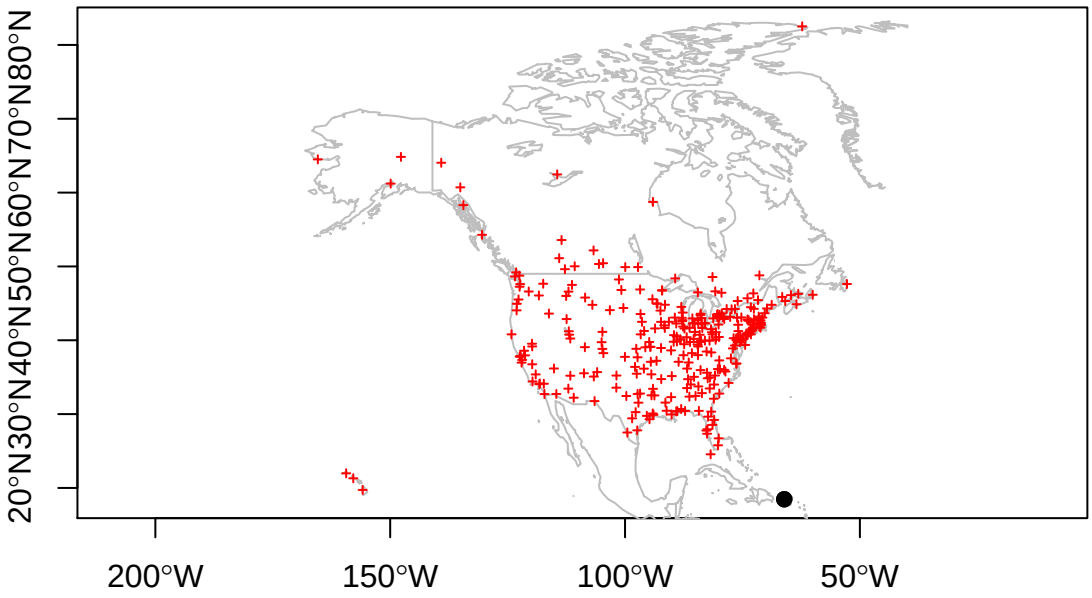

```
cluster8_ids <- match(cluster8, labels(USCA312))
plot_path(cluster8_ids)
title("Figura 32 - Cluster 8")
```

Figura 32 − Cluster 8



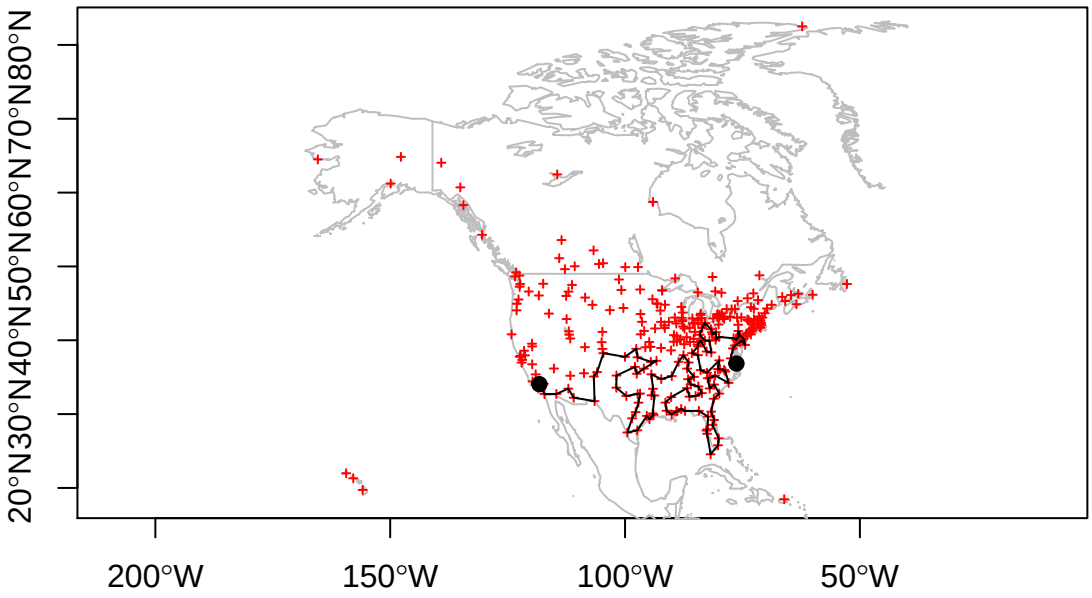

```
cluster9_ids <- match(cluster9, labels(USCA312))
plot_path(cluster9_ids)
title("Figura 33 - Cluster 9")
```

Figura 33 − Cluster 9




```
cluster10_ids <- match(cluster10, labels(USCA312))
plot_path(cluster10_ids)
title("Figura 34 - Cluster 10")
```

Figura 34 − Cluster 10



Ok, Como las distancias de viaje entre clusters no se consideran en la suma de distancia, hay lugares remotos o en islas que tiene sentido que formen su propio cluster.

Hay decisiones de negocio que tomar aquí, en cuanto a la cantidad de vendedores que se podrían contratar .... etc etc.

PRIMERO QUE NADA, yo aconsejaría quitarle restricciones al modelo, y aquí les queda el desafío:

DESAFÍO

Los desafío a realizar una nueva ruta, aplicando la estrategia de cluster a USCA312, SIN ninguna restricción de donde comenzar ni dónde terminar el recorrido de Hamilton.

En otras palabras, encontrar una solución por cluster simétrica.

CONCLUSIÓN

En este trabajo los autores del paquete TSP presentan el paquete de extensión TSP R que implementa una infraestructura para manejar y resolver el problema del viajero.El paquete introduce clases de descripciones de los problemas simétricos y asimétricos (TSP y ATSP respectivamente) y de solución (TOUR o GIRA).

Referencias:

```
citation("TSP")
```

```
##
##   Michael Hahsler and Kurt Hornik (2020). TSP: Traveling Salesperson
##   Problem (TSP). R package version 1.1-10.
##   https://CRAN.R-project.org/package=TSP
##
## Hahsler M, Hornik K (2007). "TSP - Infrastructure for the traveling
## salesperson problem." _Journal of Statistical Software_, *23*(2), 1-21.
## ISSN 1548-7660, doi: 10.18637/jss.v023.i02 (URL:
## https://doi.org/10.18637/jss.v023.i02), <URL:
## http://www.jstatsoft.org/v23/i02/>.
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
```

-fin del documento-