Tabu Search

```r
# Loss Function
L = function(A, B, C, t_c, m, omega, phi, price, t){
  loss <- sum((price - (A + B*abs(t_c - t)^m +
                        (C*abs(t_c-t)^m*cos(omega * log(abs(t_c - t)) - phi))))^2)
}

TabuSearch <- function(A, B, C){
  S <- hash() # S represents current solution
  iter_without_improvement <- 0

  # Generating a set of initial 10 * 3 parameters
  t_c_set <- runif(n = 40, min = max(base_data$t), max = max(base_data$t) + 0.6)
                # adding 0.6 year represent time horizon within the crash is expected
  m_set <- runif(n = 40, min = 0.1, max = 0.9)
  omega_set <- runif(n = 40, min = 6, max = 13)
  phi_set <- runif(n = 40, min = 0, max = 2*pi)

  # Looking for the 10 elite solutions out of 30 initial points,
  #the best one is then our starting point. Furthermore, we choose
  #the worst solution to set the taboo condition.
  elite_list <- vector()
  random_solutions <- cbind(t_c_set, m_set, omega_set, phi_set)
  losses <- apply(random_solutions, 1, function(x) L(A = A, B = B, C = C, t_c = x[1],
                                                     m = x[2], omega = x[3], phi = x[4],
                                                     price = base_data$price,
                                                     t = base_data$t))
  losses <- as.data.frame(cbind('loss' = losses, 't_c' = t_c_set,
                                'm' = m_set, 'omega' = omega_set, 'phi' = phi_set))
  losses <- losses %>%
    arrange(loss)
  elite_list <- losses[1:10,]
  taboo_condition <- losses[nrow(losses),1]
  S[['loss']] <- losses[1,'loss']
  S[['t_c']] <- losses[1,'t_c']
  S[['m']] <- losses[1,'m']
  S[['omega']] <- losses[1,'omega']
  S[['phi']] <- losses[1,'phi']
  if (min(losses, na.rm = TRUE) < 200){
    # Partitioning and setting parameters for the number of randomly drawn cells
    #and points within them
    partitions <- c(6,6,6,6)
    n_c <- 2
    n_s <- 6
    t_c.partitions <- seq(from = max(base_data$t),
                          to = max(base_data$t) + 4,
                          length.out = partitions[1] + 1)
    m.partitions <- seq(from = 0.1, to = 0.9, length.out = partitions[2] + 1)
    omega.partitions <- seq(from = 6, to = 13, length.out = partitions[3] + 1)
    phi.partitions <- seq(from = 0, to = 2*pi, length.out = partitions[4] + 1)
    partitions_matrix <- rbind(t_c.partitions, m.partitions, omega.partitions,
```

```
                              phi.partitions)

  # Searching procedure
  while (iter_without_improvement < 100){
    # Drawing n_c * n_s points for looking for new solutions
    chosen_cells <- t(sapply(partitions, function(x) sample(1:x, size = n_c,
                                                            replace = FALSE)))
    drawn_points <-
      sapply(1:nrow(partitions_matrix),
             function(row) sapply(chosen_cells[row,],
                                  function(x) as.vector(sapply(x,function(y)
                                    runif(n = n_s, min = partitions_matrix[row,y],
                                          max = partitions_matrix[row,y + 1])))))
    colnames(drawn_points) <- c('t_c', 'm', 'omega', 'phi')

    # Computing the value of loss function for new points,
    #dropping points returning losses in a taboo region
    losses <- apply(drawn_points, 1, function(x) L(A = A, B = B, C = C, t_c = x[1],
                                                  m = x[2], omega = x[3], phi = x[4],
                                                  price = base_data$price,
                                                  t = base_data$t))
    drawn_points <- cbind('loss' = losses, drawn_points)
    # dropping all points with non-defined loss function
    drawn_points <- drawn_points[complete.cases(drawn_points),,drop = FALSE]

    losses <- losses[complete.cases(losses)] # dropping points from losses, too
    non.taboo <- ifelse(losses < taboo_condition, TRUE, FALSE)
    drawn_points <- drawn_points[non.taboo,,drop = FALSE]

    # Picking the nontaboo point with the lowest move value
    if (nrow(drawn_points) > 0){
      S[['loss']] <- drawn_points[which.min(drawn_points[,'loss'] - S$loss), 'loss']
      S[['t_c']] <- drawn_points[which.min(drawn_points[,'loss'] - S$loss), 't_c']
      S[['m']] <- drawn_points[which.min(drawn_points[,'loss'] - S$loss), 'm']
      S[['omega']] <- drawn_points[which.min(drawn_points[,'loss'] - S$loss), 'omega']
      S[['phi']] <- drawn_points[which.min(drawn_points[,'loss'] - S$loss), 'phi']

      # Executing elite_list modification in case of the loss of a new points
      #is lower than the loss of the 10th element in the elite_list
      if (S$loss < elite_list[10,'loss']){
        elite_list <- rbind(elite_list[1:9,],
                            drawn_points[which.min(drawn_points[,'loss'] - S$loss),]) %>%
          arrange(loss)
        iter_without_improvement <- 0
      } else {
        iter_without_improvement <- iter_without_improvement + 1
      }
    } else {
      iter_without_improvement <- iter_without_improvement + 1
    }
  }
}
# Filling results to the Grid
```

```
  grid <- grid %>%
    filter(a == A, b == B, c == C) %>%
    mutate(loss = elite_list[1,'loss'], t_c = elite_list[1,'t_c'], m = elite_list[1,'m'],
           omega = elite_list[1,'omega'], phi = elite_list[1,'phi'])
  print(elite_list)
  return(grid)
}
```