



Rapport RN40

Alexandre BARTHELME, Nicolas AUGUSTE

Novembre 2022

Sommaire

1	Individu	3
1.1	Définition	3
1.2	Conception	3
1.3	Fonctions de l'individu	4
1.3.1	Décodage	4
1.3.2	Qualité	5
1.4	Croisement d'individu	5
2	Population	7
2.1	Définition	7
2.2	Conception	7
2.3	Manipulation	8
2.3.1	Initialisation	8
2.3.2	Trie par qualité	8
2.3.3	Sélection des meilleurs	9
2.4	Croisement de Population	10
3	Programme principal	12
3.1	Conclusion	13
4	Annexes	14

Introduction

Nous avons réalisé un projet dans le cadre de l'UV RN40 à l'UTBM consistant à la manipulation de liste chaînées au travers d'un sujet composé de deux grandes parties. Celle-ci étant composées par la définition et la manipulation de deux structures de données Individu et Population. Dans ce rapport sera détaillé ces deux éléments en détails. L'ensemble de ce projet est une base ou une introduction à l'algorithmique génétique.

L'entièreté des documents, algorithmes et codes sont accessibles sur notre page GitHub consacré au projet en cliquant [ici](#) . Pratiquement aucun code ne sera présent dans le rapport.

Chapter 1

Individu

1.1 Définition

Un individu est déclaré comme étant une liste chaînée de bits, avec une taille (un nombre de bits) et une qualité. La liste de bits est représentée comme étant une liste chaînée, la taille de celle ci est définie au préalable, elle sera nommée *LongIndiv* et la qualité sera calculer par une fonction ultérieurement.

1.2 Conception

Pour concevoir un Individu, il faut donc réaliser une liste chaînée de bits puis assigné à un individu une chaîne. La structure de l'individu (et de la liste) sera donc défini comme :

```
typedef unsigned char Bit;  
typedef struct element {  
    Bit bits;  
    struct element *suivant;  
} element;  
//element etant un maillon de la chaine de bits  
  
typedef struct Individu {  
    element *premier;  
    int LongIndiv;  
    float Qualite;
```

} Individu ;

Ensuite, nous utiliserons les primitives vues en cours pour créer, ajouter et manipuler la liste chaînée de bits et les individus. Les algorithmes correspondants sont disponibles, étant donné qu'ils sont très similaires à ceux vu en cours, ils ne seront pas détaillés ici. Une exception est la création d'un individu même. Celui-ci est aléatoire, ce qui signifie que les bits générés dans la liste de chaque individu est différent à chaque appel de la création.

Pour la création d'un individu nous l'avons donc conçu de cette manière :

Algorithm 1 Creer Individu

Require: LongIndiv : Entier

Ensure: Indiv : Individu

Individu Indiv

tete(Indiv) \leftarrow *Indefini*

i \leftarrow 0

while $i \leq \text{LongIndiv}$ **do**

 bit = 0 ou 1 \leftarrow *Entier*

 AjoutEnTete(Indiv,bit)

 Incrementer i

end while

qualite(Indiv) \leftarrow *calculQualiteIndividu(Indiv)*

creerIndividu(LongIndiv) = Indiv

LEXIQUE :

LongIndiv : Longueur d'un individu (nombre de bits)

qualite(Indiv) : Qualité d'un individu

calculQualiteIndividu : Fonction de calcul de la qualité d'un individu i :

Indice d'incrémentatation

En annexe se trouve la version récursive de cet algorithme

1.3 Fonctions de l'individu

1.3.1 Décodage

Un individu étant composé d'une liste de bits, on peut lui associer une valeur décimale. Pour cela, on décode sa "valeur" binaire :

Algorithm 2 Decodage

Require: Indiv : Individu

Ensure: res : Entier

i, res \leftarrow 0

element *actuel* \leftarrow *tete*(Indiv)

while *actuel* \neq *indefini* **do**

res \leftarrow *res* + *actuel.bits* \times *puissance*(2, *i*)

actuel = *succ*(*actuel*)

 Incrémenter *i*

end while

decodage(Indiv) = *res*

LEXIQUE :

Puissance(2,i) : Appel à une fonction puissance qui calcul 2^i

actuel.bits : le bit actuel de la liste

succ(*actuel*) : bit suivant

1.3.2 Qualité

Un des attributs d'un individu est sa qualité, elle est calculé selon la formule du sujet :

$$f(x) = -[(x/2^{LongIndiv}) \times (B - A) + A]^2 \quad (1.1)$$

Cette fonction sera changé par la suite. L'algorithme correspondant au programme qui calcul cette fonction est en annexe.

1.4 Croisement d'individu

Une des fonctions principales de ce projet pour pouvoir faire des "générations" d'individus différentes est de pouvoir croiser deux individus entre eux pour en créer un troisième. Cela consiste à prendre aléatoirement des bits de chacun des deux individus initiaux et de les mettre dans un troisième.

L'algorithme sera de la forme

Algorithm 3 CroisementIndividu

Require: indiv1, indiv2 : Individu, pCroise : Valeur de probabilité

Ensure: indiv3 : Individu résultant

```
if estVide(Indiv1) ou estVide(indiv2) then
    afficher "impossible de faire un croisement"
else
    indiv3 = creerIndividu
    actuel1 = tete(indiv1)
    actuel2 = tete(indiv2)
    if estIdentique(indiv1, indiv2) then
        CroisementIndividu() = indiv1
    end if
    while actuel1 et actuel2  $\neq$  indefini do
        proba = valeur décimale aléatoire entre 0 et 1
        if proba < pCroise then
            AjoutenTete(indiv3, actuel1(bits))
        else
            AjoutenTete(indiv3, actuel2(bits))
        end if
        actuel1 = succ(actuel1)
        actuel2 = succ(actuel2)
    end while
    qualite(Indiv3) = calculQualiteIndividu(indiv3)
    CroisementIndividu(indiv1, indiv2, Pcroise) = indiv3
end if
```

estIdentique permet de s'assurer de ne pas avoir des populations identiques à la fin de plusieurs générations.

A ce stade toutes les fonctions de manipulations d'un individu sont définies. Cependant, dans le cadre de du projet l'objectif est de créer une structure de données contenant des individus, celle-ci étant une population.

Chapter 2

Population

2.1 Définition

Une population est définie comme étant une structure de données constituée d'une liste d'individu généré aléatoirement. Elle inclut aussi une taille noté *nbrIndiv* qui définit le nombre d'individus.

2.2 Conception

Une population est constitué d'une liste chaînée d'individu, un maillon de cette chaîne étant un individu et le pointeur du suivant. Les primitives de manipulations de liste sont relativement similaires aussi et ne sont pas détaillées ici. La population, et les maillons sont donc définies tel que :

```
typedef struct elementPop{
    Individu indiv;
    struct elementPop *suivant;
} elementPop;

typedef struct Population{
    elementPop *premierPop; //pointeur sur premier individu
    int nbrIndiv;
} Population;
```


2.3 Manipulation

2.3.1 Initialisation

Pour créer une population, on a besoin d'un nombre d'individus et de leur taille. Après, on utilise les fonctions déjà existantes de création d'individus et d'ajout en tête adapté à la population. L'algorithme du sous-programme initialisant la population sera :

Algorithm 4 CreerPopulation

Require: taillePop, LongIndiv : Entier

Ensure: Pop : Population

Population Pop

for i allant de 0 a taillePop, par pas de 1 **do**

 AjoutTetePop(pop,creerIndividu(LongIndiv))

 Incrementer i

end for

CreerPopulation(TaillePop,longIndiv) = Pop

De cette manière, chaque population est initialisé avec des individus aléatoire (mais à taille définie)

2.3.2 Trie par qualité

Comme vu précédemment, chaque individu est associé à une qualité, celle-ci est aléatoire en fonction de la valeur de l'individu. Maintenant nous souhaitons trier par ordre décroissant la population en fonction de cette qualité. Pour cela, nous avons décidé de créer une fonction, récursive, qui divise à plusieurs reprise une population donnée en deux listes. A chaque appel de cette fonction, on se base sur un élément "pivot", le premier de la liste (*premierPop*) et on le compare avec l'élément suivant. A la fin , le programme concatène ces listes pour en ressortir une triée.

Voici l'algorithme correspondant :

Algorithm 5 Quicksort

Require: pop : Population**Ensure:** pop1 : Population (triée)**if** *estVide*(pop) ou *estVide*(succ(pop)) **then**
 quicksort(pop)=pop**else**

pivot = tete(pop)

pop1,pop2 = creerPopulation()

actuel = succ(pop)

while !*estVide*(actuel) **do** **if** (*qualite*(individu(actuel)) < *qualite*(individu(pivot))) **then**

AjoutTetePop(pop1,individu(actuel))

else

AjoutTetePopv(pop2,individu(actuel))

end if actuel \leftarrow succ(actuel) **if** *estVide*(tete(pop1)) **then**

tete(pop1) = pivot

else

actuel = tete(pop1)

while !*estvide*(succ(actuel)) **do**

actuel = succ(actuel)

end while

succ(actuel) = pivot

end if

succ(pivot) = tete(pop2)

quicksort(pop) = pop1

end while**end if**

2.3.3 Sélection des meilleurs

Avec une population triée par ordre décroissant,nous pouvons en retiré les meilleurs individus dans le but de créer une nouvelle population des x meilleurs individus, ce x sera nommé *select* qui correspond au nombre d'individus sélectionnés dans la liste triée. Par la suite, on duplique nos éléments choisis pour compléter une population jusqu'à *taillepop*. L'algorithme de ce sous-

programme sera :

Algorithm 6 meilleur

Require: pop : Population, select, taillepop : Entier

Ensure: pop1 : Population (constitué des meilleurs)

```
Pop1 = creerPopulation
if estVidePop(pop) then
    meilleur(pop,select,taillepop)= pop1
else
    actuel = tete(pop)
    i  $\leftarrow$  0
    while actuel  $\neq$  indefini et i < select do
        AjoutenQueue(pop1,actuel(indiv))
        actuel = succ(actuel)
        Incrémenter i
    end while
    actuel = tete(pop1)
    while i < taillepop do
        AjoutenQueue(pop1,actuel(indiv))
        actuel = succ(actuel)
        Incrémenter i
    end while
    meilleur(pop,select,taillepop)= pop1
end if
```

LEXIQUE :

select : Taux de sélection

taillePop : Taille de la population créée

actuel(indiv) : individu pointé par *actuel*

2.4 Croisement de Population

Pour finir, avec notre population désormais triée, on souhaite en créer une nouvelle constitué d'individus choisi par binome dans notre population initiale et les croiser entre eux via le sous programme *CroisementIndividu* vu précédemment. Pour ceci, il faut d'abord piocher aléatoirement deux

individus dans la population initiale, puis on les croisent en appelant notre fonction vue dans Individu avant de les ajouter dans une nouvelle population :

Algorithm 7 CroisementPopulation

Require: pop : Population, taillepop : Entier, Pcroise : probabilité de croisement

Ensure: pop1 : Population (population croisée)

Pop1 = creerPopulation

if *estVidePop*(pop) **then**

 afficher "population vide"

else

 i \leftarrow 0

while i < taillePop **do**

 alea1, alea2 = valeur aléatoire entre 0 et taillePop

 actuel1 = tete(pop)

 actuel2 = tete(pop)

 j \leftarrow 0

while j < alea1 ET succ(actuel1) \neq indefini **do**

 actuel1 = succ(actuel1)

 Incrémenter j

end while

 j \leftarrow 0

while j < alea2 ET succ(actuel2) \neq indefini **do**

 actuel2 = succ(actuel2)

 Incrémenter j

end while

 croisementIndividu(actuel1(Indiv), actuel2(Indiv), Pcroise) = indiv1

 AjoutTetePop(pop1, indiv1)

end while

 CroiementPopulation(Population, TaillePop, Pcroise) = pop1

end if

De cette manière, ce dernier sous programme nous retourne une population d'individus aléatoirement croisés.

Chapter 3

Programme principal

Avant d'utiliser toutes nos fonctions, on redéfinit la fonction de calcul comme indiqué en énoncé par :

$$f(x) = -\ln[(x/2^{LongIndiv}) \times (B - A) + A]^2 \quad (3.1)$$

avec $A = 0.1$, $B = 5$ et $LongIndiv = 16$ Maintenant, dans notre programme principale on instancie aléatoirement les variables *PSELECT* : Taux de sélection entre les bornes définies (10-90) , la taille de la population *TAILLEPOP* et le nombre de génération *NGEN* (entre 20 et 200). On fixe aussi les valeurs de longueurs d'individus *LGINDIV* à 8 et *PCROISE* à 0.5.

Algorithm 8 Main

PSELECT : Entier aléatoire entre 10 et 90

TAILLEPOP et NBGEN : Entier aléatoire entre 20 et 200

pop = creerPopulation(TAILLEPOP, LGINDIV)

for i allant de 0 à NBGEN-1 **do**

 CroisementPop(pop, PCROISE, LGINDIV)

 Quicksort(pop)

 meilleur(pop)

 Afficher : "Generation $i + 1$ "

end for

Afficher : "Meilleur individu : *meilleurIndividu(pop)*"

3.1 Conclusion

Après création du *Makefile* correspondant, un fichier *main.exe* peut s'exécuter. Celui-ci va lancer le code principal. A chaque boucle d'une génération, les meilleurs individus sont conservés, par conséquent en sortie de cette boucle, dans la majorité des cas l'individu affiché dispose d'une qualité très élevée tout comme sa valeur décodée souvent égale à 255.

Cependant, ceci dépend du nombre de génération, de la taille des populations et du taux de sélection.

De cette manière au cours du projet, nous avons introduit et expérimenté les bases d'un programme "génétique" permettant de ressortir les meilleurs individus possibles, en faisant une "évolution" sur plusieurs générations. Il aurait été intéressant d'y implémenter la mutation d'un individu pour augmenter la variété et la modification d'une population, mais également se rapprocher davantage d'un programme génétique plus complet.

L'annexe ci-dessous indique tous les algorithmes utiles mais non présentés dans le rapport.

Chapter 4

Annexes

4.1 Algorithme ajout en tête

Algorithm 9 : Ajout en tête

Require: Individu Indiv, Bits bit

Ensure: Individu

element nouveau

$nouveau.bits \leftarrow bit$

$succ(nouveau) \leftarrow tete(Indiv)$

$tete(Indiv) = nouveau$

LEXIQUE :

Element : un maillon de la chaine de bits

nouveau.bits : le bits du nouveau maillon créé

tete(Indiv) : le premier maillon de la chaîne de bits (*element * premier* dans la définition)

succ(nouveau) : le bits suivant de la chaîne

L'algorithme d'ajout en tête dans population est extrêmement similaire.

4.2 Créer individu récursivement

Algorithm 10 Creer Individu Recur

Require: LongIndiv : Entier

Ensure: Indiv : Individu

Individu Indiv

$tete(Indiv) \leftarrow Indefini$

if $LongIndiv > 0$ **then**

bit = 0 ou 1 $\leftarrow Entier$

AjoutEnTete(Indiv,bit)

$qualite(Indiv) \leftarrow calculQualiteIndividu(Indiv)$

$creerIndividuRecu = creerIndividuRecu(LongIndiv - 1)$

end if

$creerIndividu(LongIndiv) = Indiv$

LEXIQUE :

LongIndiv : Longueur d'un individu (nombre de bits)

qualite(Indiv) : Qualité d'un individu
calculQualiteIndividu : Fonction de calcul de la qualité d'un individu

4.3 Qualité

Algorithm 11 CalculQualiteIndividu

Require: Indiv: Individu

Ensure: res : float

$x = \text{decodage}(\text{Indiv})$

$\text{fct} \leftarrow (x / \text{puissance}(2, \text{longIndiv})) \times (B - A) + A$

$\text{res} = \text{fct} \times \text{fct}$

$\text{CalculQualiteIndividu}(\text{Indiv}) = -\text{res}$

LEXIQUE : B,A,LongIndiv : Variables globales définies par le sujet
Cette fonction est amené à changer légèrement à la fin du sujet.

4.4 Algorithmes de "confort"

Dans le code se trouve plusieurs fonctions non présentés dans le rapport, il s'agit uniquement de fonctions de "confort", utilisé pour le développement et la lecture de résultat. Nous avons décidé de les conserver mais de les commenter.

4.5 Informations

Si des algorithmes ne sont ni dans le projet, ni dans l'annexe,ils peuvent se trouver dans un fichier .txt sur GitHub.