14D006 Stochastic Models and Optimization
Problem Set 2 - Exercise 5

Denitsa Panova, Zsuzsa Holler, Felix Gutmann, Thomas Vicente

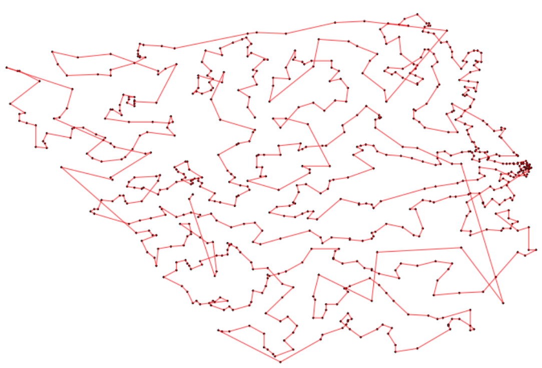
March 3, 2016

Implementation and results

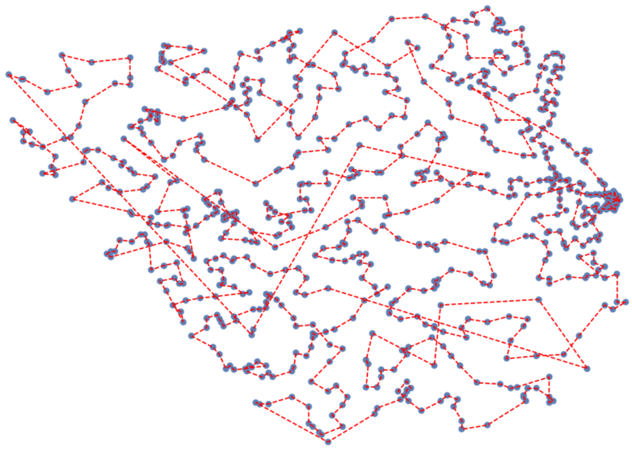
The task is to find the cycle with minimum distance among 734 cities in Uruquay (TSP-Problem). This well known problem is NP-hard. A quick and easy method to apply is a greedy algorithm, which starts at a city and chooses in every step the next smallest way to go. However, this comes with the cost that the solution is sub-optimal. The implementation performs sub-optimal in comparison to the optimal result available on the source website[1] (code on page 2 et seqq.), however gives a better result than the benchmark implementation computed with Wolfram Mathematica (see page 4).

Numerical results and plots

Methods	**Distances**
Optimal solution (website)	79114
Greedy implementation	96514
Greedy benchmark	100615



(a) Greedy implementation



(b) Greedy benchmark

[1]http://www.math.uwaterloo.ca/tsp/world/summary.html

```r
################################################################################
#################   Problemset 2 - Traveling Salesman Problem    ###############
################################################################################

# Authors:      Denitsa Panova, Zsuzsa Holler, Felix Gutmann, Thomas Vicente
# Programm:     Barcelona graduate school of economics - M.S. Data Science
# Course:       14D006 - Stochastic Models and Optimization
# Last update:  02.03.2016

################################################################################
### Preamble
################################################################################

### Clear workspace
rm( list = ls () )

### Set working directory
setwd('')

### Load data
data <- read.table('uy734.tsp.txt')[,2:3]

################################################################################
### Intialize function
################################################################################

greedySalesMan <- function( coordinates ){

  # Compute distance matrix as weighted adjacency matrix
  adjacencyMatrix  <- as.matrix( dist( coordinates ,
                          method = 'euclidean',
                          diag   = TRUE,
                          upper  = TRUE
                                   ))

  # Compute number of vertices and initialize global storage objects
  N         <- nrow( adjacencyMatrix )
  paths     <- matrix( NA , nrow = N , ncol = N )
  distances <- rep( NA, N )

  # Run greedy search for each node in the graph - start at j = 1
  for( j in 1:N ){

    # Intialize storage local objects
    source     <- j
    sequence   <- c( source )
    pathWeight <- 0

    # Find greedy cycle for node j
    for( i in 1:N  ){
      # If final step reached go back to source node
      if( i == N ){
        temp       <- sequence[i]
```

```r
        weight     <- adjacencyMatrix[ temp , ][ source ]
        pathWeight <- pathWeight + weight
      # Choose next minimum node if still unvisted nodes in the outlist of prev. node
      } else {
        # Get current node and its outlist
        temp       <- sequence[i]
        outlist    <- adjacencyMatrix[ temp , ]
        # Compute minimum weight for remaining nodes
        pos        <- setdiff( order( outlist ) , sequence )[ 1 ]
        weight     <- outlist[ pos ]
        # Update local storage objects
        pathWeight <- pathWeight + weight
        sequence   <- c( sequence , pos )
      }
    }
    # Update globol storage ubject for node j
    paths[j , ]  <- sequence
    distances[j] <- pathWeight
  }
  # Find minum and return final output
  argMin  <- which( distances == min( distances ) )
  optDist <- distances[ argMin ]
  optPath <- paths[ argMin , ]
  greedySolution <- list( minWeightPath = optPath, minDistance = optDist )
  return( greedySolution )
}

################################################################################
### Results
################################################################################

# Compute runtime and results
start.time <- proc.time()
tsp        <- greedySalesMan(data)
runtime    <- proc.time() - start.time

# Shuffle coordinates according to min. cycle for line plot
d02 <- data[ tsp$minWeightPath , ]

# Plot cycle
png('tsp.png')
  plot( data[,1] , data[,2] ,  xlab = '', ylab = '',
        axes = FALSE , cex  = .3, pch  = 19 )
  par( new = TRUE )
  lines( d02[,1] , d02[,2] , col = 'red' )
dev.off()

################################################################################

################################################################################
```

```
(* Reset printing options *)
SetOptions[SelectedNotebook[],
 PrintingStyleEnvironment → "Printout", ShowSyntaxStyles → True]

(* Set working directory and load data *)
SetDirectory[""];
data = Import["uy734.tsp.txt", "Table"];

(* Compute euclidian distance between points*)
distances =
  Outer[EuclideanDistance, data[[All, 2 ;; 3]], data[[All, 2 ;; 3]], 1];

In[5]:= (* Create object of type graph (edges are weighted by euclidian distance *)
graph = WeightedAdjacencyGraph[distances, DirectedEdges → False];

(* Compute shortest path ( Method Greedy as benchmark ) *)
tspSol = FindShortestTour[graph, Method → "Greedy"];

(* Plot results *)
p01 = ListPlot[data[[All, 2 ;; 3]],
   Axes → False, PlotStyle → PointSize[0.01], ImageSize → Full];
p02 = ListLinePlot[data[[Last[tspSol], 2 ;; 3]], Axes → False,
   PlotStyle → {Red, Dashed}, ImageSize → Full];
plotFinal = GraphicsRow[{Show[p01, p02],
   StringJoin["Distance of greedy tour: " , ToString[ First[tspSol]]]},
  PlotLabel → "Shortest greedy Tour", Frame → True]
```