Algorithme génétique pour le voyageur de commerce

Challenge M2 Data Science Algorithmique

Vincent Runge

vendredi 10 décembre 2021

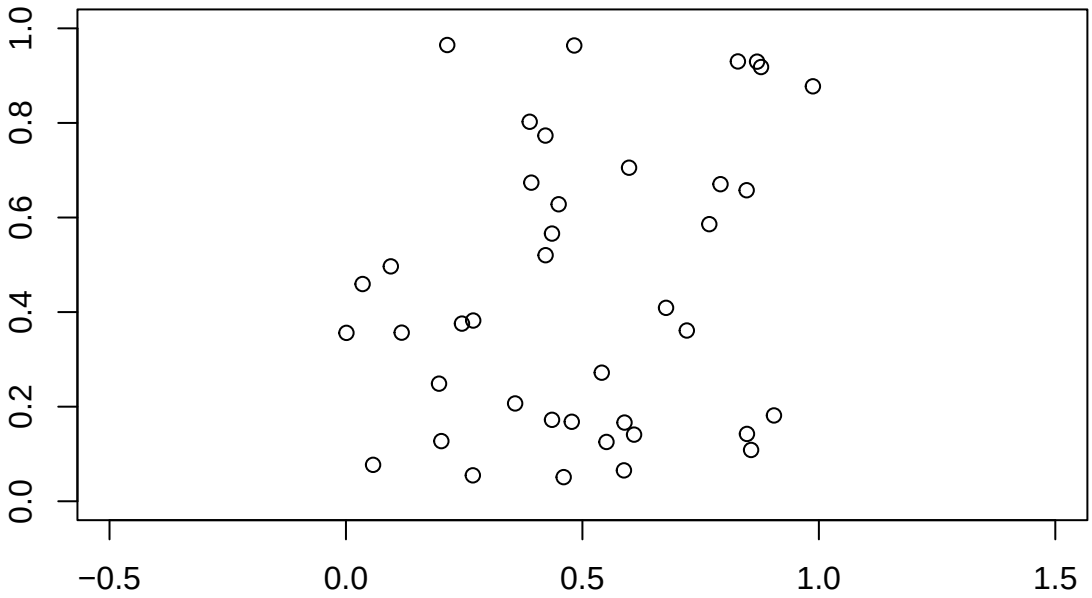
Organisation :

- Travail à réaliser par groupe de 3. Ces groupes seront les mêmes que les groupes des soutenances de projet fin janvier
- Rendre son travail mercredi 15 décembre au plus tard à 23h59 à **vincent.runge@univ-evry.fr**

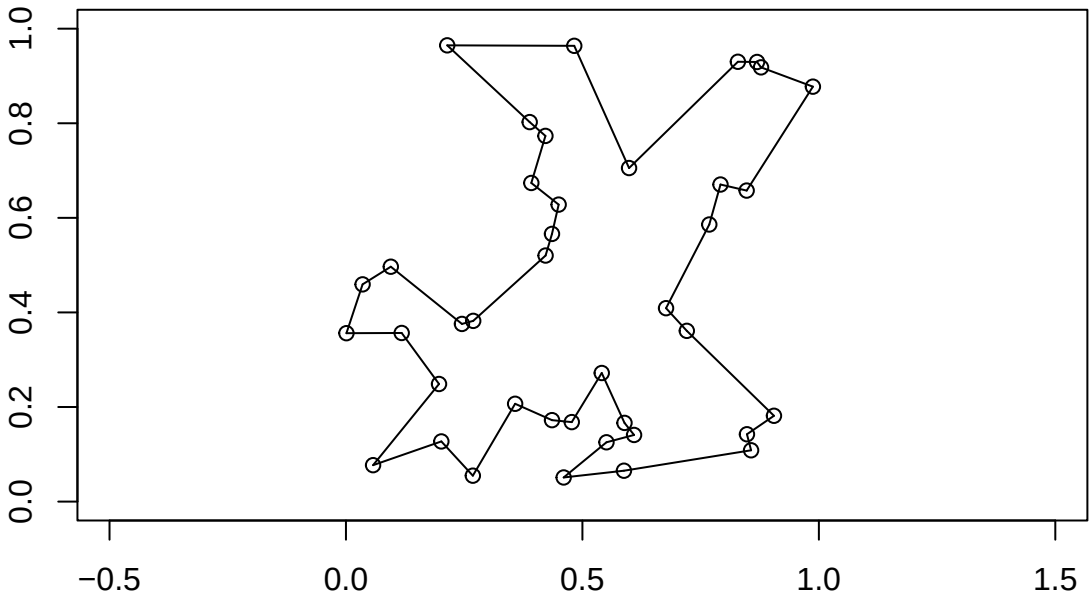
Les données :

On tire de manière aléatoire dans le carré unité selon une loi uniforme

```
n <- 40
villes <- matrix(runif(2*n), n, 2)
```



L'objectif est de contruire un "plus court chemin" par l'algorithme génétique. On n'a cependant pas de garantie que le chemin obtenu est vraiment le plus court... Cela pourra donner par exemple




```
## [1] 5.125412
```

Travail à réaliser :

1) une fonction `TSP_groupeK` (avec `K` le numéro du groupe) qui prend en argument la matrice des villes et tout autre paramètre nécessaire de votre choix. La fonction renvoie le vecteur des villes à visiter dans l'ordre "optimal"

Je présente ci-dessous un prototype possible pour une telle fonction.

```
TSP_groupe0 <- function(data, limit = 10^5, mutation = 0.01)
{

  n <- dim(data)[1]
  distances <- as.matrix(dist(data))  ### matrice des distances entre villes

  limit <- limit / 100
  nb_rep <- 5
  nb_Chrom <- 20
  iterMax <- limit
  save <- sample(n)

  for(i in 1:nb_rep)
  {
    chrom <- t(apply(matrix(0, nb_Chrom, n), 1, function(x) sample(n)))
    chrom[1,] <- save
    ###COMPLETER

    for(i in 1:iterMax)
    {
      ### 1) DISTANCES TOURS : distance du tour pour chaque chromosome
      dist_tours <- rep(0, nb_Chrom)
      for(k in 1:nb_Chrom)
```

```
      {
        dist_tours[k] <- sum(distances[chrom[k,] + n*(c(chrom[k,-1], chrom[k,1]) - 1)])
      }
      ### ATTENTION CONTRAINTE
      ### distances est utilisé  nb_Chrom x iterMax x nb_rep = limit FOIS
      ###

      ### 2) selection
      ###COMPLETER

      ### 3) croisement
      ###COMPLETER

      ### 4) mutation
      ###COMPLETER

    }
    save <- chrom[1,]
  }
  return(chrom[1,])
}
```

2) Les contraintes sont les suivantes :

- une initialisation aléatoire des chromosomes
- un nombre de calcul de distance limité par le paramètre `limit`

Vous pouvez imaginer en particulier pour les croisements toutes les stratégies de votre choix, à condition de ne pas faire appel à des comparaisons de distance (mais la position des villes peut être exploitée).

A rendre plus précisément: la fonction et les lignes de tests

```
n <- 20
villes <- matrix(runif(2*n), n, 2)
res20 <- TSP_groupe0(villes, limit = 10^4,  mutation = 0.05)

n <- 30
villes <- matrix(runif(2*n), n, 2)
res30 <- TSP_groupe0(villes, limit = 10^5,  mutation = 0.03)

n <- 40
villes <- matrix(runif(2*n), n, 2)
res40 <- TSP_groupe0(villes, limit = 10^6,  mutation = 0.01)
```

Cela permet de donner clairement quels sont les paramètres à utiliser pour chaque test.

Evaluation:

- c_1 = votre classement moyen (de 1 à 7) pour 100 (ou 1000 si possible) répétitions de 20 villes
- c_2 = votre classement moyen (de 1 à 7) pour 100 (ou 1000 si possible) répétitions de 30 villes
- c_3 = votre classement moyen (de 1 à 7) pour 100 (ou 1000 si possible) répétitions de 40 villes

Votre classement final sera donné par

$$c = \frac{c_1 + 2c_2 + 3c_3}{6}$$

afin de donner plus de poids au problème plus difficile.

La limite du nombre d'appels à la distance est de 10^4 pour `n = 20`, 10^5 pour `n = 30` et 10^6 pour `n = 40`.

Les PRIX :

- +2 points pour le 1er groupe
- +1.5 points pour le 2ème groupe
- +1 point pour le 3ème groupe