# Sprawozdanie 1 (algorytmy genetyczne)

**Autorzy:** Remigiusz Nowakowski  
**Grupa:** INMN1(hybryda)\_1.1, INMN1(hybryda)\_sp\_wszyscy   
**Data:** 12.11.2023  
**Przedmiot:** Inteligencja obliczeniowa

|  |  |
| --- | --- |
| **Cel zadania** | W gminie wiejskiej Nowe Miasto Lubawskie kuchnia szkolna w Bratianie przygotowuje obiady dla wszystkich szkół z gminy. Celem jest znalezienie jak najkrótszej możliwej trasy dostarczenia posiłków. Samochód z posiłkami rusza z Bratiana i wraca do Bratiana. |
| **Źródło danych** | Macierz odległości między miejscowościami. |
| **Struktura chromosomu** | Jest reprezentowana jako permutacja miejscowości. Chromosom zaczyna się i kończy w miejscowości "Bratian", a reszta miejscowości występuje dokładnie raz. Na poziomie implementacji, chromosom jest reprezentowany jako wektor liczb całkowitych, gdzie każda liczba oznacza indeks miejscowości. |
| **Które chromosomy są dobre?** | Długość najkrótszej znalezionej trasy: 60.5  Miejscowości na trasie:-> Bratian -> Tylice -> Gwiździny -> Skarlin -> Jamielnik -> Radomno -> Bratian > |
| **Działanie funkcji Fitness** | Funkcja fitness (f.przystosowania w tym przypadku) ocenia, jak "dobre" jest dane rozwiązanie, czyli dana trasa komiwojażera w problemie komiwojażera. W tym konkretnym przypadku, funkcja fitness jest zdefiniowana jako odwrotność sumy odległości między kolejnymi miejscowościami na trasie.  Oto jak działa funkcja fitness w tym kodzie:   1. Dodanie punktu startowego/końcowego: Linia trasa <- c(1, trasa, 1) dodaje miejsce początkowe (Bratian) na początku i na końcu trasy. To gwarantuje, że trasa zaczyna się i kończy w miejscowości "Bratian". 2. Tworzenie odcinków między miejscowościami: Linia odcinki <- embed(trasa, 2)[, 2:1] tworzy pary indeksów miejscowości, które stanowią kolejne odcinki trasy. 3. Obliczenie sumy odległości: Linia sum(macierzOdleglosci[odcinki]) oblicza sumę odległości między kolejnymi miejscowościami na trasie. 4. Odwrotność sumy odległości: Całość jest podzielona przez 1, aby uzyskać odwrotność sumy odległości. Im krótsza trasa, tym większa wartość funkcji fitness.   Funkcja fitness ma za zadanie maksymalizować tę odwrotność, co odpowiada minimalizacji sumy odległości na trasie. |
| **Wnioski i podsumowanie** | Algorytm genetyczny działa na zasadzie optymalizacji, starając się znaleźć trasę, która minimalizuje całkowity koszt (długość trasy) między miejscowościami. Podsumowując, algorytm genetyczny jest skutecznym narzędziem do rozwiązywania problemu komiwojażera, a dostosowywanie parametrów i analiza wyników są kluczowe dla osiągnięcia optymalnych tras. Warto również zauważyć, że problem komiwojażera ma wiele zastosowań w rzeczywistych sytuacjach, takich jak planowanie tras dostaw czy tras w systemach logistycznych. |
| **Bibliografia** | [1] Biblioteka GA  [2] Moodle  [3] Google maps (do stworzenia macierzy odległości między miejscowościami)  [4] gminanml.pl (informacje o placówkach) |

## Kod źródłowy rozwiązania wraz z dodatkowymi (szczegółowymi) komentarzami:

|  |
| --- |
| # Instalacja i załadowanie pakietu GA  # install.packages("GA")  library(GA)  # Miejscowości  miejscowosci <- c("Bratian", "Skarlin", "Radomno", "Gwiździny", "Tylice", "Jamielnik")  # Macierz odległości  macierzOdleglosci <- matrix(c(  0, 11.5, 8, 12, 8.5, 12,  11.5, 0, 13, 17, 16.5, 14,  8, 13, 0, 19, 15, 5,  12, 17, 19, 0, 8, 21,  8.5, 16.5, 15, 8, 0, 20,  12, 14, 5, 21, 20, 0  ), nrow = 6, byrow = TRUE)  # Funkcja oceny trasy  f.przystosowania <- function(trasa) {  trasa <- c(1, trasa, 1) # Trasa zaczyna i kończy się w Bratianie  odcinki <- embed(trasa, 2)[, 2:1]  if (any(duplicated(trasa[-1]))) {  # Kary za powtórzenie miejscowości  return(0)  }  return (1 / sum(macierzOdleglosci[odcinki]))  }  # Użycie algorytmu genetycznego  GA <- ga(type = "permutation", fitness = f.przystosowania,  lower = 2, upper = nrow(macierzOdleglosci), popSize = 40, maxiter = 2000,  run = 500, pmutation = 0.05, pcrossover = 0.6, elitism = 5, seed = 1975)  # Analiza wyników  summary(GA)  plot(GA)  # Funkcja rysująca trasę na mapie z etykietami miejscowości  rysuj <- function(trasa) {  x <- c(1, 2, 3, 4, 5, 6)  y <- c(1, 2, 3, 4, 5, 6)  plot(x, y, type = "n", asp = 1, xlab = "", ylab = "")  abline(h = pretty(range(x), 10), v = pretty(range(y), 10), col = "light gray")  trasa <- c(1, trasa, 1)  n <- length(trasa)  arrows(x[trasa[-n]], y[trasa[-n]], x[trasa[-1]], y[trasa[-1]],  length = 0.15, angle = 25, col = "steelblue", lwd = 2)  text(x, y, labels = miejscowosci, cex = 0.8)  text(x[1], y[1], "Start/End", pos = 4, col = "red", cex = 1.2)  for (index in 1:(length(trasa) - 1)) {  lines(c(x[trasa[index]], x[trasa[index + 1]]), c(y[trasa[index]], y[trasa[index + 1]]), col = "red")  }  }  # Rysowanie trasy  rysuj(GA@solution[1,])  # Funkcja dekodująca trasę  dekoduj <- function(trasa) {  trasa <- c(1, trasa, 1)  odcinki <- embed(trasa, 2)[, 2:1]  odleglosc <- sum(macierzOdleglosci[odcinki])  cat(paste("Ilość miejscowości:", nrow(macierzOdleglosci), "\n"))  cat(paste("Długość najkrótszej znalezionej trasy:", odleglosc, "\n"))  cat("Miejscowości na trasie:")  for (index in trasa) {  cat(paste("->", miejscowosci[index]), "")  }  }  # Wyświetlanie informacji o trasie  dekoduj(GA@solution[1,]) |