Traveling Salesman Problem TSP

Dominik Ramsauer - Lehrstuhl für Informationswissenschaft - Universität Regensburg

9 April 2018

In this task the aim is to find the shortest route through beer gardens in Regensburg. To solve this problem a 1-dim circular Self-Organizing Map (SOM) with 18 nodes shall be applied.

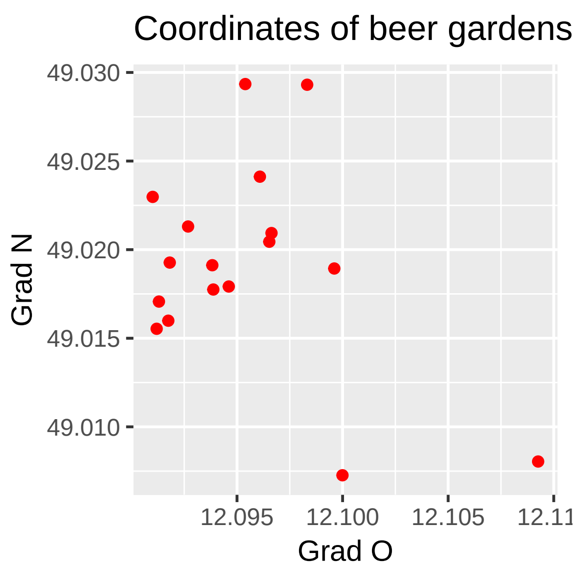
Dataset

Given is a dataset with coordinates of 17 beer gardens in Regensburg.

```
beer_gardens
```

```
##      Grad O   Grad N
## 1  12.09999 49.00727
## 2  12.09961 49.01894
## 3  12.09101 49.02298
## 4  12.09539 49.02934
## 5  12.09268 49.02131
## 6  12.09461 49.01792
## 7  12.09653 49.02045
## 8  12.09130 49.01707
## 9  12.09175 49.01599
## 10 12.09609 49.02412
## 11 12.09832 49.02931
## 12 12.09120 49.01553
## 13 12.09383 49.01912
## 14 12.09663 49.02094
## 15 12.10926 49.00804
## 16 12.09388 49.01775
## 17 12.09182 49.01927
```

```
ggplot(beer_gardens, aes(x=`Grad O`, y=`Grad N`))+
  geom_point(aes(),color="red")+
  ggtitle("Coordinates of beer gardens in Regensburg")
```



Geo-Map created with orange3 Geo widget.

Implementation SOM

As functions the following algorithm were implemented.

```
neighborhood <- function(n_nodes, winner, k, sigma) {
  #'
  #' @param n_nodes amount of nodes used for SOM
  #' @param winner winning neuron in this iteration step
  #' @param k iteration parameter over the nodes
  #' @param sigma radius of the neighborhood function
  #' @return neighborhood-factor influencing the SOM learning rule
  nodes <- c(1:n_nodes)
  distance  <- sqrt((nodes[winner] - nodes[k]) ** 2)
  distance  <- min(distance, n_nodes-distance)
  h <- 1 / exp((distance ** 2) / (2 * sigma ** 2))
  return(h)
}
determine_winner <- function(n_nodes, coordinates, w, current_coordinates) {
  #' Function to determine the winning neuron
  #' @param n_nodes amount of nodes used for SOM
  #' @param coordinates matrix containing coordinations of places in TSP
  #' @param w matrix which will get updated in SOM and contain winning neurons later
  #' @param current_coordinates currently considered place in TSP
  #' @return the current winning neuron
  differences <- c()
  for (i in c(1:n_nodes)) {
    differences[i] <-
      sqrt(
        (w[i, 1] - coordinates[current_coordinates, 1]) ** 2 +
        (w[i, 2] - coordinates[current_coordinates, 2]) ** 2)
  }
  minimum <- min(differences)
  winner <- as.integer(which(differences == minimum))
  return(winner)
}

learning_rate <- function(rate, t) {
  #' @param rate learning rate value
  #' @param t iteration step
  #' @return the learning rate or step width
  return(0.01 + rate * t ** (-0.5))
}
sigma <- function(sigma, t) {
  #' Halbwertsbreite der neighborhood-function
  #' @param sigma radius value
  #' @param t iteration step
  #' @return radius at step t
  return(sigma / (t ** 0.5))
}

som <- function(n_nodes, coordinates, w, sigma, rate, iterations=8000, save_plots=FALSE){
  #' Main algorithm for a Self-Organizing-Map for the Travelling Salesman Problem
  #' @param n_nodes amount of nodes used for SOM
  #' @param coordinates matrix containing coordinations of places in TSP
```

```r
  #' @param w matrix which will get updated in SOM and contain winning neurons later
  #' @param sigma radius value
  #' @param rate learning rate value
  #' @param iterations value how often SOM algorithm shall be executed
  #' @param save_plots if set to TRUE, the algorithm will save a plot for each iteration
  #' @return matrix w which contains the winning neurons with their coordinates

  # Iteration for each learning cycle
  for (t in c(1:iterations)) {
    # Iteration over all places in travelling salesman problem
    for (j in c(1:nrow(coordinates))) {

      # Determine the current winning neuron
      winner <- determine_winner(n_nodes,coordinates, w, j)

      # Update neurons
      sig <- sigma(sigma, t)
      for (k in c(1:n_nodes)) {
        # Update equatation:  w + (learning rate * neighborhood * coordinates)
        w[k,] <-
          w[k,] + learning_rate(rate, t) *
          neighborhood(n_nodes, winner, k, sig) *
          (coordinates[j,] - w[k,])
      }

    }

    # Plot every graph
    if (save_plots) {
      if (t %% 10 == 0) {
        gname <- paste(c("N", n_nodes, "-", t, ".png"), collapse = "")
        coord.df  <- as.data.frame(coordinates)
        w.df      <- as.data.frame(w)
        colnames(coord.df) <- c("Grad O", "Grad N")
        colnames(w.df)     <- c("Grad O", "Grad N")

        g <- ggplot() +
          geom_point(data=coord.df,aes(`Grad O`,`Grad N`),color = "red",shape=16,size=4)+
          geom_point(data=w.df,aes(`Grad O`,`Grad N`),color = "green", shape=4, size=5)+
          geom_path(data=w.df,aes(`Grad O`,`Grad N`),color = "green")+
          ggtitle(paste(c("Iteration: ", t, " Nodes: ", n_nodes), collapse = ""))
        ggsave(filename = gname, plot = g, path = "Exercise II - ANN/plots")
      }
    }
  }
  return(w)
}

initialize_w_random <- function(n_nodes, grad_o_low, grad_o_high,
                                          grad_n_low, grad_n_high) {
  #' Function for initializing matrix w with random values between certain ranges
  #' w which will contain the winning neurons later
  #' @param n_nodes amount of nodes used for SOM - matrix w will contain as many entries
```

```r
  #' @param grad_o_low min value for Grad O
  #' @param grad_o_high max value for Grad O
  #' @param grad_n_low min value for Grad N
  #' @param grad_n_high max value for Grad N
  #' @return matrix w with start points
  w <- matrix(nrow = n_nodes, ncol = 2)

  for (i in c(1:n_nodes)) {
    w[i, 1] <- runif(1, grad_o_low, grad_o_high)
    w[i, 2] <- runif(1, grad_n_low, grad_n_high)
  }
  colnames(w, c("Grad O","Grad N"), do.NULL = FALSE)
  return(w)
}
initialize_w_from_df <- function(n_nodes, coordinates.df) {
  #' Function for initializing matrix w with samples from given dateframe
  #' w which will contain the winning neurons later
  #' @param n_nodes amount of nodes used for SOM - matrix w will contain as many entries
  #' @param coordinates.df data.frame containing coordinations of places in TSP
  #' @return matrix w with start points
  w <- as.matrix(dplyr::sample_n(coordinates.df, n_nodes, replace = TRUE)) * 1000
  return(w)
}
```

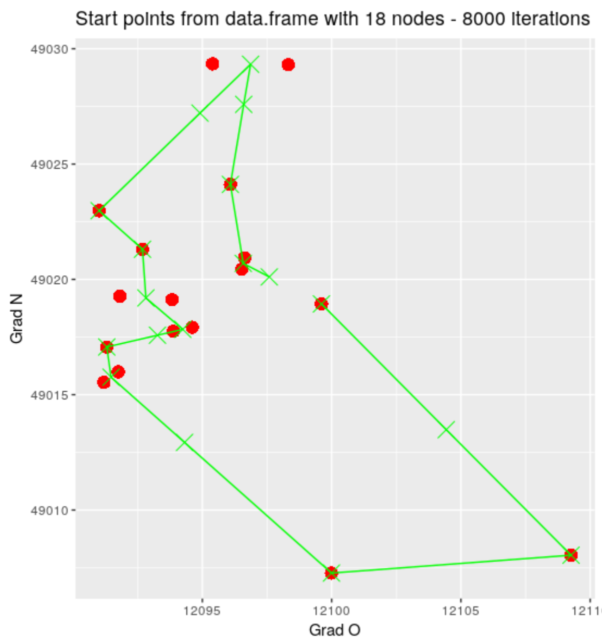
Parameters

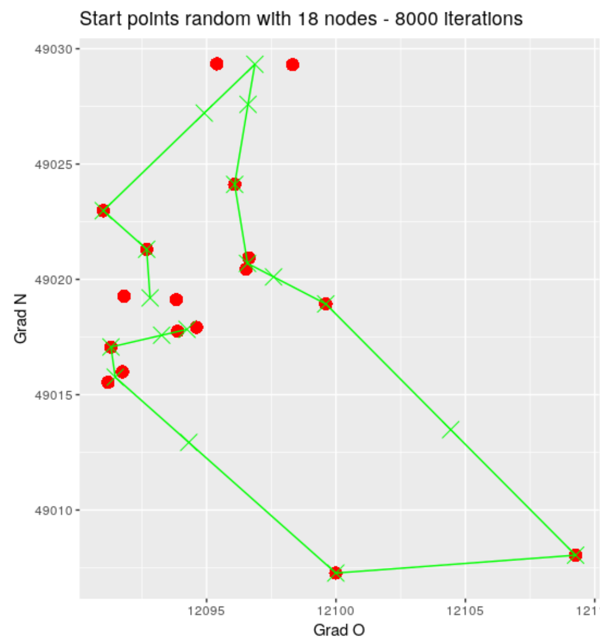
For the execution of the SOM-algorithm the following hyper-parameters were chosen. Furthermore the coordinates from beer_gardens were transformed into a matrix for a simpler computing.

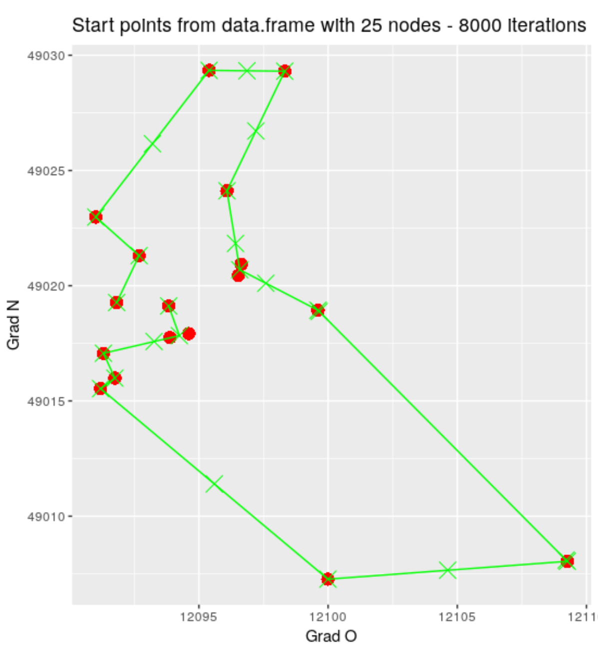
```r
# Create matrices for easier calculation
coordinates <- as.matrix(beer_gardens)
coordinates <- coordinates * 1000

# Learning rate value
rate <- 0.2
# Sigma - radius value for neighborhood function
s <- 18
# Amount of nodes
n_nodes <- 18
# Iterations for SOM executions
iterations <- 8000
```

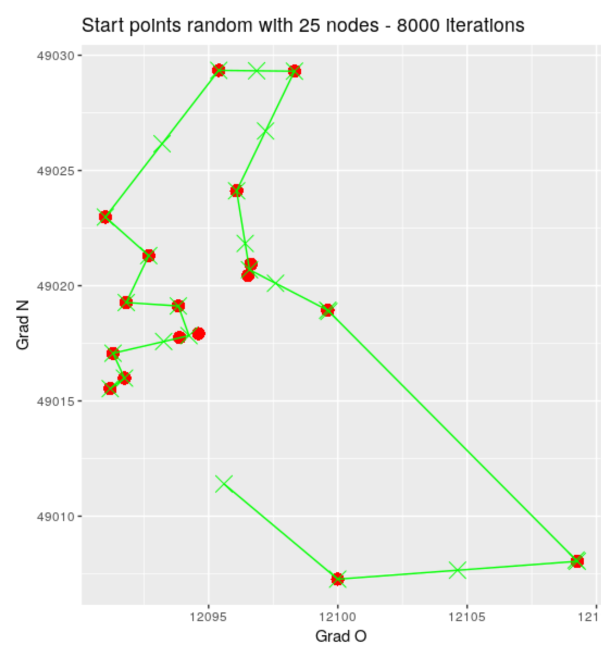
Results

The main aim was to create a self-organizing map mit 18 nodes, which you can see in the upper two plots. Moreover the SOM algorithm was executed with a larger amount of nodes with 30 nodes (lower two plots), to see how the map behaves then. To test if a complete random initialisation works worse or better than an initialisation with sample coordinates from the data set, two runs were done: the left plots are initialised with sample coordinates from beer_gardens, the right ones were initialised completely randomly within certain ranges (Grad 0: 12092-12180, Grad N: 49009-49028).









It seems like the initialisation with sample coordinates creates slightly better maps. But for a real evidence, a larger amount of data should be brought in. Besides that, it is outstanding that a larger amount of nodes makes the maps way better - the shortest way trough all vectors is modeled better and there no “dead units” that lay between two vectors and won’t optimize at a certain point.