Stochastic Modeling and Optimization Problemset 2

Problem 5: Traveling Salesman

Aimee Barciauskas, Andreas Lloyd, Francis Thomas Moynihan IV, and Seda Yilmaz

March 2, 2016

Below is code for implementing the Held-Karp Algoritm on the Uruguay data set.

The functions are in order from highest to lowest abstraction.

```
cities <- as.matrix(read.csv('uruguay.tsv', sep = ' ', header = FALSE))
path <- travelling.salesman(cities)

# references:
# https://en.wikipedia.org/wiki/Held-Karp_algorithm
# http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/CG-Applets/TSP/notspcli.htm
# http://www.math.uwaterloo.ca/tsp/world/uytour.html
# this greedy solution returns a distance of 96514.81
# current optimal distance is 79114
```

travelling.salesman finds a minimal path from any origin to return without visiting a city twice.

```
source('distances.R')
source('greedySpanningTree.R')

# Find a minimal path from any origin to return without visiting a city twice
#
travelling.salesman <- function(cities) {
  # Initialize the number of cities to work through and distances between them
  num.cities <- nrow(cities)
  distances <- distances(cities)

  # initialize data objects for storing distances and paths to enable returning the best solution
  all.distances <- rep(0, length(num.cities))
  # paths is a matrix to store each path returned from greedy spanning tree
  # will be a matrix num.cities x num.cities matrix, each city having itself as the origin
  # and a greedy path stemming from it
  all.paths <- matrix(0, nrow = num.cities, ncol = num.cities)

  # Loop through each city, find a greedy spanning tree
  # add it's total distance and path to all.distances and all.paths
  for (i in 1:num.cities) {
    greedy.path <- greedy.spanning.tree(i, distances)
    all.distances[i] <- greedy.path[['total.distance']]
    all.paths[i,] <- greedy.path[['path']]
  }

  min.distance.idx <- which.min(all.distances)
  min.distance <- all.distances[min.distance.idx]
  min.path <- all.paths[min.distance.idx,]
```

```
  return(list(min.path = min.path, min.distance = min.distance))
}
```

greedy.spanning.tree greedily finds a minimized cost path from origin node to every other node (w/o cycles).

```
# `greedy.spanning.tree`
#
#  greedy.spanning.tree greedily finds a path from origin node
#   to every other node (w/o cycles)
#   but this is not the minimal cost
#
# Naming is purposefully generic, to facilitate use outside of the travelling salesman problem
#  Although, given it computes a complete path from origin to return,
#  does seem to purposed for solving that problem
#
# greedy.spanning.tree takes 2 arguments:
#   `origin.node`: the index of the origin node
#   `costs`: a matrix of dimension of the number of nodes in the graph
#      indiciating the cost to travel from each node to any other nodes in the graph
#
# greedy.spanning.tree returns a list with elements
#   `path`: an ordered list of nodes visited
#   `total.distance`: the total distance from the first node through each subsequent node
#
greedy.spanning.tree <- function(origin.node = 1, costs = matrix()) {
  current.node <- origin.node
  total.distance <- 0
  num.nodes <- nrow(costs)
  visited <- c(origin.node)

  while (length(visited) < num.nodes) {
    ordered.costs <- order(costs[,current.node])
    nearest.neighbor <- ordered.costs[which(!(ordered.costs %in% visited))][1]
    total.distance <- total.distance + costs[current.node, nearest.neighbor]
    visited <- append(visited, nearest.neighbor)
    current.node <- nearest.neighbor
  }

  # Add final distance from last node to return ot the origin node
  final.distance <- total.distance + costs[origin.node, visited[num.nodes]]
  return(list(total.distance = final.distance, path = visited))
}
```

distances calculates the euclidean distance between every row in a matrix.

```
if (!require('Rcpp')) install.packages('Rcpp')
Rcpp::sourceCpp('distance.cpp')

# Calculate the euclidean distance between every row in a matrix
distances <- function(matrix) {
  n <- nrow(matrix)
  dist.matrix <- matrix(NA, n, n)
```

```
  for (idx in 1:n) {
    x.current <- matrix[idx,]
    x.current.expanded <- matrix(x.current, nrow = n, ncol = 2, byrow = TRUE)
    dist.matrix[idx, ] <- calcDist(matrix, x.current.expanded, 2)
  }
  return(dist.matrix)
}
```

```
#include <Rcpp.h>
#include <math.h>
using namespace Rcpp;

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar). Learn
// more about Rcpp at:
//
//   http://www.rcpp.org/
//   http://adv-r.had.co.nz/Rcpp.html
//   http://gallery.rcpp.org/
//

// [[Rcpp::export]]
// Credit: http://adv-r.had.co.nz/Rcpp.html
NumericVector rowSumsC(NumericMatrix x) {
  int nrow = x.nrow(), ncol = x.ncol();
  NumericVector out(nrow);

  for (int i = 0; i < nrow; i++) {
    double total = 0;
    for (int j = 0; j < ncol; j++) {
      total += x(i, j);
    }
    out[i] = total;
  }
  return out;
}

// [[Rcpp::export]]
NumericVector calcDist(NumericMatrix x, NumericMatrix y, int p) {
  NumericMatrix xx(x.nrow(), x.ncol());
  for (int i=0; i<x.ncol(); ++i) {
    xx(_,i) = pow(x(_,i)-y(_,i), p);
  }
  float power = 1/(float)p;
  return pow(rowSumsC(xx),power);
}
```