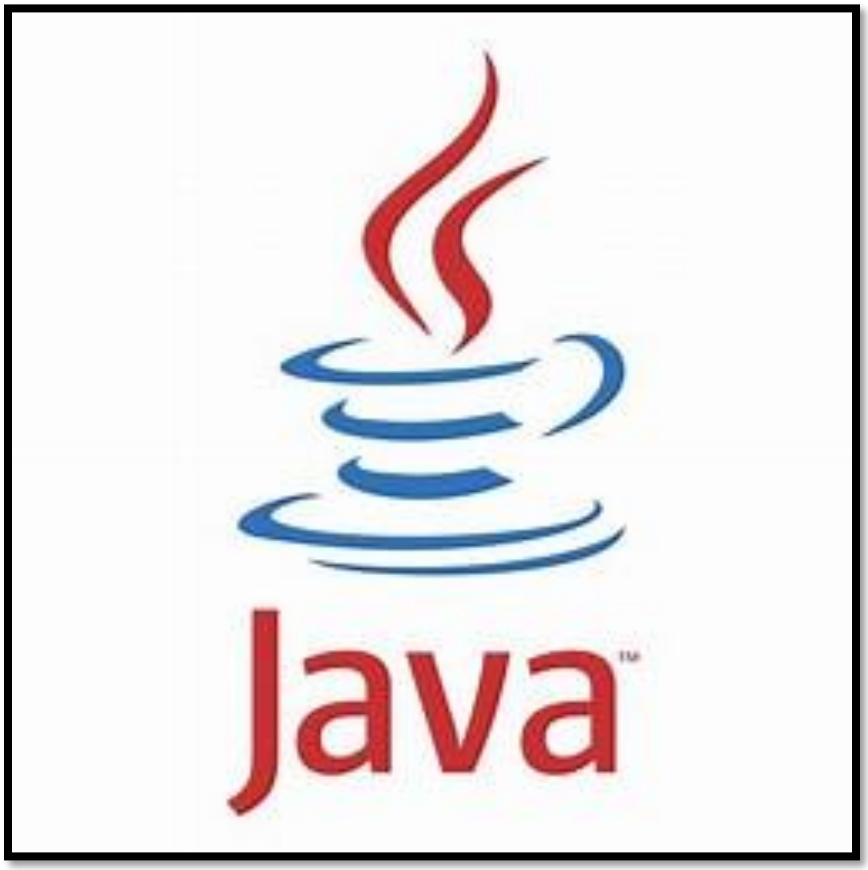
Systèmes Multi Agents et Intelligence Artificielle Distribuée

Implémentation de l'algorithme Générique



Pr. ABDELMAJID BOUSSELHAM.

Réalisé par :

ASMAE EL HYANI
HAMZA SANBOULI
Master : Systèmes Distribué et Intelligence Artificiel (SDIA).
Département : Mathématiques et Informatique.

Introduction :

A partir d'une population initiale, dont les gènes des chromosomes sont des caractères aléatoires, Notre algorithme génétique doit générer des générations dont les chromosomes convergent vers la chaine {'b','o','n','j','o','u','r'}.

Dans ce qui suit, la démarche suivie :

Conception :

Les opérations de l'algorithme :

Notre algorithme doit être capable de faire des opérations de ***croissement*** entre les chromosomes des individu existants dans la population, mais dans notre cas, il est possible que les individues initiaux ne possèdent pas certains caractères de la chaine qu'on veut atteindre ({'b','o','n','j','o','u','r'}). Donc la mutation ne peut pas suffire a atteindre à notre objectif.

Donc notre algorithme être capable de faire des opérations de ***mutations*** pour introduire des nouveaux caractères a ceux déjà existant pour couvrir tous les cas possibles.

Aussi, l'algorithme doit être capable de faire l'opération de ***sélection*** pour déterminer quels individus sont plus enclins à obtenir les meilleurs résultats (qui ont les meilleurs évaluations).

La fonction d'évaluation (fitness) :

On peut implémenter la fonction d'évaluation des individues en deux manières :

Fonction d'évaluation avec la distance euclidienne :

On calcule la distance euclidienne (en code asci) entre chaque gène du chromosome avec le caractère correspondant en ordre dans la chaine ({'b','o','n','j','o','u','r'}). Et on fait la somme de ces distances. Et d'une génération a l'autre , on cherche à minimiser cette somme.

(si la somme est proche de 0 , le chromosome est proche de l'objectif)

L'implémentation sera comme suite :

```
no usages
public void calculateFitness2(){
    sumFitness=0;
    for(int i = 0; i< genes.length; i++){
        fitness[i]= Math.abs(genes[i]-goal[i]);
        sumFitness+=fitness[i];
    }
}
```

On prend des deux chromosomes potentiel suivants :

Chromosome 1 : {'z','o','n','j','o','u','r'})

Chromosome 2 : {'c','p','o','k','p','v','s'}

En appliquant cette fonction d'évaluation a ces deux chromosomes, on va trouver :

Fitnesse(Chromosome 1) = 24

Fitnesse(Chromosome 2) = 7

Alors que visuellement, on peut voir que le chromosome 1 a 6 caractères justes, et un seul qui doit être changé, et le chromosome 2 n'a aucun caractère juste.

Fonction d'évaluation avec la comparaison des caractères :

On fait une comparaison entre chaque gène du chromosome avec le caractère correspondant en ordre dans la chaine ({'b','o','n','j','o','u','r'}).

Si les caractères ne sont pas égaux, on ajoute 1 a la fitneess, cette somme aura comme valeur maximal 7 (si aucun caractère n'est correcte), et comme valeur minimal 0 (si le chromosome est entièrement correcte) on cherche à minimiser cette somme.

L'implémentation sera comme suite :

```
4 usages
public void calculateFitness(){
    sumFitness=0;
    for(int i = 0; i< genes.length; i++){
        if(Math.abs(genes[i]-goal[i])!=0)
            fitness[i]= 1;
        else
            fitness[i]= 0;
        sumFitness+=fitness[i];
    }
}
```

On prend les chromosomes de l'exemple précèdent :

Cette fois-ci :

Fitnesse(Chromosome 1) = 1

Fitnesse(Chromosome 2) = 7

Maintenant le chromosome le plus proche de l'objectif est celui avec la fitnesse la plus petite.

Dans notre implémentation on va utiliser cette méthode de calcule de fitnesse.

Implémentation :

On commence par l'implémentation de la classe Individual :

On commence par donner les caractères que possible et la chaine objectif :

```java
package ga.sequenciel;
import java.util.Random;
13 usages
public class Individual implements Comparable {
    //chromosome
    8 usages
    char[] genes;
    7 usages
    char[] goal= {'b','o','n','j','o','u','r'};
    3 usages
    char[] alphabet={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};
    8 usages
    private int[] fitness;
    9 usages
    private int sumFitness=0;
```

La construction d'un individu aléatoirement et la fonction d'évaluation

```java
    public Individual(){
        Random rnd=new Random();
        genes=new char[goal.length];
        fitness=new int[goal.length];

        for(int i = 0; i< goal.length; i++){
            genes[i]=alphabet[rnd.nextInt( bound: 25)];
        }
    }
    4 usages
    public void calculateFitness(){
        sumFitness=0;
        for(int i = 0; i< genes.length; i++){
            if(Math.abs(genes[i]-goal[i])!=0)
                fitness[i]= 1;
            else
                fitness[i]= 0;
            sumFitness+=fitness[i];
        }
    }
    no usages
```

La comparaison entre les fitness des individues :

```java
@Override
public int compareTo(Object o) {
    Individual individual=(Individual) o;

    if(this.sumFitness>individual.sumFitness)
        return 1;
    else if (this.sumFitness<individual.sumFitness)
        return -1;
    return 0;
}
```

Les getters et setters :

```java
    3 usages
    public int[] getFitness() { return fitness; }

    no usages
    public void setGenes(char[] genes) { this.genes = genes; }

    no usages
    public void setFitness(int[] fitness) { this.fitness = fitness; }

    15 usages
    public char[] getGenes() { return genes; }

    2 usages
    public int getSumFitness() { return sumFitness; }
}
```

Maintenant on va passer à l'implémentation de la classe Population :

La population est une liste d'individues, on fait l'initialisation de la population. Et e calcul de la fitnesse des individues.

```
public class Population {
    3 usages
    private static final int MAX_IND = 5;
    30 usages
    List<Individual> individuals = new ArrayList<>();
    3 usages
    Individual firstFinest;
    3 usages
    Individual secondFinest;
    5 usages
    Random random = new Random( seed: 4);
    1 usage
    public void initializePopulation() {
        for (int i = 0; i < MAX_IND; i++) {
            individuals.add(new Individual());
        }
    }
    3 usages
    public void calculateIndividualFitness() {
        for (int i = 0; i < MAX_IND; i++) {
            individuals.get(i).calculateFitness();
        }
    }
```

Le nombre d'individu dans notre population est 5.

L'opération de ***sélection*** :

```
public void selection() {
    firstFinest = individuals.get(0);
    secondFinest = individuals.get(1);
}
3 usages
public void sortPopulation() { Collections.sort(individuals); }
```

L'operation de ***croissement*** :

```
public void crossover() {
    int pointCroisement = random.nextInt( bound: MAX_IND-3);
    pointCroisement+=2;
    //System.out.println("___pointCroisement : "+pointCroisement);
    Individual individual1 = new Individual();
    Individual individual2 = new Individual();
    String ind1="";
    String ind2="";
    for (int i = 0; i < individual2.getGenes().length; i++) {
        individual1.getGenes()[i] = firstFinest.getGenes()[i];
        ind1+=individual1.getGenes()[i];
        individual2.getGenes()[i] = secondFinest.getGenes()[i];
        ind2+=individual2.getGenes()[i];
    }
    //System.out.println("ind1 "+ind1);
    //System.out.println("ind2 "+ind2);
    for (int i = 0; i < pointCroisement; i++) {
        individual1.getGenes()[i] = secondFinest.getGenes()[i];
        individual2.getGenes()[i] = firstFinest.getGenes()[i];
    }
    individuals.set(individuals.size() - 1, individual1);
    individuals.set(individuals.size() - 2, individual2);
    individual1.calculateFitness();
    individual2.calculateFitness();
}
```

L'opération de ***mutation*** :

```
public void mutation() {
    int index = random.nextInt(individuals.get(0).goal.length);
    while(individuals.get(individuals.size() - 1).getFitness()[index] ==0){
        index = random.nextInt(individuals.get(0).goal.length);
    }
    if (individuals.get(individuals.size() - 1).getFitness()[index] !=0){
        char newAlpha=individuals.get(0).alphabet[random.nextInt( bound: 24)];
        while(individuals.get(individuals.size() - 1).getGenes()[index]==newAlpha){
            newAlpha=individuals.get(0).alphabet[random.nextInt( bound: 24)];
        }
        System.out.println(newAlpha+" "+individuals.get(individuals.size() - 1).getGenes()[index]);
        individuals.get(individuals.size() - 1).getGenes()[index] = newAlpha;
        individuals.get(individuals.size() - 1).calculateFitness();
    }
}
1 usage
public Individual getBestFitnessInd() { return individuals.get(0); }
}
```

Maintenant on passe à l'implémentation de la classe AGApp :

```
public class AGApp {
    1 usage
    private static final int MAX_IT=1000;
    1 usage
    private  static final int MAX_FITNESS=10;
    no usages
    private static final char[] goal= {'b','o','n','j','o','u','r'};
    public static void main(String[] args) {
        Population population=new Population();
        population.initializePopulation();
        population.calculateIndividualFitness();
        population.sortPopulation();
        int it=0;
        while (it<MAX_IT || population.getBestFitnessInd().getSumFitness()>MAX_FITNESS){
            System.out.println("---------------------------iteration-"+it+"-----------------------------------");
            population.selection();
            population.crossover();
            population.calculateIndividualFitness();
            population.sortPopulation();
            Random random=new Random();
            if(random.nextInt( bound: 101)<50){
                System.out.println("mutation");
                population.mutation();
            }
            population.calculateIndividualFitness();
            population.sortPopulation();
            for(int i=0;i<population.individuals.size();i++){
                System.out.println(i+" Chromosome : "+ Arrays.toString(population.individuals.get(i).getGenes())+" fitness : "+population.individu
            }
            it++;
        }
    }
}
```

Exécution :

On exécute le programme :

La population initial et les premières générations :

```
mutation
p h
0 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
1 Chromosome : [o, d, h, j, u, s, v] fitness : 6 fitness [1, 1, 1, 0, 1, 1, 1]
2 Chromosome : [b, y, l, c, e, h, w] fitness : 6 fitness [0, 1, 1, 1, 1, 1, 1]
3 Chromosome : [h, b, v, r, g, a, r] fitness : 6 fitness [1, 1, 1, 1, 1, 1, 0]
4 Chromosome : [o, d, p, c, e, h, w] fitness : 7 fitness [1, 1, 1, 1, 1, 1, 1]
---------------------------iteration-1-----------------------------------
mutation
j h
0 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
1 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
2 Chromosome : [o, d, h, j, u, s, v] fitness : 6 fitness [1, 1, 1, 0, 1, 1, 1]
3 Chromosome : [b, y, l, c, e, h, w] fitness : 6 fitness [0, 1, 1, 1, 1, 1, 1]
4 Chromosome : [o, d, j, j, u, s, v] fitness : 6 fitness [1, 1, 1, 0, 1, 1, 1]
---------------------------iteration-2-----------------------------------
0 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
1 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
2 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
3 Chromosome : [b, y, l, j, u, s, v] fitness : 5 fitness [0, 1, 1, 0, 1, 1, 1]
4 Chromosome : [o, d, h, j, u, s, v] fitness : 6 fitness [1, 1, 1, 0, 1, 1, 1]
---------------------------iteration-3-----------------------------------
```

Les résultats :

```
--------------------------iteration-181-------------------------------------
0 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
1 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
2 Chromosome : [b, o, n, j, u, u, r] fitness : 1 fitness [0, 0, 0, 0, 1, 0, 0]
3 Chromosome : [b, o, n, j, u, u, r] fitness : 1 fitness [0, 0, 0, 0, 1, 0, 0]
4 Chromosome : [b, o, n, j, u, u, r] fitness : 1 fitness [0, 0, 0, 0, 1, 0, 0]
--------------------------iteration-182-------------------------------------
0 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
1 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
2 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
3 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
4 Chromosome : [b, o, n, j, u, u, r] fitness : 1 fitness [0, 0, 0, 0, 1, 0, 0]
--------------------------iteration-183-------------------------------------
0 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
1 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
2 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
3 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
4 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
--------------------------iteration-184-------------------------------------
0 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
1 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
2 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
3 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
4 Chromosome : [b, o, n, j, o, u, r] fitness : 0 fitness [0, 0, 0, 0, 0, 0, 0]
```