Anne Lassabe
Aymeric Lamboley
GI01

RAPPORT DE PROJET : ALGORITHMES GÉNÉTIQUES

1. Compilation et exécution du projet

La compilation est réalisée grâce à un makefile. Il suffit donc de taper la commande `make`, puis d'exécuter le programme avec la commande `./projet`, projet étant le nom de l'exécutable. On peut passer en argument à l'exécution la fonction de qualité à utiliser (cf 5.) : `./projet f2` ou `./projet f3`. La fonction utilisée par défaut est la fonction f1.

2. Analyse du projet

Le projet a pour but de présenter une approche simplifiée des algorithmes génétiques. D'après Wikipedia, un algorithme génétique a pour but « d'obtenir une solution approchée, en un temps correct, à un problème d'optimisation[...]. Les algorithmes génétiques utilisent la notion de sélection naturelle développée au XIXe siècle par le scientifique Darwin et l'appliquent à une population de solutions potentielles au problème donné ». C'est-à-dire que l'algorithme génétique permet de proposer une solution à un problème issue des meilleures solutions potentielles. Ici on considère une population d'individus aléatoirement définis dont on sélectionnera le meilleur après des opérations de sélection.
Afin de reproduire la sélection naturelle on fait intervenir des paramètres tels que la taille de la population, la qualité d'un individu, le nombre de générations, la probabilité de croisement des bits d'un individu, le pourcentage d'individus sélectionnés dans une population.

3. Structures de données

Ce projet permet de manipuler la principale structure de données étudiée pendant l'UV de LO44, à savoir la liste chaînée. On utilise en effet la représentation chaînée pour définir un individu en tant que liste de bits, et pour définir une population comme liste d'individus. Cela permet de créer un individu ou une population de manière dynamique, ce qui est ici nécessaire du fait de l'utilisation de nombreux paramètres définis aléatoirement.
Un individu étant constitué de bits, on a également défini un type Bit prenant pour valeur 0 ou 1. Néanmoins, comme il l'est spécifié dans le sujet, un Bit est du type unsigned char.

4. Algorithmes utilisés

4-1. Type abstrait Individu

Fonction initialiser
Algorithme itératif :
Données : néant

Résultat : la liste de bits individu ayant récupéré des valeurs de manière aléatoire
Lexique :
Individu : liste de bits de longueur fixée par la constante longIndiv
Bits : caractère ayant une valeur binaire
Algorithme :

```
initialiserIT () → Individu
Debut
        Pour i allant de 1 à longIndiv avec un pas de 1 Faire
                individu= insererEnQueue(individu, valeur aléatoire (0..1))
        Fait
Fin
```

Algorithme récursif :
Paramétrage : Individu indiv : liste à initialiser – entier l ayant pour valeur longIndiv : la longueur de la liste finale
Résultat : Individu initialisé aléatoirement
Algorithme :

```
initialiserRE (indiv : Individu, l : entier) → Individu
Debut
        //Cas trivial
        Si l = 1 Alors
                indiv= valeur aléatoire (0..1)
        //Cas général
        Sinon
                valeur_tete(indiv)= valeur aléatoire (0..1)
                initialiser(reste(indiv), l-1)
        Finsi
Fin
```

Fonction decoder
Résultat : l'entier v, valeur en base 10 de la liste de bits/Individu
Données : l'Individu indiv à décoder
Lexique : v s'obtient en faisant la somme des valeurs de chaque bit multipliées par 2 à la puissance du rang du bit correspondant
Algorithme :

```
decoder(indiv : Individu) → entier
Debut
        entier i = longIndiv
        Tant que (non vide(reste(individu))) Faire
                v = v+(2^i)*valeur_tete(indiv)
                i = i-1
                indiv = reste(indiv)
        Fait
Fin
```

Fonction croiserIndividu
Résultat : 2 Individus qui ont été croisés.
Données : Individu1 et Individu2 qui sont des individus, pCroise une probabilité de croisement
Algorithme :

```
croiserIndividu(individu : Individu1, individu : Individu2)
Debut
        entier tmp
        Pour i variant de 1 à longIndiv avec un pas de 1 Faire
                pCroise = valeur aléatoire(0,1)
                Si pCroise = 1 Alors
```

```
                    tmp = individu1(i)
                    individu1(i) = individu2(i)
                    individu2(i) = tmp
                Finsi
        Fait
Fin
```

Fonction de qualité : f1
Résultat : réel qualite représentant la qualité d'un individu
Données : la valeur v d'un Individu, et la longueur longIndiv, A = -1 et B = 1 des entiers servant à calculer la qualité
Lexique : la qualité d'un individu se calcule par la formule : f1 (x) = -((x/2^longIndiv)*(B-A)+A)²
Algorithme :

```
f1 (v : entier) → réel
Debut
        qualite = -((v/2^longIndiv)*(B-A)+A)²
Fin
```

4-2. Type Abstrait Population

Initialiser la population
Résultat : newPop la liste châinée de type population, contenant des Individus
Données : Taille de la population (nombre d'individus dans une population), entier
Algorithme :

```
initialiserPop (entier taillePop) → Population
Debut
        pour i allant de 1 à taillePop Faire
                newPop= insererEnQueue(newPop, initialiserIndiv())
        Fait
Fin
```

Quicksort (Tri rapide)
Paramétrage : Population aTrier dont on doit trier les individus par ordre décroissant de leur qualité
Résultat : Population triée
Cas trivial :

```
Si est_vide(aTrier) ou est_vide(reste(aTrier)) Alors
        quicksort = aTrier
```

Cas général :

```
Si (non vide(aTrier) ou non vide(reste(aTrier))) Alors
        Population petits, grands, gauche, droite
        reel valeur, pivot
        Individu indivValeur, indivPivot
        indivPivot = valeur_tete(aTrier)
        pivot = qualite(valeur_tete(aTrier))

        Tantque(non vide(aTrier) et non_vide(reste(aTrier)))
                valeur = qualite(valeur_tete(reste(aTrier)));

                indivValeur = valeur_tete(reste(aTrier));
                Si valeur <= pivot alors
                        petits = insererIndividu(petits, indivValeur)
                sinon
                        grands = insererIndividu(grands, indivValeur)
```

```
            Finsi
            aTrier = reste(aTrier)
        FinTantQue

        gauche = quicksort(gauche)
        droite = quicksort(droite)

        gauche = insererIndividu(gauche, indivPivot)
        ajout_file(gauche, droite)
Finsi
```

Sélection dans la population
Données : Population elite dont on doit sélectionner les meilleurs individus, tSelect le pourcentage d'individus à sélectionner
Résultat : Population resultat constituée des meilleurs individus d'elite uniquement
Algorithme :

```
selection (elite : Population) → Population
Debut
        entier nbIndiv = taillePop * tSelect
        entier i = 0
        Population resultat, tmp
        Pour i variant de 0 à nbIndiv avec un pas de 1 Faire
            Individu in = elite->valeur
            resultat = insererIndividu(resultat, in)
            elite = elite->suivant
        Fait
        tmp = resultat
        Pour i variant de 0 à taillePop – nbIndiv avec un pas de 1 Faire
            Individu in = tmp->valeur
            resultat = insererIndividu(resultat, in)
            tmp = tmp->suivant
        Fait
Fin
```

Croisement de Population
Données : Population parent dont on croise les individus aléatoirement deux à deux
Résultat : Population enfant constituée du croisement d'individus de la Population parent
Algorithme :

```
croiserPop (parent : Population) → Population
Debut
        Pour i variant de 0 à taillePop avec un pas de 2 Faire
            entier choixPetit, choixGrand;

            Individu petit, grand

            Population tmp = parent
            choixPetit = valeur aléatoire(0..taillePop – 1)
            choixGrand =valeur aléatoire(choixPetit..taillePop - 1)
            Pour i variant de 0 à choixGrand avec un pas de 1 Faire
                Si j = choixPetit Alors
                    petit = tmp->valeur
                Finsi
                Si j = choixGrand Alors
```

```
                    grand = tmp->valeur
                Finsi
                tmp = tmp->suivant
            Fait
            croiserIndividu(petit, grand)
            enfant = insererIndividu(enfant, petit)
            enfant = insererIndividu(enfant, grand)
        Fait
Fin
```

5. Jeux d'essai et analyse des résultats

5-1. Paramètres

Comme on l'a expliqué dans la présentation du projet on utilise plusieurs paramètres pour effectuer la sélection au sein de la population.
longIndiv : nombre de bits constituant un individu ; selon la fonction de qualité utilisée (cf ci-dessous) longIndiv peut valoir 8, 16 ou 32 bits.
pCroise : probabilité de croisement de deux bits d'un individu ; par défaut on définit une équiprobabilité (0.5).
taillePop : nombre d'individus d'une population ; la taille varie entre 20 et 200 individus.
tSelect : Pourcentage des individus à sélectionner dans une population ; ce pourcentage varie entre 10 et 90% de la population.
nGen : nombre de générations que subit la population ; il peut varier entre 20 et 200 générations.

Étant donné que certains paramètres doivent varier, lors de l'implémentation on les a définis par un nombre aléatoire. Ainsi, pour chaque essai les paramètres sont différents et cela permet de souligner les différences induites.

Parallèlement, on a défini trois fonctions différentes pour calculer la qualité. Avec
X = (x / 2 ^ longIndiv) * (B - A) + A on a :

- f1(x) = -X² avec A = -1, B = 1, longIndiv = 8
- f2(x) = -ln(X) avec A = 0.5, B = 5, longIndiv = 16
- f3(x) = -cos(X) avec A = -π, B = π, longIndiv = 32

où x est la valeur décimale de l'individu.

On peut choisir la fonction à utiliser dans le programme en passant en argument à l'exécution f1, f2 ou f3. Exemple :
`~$ ./projet f2` exécutera le programme en utilisant la fonction de qualité f2 sur des individus de 16 bits.

Quelle que soit la fonction choisie, la meilleure qualité représente le maximum de la fonction dans l'intervalle [0 ; (2^longIndiv)-1]. Ainsi, pour la fonction f1, le maximum est f(128) = 0. Pour la fonction f2, le maximum est f(0), soit environ 2,3. Enfin, pour la fonction f3, le maximum est f(0) = f(2^32) = 1.
On observe donc que, selon la fonction choisie, le meilleur individu peut être celui dont la valeur est la plus élevée, la moins élevée ou la valeur intermédiaire. Le résultat du meilleur individu est donc directement dépendant de la fonction de qualité.

5-2. Résultats obtenus

Les individus sont toujours générés aléatoirement. Ce sont donc les paramètres exposés ci-

dessus qui font varier les résultats.
En utilisant la fonction de qualité par défaut f1, on observe que plus les paramètres aléatoires ont de petites valeurs, meilleure peut être la qualité du meilleur individu.
On observe que les résultats sont plus homogènes et que l'on obtient des individus de qualité moyenne en manipulant une population nombreuse, sur un grand nombre de générations, avec une forte probabilité de croisement et un taux de sélection élevé.
Par exemple, en fixant les données suivantes, on obtient :

```
Taux de sélection : 0.9 ; probabilité de croisement : 0.8
Meilleur individu dans une population de 200 après 200 générations :
0111 1011
Sa qualité vaut -0.001526
```

Avec des valeurs faibles les résultats sont hétérogènes mais meilleurs :

```
Taux de sélection : 0.1 ; probabilité de croisement : 0.2
Meilleur individu dans une population de 20 après 20 générations :
1000 0000
Sa qualité vaut -0.000000
```

En utilisant la fonction f2, les meilleurs résultats sont obtenus avec des paramètres ayant des valeurs élevées. Exemple :

```
Taux de sélection : 0.9 ; probabilité de croisement : 0.2
Meilleur individu dans une population de 200 après 200 générations :
0000 0000 0000 0000
Sa qualité vaut 2.302585
```

Les résultats sont plus médiocres lorsque l'on utilise de petites valeurs.

```
Taux de sélection : 0.1 ; probabilité de croisement : 0.8
Meilleur individu dans une population de 20 après 20 générations :
0010 0101 0101 1101
Sa qualité vaut 0.204375
```

Avec la fonction f3 les meilleurs individus sont obtenus avec des valeurs faibles en paramètre :

```
Taux de sélection : 0.1 ; probabilité de croisement : 0.2
Meilleur individu dans une population de 20 après 20 générations :
1111 1111 1001 1011 0101 1010 1000 1110
Sa qualité vaut 0.999953
```

De manière générale, on observe qu'avec un taux de sélection faible et un grand nombre de générations la population devient homogène, ce qui réduit la possibilité d'obtenir de meilleurs individus. Ce phénomène est d'autant plus remarquable que le nombre d'individus est faible.