Problem Komiwojażera - Algorytmy Genetyczne

Ludwik Przyrowski

9 kwietnia 2017

Problem Komiwojażera (ang. Travelling Salesman Problem)

Poniższy skrypt jest rozwiązaniem zadań z laboratorium Sztucznej inteligejncji będącego przedmiotem na kierunku Data Science w Politechnice Warszawskiej prowadzonego przez Ph.D. Kamila Żbikowskiego [1].

Celem ćwiczenia jest rozwiązanie problemu komiwojażera (cyklu Hamiltona w pełnym grafie ważonym) z wykorzystaniem algorytmów genetycznych.

Zadanie 1

```
cities = read.csv("data/cities.csv", header = TRUE, sep=",")
head(cities)
```

```
##           Barcelona Belgrade  Berlin Brussels Bucharest Budapest
## Barcelona      0.00  1528.13 1497.61  1062.89   1968.42  1498.79
## Belgrade    1528.13     0.00  999.25  1372.59    447.34   316.41
## Berlin      1497.61   999.25    0.00   651.62   1293.40   689.06
## Brussels    1062.89  1372.59  651.62     0.00   1769.69  1131.52
## Bucharest   1968.42   447.34 1293.40  1769.69      0.00   639.77
## Budapest    1498.79   316.41  689.06  1131.52    639.77     0.00
##           Copenhagen  Dublin Hamburg Istanbul    Kiev  London  Madrid
## Barcelona    1757.54 1469.29 1471.78  2230.42 2391.06 1137.67  504.64
## Belgrade     1327.24 2145.39 1229.93   809.48  976.02 1688.97 2026.94
## Berlin        354.03 1315.16  254.51  1735.01 1204.00  929.97 1867.69
## Brussels      766.67  773.20  489.76  2178.85 1836.20  318.72 1314.30
## Bucharest    1571.54 2534.72 1544.17   445.62  744.44 2088.42 2469.71
## Budapest     1011.31 1894.95  927.92  1064.76  894.29 1450.12 1975.38
##             Milan  Moscow  Munich   Paris  Prague    Rome Saint.Petersburg
## Barcelona  725.12 3006.93 1054.55  831.59 1353.90  856.69          2813.02
## Belgrade   885.32 1710.99  773.33 1445.70  738.10  721.55          1797.75
## Berlin     840.72 1607.99  501.97  876.96  280.34 1181.67          1319.62
## Brussels   696.61 2253.26  601.87  261.29  721.08 1171.34          1903.66
## Bucharest 1331.46 1497.56 1186.37 1869.95 1076.82 1137.38          1740.39
## Budapest   788.56 1565.19  563.93 1247.61  443.26  811.11          1556.51
##             Sofia Stockholm  Vienna  Warsaw
## Barcelona 1745.55   2276.51 1347.43 1862.33
## Belgrade   329.46   1620.96  489.28  826.66
## Berlin    1318.67    810.38  523.61  516.06
## Brussels  1697.83   1280.88  914.81 1159.85
## Bucharest  296.68   1742.25  855.32  946.12
## Budapest   629.63   1316.59  216.98  545.29
```

Zadanie 2

Napisz funkcję, która obliczy dystans pomiędzy wszystkimi odwiedzanymi miastami.

Funkcja przyjmuje dwa parametry:

- visitedCities - wektor od długości k, gdzie k to liczba odwiedzonych miast; kolejność wartości w ramach tego wektora będzie odzwierciedlała kolejność odwiedzania miast,
- distances - macierz odległości między dowolnymi dwoma miastami

```
totalDistance = function(visitedCities, distances=cities) {
visitedCities = c(visitedCities, visitedCities[1])
route = embed(visitedCities, 2)[, 2:1]
distancesSum = sum(distances[route])
return(distancesSum)
}
```

szybki test:

```
totalDistance(c('Warsaw', 'London'))
```

```
## [1] 2891.7
```

Zadanie 3

Zastanówmy się nad rozwiązaniem, które przeszukiwałoby przestrzeń wszystkich możliwych połączeń pomiędzy miastami. W tym celu skorzystaj z poniższej funkcji:

```
permu <- function(perm, fun, current=NULL){
  for(i in 1: length(perm)){
    fix <- c(current, perm[i]) # calculated elements; fix at this point
    rest <- perm[-i] # elements yet to permutate
    #Call callback.
    if(!length(rest)){
      result <- fun(fix)
    if(result<bestResult){
      assign("bestResult", result, envir = .GlobalEnv)
      print(bestResult)
      }
    }
    if(length(rest)){
      result <- permu(rest, fun, fix)
    }
  }
}
```

Funkcja "permu" korzysta ze zmiennej globalnej bestResults zdefiniowanej jako:

```
bestResult <- 99999
```

(Definiowanie i korzystanie ze zmiennych globalnych jest antywzorcem)

Wywołaj funkcję "permu" dla fun=totalDistance i podaj adekwatną wartość parametru "perm" tak, aby przeszukiwana była cała przestrzeń rozwiązań.

```
permu(c('Belgrade','Berlin','Brussels','Bucharest',
        'Budapest','Copenhagen','Dublin', 'Hamburg')
      , totalDistance)
```

```
## [1] 8613.31
## [1] 8578.36
## [1] 8548.17
```

```
## [1] 8535.37
## [1] 8500.42
## [1] 8096.95
## [1] 8057.55
## [1] 8040.93
## [1] 7229.63
## [1] 6744.31
## [1] 6450.77
## [1] 5965.45
## [1] 5883.82
## [1] 5869.65
## [1] 5706.44
## [1] 5702.82
## [1] 5637.32
```

Widać, że obliczenia trwają bardzo długo więc przykład tylko dla ograniczonej listy

Zadanie 4

Teraz kiedy mamy już dosyć dobre zrozumienie problemu zastanówmy się nad jego złożonością obliczeniową w zależności od liczby miast. Czy jesteś w stanie podać aproksymację złożoności? Skorzystaj w tym celu z funkcji "factorial".
Złożoność można określić jako [2]:
Startując z pierwszego miasta mamy (n-1) miast do wyboru, z drugiego (n-2) miast do wyboru i tak dalej (n-1)(n-2)(n-3)...x3x2x1
itd. Jako, że koszt przejazdu mierzony jest w jedną stronę dzielimi wszystko przez dwa i finalnie:
(n-1)!/2

Zadanie 5

W tym i kolejnych zadaniach korzystać będziemy z funkcji z biblioteki "GA"[3]. Zainstaluj i załaduj ją przy pomocy poleceń:

```
library(GA)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Package 'GA' version 3.0.2
```

```
## Type 'citation("GA")' for citing this R package in publications.
```

... wprowadzenie do Algorytmów Genetycznych

Zadanie 6

Zaproponuj funkcję kosztu dla algorytmu genetycznego. ** Najprostrza funkcja do minimalizacji to -1 x koszt podróży

```
costFunction <-function(cfVisitedCities, ...){
  -(totalDistance(cfVisitedCities,...))
}
```

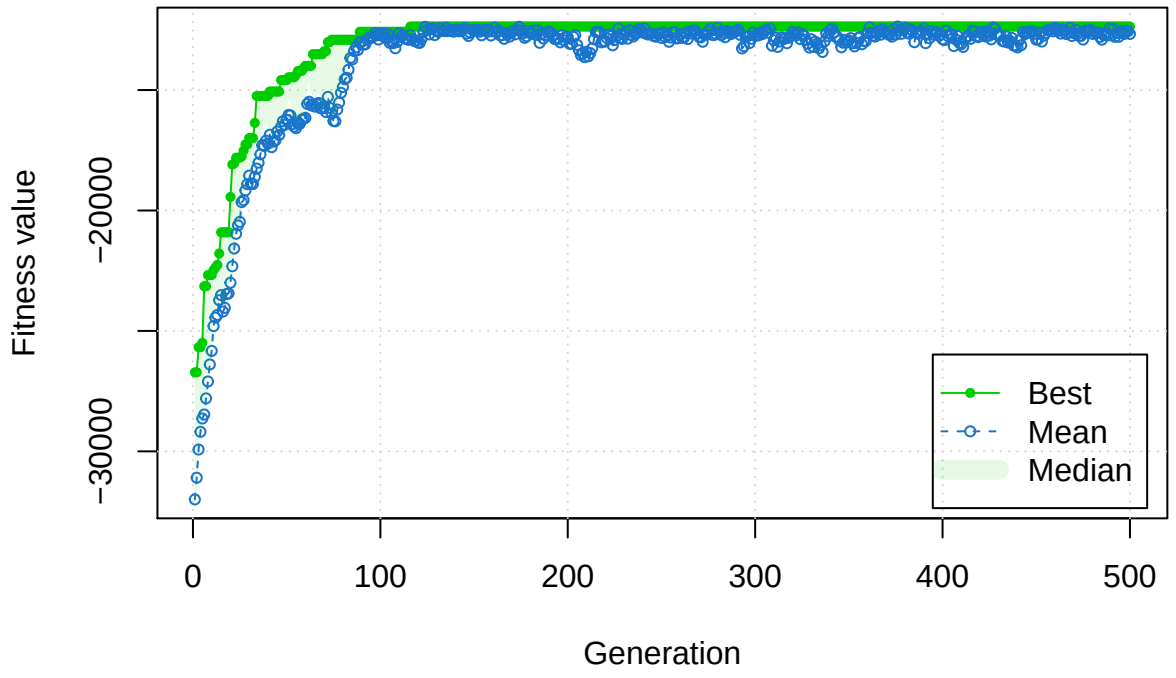
Zadanie 7

Skorzystaj z funkcji "ga" w celu wybrania optymalnej wartości trasy z ograniczoną liczbą iteracji do 500. Zapoznaj się z parametryzacją funkcji "min", "max" oraz "maxiter". Ustaw typ parametryzacji na "permutation". Zapoznaj się z innymi typami zmiennych decyzyjnych. Zakładając, że GA.fit zawiera wynik wykonania funkcji "ga" sprawdź najlepsze rozwiązanie oraz narysuj wykres dochodzenia do rozwiązania przez algorytm genetyczny.

```
GA.fit <- ga(
  type = "permutation"
  ,fitness = costFunction
  ,min = 1
  ,max = length(cities)
  ,maxiter = 500)
GA.fit@fitnessValue
```

```
## [1] -12362.92
```

```
plot(GA.fit)
```



a) Czy zauważyłeś problem z taką definicją zmiennych decyzyjnych w kontekście badanego problemu? Jaką faktycznie złożoność ma takie podejście?
Szczególnymi warunkami tego problemu są wymagania aby każde miejsce zostanie odwiedzone tylko raz. Jeśli nie będziemy mieli któregoś z miast lub pojawi się ono dwa razy to nie spełnimy warunków zagadnienia

b) Zwróć uwagę na fakt, iż zastosowanie operatorów krzyżowania i mutacji w wersji podstawowej spowodowałoby złamanie warunków określonych dla zmiennych decyzyjnych.

c) • Zaproponuj operator mutacji dla omawianego problemu.
Przedewszystkim powinniśmy zapewnić aby mutacja odbywała się poprzez tasowanie a nie losowanie które mogłoby prowadzić do niespełnienia założeń problemu. Jednym z rozwiązań [4] jest Mutacja Wymiany (ang. Swamp Mutation). W tej metodzie dwa losowo wybrane elementy (miasta) są zamieniane ze sobą miejscami

d) • Zaproponuj operator krzyżowania dla omawianego problemu.
Jednym ze sposobów jest Krzyżowanie Posortowane [4] (ang. Ordered Crossover). Wybieramy pozdzbiór kolejnych miast od rodzica a następnie dodajemy go do dzieci. Jakiekolwiek brakujące wartości dodawane są z innego rodzica z zachowaniem kolejności przy omijaniu powtarzających się miast z rodzica pierwszego.

Bibliografia

[1] K.Żbikowski, Materiały do laboratorium ze Sztucznej Inteligencji, Politechnika Warszawska, 2017
[2] http://www.math.uwaterloo.ca/tsp/problem/pcb3cnt.html
[3] Luca Scrucca (2013). GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, 53(4), 1-37. URL http://www.jstatsoft.org/v53/i04/.
[4] http://www.theprojectspot.com/tutorial-post/applying-a-genetic-algorithm-to-the-travelling-salesman-problem/5