Rapport projet LO21

de Julien Biermé

Description des choix de conception et d'implémentation

Le projet en C se compose de 6 fichiers :

- main.c : le programe principal correspondant à la partie 4 de l'énoncé
- individu.c et individu.h : La définition et les fonctions associées du Type abstrait « Individu » qui correspond à la partie 1 de l'énoncé
- population.c et popuation.h : La définition et les fonctions associées du Type abstrait « Population » qui correspond à la partie 2 de l'énoncé
- parametre.h : L'ensemble des constantes pouvant être manipulés pour le projet

Algorithme et Description des Fonctions

Paramètre du programe

Paramètres du Type abstrait « Individu » (Partie 1) : longIndiv = 16 :Entier *(Longeur de l'individu en nombre de bits)* pCroise = 0.5 : Réel ∈ [0,1] *(Probabilité indiqué pour croiser une liste de bits)* A = 0.1 : Réel *(paramètre A de la fonction qualité)* B = 5 : Réel *(paramètre B de la fonction qualité)* f(x) = -ln(x) : fonction *(fonction qualité de X f1(x)= -x*x et f2(x)=-log(x))*

Paramètres du Type abstrait « Population » (Partie 2) : TaillePop_min = 20 : Entier *(Taille minimum de la population)* TaillePop_max = 200 :Entier (Taille maximum de la population) //#define tselect = 4 : Entier *(Taux de selection de la fonction selection)*

Paramètres du programme principal tSelect_min = 10 : Entier *(Taux minimal de selection de la fonction selection)*

tSelect_max = 90 : Entier *(Taux maximal de selection de la fonction selection)* nGen_min = 10 : Entier *(Nombre de générations minimal)* nGen_max = 90 : Entier (Nombre de générations maximal)

Algorithme du sous programme individu

TYPE element = enregistrement value : BIT next : pointeur sur element Fin enregistrement INDIVIDU : pointeur sur element

---

Fonction Initialisation() : INDIVIDU

Données :

- longIndiv : int

Lexique :

- Individu_new : INDIVIDU
- value_bit : int

Résultats :

- Initialisation : INDIVIDU

DEBUT

1. Individu_new ← ALLOUER (taille (element))
2. Pour i ← 1 à longIndiv faire
    1. value_bit ← rand() % 2
    2. Individu_new ← insert_head(Individu_new , value_bit)
3. Fin Pour
4. Initialisation ← Individu_new

FIN

Cette fonction initialise une liste chainée de manière aléatoire en générant des bits aléatoires et en les insérant en tête de la liste. Elle commence par allouer de la mémoire pour le premier élément de la liste et déclare une variable "value_bit" qui stockera la valeur du bit généré aléatoirement. Elle utilise ensuite une boucle pour générer "longIndiv" bits aléatoirement et les insérer en tête de la liste. Enfin, elle renvoie l'individu qui pointe vers le début de la liste initialisée.

Fonction insert_head() : INDIVIDU Données :

Données :

- l : INDIVIDU
- e : int

Lexique :

- newel : INDIVIDU

Résultats :

- insert_head : INDIVIDU

DEBUT

1. newel ← ALLOUER (taille (element))
2. newel → value ← e
3. newel → next ← l
4. insert_head ← newel

FIN

Cette fonction insère un nouvel élément en tête d'une liste chainée

---

Fonction Decodage_liste_bit(Individu : INDIVIDU) : int

Données :

- Individu : INDIVIDU

Lexique :

- p : INDIVIDU
- valeur : int

Résultats :

- Decodage_liste_bit : int

DEBUT

1. Si is_empty(Individu) alors
    1. Decodage_liste_bit ← 0
2. Sinon
    1. p ← Individu
    2. valeur ← 0
    3. Tant que p → next ≠ NULL faire
        1. valeur ← valeur + p → value

        2. p ← p → next
    4. Fin Tant que
    5. Decodage_liste_bit ← valeur

FIN

Cette fonction calcule la valeur numérique d'une liste chainée de bits passée en entrée. Si la liste est vide, la fonction renvoie 0. Si la liste n'est pas vide, elle parcourt la liste à l'aide d'un pointeur et calcule la valeur numérique de la liste en additionnant les valeurs de chaque bit de la liste. Enfin, elle renvoie la valeur numérique de la liste.

Fonction Intervertir_prob(individu1 : INDIVIDU, individu2 : INDIVIDU)

Données :

- individu1 : INDIVIDU
- individu2 : INDIVIDU
- pCroise : int

Lexique :

- p : INDIVIDU
- q : INDIVIDU
- n : float

DEBUT

1. Si is_empty(individu1) ou is_empty(individu2) alors
    1. **FIN**
2. Sinon
    1. n ← (float)rand() / RAND_MAX
    2. Si n > pCroise alors
        1. swap_bit(individu1 → value, individu2 → value)
    3. p ← individu1
    4. q ← individu2

5. Si p → next ≠ NULL ou q → next ≠ NULL alors
    1. Tant que p → next ≠ NULL ou q → next ≠ NULL faire
    2. n ← (float)rand() / RAND_MAX
    3. Si n < pCroise alors
        1. swap_bit(p → value, q → value)
    4. p ← p → next
    5. q ← q → next
    6. Fin Tant que
    7. n ← (float)rand() / RAND_MAX
    8. Si n < pCroise alors
        1. swap_bit(p → value, q->value)
6. individu1 ← p
7. individu2 ← q

3. **FIN**

Cette fonction intervertit de manière aléatoire les bits de deux listes chainées passées en entrée. Si l'une des listes est vide, la fonction ne fait rien et s'arrête. Sinon, elle parcourt les deux listes et intervertit de manière aléatoire les bits de chaque élément. Pour chaque bit, elle génère un nombre aléatoire et le compare à "pCroise". Si le nombre est inférieur à "pCroise", les bits sont intervertis. Sinon, ils ne le sont pas. La fonction utilise la fonction "swap_bit()" pour effectuer l'interversion.

Fonction swap_bit(x : BIT, y : BIT)

Données :

- x : BIT
- y : BIT

Lexique :

- temp : BIT

Resultat :

DEBUT

1. temp ← x
2. x ← y

3. y ← temp
4. **FIN**

Cette fonction échange les valeurs de deux variables de type BIT

Fonction Qualite_individu(Individu : INDIVIDU) : float

Données :

- Individu : INDIVIDU
- A : float
- B : float
- f(x) : fonction (-ln(x) ou -x²)

Lexique :

- valeur : int
- qualite : double
- X : float

Resultat :

- Qualite_individu : float

DEBUT

1. valeur ← Decodage_liste_bit(Individu)
2. qualite ← 0
3. X ← (valeur / 2^longIndiv) * (B - A) + A
4. qualite ← f(X)
5. Qualite_individu ← qualite
6. **FIN**

Cette fonction calcule la qualité d'un INDIVIDU, qui est représenté par une liste chainée de bits. La qualité est calculée à l'aide de la valeur décodée de l'INDIVIDU (obtenue grâce à la fonction "Decodage_liste_bit()") et d'une formule qui utilise cette valeur. La valeur décodée est convertie en un nombre réel grâce à une formule qui utilise les variables A et B, puis la fonction "f()" est appliquée à ce nombre pour obtenir la qualité de l'INDIVIDU. La fonction renvoie ensuite la valeur de cette qualité.

Fonction print_list(l : INDIVIDU)

Données :

- l : INDIVIDU

Lexique :

- p : INDIVIDU

Résultat :

DEBUT

1. Si is_empty(l) alors
    1. Ecrire ("*** EMPTY LIST ***")
2. Sinon
    1. p ← l
    2. Ecrire ("[")
    3. Tant que p.next ≠ NULL faire
    4. Ecrire (p.value)
    5. p ← p.next
    6. Ecrire ("]")
3. **FIN**

La fonction `print_list` est une fonction qui affiche le contenu d'une liste chainée de bits. Elle prend en paramètre une liste chainée de bits (`INDIVIDU l`) et ne renvoie aucun résultat.

La fonction vérifie d'abord si la liste passée en paramètre est vide en utilisant la fonction `is_empty`. Si c'est le cas, elle affiche un message indiquant que la liste est vide. Si la liste n'est pas vide, la fonction parcourt la liste à l'aide d'un pointeur `p` et affiche le contenu de chaque élément de la liste à l'aide de la fonction `printf`. Une fois le parcours de la liste terminé, la fonction affiche un retour à la ligne pour séparer les différentes listes affichées.

Algorithme du sous programme population

TYPE structure_pop = enregistrement value_i_ : INDIVIDU next : pointeur sur structure_pop Fin enregistrement POPULATION : pointeur sur structure_pop

---

Fonction Initialisation_pop() : POPULATION

Données :

- longIndiv : int

Lexique:

- Population_new : POPULATION
- Individu_newel : INDIVIDU
- TaillePop_min : int
- TaillePop_max : int

Résultats :

- Initialisation_pop : POPULATION

DEBUT

1. Population_new ← ALLOUER (taille (structure_pop))
2. TaillePop ← Random_borne()
3. Pour i de 1 à TaillePop faire
    1. Individu_newel ← Initialisation()
    2. Population_new ← insert_head_pop(Population_new , Individu_newel)
4. Fin Pour
5. Initialisation_pop ← Population_new
6. **FIN**

Cette fonction permet d'initialiser une population de manière aléatoire en créant des individus de manière aléatoire et en les ajoutant en tête de la population. La taille de la population est déterminée par la fonction Random_borne(), qui retourne un entier compris entre deux bornes prédéfinies. Chaque individu est créé grâce à la fonction Initialisation(), qui génère de manière

aléatoire une liste chainée de bits. La population créée est retournée à la fin de la fonction, qui libère également la mémoire allouée pour la structure de la population.

Fonction Random_borne() : int

Données :

- Aucune

Lexique :

- resultat_random_borne : int
- TaillePop_min : int
- TaillePop_max : int

Résultats :

- Random_borne : int

DEBUT

1. resultat_random_borne ← rand() %(TaillePop_max - TaillePop_min + 1) + (TaillePop_min)
2. Random_borne ← resultat_random_borne
3. **FIN**

La fonction `Random_borne` est une fonction qui génère un nombre entier aléatoire compris entre `TaillePop_min` et `TaillePop_max`, inclus. Cela se fait grâce à l'utilisation de la fonction `rand()` qui génère un nombre aléatoire entier compris entre 0 et `RAND_MAX` (une constante prédéfinie en C). Le nombre généré par `rand()` est ensuite modifié de manière à être compris entre `TaillePop_min` et `TaillePop_max`, inclus, grâce à une expression arithmétique. Le résultat final est stocké dans la variable `resultat_random_borne` et renvoyé par la fonction.

Fonction getTail(cur : POPULATION) : POPULATION Données :

- cur : POPULATION

Lexique :

- tmp : POPULATION

Résultats :

- getTail : POPULATION

DEBUT

1. tmp ← cur
2. cur ← cur.next
3. Tant que (cur ≠ NULL) et (cur.next ≠ NULL) faire
    1. cur ← cur.next
    2. tmp ← tmp.next
4. Fin Tant que
5. getTail ← tmp
6. **FIN**

Cette fonction retourne l'élément de fin de liste de la population

---

Fonction Trie_Quicksort(Population : POPULATION) : POPULATION Données :

- Population : POPULATION

Lexique :

- pivot : POPULATION
- indiv_inf : POPULATION
- indiv_sup : POPULATION

Résultats :

- Trie_Quicksort : POPULATION

DEBUT

1. Si (Population ≠ NULL) et (Population.next ≠ NULL) alors
    1. pivot ← element_pivot(Population)
    2. indiv_inf ← NULL
    3. indiv_sup ← NULL
    4. Pour chaque element de Population faire
        1. Si (element.qualite < pivot.qualite) alors
            1. indiv_inf ← insert_head_pop(indiv_inf, element)
        2. Sinon
            1. indiv_sup ← insert_head_pop(indiv_sup, element)
    5. Fin Pour
    6. indiv_inf ← Trie_Quicksort(indiv_inf)

7. indiv_sup ← Trie_Quicksort(indiv_sup)
8. Population ← merge(indiv_inf, pivot, indiv_sup)
9. Trie_Quicksort ← Population
10. **FIN**

Fonction element_pivot(Population : POPULATION) : POPULATION Données :

- Population : POPULATION

Lexique :

- element_pivot : POPULATION

Résultats :

- element_pivot : POPULATION

DEBUT

1. element_pivot ← dernier element de Population
2. element_pivot ← element_pivot.precedent
3. Population.suivant.precedent ← Population.precedent
4. Population.precedent.suivant ← Population.suivant
5. element_pivot.suivant ← NULL
6. element_pivot.precedent ← NULL
7. element_pivot ← element_pivot
8. **FIN**

Fonction merge(indiv_inf : POPULATION, pivot : POPULATION, indiv_sup : POPULATION) : POPULATION Données :

- indiv_inf : POPULATION
- pivot : POPULATION
- indiv_sup : POPULATION

Lexique :

- merge : POPULATION

  Résultats :

- merge : POPULATION

  DEBUT

1. merge ← indiv_inf
2. dernier element de indiv_inf.suivant ← pivot
3. pivot.precedent ← dernier element de indiv_inf
4. pivot.suivant ← indiv_sup
5. indiv_sup.precedent ← pivot
6. merge ← merge
7. **FIN**

La fonction `Trie_Quicksort` utilise la technique de tri par partitionnement de l'algorithme Quicksort pour trier la liste chaînée de type `POPULATION`. Le tri s'effectue selon la qualité de chaque individu de la population. Elle prend en entrée une population et renvoie cette même population triée. L'algorithme fonctionne en sélectionnant un élément de la population comme pivot, puis en répartissant les autres éléments de la population en deux sous-populations en fonction de leur qualité par rapport au pivot. Les éléments de qualité inférieure au pivot sont placés dans une première sous-population, tandis que ceux de qualité supérieure sont placés dans une seconde sous-population. L'algorithme réalise ensuite un tri rapide sur chacune de ces sous-populations, avant de les fusionner avec le pivot pour former la population triée finale.procède au tri.

Fonction selectionner(population : POPULATION) : POPULATION Données :

- population : POPULATION
- tSelect_min : int
- tSelect_max : int

Lexique :

- marqueur : int
- pop : POPULATION
- marqueur_pop : POPULATION
- p_select : POPULATION

- tselect : int

Résultats :

- selectionner : POPULATION

DEBUT

1. marqueur ← 1
2. pop ← population
3. marqueur_pop ← population.next
4. p_select ← Creation_pop(pop.value_i)
5. tselect ← nombre aléatoire entre tSelect_min et tSelect_max
6. Tant que (marqueur_pop.next ≠ NULL) faire
    1. Si (marqueur < tselect) alors
        1. pop ← pop.next
        2. p_select ← insert_tail_pop(p_select, pop.value_i)
        3. marqueur ← marqueur + 1
    2. Sinon
        1. marqueur ← 1
        2. pop ← population
        3. p_select ← insert_tail_pop(p_select, pop.value_i)
    3. marqueur_pop ← marqueur_pop.next
7. selectionner ← p_select
8. **FIN**

La fonction `selectionner` permet de sélectionner aléatoirement une liste de `tselect` individus dans une population donnée. La population est triée selon la qualité de chaque individu. La fonction commence par déterminer le nombre d'individus à sélectionner en générant un nombre aléatoire compris entre `tSelect_min` et `tSelect_max`. Ensuite, elle parcourt la population et sélectionne les individus un à un. Si le compteur `marqueur` est inférieur au nombre d'individus à sélectionner, la fonction sélectionne l'individu actuel et incrémente le compteur. Si le compteur est égal ou supérieur au nombre d'individus à sélectionner, la fonction remet le compteur à 0 et sélectionne le premier individu de la population. La fonction continue de parcourir et de sélectionner les individus jusqu'à ce que le pointeur `marqueur_pop` atteigne la fin de la populationet renvoit `p_select`

Fonction Choix_aleatoire_individu(pop : POPULATION, longueur : int) : INDIVIDU
Données :

- pop : POPULATION
- longueur : int

Lexique :

- nb_aleatoire : int
- marqueur : int

Résultats :

- Choix_aleatoire_individu : INDIVIDU

DEBUT

1. nb_aleatoire ← nombre aléatoire compris entre 1 et longueur
2. marqueur ← 0
3. Tant que (marqueur < nb_aleatoire) faire
    1. pop ← pop.next
    2. marqueur ← marqueur + 1
4. Choix_aleatoire_individu ← pop.value_i
5. **FIN**

La fonction `Choix_aleatoire_individu` permet de choisir un individu de manière aléatoire dans une population donnée. Elle génère un nombre aléatoire compris entre 1 et la longueur de la population, puis parcourt la population jusqu'à atteindre l'individu correspondant à ce nombre. La fonction renvoie ensuite cet individu.

Fonction Croiser_pop(P1 : POPULATION) : POPULATION Données :

- P1 : POPULATION, liste chainée représentant une population

Lexique :

- P2 : POPULATION, liste chainée représentant une population
- i1 : INDIVIDU
- i2 : INDIVIDU
- pop_marqueur : POPULATION, pointeur sur la tête de P1
- longueur : int, longueur de P1

Résultats :

- P2 : POPULATION, liste chainée représentant une population avec les INDIVIDUS croisés

DEBUT

1. Si (is_empty_pop(P1) = VRAI) ou (is_empty_pop(P1) = VRAI) Alors
    1. P2 ← P1
    2. Renvoyer P2
2. Fin Si
3. longueur ← 0
4. pop_marqueur ← P1
5. Tant que (pop_marqueur->next ≠ NULL) Faire
    1. longueur ← longueur + 1
    2. pop_marqueur ← pop_marqueur->next
6. Fin Tant que
7. i1 ← Choix_aleatoire_individu(P1, longueur)
8. i2 ← Choix_aleatoire_individu(P1, longueur)
9. Intervertir_prob(i1, i2)
10. P2 ← Creation_pop(i1)
11. P2 ← insert_head_pop(P2, i2)
12. pop_marqueur ← P1->next->next
13. Si (is_empty_pop(pop_marqueur) = FAUX) ou (is_empty_pop(pop_marqueur->next) = FAUX)
    1. Tant que (is_empty_pop(pop_marqueur) = VRAI) ou (is_empty_pop(pop_marqueur->next) = VRAI) faire
        1. i1 ← Choix_aleatoire_individu(P1, longueur)
        2. i2 ← Choix_aleatoire_individu(P1, longueur)

3. Intervertir_prob(i1, i2)
4. P2 ← insert_head_pop(P2, i1)
5. P2 ← insert_head_pop(P2, i2)
6. pop_marqueur ← pop_marqueur->next->next

2. Fin Tant que

14. Fin Si
15. Renvoyer P2
16. **FIN**

Cette fonction effectue un croisement sur des individus d'une population passée en paramètre, et crée une nouvelle population avec ces individus croisés. Le croisement consiste à intervertir les bits de deux individus choisis de manière aléatoire dans la population. La fonction parcourt la population passée en paramètre et utilise une boucle pour croiser tous les individus de cette population deux par deux. La nouvelle population est créée en ajoutant les individus croisés à sa tête. Enfin, la fonction renvoie la nouvelle population ainsi créée.

Résultats et Interprétation

Avec $f_1(x) = -X^2$ et ses paramètres associées :

- longIndiv = 8
- tSelect = ∈ [0.1 ; 0.9]
- pCroise = 0.5
- A=−1
- B=1
- nGen ∈ [20 ; 200]

Le Meilleur Individu obtenue est toujours : `[00000000]`

En effet cela s'explique par le fait que la selection cherche à maximiser f_1(x)=−X² donc à minimiser X. v

Avec $f_2(x) = -ln(X)$ et ses paramètres associées :

- longIndiv = 16
- tSelect ∈ [0.1 ; 0.9]
- TaillePop ∈ [20 ; 200]
- pCroise = 0.5
- A=0.1
- B=5
- nGen ∈ [20 ; 200]
- TaillePop_max : int

Le Meilleur Individu obtenue tends à etre : `[0000000000000000]`

En effet cela s'explique par le fait que la selection cherche à maximiser $f_2(x) = -ln(X)$ donc à minimiser X. Or X est juste une transformation affine avec A positif elle est donc minimale quand l'individu vaut `[0000000000000000]`.

La longueur de l'individu est de 16, ainsi avec un nombre de génération nGen ∈ [20;200] et TaillePop ∈ [20 ; 200], cela n'est pas assez pour obtenir l'individu optimum `[0000000000000000]` dans la population généré. Cela explique le fait que le meilleur individu ne soit pas entièrement nul après test.