Loric Ravassard
Louis Rolland

Projet RN40 – Rapport

Automne 2022

Table des matières

1. Introduction

Tout d'abord, l'objectif de ce projet est de réaliser une version simplifiée d'un algorithme génétique, sous la forme d'une sélection des meilleurs individus d'une population sur plusieurs générations.
Ici, un individu sera représenté par une suite de bits de taille constante, et une population par un groupe d'individus lui-même stable.
Ce projet a été réalisé par Loric RAVASSARD et Louis ROLLAND.

2. Description des choix de conception et d'implémentation

Les structures de données utilisées pour représenter les types abstraits Individu et Population sont des listes chaînées.

En effet, afin de pouvoir appliquer les connaissances acquises en cours, nous avons jugé judicieux qu'un Individu soit défini par une liste chaînée de Bit, où un Bit est décrit comme un unsigned char en C (ce type de variable ne peut prendre que 0 ou 1 comme valeur) et de la même manière, qu'une Population soit représentée par une liste chaînée d'Individus.

Pour pouvoir utiliser ces deux nouveaux types de données, nous avons implémenté les primitives suivantes : ***vide***, ***afficher***, ***ajoutt***, ***ajoutq*** pour Individu et Population et ***taillePop***, ***randIndiv***, ***dernierIndiv*** pour Population.

3. Algorithmes

3.1. Individu

Comme précisé précédemment, un Individu est caractérisé par une liste de Bits de longueur constante.
C'est pourquoi la majorité des constructeurs ci-dessous prennent très souvent en paramètres :

- La taille d'un Individu
- La liste de bits représentant un Individu

Ci-dessous les différents algorithmes des fonctions applicables à un Individu :

initIndiv

```
initIndiv() -> Individu
Données:
Résultat: liste de bits générée aléatoirement
Lexique:
    creer_indiv() -> Individu, Fonction qui retourne un Individu vide
    longIndiv, entier correspondant à la taille d'Individu voulue
    ajoutq_indiv(Individu, Bit) -> Individu, Fonction qui ajoute un Bit en fin d'une liste de Bits
    != correspond au test "différent"
    % correspond à l'opération "reste de la division euclidienne" ou "modulo"
    rand(), fonction qui retourne un entier aléatoire


    l = creer_indiv()
    Tant que (longIndiv != 0) Faire
        l = ajoutertq_indiv(l,rand()%2)
        longIndiv = longIndiv - 1
    Fin Tant que
```

L'algorithme crée une liste vide, et tant que la longueur de la liste de bits constituant l'Individu n'est pas atteinte, on génère un bit aléatoire que l'on ajoute à l'Individu en question.

initIndiv_recursif

```
initIndiv_recursif(longIndiv : entier) -> Individu
Données: longIndiv, la longueur souhaitée de la liste de bits
Résultat: liste de bits générée aléatoirement
Lexique:
    creer_indiv() -> Individu, Fonction qui retourne un Individu vide
    ajoutt_indiv(Individu, Bit) -> Individu, Fonction qui ajoute un Bit en tête d'une liste de Bits
    rand(), fonction qui retourne un entier aléatoire
    % correspond à l'opération "reste de la division euclidienne" ou "modulo"
    == correspond au test d'égalité


    Si longIndiv == 0 Alors
        initIndiv_recursif(longIndiv) = creer_indiv()
    Sinon
        initIndiv_recursif(longIndiv) = ajoutertt_indiv(initIndiv_recursif(longIndiv-1), rand()%2)
    Fsi
```

decodeList

```
decodeList(l : Individu) -> entier
Données: l la liste de bits à décoder
Résultat: l'entier correspondant à la liste de bits donnée
Lexique:
    vide_indiv(Individu) -> booléen, Fonction qui retourne vrai si Individu est vide
    val_indiv(Individu) -> Bit, Fonction qui retourne la valeur d'un Individu
    suc_indiv(Individu) -> Individu, Fonction qui retourne l'élément suivant


    Si vide_indiv(l) Alors
        decodeList(l) = 0
    Sinon
        decodeList(l) = val_indiv(l) + 2*decodeList(suc_indiv(l))
    Fsi
```

L'algorithme prend un individu en paramètres et calcule sa valeur, fonction qui sera utile pour calculer la qualité d'un individu par la suite.

croiserList

```
croiserListes(pCroise : réel, l1 : Individu, l2 : Individu) -> Individu
Données: pCroise probabilité d'intervertir des éléments des 2 listes, l1 et l2 deux liste de bits
Résultat: l3 la liste de bits caractérisant l'individu résultant des éléments intervertis de l1 et l2
Lexique:
    vide_indiv(Individu) -> booléen, Fonction qui retourne vrai si Individu est vide
    ajoutt_indiv(Individu, Bit) -> Individu, Fonction qui ajoute un Bit en tête d'une liste de Bits
    val_indiv(Individu) -> Bit, Fonction qui retourne la valeur d'un Individu
    suc_indiv(Individu) -> Individu, Fonction qui retourne l'élément suivant
    creer_indiv() -> Individu, Fonction qui retourne un Individu vide
    rand_reel(), fonction qui retourne un réel aléatoire entre 0 et 1
```

```
Si non(vide_indiv(l1)) et non(vide_indiv(l2)) Alors
    Si rand_reel() <= pCroise Alors
        croiserListes(pCroise, l1, l2) = ajoutt_indiv(croiserListes(pCroise, suc_indiv(l1), suc_indiv(l2)), val_indiv(l1))
    Sinon
        croiserListes(pCroise, l1, l2) = ajoutt_indiv(croiserListes(pCroise, suc_indiv(l1), suc_indiv(l2)), val_indiv(l2))
    Fsi
Sinon
    croiserListes(pCroise, l1, l2) = creer_indiv()
Fsi
```

qualiteIndiv

```
qualiteIndiv(l : Individu) -> réel
Données: la liste de bits Individu à calculer sa qualité
    A = -1
    B = 1
Résultat: réel correspondant à la qualité de l'individu donné
Lexique:
    longIndiv, entier correspondant à la taille de l'Individu

 qualiteIndiv(l) = - ((decodeList(l) / (2^(longIndiv))) * (B - A) + A)^2
```

longIndiv

```
longIndiv(l : Individu) -> entier
Données : l'Individu dont on souhaite connaître le nombre de Bits
Résultat : le nombre de Bits composant l'Individu en paramètre
Lexique :
    != correspond au test "différent"
    suc_indiv(Individu) -> Individu, Fonction qui retourne l'élément suivant
    creer_indiv() -> Individu, Fonction qui retourne un Individu vide


Si l != creer_indiv() Alors
    longIndiv(l) = longIndiv(suc_indiv(l)) + 1
Sinon
    longIndiv(l) = 0
FSi
```

Cet algorithme est utilisé pour calculer la longueur d'un Individu (nombre de Bits).

3.2. Population

initPop

```
initPop(taillePop : entier) -> Population
Donnees: taillePop un entier representant la taille voulue de la population
Resultat: Population (une liste d'Individus) generee aleatoirement
Lexique:
    creer_pop() -> Population, Fonction qui retourne une Population vide
    ajoutt_pop(Population, Individu) -> Population, Fonction qui ajoute un Individu
    en tete d'une liste d'Individus
    initIndiv_recursif(entier) -> Individu, Fonction qui retourne
    un Individu de la taille du parametre entre
    == correspond au test d'égalité


  Si taillePop == 0 Alors
      initPop(taillePop) = creer_pop()
  Sinon
      initPop(taillePop) = ajoutt_pop(initPop(taillePop-1), initIndiv_recursif(8))
  Fsi
```

L'algorithme créé de manière récursive une Population composée de taillePop Individus de taille préalablement définie.

triQualiteDec

```
triQualiteDec(p1 : Population, p2 : Population) -> void
Donnees: p1 le dernier élément de la population à trier et p2 le dernier
élément de la population à trier
Resultat: Population triee par qualite decroissante des Individus
Lexique:
    partition(Population p1, Population p2) la fonction qui echange des
    individus dans une population sous certaines conditions
    creer_pop() -> Population, Fonction qui retourne une Population vide
    suc_pop(Population) -> Population, Fonction qui retourne l'element suivant
    != correspond au test "différent"

 Si p1 != p2 Alors
   pivot = partition(p1,p2)
   Si pivot != creer_pop() et suc(pivot) != creer_pop() Alors
     triQualiteDec(p1,p2) = triQualiteDec(suc_pop(pivot), p2)
   Sinon Si pivot != creer_pop() et p1 != pivot Alors
     triQualiteDec(p1,p2) = triQualiteDec(p1, pivot)
   FSi
 Fsi
```

Cet algorithme trie une Population par ordre décroissant de qualité d'Individu.

Partition

```
partition(p1 : Population, p2 : Population) -> Population
Donnees: p1 et p2 deux Individus d'une population
Resultat: Population avec certains individus echanges
Lexique:
    qualiteIndiv(Individu i) la fonction qui renvoie la qualite associee a un individu
    creer_pop() -> Population, Fonction qui retourne une Population vide
    val_pop(Population) -> Individu, Fonction qui retourne la valeur d'une Population
    suc_pop(Population) -> Population, Fonction qui retourne l'element suivant
    != correspond au test "différent"


    pivot = p1
    actuel = p1
    Tant que actuel != creer_pop() et actuel != p2 Faire
        Si(qualiteIndiv(val_pop(actuel))) > qualiteIndiv(val_pop(p2))) Alors
            pivot = p1
            temp = val_pop(p1)
            val_pop(p1) = val_pop(actuel)
            val_pop(actuel) = temp
        FSi
        actuel = suc_pop(actuel)
    FTantQue
    temp = val_pop(p1)
    val_pop(p1) = val_pop(p2)
    val_pop(p2) = temp

    partition(p1,p2) = pivot
```

meilleursIndiv

```
meilleursIndiv(l : Population, tSelect : entier) -> Population
Donnees: l la Population de base
         tSelect l'entier correspondant a l'endroit ou il faut tronquer la Population
Resultat: Population des meilleurs Individus en tronquant la liste et en la completant
par recopie des tSelect premiers elements
Lexique:
     TriQualiteDec(Population) -> Population, Fonction qui trie une Population par
     ordre decroissant de qualite d'Individu
     vide_pop(Population) -> booleen, Fonction qui retourne vrai si une Population est vide
     creer_pop() -> Population, Fonction qui retourne une Population vide
     ajoutq_pop(Population, Individu) -> Population, Fonction qui ajoute un Individu
     en fin d'une liste d'Individus
     val_pop(Population) -> Individu, Fonction qui retourne la valeur d'une Population
     suc_pop(Population) -> Population, Fonction qui retourne l'element suivant
     tete_pop(Population) -> Population, Fonction qui retourne la tete d'une Population


    l = triQualiteDec(l)
    i = creer_pop()
    TantQue tSelect > 0 Faire
        i = ajoutq_pop(i, val(l))
        l = suc(l)
        tSelect = tSelect-1
    FTantQue
    TantQue non(vide_pop(l)) Faire
        TantQue non(vide_pop(i)) et non(vide_pop(l)) Faire
                val_pop(l) = val_pop(i)
                i = suc_pop(i)
                l = suc_pop(l)
        FTantQue
        i = tete_pop(i)
    FTantQue
```

Cet algorithme sélectionne les meilleurs Individus d’une Population et complète le reste de la Population avec ces mêmes Individus.

croiserPop

```
croiserPop(p : Population, pCroise : reel) -> Population
Donnees: p la Population qu'il faut croiser
         pCroise la probabilite de croisement pour un Individu
Resultat: Population constituee d'Individus selectionnes aleatoirement
deux a deux dans la Population de base et croises entre eux
Lexique :
     creer_indiv() -> Individu, Fonction qui retourne un Individu vide
     randIndiv(Population) -> Individu, Fonction qui choisit un individu aleatoire
     dans une Population
     compteur_pop(Population) -> Entier, Fonction qui mesure la taille de la population p
     creer_pop() -> Population, Fonction qui retourne une Population vide
     ajoutq_pop(Population, Individu) -> Population, Fonction qui ajoute un Individu
     en fin d'une liste d'Individus
     croiserListes(reel, Individu, Individu) -> Individu, Fonction qui renvoie un Individu
     comme melange des deux Individus en parametres
     == correspond au test d'égalité
     != correspond au test "différent"


    i = creer_indiv()
    j = creer_indiv()
    p2 = creer_pop()
    TantQue compteur_pop(p2) != compteur_pop(p) Faire
        Faire
            i = randIndiv(p)
            j = randIndiv(p)
        TantQue i == j
        FinFaireTantQue

        p2 = ajoutq_pop(p2, croiserListes(pCroise, i, j)
    FTantQue
```

Cet algorithme croise aléatoirement des Individus d'une Population et répertorie ces mélanges dans une Population fille jusqu'à qu'elle soit de même taille que la Population initiale.

4. Jeux d'essais

4.1. Exécutions

Comme précisé dans l'énoncé du projet, le nombre de génération et la taille de la Population sont générés aléatoirement entre 20 et 200. Ensuite, pour ce qui est du taux de sélection, nous avons fait en sorte qu'il soit compris entre 10% et 90%.

Calcul de la qualité d'un Individu en fonction de sa valeur x :

$$f_1(x) = -X^2$$

$$Où\ X = \left(\frac{x}{2^{longIndiv}}\right) * (B - A) + A$$

$$Avec\ A = -1$$

$$B = 1$$

$$longIndiv = 8$$

Avec une probabilité de croisement pCroise = 0.5 et une longueur d'Individu de 8, nous obtenons les 4 résultats suivants :

```
0 0 0 0 0 0 0 1
qualite : -0.000000
nGen : 71 / taillePop : 48 / tSelect : 39
1 1 1 1 1 1 1 0
qualite : -0.000061
nGen : 26 / taillePop : 110 / tSelect : 86
1 1 1 1 1 1 1 0
qualite : -0.000061
nGen : 117 / taillePop : 30 / tSelect : 15
0 0 0 0 0 0 0 1
qualite : -0.000000
nGen : 167 / taillePop : 111 / tSelect : 61
```

4.2. Manipulation

Pour la manipulation, la formule du calcul de qualité d'un Individu n'est plus la même. En effet :

$$f_2(x) = -\ln(X)$$

$$Où\ X = \left(\frac{x}{2^{longIndiv}}\right) * (B - A) + A$$

$$Avec\ A = 0.1$$

$$B = 5$$

$$longIndiv = 16$$

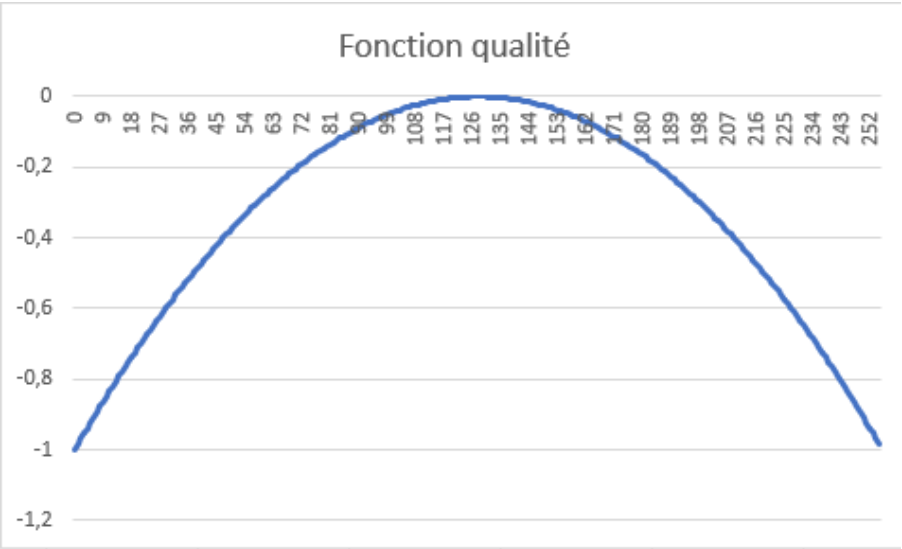
Avec cette nouvelle formule et les paramètres précisés ci-dessus, nous obtenons les 4 résultats suivants :

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
qualite : 2.302585
nGen : 80 / taillePop : 22 / tSelect : 12
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
qualite : 2.302585
nGen : 34 / taillePop : 89 / tSelect : 72
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
qualite : 2.302585
nGen : 175 / taillePop : 157 / tSelect : 119
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
qualite : 2.301838
nGen : 165 / taillePop : 28 / tSelect : 17
```

5. Commentaires sur les résultats

Nous observons pour les différentes exécutions que le résultat du meilleur Individu est quasi-systématiquement l'individu avec la qualité nulle (composé uniquement de 0 avec un 1 en fin de liste ce qui est égal à 128 en décimal). En effet, lorsqu'il y a un nombre de génération et une taille de population élevée, comme nous trions la liste par ordre de qualité décroissante puis sélectionnons les meilleurs Individus pour chaque génération, la probabilité que le

meilleur Individu soit l'Individu avec la qualité nulle (maximale car le maximum de la fonction est 0) est très élevée.
En effet, nous pouvons voir sur la courbe de la fonction de qualité que le maximum (0) est atteint pour la valeur 128 (donc l'Individu avec des 0 et un seul 1 à la fin).



Quant à la manipulation, nous obtenons des résultats similaires, à l'exception près que l'Individu avec la qualité maximale est cette fois-ci l'Individu dont la suite de bits ne comporte que des 0 (qualité de 2.302585).
Pour la fonction de qualité de la manipulation, nous voyons sur la courbe que le maximum est 2.302585 et ce maximum est atteint pour la valeur 0 (donc un Individu comportant seulement des 0).