Reinforcement Learning

Objective: using travelling salesman problem to build an itinerary for a European vacation

Data: Travel distances and coordinates for a selection of European cities in Distances.csv and Coordinates.csv.

Additional sources: Approaches to solving the traveling salesman problem were adapted from https://www.r-bloggers.com/travelling-salesman-with-ggmap/ .

The travelling salesman problem represents a classical programming task that has many practical applications. It implies optimizing a travelling path between N cities with the conditions that each city has to be visited once and only once, and the trip has to end where it started. It required a sequential desicion-making process with the overall goal of optimizing trip performance.

Reinforcement learning components as applied to the travelling salesman problem:

Objective – to find the shortest tour that visits each city on a list exactly once and returns to the starting point.

Policy – describes behaviors, ways of choosing next city to visit (can be random, or shortest distance, longest distance, next on the list etc.).

Value function – overall minimization of the travel distance.

Reward function – distance added to the trip on each of the travel segments.

Exploration - exploration is represented by random choice of the initial point of the trip (origin) and the possibility of the random choice of the next city to visit.

In the situation where a direction of the trip is important, or path may not exist between the two points in both directions, it is possible to use an asymmetric algorithm where the distance between the two cities depends on a direction of travel. In this assignment, only a symmetrical algorithm is considered (distances between the cities are equal no matter the direction and no rewards/penalties for transitions between particular states).

In this assignment, I used the TSP package that provides a convenient infrastructure for practical implementation of the travelling salesman problem in order to find a path between nine European cities. I used a blog post by Collier (2018) as inspiration for approaching the problem, however due to the recent changes in the way Google manages its mapping APIs I had to make adjustments aimed at making the code independent of the Google geo-coding services. Because of exciding the limits for my existing access keys, I used csv files as a source of the input data and was limited in my visualization options as I could not use any of the ggmap package functionality reverting back to the rworldmap package.

First, I prepared the environment and loaded the packages using the following code:

```
> ###MSDS680 Week 8 Assignment:  Traveling salesman Problem   Weakly,Natalia
> ###Chose Itenerary for a Dream Vacation in Europe
>
> #Used as an inspiration
> #https://www.r-bloggers.com/travelling-salesman-with-ggmap/
> #https://github.com/mhahsler/TSP
> #Driving distances data from https://www.engineeringtoolbox.com/driving-distances-d_1029.html
> #Coordinates daty from https://www.latlong.net/
>
> rm(list=ls()) #Clear the environment
> setwd("E:/Dropbox/RU DataScience/MSDS680/Week8/Assignment") #Set working directory for the assi
gnment
> getwd() #Check working directory
[1] "E:/Dropbox/RU DataScience/MSDS680/Week8/Assignment"
>
> #Load Libraries
> library(TSP)
> library(dplyr)
```

```
> library(purrr)
> library(rworldmap)
```

Next, I loaded the data about driving distances (in km) between the cities from a csv file and converted it to a symmetrical distance matrix using the following code:

```
> #Load distances  between the cities in km
> distances <- read.csv("Distances.csv", header = FALSE, sep = ",", stringsAs
Factors = FALSE)
> distances
            V1    V2     V3     V4     V5      V6         V7     V8     V9    V1
0
1                Rome Athens Vienna Munich Hamburg Copenhagen Paris Lisbon Madri
d
2         Rome     0   2551   1168    969    1903       2352   1531   2737   209
9
3       Athens 2551       0   1886   2210    2758       3414   3140   4578   394
0
4       Vienna 1168   1886       0    458     896       1345   1285   3255   261
7
5       Munich  969   2210     458      0     755       1204    827   2515   187
7
6      Hamburg 1903   2758    1285    755       0        321    880   2666   240
9
7   Copenhagen 2352   3414    1345   1204     321          0   1329   3115   259
7
8        Paris 1531   3140    1285    827     880       1329      0   1786   126
8
9       Lisbon 2737   4578    3255   2515    2666       3115   1786      0    63
8
10      Madrid 2099   3940    2617   1877    2409       2597   1268    638
0
> distances <- distances[, -1] #Delete first column
> distances <- distances[-1, ] #Delete first row
> distances <- as.dist(distances) #create distance matrix
> #Check the matrix
> distances
      2    3    4    5    6    7    8    9
3  2551
4  1168 1886
5   969 2210  458
6  1903 2758 1285  755
7  2352 3414 1345 1204  321
8  1531 3140 1285  827  880 1329
9  2737 4578 3255 2515 2666 3115 1786
10 2099 3940 2617 1877 2409 2597 1268  638
```

Next, I loaded the coordinates for the cities:

```
> #Load coordinates
> coordinates <- read.csv("Coordinates.csv", header = FALSE, sep = ",", stringsAsFactors = FALSE)
> names(coordinates) <- c("city","country", "lat", "lon")
```

```
> #check results
> coordinates
         city  country      lat       lon
1        Rome    Italy 45.46362 12.496365
2      Athens   Greece 37.98381 23.727539
3      Vienna  Austria 48.20921 16.372780
4      Munich  Germany 48.13512 11.581981
5     Hamburg  Germany 53.55109 53.551086
6   Stockholm   Sweden 59.33279 18.064489
7  Copenhagen  Denmark 55.67610 12.568337
8       Paris   France 48.85661 48.856613
9      Lisbon Portugal 38.72225 -9.139337
10     Madrid    Spain 40.41678 -3.703790
```

Next, I used the TSP package to construct a symmetrical TSP problem and solve it to design a tour:

```
> ####Traveling Salesman Problem
> ###use TSP package
>
> #constructor creating an instance od a symmetric TSP problem
> tsp <- TSP(distances)
>
> methods <- c(
+   "nearest_insertion",
+   "farthest_insertion",
+   "cheapest_insertion",
+   "arbitrary_insertion",
+   "nn",
+   "repetitive_nn",
+   "two_opt"
+ )
>
> tours <- methods %>% map(function(method) {
+   solve_TSP(tsp, method)
+ })
>
> #tour - stores solution of the TSP
> tour <- solve_TSP(tsp)
> # Order cities
> tour_order <- as.integer(tour)
```

The resulting solution found by the algorithm has a total driving distance of 12134 km:

```
> #Total driving distance
> tour_length(tour)
[1] 12134
```

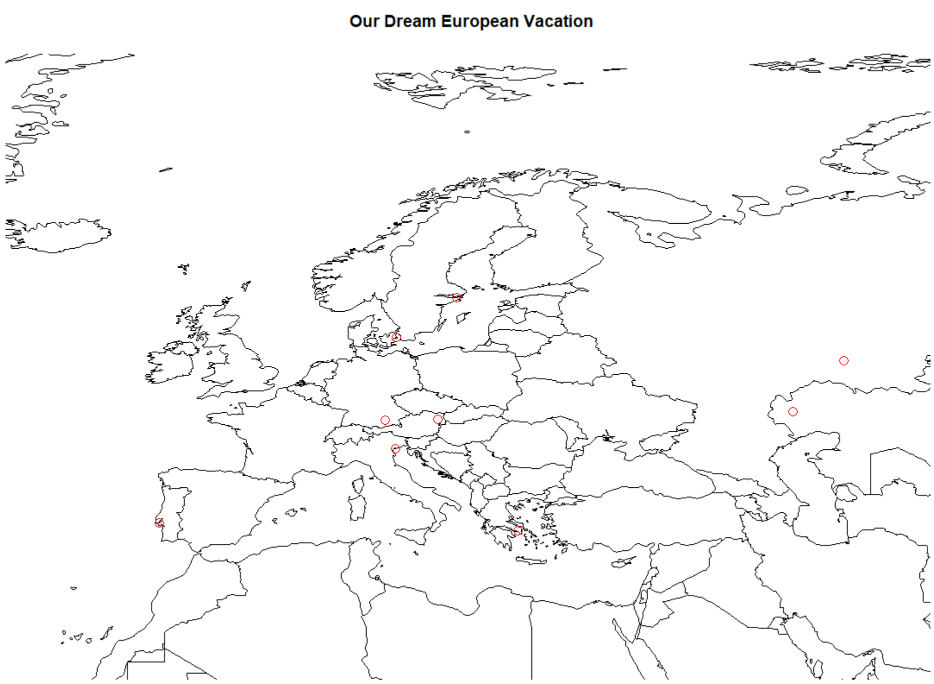
The trip starts and ends in Copenhagen:

```
> # Sort destination cities
> coordinates <- coordinates[tour_order, ]
>
> #Print vacation itenerary
```

```
> coordinates[1:2]
       city  country
7 Copenhagen  Denmark
8      Paris   France
9     Lisbon Portugal
1       Rome    Italy
4     Munich  Germany
2     Athens   Greece
3     Vienna  Austria
6  Stockholm   Sweden
5    Hamburg  Germany
```

To visualize the results, I used the rworld package and a low resolution map of Europe, drawing the map and placing the destination points on it using the following code:

```
> #####Visualization using rworldmap library######
>
> #Get low resolution map of Europe
> euro_map<-getMap(resolution="low")
>
> #Plot the map
> plot(euro_map, xlim=c(-20,59), ylim = c(35,71), asp = 1)
>
> #Add trip destinations to the map
> points(coordinates$lon, coordinates$lat, col="red", cex = 1.5)
>
> #Add  title to the map
> title(main=paste("Our Dream European Vacation"), cex=3)
```



Conclusions

The TSP package in R provides a very convenient way of approaching the traveling salesman problem.

Due to external limitations, I was not able to fully use the opportunities of the ggmap and ggplot 2 packages for this assignment. For applications designed for production that require external geocoding services, I would recommend looking into paid integration options with ESRI (ArcGIS online) services.

References

Collier A. B. (2018). Travelling salesman with ggmap. (2018) R-bloggers. Retrieved from https://www.r-bloggers.com/travelling-salesman-with-ggmap/