Simulated Annealing

Nik Bear Brown

In this lesson we'll learn the theory behind using simulated annealing as an optimization and search technique. We'll then use simulated annealing to search for a solution to the famous Travelling Salesman Problem in R.

# Additional packages needed

To run the code in M07\_Lesson\_01.Rmd you may need additional packages.

* If necessary install the followings packages.

install.packages("ggplot2");  
install.packages("stats");

require(ggplot2)

## Loading required package: ggplot2

require(stats)

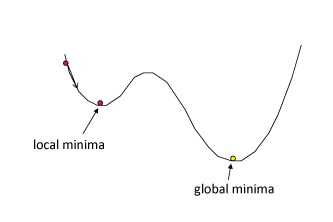
# Data

We will be using the R stats library mapping of distances between European cities to generate out data.

# Simulated Annealing

A Simulated Annealing (SA) is a probabilistic search heuristic that mimics the process of cooling in thermodynamic systems. This heuristic is often used to generate useful solutions to optimization and search problems. The method is an adaptation of the [Metropolis–Hastings algorithm](https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm), a Monte Carlo method to generate sample states of a thermodynamic system, invented by M.N. Rosenbluth and published in a paper by N. Metropolis et al. in 1953.

Basically simulated annealing perturbs the current solution and then checks to see whether the new solution is good or not. If its an improvement it will accept it and if the new solution is worse it may accept it with a probability inversely proportional to how much worse the new solution changes the current one. Compared to pure gradient descent the main difference is that SA allows "uphill" steps. Simulated annealing also differs from gradient descent in that a move is selected at random.

 *Gradient Descent*

## Metropolis–Hastings algorithm

[Metropolis algorithm](https://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm) (symmetric proposal distribution) Let f(x) be a function that is proportional to the desired probability distribution P(x)

This algorithm proceeds by randomly attempting to move about the sample space, sometimes accepting the moves and sometimes remaining in place.

Perturb (randomly) the current state to a new state. is the difference in energy between current and new state.  
If (new state is lower), accept new state as current state If accept new state with probability inversely proportional to the increase in system energy. Traditionally the change in [Gibbs free energy](https://en.wikipedia.org/wiki/Gibbs_free_energy) is used for thermodynamic free energy systems.

This can be run for a fixed number of iterations or if the overall system energy can measured then it can be run until the overall system energy settles.

## Simulated Annealing Pseudocode

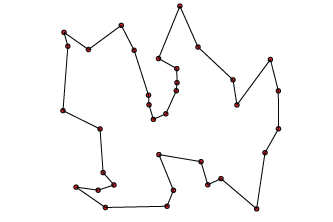
Simulated Annealing uses the Metropolis–Hastings algorithm with a temperature parameter that effects the acceptance probability of an "uphill" transition. At higher"temperatures" accpeting "uphill" transitions is more probable. The algorithm starts initially with set to a high value , and then it is decreased at each step following some annealing schedule—which is often specified by the user, but must end with . At there is no chance of accpeting "uphill" transitions and so it becomes gradient descent.

At a fixed temperature T:  
Perturb (randomly) the current state to a new state. is the difference in energy between current and new state.  
If (new state is lower), accept new state as current state If accept new state with probability inversely proportional to the increase in system energy as a function of T.

Eventually the systems evolves into thermal equilibrium at temperature T ; then the formula mentioned before holds When equilibrium is reached, temperature T can be lowered and the process can be repeated.

# Travelling Salesman Problem

The travelling salesman problem (TSP) asks the following question: 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? It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

 *Travelling Salesman Problem*

# Simulated Annealing to solve the Travelling Salesman Problem in R

Simulated Annealing to solve the Travelling Salesman Problem in R

CityDistMatx <- as.matrix(eurodist)  
CityDistMatx

## Athens Barcelona Brussels Calais Cherbourg Cologne  
## Athens 0 3313 2963 3175 3339 2762  
## Barcelona 3313 0 1318 1326 1294 1498  
## Brussels 2963 1318 0 204 583 206  
## Calais 3175 1326 204 0 460 409  
## Cherbourg 3339 1294 583 460 0 785  
## Cologne 2762 1498 206 409 785 0  
## Copenhagen 3276 2218 966 1136 1545 760  
## Geneva 2610 803 677 747 853 1662  
## Gibraltar 4485 1172 2256 2224 2047 2436  
## Hamburg 2977 2018 597 714 1115 460  
## Hook of Holland 3030 1490 172 330 731 269  
## Lisbon 4532 1305 2084 2052 1827 2290  
## Lyons 2753 645 690 739 789 714  
## Madrid 3949 636 1558 1550 1347 1764  
## Marseilles 2865 521 1011 1059 1101 1035  
## Milan 2282 1014 925 1077 1209 911  
## Munich 2179 1365 747 977 1160 583  
## Paris 3000 1033 285 280 340 465  
## Rome 817 1460 1511 1662 1794 1497  
## Stockholm 3927 2868 1616 1786 2196 1403  
## Vienna 1991 1802 1175 1381 1588 937  
## Copenhagen Geneva Gibraltar Hamburg Hook of Holland Lisbon  
## Athens 3276 2610 4485 2977 3030 4532  
## Barcelona 2218 803 1172 2018 1490 1305  
## Brussels 966 677 2256 597 172 2084  
## Calais 1136 747 2224 714 330 2052  
## Cherbourg 1545 853 2047 1115 731 1827  
## Cologne 760 1662 2436 460 269 2290  
## Copenhagen 0 1418 3196 460 269 2971  
## Geneva 1418 0 1975 1118 895 1936  
## Gibraltar 3196 1975 0 2897 2428 676  
## Hamburg 460 1118 2897 0 550 2671  
## Hook of Holland 269 895 2428 550 0 2280  
## Lisbon 2971 1936 676 2671 2280 0  
## Lyons 1458 158 1817 1159 863 1178  
## Madrid 2498 1439 698 2198 1730 668  
## Marseilles 1778 425 1693 1479 1183 1762  
## Milan 1537 328 2185 1238 1098 2250  
## Munich 1104 591 2565 805 851 2507  
## Paris 1176 513 1971 877 457 1799  
## Rome 2050 995 2631 1751 1683 2700  
## Stockholm 650 2068 3886 949 1500 3231  
## Vienna 1455 1019 2974 1155 1205 2937  
## Lyons Madrid Marseilles Milan Munich Paris Rome Stockholm  
## Athens 2753 3949 2865 2282 2179 3000 817 3927  
## Barcelona 645 636 521 1014 1365 1033 1460 2868  
## Brussels 690 1558 1011 925 747 285 1511 1616  
## Calais 739 1550 1059 1077 977 280 1662 1786  
## Cherbourg 789 1347 1101 1209 1160 340 1794 2196  
## Cologne 714 1764 1035 911 583 465 1497 1403  
## Copenhagen 1458 2498 1778 1537 1104 1176 2050 650  
## Geneva 158 1439 425 328 591 513 995 2068  
## Gibraltar 1817 698 1693 2185 2565 1971 2631 3886  
## Hamburg 1159 2198 1479 1238 805 877 1751 949  
## Hook of Holland 863 1730 1183 1098 851 457 1683 1500  
## Lisbon 1178 668 1762 2250 2507 1799 2700 3231  
## Lyons 0 1281 320 328 724 471 1048 2108  
## Madrid 1281 0 1157 1724 2010 1273 2097 3188  
## Marseilles 320 1157 0 618 1109 792 1011 2428  
## Milan 328 1724 618 0 331 856 586 2187  
## Munich 724 2010 1109 331 0 821 946 1754  
## Paris 471 1273 792 856 821 0 1476 1827  
## Rome 1048 2097 1011 586 946 1476 0 2707  
## Stockholm 2108 3188 2428 2187 1754 1827 2707 0  
## Vienna 1157 2409 1363 898 428 1249 1209 2105  
## Vienna  
## Athens 1991  
## Barcelona 1802  
## Brussels 1175  
## Calais 1381  
## Cherbourg 1588  
## Cologne 937  
## Copenhagen 1455  
## Geneva 1019  
## Gibraltar 2974  
## Hamburg 1155  
## Hook of Holland 1205  
## Lisbon 2937  
## Lyons 1157  
## Madrid 2409  
## Marseilles 1363  
## Milan 898  
## Munich 428  
## Paris 1249  
## Rome 1209  
## Stockholm 2105  
## Vienna 0

# Distance function  
distance <- function(sq)   
 { # Target function  
 sq2 <- embed(sq, 2)  
 return(as.numeric(sum(CityDistMatx[cbind(sq2[,2],sq2[,1])])))  
}  
  
# Generate new candidates  
GenSeq <- function(sq) { # Generate new candidate sequence  
 idx <- seq(2, NROW(CityDistMatx)-1, by=1)  
 ChangePoints <- sample(idx, size=2, replace=FALSE)  
 tmp <- sq[ChangePoints[1]]  
 sq[ChangePoints[1]] <- sq[ChangePoints[2]]  
 sq[ChangePoints[2]] <- tmp  
 return(as.numeric(sq))  
}  
  
cities<-labels(eurodist)  
cities

## [1] "Athens" "Barcelona" "Brussels"   
## [4] "Calais" "Cherbourg" "Cologne"   
## [7] "Copenhagen" "Geneva" "Gibraltar"   
## [10] "Hamburg" "Hook of Holland" "Lisbon"   
## [13] "Lyons" "Madrid" "Marseilles"   
## [16] "Milan" "Munich" "Paris"   
## [19] "Rome" "Stockholm" "Vienna"

initial.tour <- c(1,2:NROW(CityDistMatx),1)   
# Initial sequence  
initial.tour

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 1

initial.d<-distance(initial.tour)  
initial.d

## [1] 29625

for(i in 1:length(initial.tour))  
{  
 print(cities[initial.tour[i]])   
}

## [1] "Athens"  
## [1] "Barcelona"  
## [1] "Brussels"  
## [1] "Calais"  
## [1] "Cherbourg"  
## [1] "Cologne"  
## [1] "Copenhagen"  
## [1] "Geneva"  
## [1] "Gibraltar"  
## [1] "Hamburg"  
## [1] "Hook of Holland"  
## [1] "Lisbon"  
## [1] "Lyons"  
## [1] "Madrid"  
## [1] "Marseilles"  
## [1] "Milan"  
## [1] "Munich"  
## [1] "Paris"  
## [1] "Rome"  
## [1] "Stockholm"  
## [1] "Vienna"  
## [1] "Athens"

set.seed(333) # chosen to get a good soln relatively quickly  
  
# box-constrained optimization and simulated annealing  
# method = "SANN" performs simulated annealing  
# Method "SANN" is by default a variant of simulated annealing given in Belisle (1992)  
res <- optim(initial.tour, distance, GenSeq, method = "SANN",  
 control = list(maxit = 30000, temp = 2000, trace = TRUE,  
 REPORT = 500))

## sann objective function values  
## initial value 29625.000000  
## iter 5000 value 13044.000000  
## iter 10000 value 13044.000000  
## iter 15000 value 12907.000000  
## iter 20000 value 12907.000000  
## iter 25000 value 12907.000000  
## iter 29999 value 12907.000000  
## final value 12907.000000  
## sann stopped after 29999 iterations

res # Near optimum distance around 12842

## $par  
## [1] 1 19 16 15 2 14 9 12 13 8 18 5 4 3 11 7 20 10 6 17 21 1  
##   
## $value  
## [1] 12907  
##   
## $counts  
## function gradient   
## 30000 NA   
##   
## $convergence  
## [1] 0  
##   
## $message  
## NULL

final.tour<-res$par  
final.tour

## [1] 1 19 16 15 2 14 9 12 13 8 18 5 4 3 11 7 20 10 6 17 21 1

final.d<-distance(final.tour)  
final.d

## [1] 12907

initial.d

## [1] 29625

final.d/initial.d

## [1] 0.4356793

cities.xy <- cmdscale(eurodist)  
cities.xy

## [,1] [,2]  
## Athens 2290.274680 1798.80293  
## Barcelona -825.382790 546.81148  
## Brussels 59.183341 -367.08135  
## Calais -82.845973 -429.91466  
## Cherbourg -352.499435 -290.90843  
## Cologne 293.689633 -405.31194  
## Copenhagen 681.931545 -1108.64478  
## Geneva -9.423364 240.40600  
## Gibraltar -2048.449113 642.45854  
## Hamburg 561.108970 -773.36929  
## Hook of Holland 164.921799 -549.36704  
## Lisbon -1935.040811 49.12514  
## Lyons -226.423236 187.08779  
## Madrid -1423.353697 305.87513  
## Marseilles -299.498710 388.80726  
## Milan 260.878046 416.67381  
## Munich 587.675679 81.18224  
## Paris -156.836257 -211.13911  
## Rome 709.413282 1109.36665  
## Stockholm 839.445911 -1836.79055  
## Vienna 911.230500 205.93020

for(i in 1:length(final.tour))  
{  
 print(cities[final.tour[i]])   
}

## [1] "Athens"  
## [1] "Rome"  
## [1] "Milan"  
## [1] "Marseilles"  
## [1] "Barcelona"  
## [1] "Madrid"  
## [1] "Gibraltar"  
## [1] "Lisbon"  
## [1] "Lyons"  
## [1] "Geneva"  
## [1] "Paris"  
## [1] "Cherbourg"  
## [1] "Calais"  
## [1] "Brussels"  
## [1] "Hook of Holland"  
## [1] "Copenhagen"  
## [1] "Stockholm"  
## [1] "Hamburg"  
## [1] "Cologne"  
## [1] "Munich"  
## [1] "Vienna"  
## [1] "Athens"

rx <- range(x <- cities.xy[,1])  
ry <- range(y <- -cities.xy[,2])  
rx

## [1] -2048.449 2290.275

ry

## [1] -1798.803 1836.791

x

## Athens Barcelona Brussels Calais   
## 2290.274680 -825.382790 59.183341 -82.845973   
## Cherbourg Cologne Copenhagen Geneva   
## -352.499435 293.689633 681.931545 -9.423364   
## Gibraltar Hamburg Hook of Holland Lisbon   
## -2048.449113 561.108970 164.921799 -1935.040811   
## Lyons Madrid Marseilles Milan   
## -226.423236 -1423.353697 -299.498710 260.878046   
## Munich Paris Rome Stockholm   
## 587.675679 -156.836257 709.413282 839.445911   
## Vienna   
## 911.230500

y

## Athens Barcelona Brussels Calais   
## -1798.80293 -546.81148 367.08135 429.91466   
## Cherbourg Cologne Copenhagen Geneva   
## 290.90843 405.31194 1108.64478 -240.40600   
## Gibraltar Hamburg Hook of Holland Lisbon   
## -642.45854 773.36929 549.36704 -49.12514   
## Lyons Madrid Marseilles Milan   
## -187.08779 -305.87513 -388.80726 -416.67381   
## Munich Paris Rome Stockholm   
## -81.18224 211.13911 -1109.36665 1836.79055   
## Vienna   
## -205.93020

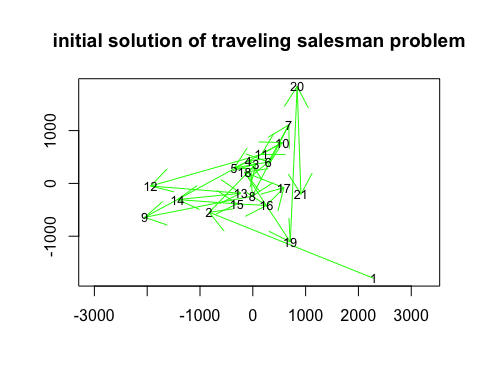
initial.tour

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 1

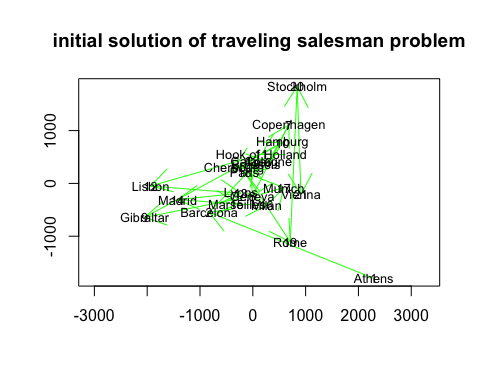
## remove last element to draw arrows from point to point  
s <-head(initial.tour, -1)  
s

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

plot(x, y, type="n", asp=1, xlab="", ylab="", main="initial solution of traveling salesman problem")  
arrows(x[s], y[s], x[s+1], y[s+1], col="green")  
text(x, y, labels(cities), cex=0.8)



plot(x, y, type="n", asp=1, xlab="", ylab="", main="initial solution of traveling salesman problem")  
arrows(x[s], y[s], x[s+1], y[s+1], col="green")  
text(x, y, labels(cities), cex=0.8)  
text(x, y, labels(eurodist), cex=0.8)



final.tour

## [1] 1 19 16 15 2 14 9 12 13 8 18 5 4 3 11 7 20 10 6 17 21 1

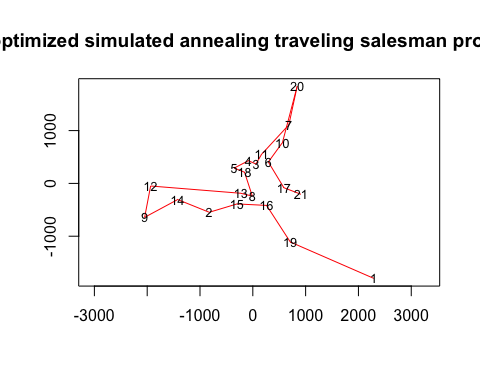
## draw lines from point to point  
s <-head(final.tour, -1)  
s

## [1] 1 19 16 15 2 14 9 12 13 8 18 5 4 3 11 7 20 10 6 17 21

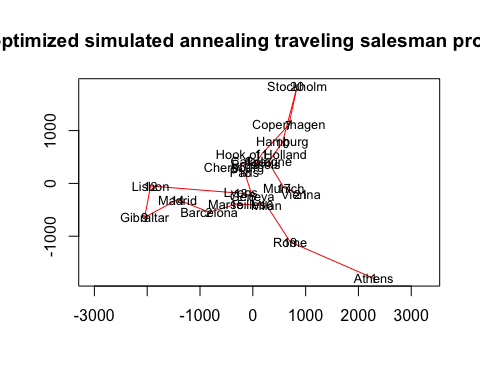
df = data.frame(x[s],y[s])  
df

## x.s. y.s.  
## Athens 2290.274680 -1798.80293  
## Rome 709.413282 -1109.36665  
## Milan 260.878046 -416.67381  
## Marseilles -299.498710 -388.80726  
## Barcelona -825.382790 -546.81148  
## Madrid -1423.353697 -305.87513  
## Gibraltar -2048.449113 -642.45854  
## Lisbon -1935.040811 -49.12514  
## Lyons -226.423236 -187.08779  
## Geneva -9.423364 -240.40600  
## Paris -156.836257 211.13911  
## Cherbourg -352.499435 290.90843  
## Calais -82.845973 429.91466  
## Brussels 59.183341 367.08135  
## Hook of Holland 164.921799 549.36704  
## Copenhagen 681.931545 1108.64478  
## Stockholm 839.445911 1836.79055  
## Hamburg 561.108970 773.36929  
## Cologne 293.689633 405.31194  
## Munich 587.675679 -81.18224  
## Vienna 911.230500 -205.93020

plot(x, y, type="n", asp=1, xlab="", ylab="", main="optimized simulated annealing traveling salesman problem")  
lines(df$x, df$y, col="red")  
text(x, y, labels(cities), cex=0.8)



plot(x, y, type="n", asp=1, xlab="", ylab="", main="optimized simulated annealing traveling salesman problem")  
lines(df$x, df$y, col="red")  
text(x, y, labels(cities), cex=0.8)  
text(x, y, labels(eurodist), cex=0.8)



# Assingment

Use simulated annealing to search for a solution to the famous Travelling Salesman Problem in R.  
\* . Answer the following questions:  
 \* Write a different distance function. How does the effect the final solution?  
 \* Does scaling, normalization or leaving the data unscaled make a difference for simulated annealing?  
 \* Try a few differnt initial tours. How does the effect the final solution?

# Resources

* [The Traveling Salesman with Simulated Annealing, R, and Shiny](http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/)
* [Simulated Annealing Feature Selection](http://www.r-bloggers.com/simulated-annealing-feature-selection/)

```