

Appendix - R Code

```
# -----  
# Compute distance matrix  
# -----  
distance.matrix <- function(data) {  
  # use high dummy value for distance to itself  
  dm <- matrix(10000, N, N)  
  for (i in 1:N) {  
    for (j in 1:N) {  
      if (i != j)  
        # Euclidian distance  
        dm[i,j] <- round(sqrt((data[i,2] - data[j,2])**2 +  
                               (data[i,3] - data[j,3])**2),3)  
    }  
  }  
  return(dm)  
}  
  
dm <- distance.matrix(data)  
  
# -----  
# Compute path length  
# -----  
path.length <- function(dm, path) {  
  length <- 0  
  N <- length(path)  
  
  # walk through path and add up distance  
  for (i in 2:N) {  
    length <- length + dm[path[i-1],path[i]]  
  }  
  
  # add distance back to origin  
  length <- length + dm[path[N],path[1]]  
  return(length)  
}  
  
# -----  
# Plot path  
# -----  
plot.path <- function(data, path, color="red") {  
  N <- length(path)  
  
  # plot cities  
  par( mar=c(0.5,0.5,0.5,0.5), mfrow=c(1,1) )  
  plot(-data[,3:2], type='p', pch=16, cex=0.5, asp=1, xaxt='n', yaxt='n')  
  
  # plot path through cities  
  for (i in 2:N) {  
    segments(-data[path[i-1],3], -data[path[i-1],2],  
            -data[path[i],3], -data[path[i],2], col=color)  
  }  
}
```

```

}
segments(-data[path[N],3], -data[path[N],2],
         -data[path[1],3], -data[path[1],2], col=color)
}

```

```

# -----
# Nearest neighbor algorithm
# -----
nn.path <- function(dm) {
  visited <- rep(0,N)
  path <- rep(0,N)

  # starting point
  city <- 1
  visited[city] <- TRUE
  path[1] <- 1

  # repeatedly visit nearest not-yet visited neighbor
  for (i in 2:N) {
    leg <- min(dm[city, !visited])

    # find index of nearest city and add it to path
    potentials <- which(dm[city,] == leg)
    for (j in 1:length(potentials)) {
      if (!visited[potentials[j]]) {
        city <- potentials[j]
        visited[city] <- TRUE
        path[i] <- city
        break
      }
    }
  }
  return(path)
}

```

```

# -----
# Insertion Heuristics
# -----
insertion.path <- function(dm) {
  visited <- rep(0,N)

  # start with initial subtour
  visited[1] <- visited[2] <- visited[3] <- TRUE
  path <- c(1,2,3,1)

  while (0 %in% visited){
    # find closest unvisited city to subtour
    leg <- min(dm[!!visited, !visited])

    # find index of closest unvisited city
    potentials <- which(dm[,] == leg, arr.ind=TRUE)
    for (k in 1:nrow(potentials)) {
      if (!!visited[potentials[k,1]] & !visited[potentials[k,2]]) {

```

```

        city <- as.numeric(potentials[k,2])
        visited[city] <- TRUE
        break
    }
}

# find shortest detour to new city
detour <- list(len=1000000, from=0, to=0)
for (i in 1:(length(path)-1)) {
    pot.detour.len <- dm[path[i],city] + dm[path[i+1],city] - dm[path[i],path[i+1]]
    if (pot.detour.len < detour$len) {
        detour$len <- pot.detour.len
        detour$from <- i
        detour$to <- i+1
    }
}
# add detour to new path
path <- c(path[1:detour$from], city, path[detour$to:length(path)])
}
return(path[1:N])
}

```

```

# -----
# 2-opt path improvement
# -----
two.opt.path <- function(dm, path) {
    path <- c(path, path[1])
    N <- length(path)
    changes <- TRUE

    # keep updating path until it is 2-opt path
    while (changes) {
        changes <- FALSE

        for (i in 1:(N-3)) {
            for (j in (i+2):(N-1)) {
                # if alternative path is segment is shorter than current segment
                curr.len <- dm[path[i],path[i+1]] + dm[path[j], path[j+1]]
                new.len <- dm[path[i],path[j]] + dm[path[i+1],path[j+1]]

                if (new.len < curr.len) {
                    # swap positions
                    path[(i+1):j] <- path[j:(i+1)]
                    changes <- TRUE
                }
            }
        }
    }
    return(path[1:(N-1)])
}

```

```

# -----
# Run path computations and plot solutions

```

```

# -----

# compute distance matrix
dm <- distance.matrix(data)

# compute nearest neighbor path
path.nn <- nn.path(dm)
path.length(dm, path.nn)
plot.path(data, path.nn, color="red")

# compute insertion heuristics path
path.insert <- insertion.path(dm)
path.length(dm, path.insert)
plot.path(data, path.insert, color="blue")

# compute 2-opt improved nearest neighbor path
path.nn.2opt <- two.opt.path(dm, path.nn)
path.length(dm, path.nn.2opt)
plot.path(data, path.nn.2opt, color="red")

# compute 2-opt improved insertion heuristics path
path.insert.2opt <- two.opt.path(dm, path.insert)
path.length(dm, path.insert.2opt)
plot.path(data, path.insert.2opt, color="blue")

```