Technical Report: OPR Final Project

What is the Travelling Salesman Problem?

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Packages

- **lpSolve** - This is going to be the package that we are going to use to solve all of the linear programming. This package is the heart and soul of our solution. It will be utilized multiple times throughout the program.
- **ggplot2** - In our solution there will be a lot of visualization that we will show. This is where ggplot2 comes in. This package enables us to plot routes and points in the map.
- **ggmap** - This package is responsible for supplying us with the different maps that we will be needing. It has a wide range of maps in terms of aesthetics. Another use for this package is finding the walking, driving and biking distance between two points. Furthermore, it provides us the coordinates of the path that ggplot2 will be using. Unfortunately, this service is only partly free it limitations are listed below.

Google Maps Geocoding API Usage Limits

- 2500 Free requests per day
- 50 requests per second

There is an option for a premium account but was not used for this project.

To check how many requests you have left you can use the function "distQueryCheck".

```
library(ggmap)
```

```
## Loading required package: ggplot2
```

```
distQueryCheck()
```

```
## 2500 distance queries remaining.
```

Plotting a Route using ggmap and ggplot2

To show how well integrated ggmap and ggplot2 are, we are going to plot the route from Lebow to the Philadelphia Art Museum. Keep in mind that you have to use the complete address. If you are unsure about the complete address, you can check it by searching it on the Google maps and it will provide you with the complete address. Firstly, we have to use the route function which gives us all the information about the route

```
route <- route("LeBow Hall, 3220 Market St, Philadelphia, PA 19104","Philadelphia Museum
      of Art, 2600 Benjamin Franklin Pkwy, Philadelphia, PA 19130", structure = "route")
```

```
## Information from URL : http://maps.googleapis.com/maps/api/directions/json?origin=LeBow+Hall,+3220+Ma
```

```
route
```

```
##     m    km     miles seconds   minutes       hours leg       lon      lat
## 1 138 0.138 0.0857532      31 0.5166667 0.008611111   1 -75.18788 39.95542
## 2 532 0.532 0.3305848     122 2.0333333 0.033888889   2 -75.18948 39.95562
## 3 200 0.200 0.1242800      36 0.6000000 0.010000000   3 -75.18944 39.96036
## 4 301 0.301 0.1870414      52 0.8666667 0.014444444   4 -75.18717 39.96069
## 5 466 0.466 0.2895724      84 1.4000000 0.023333333   5 -75.18744 39.96337
## 6 246 0.246 0.1528644      56 0.9333333 0.015555556   6 -75.18224 39.96445
## 7  NA    NA        NA      NA        NA          NA  NA -75.18202 39.96638
```

Next is we are going to look up the geocode for these location that is to be plotted later

```
gcode<-as.data.frame(geocode(c("LeBow Hall, 3220 Market St, Philadelphia,
      PA 19104","Philadelphia Museum of Art, 2600 Benjamin Franklin Pkwy,
      Philadelphia, PA 19130")))
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=LeBow%20Hall,%203220%
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=Philadelphia%20Museum
```

```
gcode
```

```
##         lon      lat
## 1 -75.18794 39.95511
## 2 -75.18097 39.96557
```

Lastly, we are going to plot it on the map

```
map <- qmap(location = "LeBow Hall, 3220 Market St, Philadelphia, PA 19104",
            source = "stamen", maptype = "toner",zoom=14)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=LeBow+Hall,+3220+Market+St,+Phila
```

```
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=LeBow%20Hall,%203220%
```

```
## Map from URL : http://tile.stamen.com/toner/14/4768/6204.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4769/6204.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4770/6204.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4771/6204.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4768/6205.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4769/6205.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4770/6205.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4771/6205.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4768/6206.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4769/6206.png
```

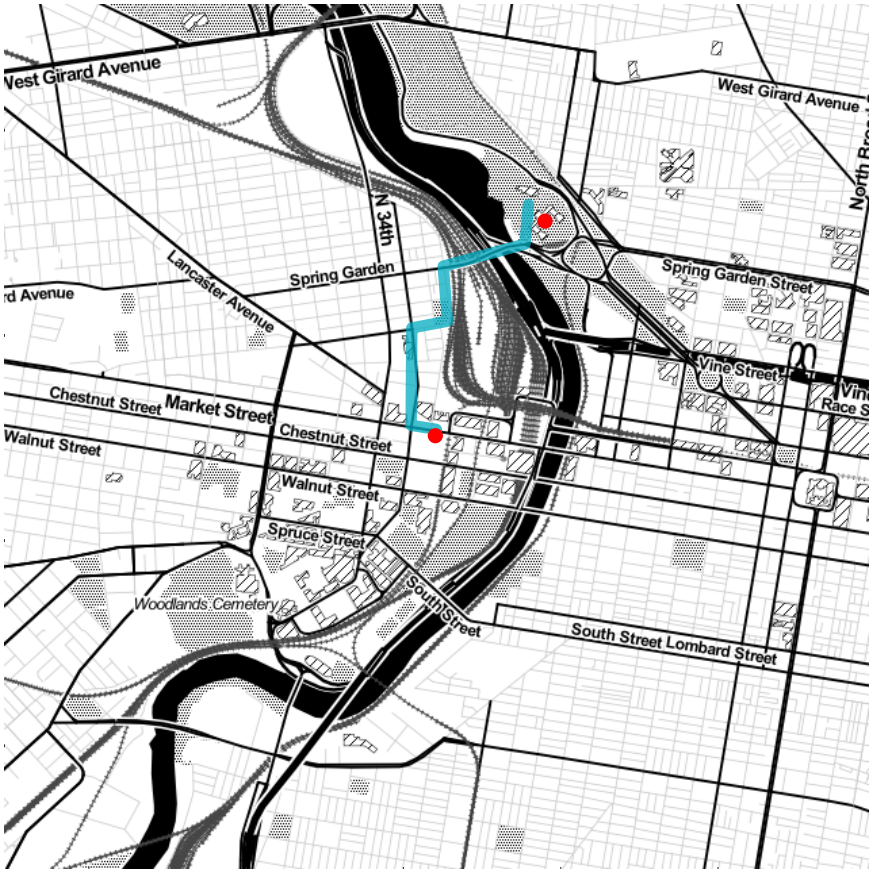
```
## Map from URL : http://tile.stamen.com/toner/14/4770/6206.png
```

```
## Map from URL : http://tile.stamen.com/toner/14/4771/6206.png
```

```
## Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
## instead
```

```
map <- map + geom_path(aes(x = lon, y = lat),  colour = "#00ACC1", size = 2,
                       data = route, lineend = "round",alpha=.75)
map <- map + geom_point(data=gcode, aes(x=gcode$lon, y=gcode$lat),size=2,
```

```
                        colour= "red")
map
```



Selecting the Destinations:

Here we select our top choices for our optimal road trip. For the sake of simplicity we will only use six destinations for this example. We will input the complete address that Google maps accepts as strings and we will add them to the "destinations" vector.

```
destinations<-"Times Square,New York,New York"
destinations<-append(destinations,"Golden Gate Bridge,San Francisco Bay Area,California")
destinations<-append(destinations,"Union Station,Washington, D.C.")
destinations<-append(destinations,"Great Smoky Mountains National Park,Tennessee")
destinations<-append(destinations,"Epcot,Orlando,Florida")
destinations<-append(destinations,"Pike Place Market,Seattle,Washington")
```

Building the Distance Matrix:

```
library(ggmap)
bignumber<-10000

destinationsDist<-matrix(nrow=length(destinations),ncol=length(destinations))
destinationsDistv<-vector()
```

```
for(i in 1:length(destinations)){
  for(j in 1:length(destinations)){
    if(i==j){
      destinationsDist[i,j]<-bignumber;
    }
    else if (i<j){
      destinationsDist[i,j]<-mapdist(destinations[i],destinations[j],mode="driving")$miles
    }
    else{
      destinationsDist[i,j]<-destinationsDist[j,i]
    }
  }
  Sys.sleep(2)
}
```

We create a Matrix called "destinationDist" which has all the distance information from city i (row) to city j (column). We are pulling the data from the ggmap library which is a collection of functions to visualize spatial data and models on top of static maps from various online resources (e.g Google Maps and Stamen Maps). It includes tools common to those tasks, including functions for geolocation and routing. We call the function "mapdist" to get the shortest driving distance from Google maps in terms of distance. One by one we get the values of the upper triangular of the matrix. We put the variable "bignumber" in the diagonal which in this case has the value of 10000. Lastly, for the lower triangular of the matrix, also excluding the diagonal, we just copy the upper half to save time and avoid using up the limits of the free distance checks from ggmap. We put Sys.sleep(3) at the end of every cycle of the inner loop so that R will pause for 3 seconds which will prevent us from reaching our 50 request per second limit. Below is out destinationDist matrix:

```
destinationsDist
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,] 10000.0000  2913.6980   227.2559   704.1817  1093.4018  2854.4351
## [2,]  2913.6980 10000.0000  2828.3462  2503.7430  2904.6883   816.3462
## [3,]   227.2559  2828.3462 10000.0000   485.7254   866.2341  2765.0510
## [4,]   704.1817  2503.7430   485.7254 10000.0000   627.6246  2582.9746
## [5,]  1093.4018  2904.6883   866.2341   627.6246 10000.0000  3085.5561
## [6,]  2854.4351   816.3462  2765.0510  2582.9746  3085.5561 10000.0000
```

The matrix must be converted into a vector for the lp function only accepts vectors

```
destinationsDistv<-as.vector(t(destinationsDist))
```

First Constraint

The first constraint is about having only one path that enter one node. Below is a partial view of the constraint.

```
constraints1<-matrix(nrow = ncol(destinationsDist),ncol = ncol(destinationsDist)^2)
for(i in 1:ncol(destinationsDist)){
  for(j in 1:ncol(destinationsDist)^2){
    if((j>=(i-1)*ncol(destinationsDist)+1)&&
       (j<=i*ncol(destinationsDist))){
      constraints1[i,j]<-1
    }
    else{
      constraints1[i,j]<-0
    }
```

```
  }
}
```

Table 1: constraints1

1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Second Constraint

The second constraint is about having only one path that exits from every single node.Below is the partial view of the constraint.

```
constraints2<-diag(x = 1, nrow=ncol(destinationsDist), ncol = ncol(destinationsDist))
for(i in 2:ncol(destinationsDist)){
  constraints2<-cbind(constraints2,diag(x = 1, nrow=ncol(destinationsDist), ncol = ncol(destinationsDis
}
```

Table 2: constraints2

1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0

Next, is we combine the two constraints together into a single matrix. Below, is the resulting matrix.

```
constraints<-matrix()
constraints<-rbind(constraints1,constraints2)
```

Table 3: Constraints

1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0

```
rhs<-rep(1,nrow(constraints));
signs<-rep('=',nrow(constraints))
```

And then, we add the right hand side of the equation which we will call "RHS". These are just the number one in the same amount of rows of the constraints. We will also add the variable "signs" which is the signs for our equation that is dependent upon the number of rows of the constraints.

Solving Function

The Function below is the "solving" function. This Function not only solves the lp but also builds a matrix (ressolution)from the resulting vector. From this matrix it determines the order of path which results in the vector "ord.destinations_address" which is printed below.

```
library(lpSolve)
solving<-function(){
  res<<-lp("min",
           destinationsDistv,
           constraints,
           signs,
           rhs,
           all.bin = TRUE,
           presolve = TRUE
  )
  ressolution<<-matrix(res$solution,ncol = ncol(destinationsDist),byrow = TRUE)
  routeorder<<-which(ressolution==1, arr.ind=TRUE)
  ord.destinations<<-vector()
  for(i in 1:(nrow(routeorder))){
    if(i==1){
      ord.destinations[i]<<-routeorder[i,1]
    }else{
      ord.destinations[i]<<-routeorder[ord.destinations[i-1],1]
    }
  }
  ord.destinations_address<<-destinations[ord.destinations]
}
```

If we read the function the first thing that stands out is the use of "<<-". This is not the same as "<-", for this declares a global variable when used inside a function. We used this, in order for us to be able to call the res variable outside of this function. We also used the all.bin parameter which only solves for binary solutions. Lastly, we set the presolve parameter to true. How this works is that it does preprocessing work to the linear equations so that the CPU will have an easier time solving. The only disadvantage of this is that sometimes it may take longer to simplify the linear equations than to just leave it as is.

Now, lets take a look at it in action.

```
solving()
```

Lets take a look at the solution

```
ressolution
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    1    0    0    0
## [2,]    0    0    0    0    0    1
## [3,]    1    0    0    0    0    0
## [4,]    0    0    0    0    1    0
```

```
## [5,]    0    0    0    1    0    0
## [6,]    0    1    0    0    0    0
```

But what does this matrix mean? Were going to have to transform it into its text form.

```
ord.destinations_address
```

```
## [1] "Union Station,Washington, D.C." "Times Square,New York,New York"
## [3] "Union Station,Washington, D.C." "Times Square,New York,New York"
## [5] "Union Station,Washington, D.C." "Times Square,New York,New York"
```

If you look at ord.destinations_address, it is telling us that we are just going back and fourth between two cities. This is called a subroute. We would need to eliminate all the subroute in order for us to find the optimal order of destinations such that we would pass by all of the destinations and would have zero subroutes.

Identifying and Eliminating the Subroutes

The function "getsubroute" is a multi step function that takes care of the subroute problem. These are the steps: 1. It identifies the subroutes. 2. Labels said subroutes with numbers. 3. Adds a modifies the constraint, rhs and signs so that when we use the solve function the subroute will be eliminated.

```
getsubroute<-function(routeorder){
  routeorder1<-routeorder
  routeorder1<-as.data.frame(routeorder1)
  routeorder1$subroute<-0
  routeorder1<-as.data.frame(routeorder1)
  ord.destinations1<-rep(0,nrow(routeorder1))
  i<-1
  subroutenum<-1
  while(sum(routeorder1$subroute==0)!=0){
    if(i==1){
      ord.destinations1[i]<-routeorder1[i,1]
      routeorder1[i,3]<-subroutenum
    }else{
      ord.destinations1[i]<-routeorder1[ord.destinations1[i-1],1]

      if(routeorder1[ord.destinations1[i-1],3]==0){
        routeorder1[ord.destinations1[i-1],3]<-subroutenum

      }else{
        routeorder1[ord.destinations1[i-1],3]<-subroutenum
        subroutenum<-subroutenum+1
        ord.destinations1[i]<-routeorder1[match(0,routeorder1$subroute),1]
      }
    }
    i<-i+1
  }
  numsubroute<-unique(routeorder1[,3])
  for(i in 1:length(numsubroute)){#eliminate single city subroute
    if(sum(routeorder1[,3]==i) == 1){
      routeorder1<-routeorder1[!(routeorder1[,3])==i,]
    }
  }
  printrouteorder<<-routeorder1
```

```
  names(printrouteorder)<-c("City I","City J","SubRoute")
  for(i in 1:length(unique(routeorder1[,3]))){
    newconstraint<-rep(0,ncol(constraints))
    for(j in 1:sum(routeorder1[,3]==i)){
      newconstraint[((routeorder1[routeorder1[,3]==i,1][j]-1)*length(destinations))
                    +routeorder1[routeorder1[,3]==i,2][j]]<-1
      newconstraint[((routeorder1[routeorder1[,3]==i,2][j]-1)*length(destinations))
                    +(routeorder1[routeorder1[,3]==i,1][j])]<-1
    }
    constraints<<-rbind(constraints,newconstraint)
    signs<<-append(signs,"<=")
    rhs<<-append(rhs,sum(routeorder1[,3]==i)-1)
  }
}
```

Eliminating the Subtour

Here we are putting everything together, we are identifying the different subroutes, adding the constraints, and then resolving them. We are putting this in a while loop such that it would only stop if the solution does not have any subroutes anymore. Fortunately, since we only have 6 nodes in this example we only got two iterations. The number of iterations is dependent upon a lot of factors such as the number of nodes and the disparity between the distances of the different nodes. But as the more nodes you add, the amount of iterations exponentially increases. You can also notice that each iteration that we add, the objective function increases. If you have iterations that is large enough, you can notice that the increase in the objective function also slightly decreases.

```
iteration<-0
getsubroute(routeorder)
a<-length(unique(printrouteorder[,3]))>1 && length(unique(ord.destinations))<length(destinations)
while(a!=0){

  getsubroute(routeorder)

  solving()
  iteration<-1+iteration
  print(paste("ITERATION: ",iteration))
  print("Subtours Eliminated:")
  print(printrouteorder)
  print(paste("Num of Constraints:",nrow(constraints)))
  print(res)
  cat("\n")

  a<-sum(length(unique(printrouteorder[,3]))>1 ,
         length(unique(ord.destinations))<length(destinations))
}
```

```
## [1] "ITERATION:  1"
## [1] "Subtours Eliminated:"
##   row col subroute
## 1   3   1        1
## 2   6   2        2
## 3   1   3        1
## 4   5   4        3
```

```
## 5   4   5        3
## 6   2   6        2
## [1] "Num of Constraints: 18"
## Success: the objective function is 7895.639
##
## [1] "ITERATION:  2"
## [1] "Subtours Eliminated:"
##   row col subroute
## 1   3   1        1
## 2   6   2        1
## 3   5   3        1
## 4   2   4        1
## 5   4   5        1
## 6   1   6        1
## [1] "Num of Constraints: 19"
## Success: the objective function is 7916.075
```

Plotting the Route in a Map

```
ord.destinations_address<-append(ord.destinations_address,ord.destinations_address[1])

remove(route_df1)
```

```
## Warning in remove(route_df1): object 'route_df1' not found
```

```
remove(route_df)
```

```
## Warning in remove(route_df): object 'route_df' not found
```

```
route_df<-vector()
route_df1<-vector()
for(i in 1:(length(ord.destinations_address)-1)){
  route_df1 <- route(ord.destinations_address[i], ord.destinations_address[i+1], structure = "route")
  route_df<-rbind(route_df,route_df1)
}

route_df<-route_df[,8:9]
gcode<-as.data.frame(geocode(ord.destinations_address))

map <- qmap(location = "USA", source = "stamen", maptype = "toner",zoom=4)
```

```
## Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
## instead
```

```
map <- map + geom_path(aes(x = lon, y = lat),  colour = "#00ACC1", size = 2,
                       data = route, lineend = "round",alpha=.75)
map <- map + geom_point(data=gcode, aes(x=gcode$lon, y=gcode$lat),size=2,
                        colour= "red")
map
```

The following is the order of our trip: * Epcot * Great Smokey Mountains National Park * Golden Gate Bridge * Pike Place Market * Union Station * Times Square

Now that we've showed how the code works, let us now plug in the whole 40 destinations

"'{r ,fig.width=8, fig.height=7,fig.align='center'}

remove(destinations) destinations<-"13000 SD-244, Keystone, SD 57751" destinations<-append(destinations,"Golden Gate Bridge, San Francisco, CA") destinations<-append(destinations,"1401 John F Kennedy Blvd, Philadelphia, PA 19107") destinations<-append(destinations,"Times Square, Manhattan, NY 10036") destinations<-append(destinations,"The Gateway Arch, St. Louis, MO 63102") destinations<-append(destinations,"Freedom Trail, Boston, MA") destinations<-append(destinations,"Hoover Dam, Nevada 89005") destinations<-append(destinations,"The Alamo, 300 Alamo Plaza, San Antonio, TX 78205") destinations<-append(destinations,"600 Independence Ave SW, Washington, DC 20560") destinations<-append(destinations,"Yellowstone National Park, WY") destinations<-append(destinations," 28210 W Park Hwy, Ashland, NE 68003") destinations<-append(destinations," 200 E 3rd St, Little

Rock, AR 72201“) destinations<-append(destinations,”Craters of the Moon National Monument & Preserve, Idaho“) destinations<-append(destinations,”Theodore Roosevelt National Park“) destinations<-append(destinations,” 100 W 14th Ave Pkwy, Denver, CO 80204“) destinations<-append(destinations,” 3650 E Ave G, Hutchinson, KS 67501“) destinations<-append(destinations,”2006 Riverside Dr, Monroe, LA 71201“) destinations<-append(destinations,” 306 Elvis Presley Dr, Tupelo, MS 38801“) destinations<-append(destinations,”1001 Parkway, Gatlinburg, TN 37738“) destinations<-append(destinations,”Redstone Arsenal, 1 Tranquility Base, Huntsville, AL 35805“) destinations<-append(destinations,”121 Baker St NW, Atlanta, GA 30313“) destinations<-append(destinations,”700 S Victory Way, Kissimmee, FL 34747“) destinations<-append(destinations,” 500 Wildlife Pkwy, Columbia, SC 29210“) destinations<-append(destinations,” 1 Lodge St, Asheville, NC 28803“) destinations<-append(destinations,” 310 S England St, Williamsburg, VA 23185“) destinations<-append(destinations,” 2820 SE Ferry Slip Rd, Newport, OR 97365“) destinations<-append(destinations,”Desert Botanical Garden, Papago Park, 1201 N Galvin Pkwy, Phoenix, AZ 85008“) destinations<-append(destinations,”1601 NASA Road 1, Houston, TX 77058“) destinations<-append(destinations,”Hollywood Sign, Griffith Park, Los Angeles, CA 90068“) destinations<-append(destinations,” 452 E Christmas Blvd, Santa Claus, IN 47579“) destinations<-append(destinations,”131 Discovery Rd, Hopewell Cape, NB E4H 4Z5, Canada“) #destinations<-append(destinations,”Canadian Rockies, Canada“) destinations<-append(destinations,”Fairmont Le Chateau Frontenac, 1 Rue des Carrieres, Quebec City, QC G1R 4P5, Canada“) destinations<-append(destinations,”Banff National Park, Improvement District No. 9, AB T0L, Canada“) destinations<-append(destinations,”CN Tower, 301 Front St W, Toronto, ON M5V 2T6, Canada“) destinations<-append(destinations,”Xel-Ha Park, Carretera Chetumal Puerto Juarez Km 240, locales 1 & 2, modulo B, 77780 Q.R., Mexico“) destinations<-append(destinations,”The Arch of Cabo San Lucas, Cabo San Lucas, B.C.S., Mexico“) destinations<-append(destinations,”Biosphere Reserve El Vizcaino, Heroica Mulege, BCS, Mexico“) destinations<-append(destinations,”Medano Beach, El Medano Ejidal, 23479 Cabo San Lucas, BCS, Mexico“) destinations<-append(destinations,”National Palace, Plaza de la Constitucion S/N, Centro, Cuauhtemoc, 06066 Ciudad de Mexico, CDMX, Mexico“) destinationsDist<-read.csv(”dist.csv“) bignumber<-10000

library(ggmap)

Building the distance matrix

destinationsDist<-matrix(nrow=length(destinations),ncol=length(destinatio

destinationsDistv<-vector()

for(i in 1:length(destinations)){

for(j in 1:length(destinations)){

if(i==j){

destinationsDist[i,j]<-bignumber;

}

else if (i=(i-1)*ncol(destinationsDist)+1)&& (j<=*i*ncol(destinationsDist))){

```
  constraints1[i,j]<-1
}
else{
  constraints1[i,j]<-0
}
```

} }

Columns equal to 1 constraint

constraints2<-diag(x = 1, nrow=ncol(destinationsDist), ncol = ncol(destinationsDist)) for(i in 2:ncol(destinationsDist)){ constraints2<-cbind(constraints2,diag(x = 1, nrow=ncol(destinationsDist), ncol = ncol(destinationsDist))) }

constraints<-matrix() constraints<-rbind(constraints1,constraints2)

remove(signs) rhs<-rep(1,nrow(constraints)); signs<-rep('=',nrow(constraints))

Solving the LP

solving<-function(){ library(lpSolve) res<<-lp("min", destinationsDistv, constraints, signs, rhs, all.bin = TRUE, presolve = TRUE) ressolution<<-matrix(res$solution,ncol = ncol(destinationsDist),byrow = TRUE)

#converting matrix into order routeorder<<-which(ressolution==1, arr.ind=TRUE) ord.destinations<<-vector() for(i in 1:(nrow(routeorder))){ if(i==1){ ord.destinations[i]<<-routeorder[i,1] }else{ ord.destinations[i]<<-routeorder[ord.destinations[i-1],1] } } ord.destinations_address<<-destinations[ord.destinations] }

solving()

getsubroute<-function(routeorder){ routeorder1<-routeorder routeorder1<-as.data.frame(routeorder1) routeorder1$subroute < −0routeorder1 < −as.data.frame(routeorder1)ord.destinations1 < −rep(0, nrow(routeorder1))i < −1subroutenum < −1while(sum(routeorder1subroute==0)!=0){#-1 if(i==1){ ord.destinations1[i]<-routeorder1[i,1] routeorder1[i,3]<-subroutenum }else{ ord.destinations1[i]<-routeorder1[ord.destinations1[i-1],1]

```
  if(routeorder1[ord.destinations1[i-1],3]==0){
    routeorder1[ord.destinations1[i-1],3]<-subroutenum

  }else{
    routeorder1[ord.destinations1[i-1],3]<-subroutenum
    subroutenum<-subroutenum+1
    ord.destinations1[i]<-routeorder1[match(0,routeorder1$subroute),1]
  }
}
i<-i+1
```

} #routeorder1<-routeorder1[order(routeorder1[,3],routeorder1[,2]),] numsubroute<-unique(routeorder1[,3]) for(i in 1:length(numsubroute)){#eliminate single city subroute if(sum(routeorder1[,3]==i) == 1){ routeorder1<-routeorder1[!(routeorder1[,3])==i,] } } printrouteorder<<-routeorder1 names(printrouteorder)<-c("City I","City J","SubRoute") for(i in 1:length(unique(routeorder1[,3]))){ newconstraint<-rep(0,ncol(constraints)) for(j in 1:sum(routeorder1[,3]==i)){ newconstraint[((routeorder1[routeorder1[,3]==i,1][j]-1)*length(destinations))+routeorder1[1 newconstraint[((routeorder1[routeorder1[,3]==i,2][j]-1)*length(destinations))+(routeorder1[routeorder1[,3]==i,1][j])]<-1 } constraints<<-rbind(constraints,newconstraint) signs<<-append(signs,"<=") rhs<<-append(rhs,sum(routeorder1[,3]==i)-1) } }

Subtour Elimination

iteration<-0 ptm <- proc.time() #while(length(unique(ord.destinations))1 && length(unique(ord.destinations))1 , length(unique(ord.destinations))1 print(paste("a:",a)) }

proc.time() - ptm

Go back to first node

ord.destinations_address<-append(ord.destinations_address,ord.destinations_address[1])

driving route

remove(route_df1) remove(route_df) route_df<-vector() route_df1<-vector() for(i in 1:(length(ord.destinations_address)-1)){ route_df1 <- route(ord.destinations_address[i], ord.destinations_address[i+1], structure = "route") route_df<-rbind(route_df,route_df1) Sys.sleep(.1) }

route_df<-route_df[,8:9] gcode<-as.data.frame(geocode(ord.destinations_address))

map1 <- qmap(location = c(-130,8,-60,53),source = "stamen", maptype = "toner",zoom=3) map1 <- map1 + geom_path(aes(x = lon, y = lat), colour = "#00ACC1", size = 2,data = route_df, lineend = "round",alpha=.75) map1 <- map1 + geom_point(data=gcode, aes(x=gcode$lon, y = gcode$lat),size=2, colour= "red") map1

‘‘‘

Exceeding the Limitaions of ggmap

Since ggmap can only give us 2500 queries per day, we will get rid of this and just use euclidean distance to show how much our code can do. Below is the small sample of our new Distance Matrix

```
numberofdest<-200
coordinates_df <- data.frame(long=runif(numberofdest,0,10), lat=runif(numberofdest,0,10))
coordinates_mat <- as.matrix(coordinates_df)
distance_mat <- dist(coordinates_mat)
round(runif(10,0,100))
```

```
##  [1] 90 27 25 21 71 18 30 68 66 21
```

```
distance_mat<-as.matrix(distance_mat)
destinationsDist<-distance_mat
for(i in 1:ncol(destinationsDist)){
  destinationsDist[i,i]<-100
}
destinationsDist[1:5,1:5]
```

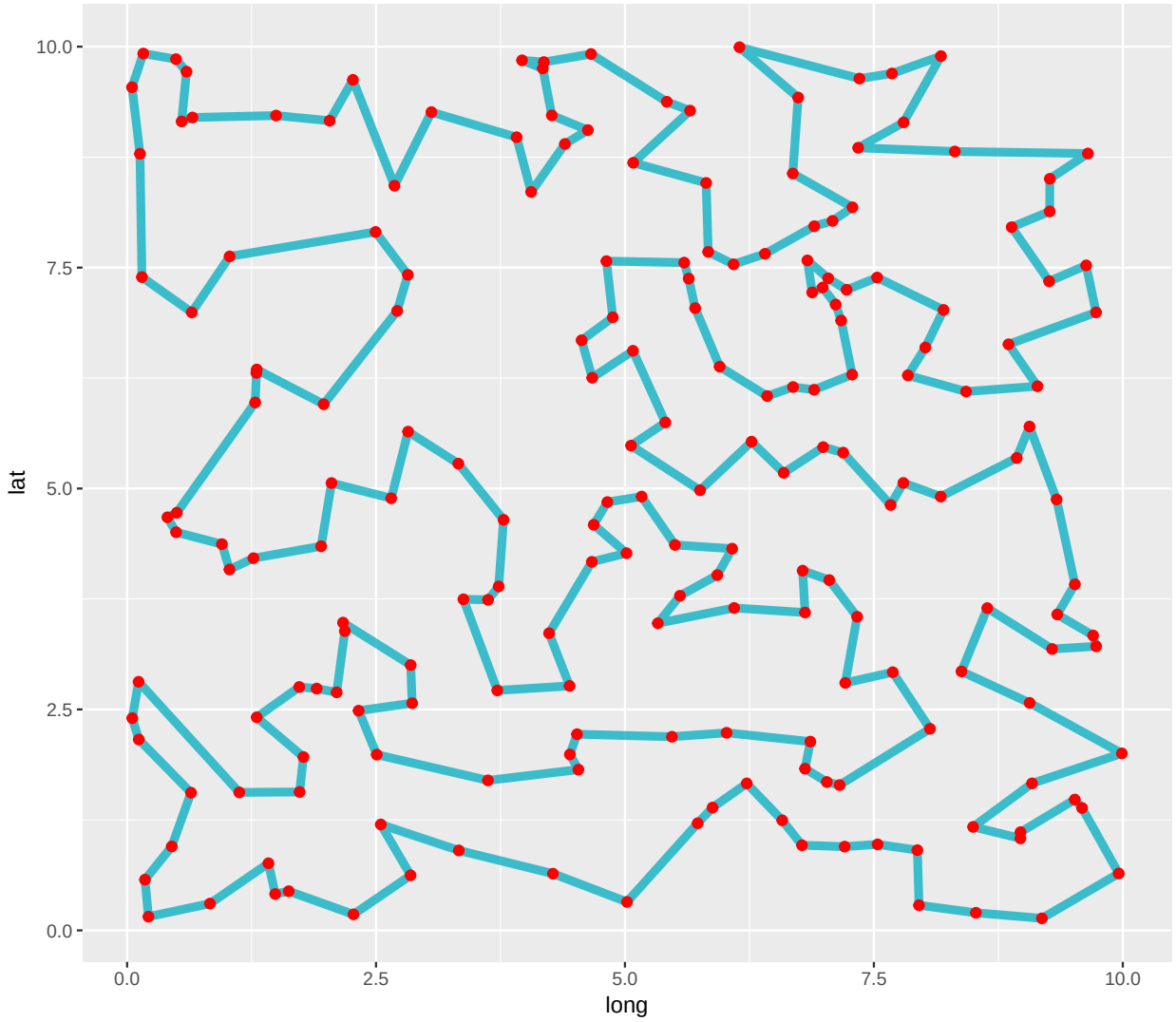
```
##            1          2          3          4          5
## 1 100.000000   4.974327   8.984581   5.813911   7.683418
## 2   4.974327 100.000000   5.082243   3.033320   5.337958
## 3   8.984581   5.082243 100.000000   3.219767   9.699842
## 4   5.813911   3.033320   3.219767 100.000000   8.360694
## 5   7.683418   5.337958   9.699842   8.360694 100.000000
```

In total it has 200 nodes but we only showed 5 of it. The plot below is the solution using the 200 nodes.

```
## Loading required package: TSP
```

```
## Loading required package: MASS
```

```
map
```


10.0
7.5
5.0
2.5
0.0
lat
0.0
2.5
5.0
7.5
10.0
long