

Algorithme Génétique

I. Pseudo Code :

```
//Main class
public class SimpleDemoGA {

    Population population = new Population();
    Individual fittest;
    Individual secondFittest;
    int generationCount = 0;

    public static void main(String[] args) {

        Random rn = new Random();

        SimpleDemoGA demo = new SimpleDemoGA();

        //Initialize population
        demo.population.initializePopulation(10);

        //Calculate fitness of each individual
        demo.population.calculateFitness();

        System.out.println("Generation: " + demo.generationCount + "
Fittest: " + demo.population.fittest);

        //While population gets an individual with maximum
fitness
        while (demo.population.fittest < 5) {
            ++demo.generationCount;

            //Do selection
            demo.selection();

            //Do crossover
            demo.crossover();

            //Do mutation under a random probability
            if (rn.nextInt()%7 < 5) {
                demo.mutation();
            }

            //Add fittest offspring to population
            demo.addFittestOffspring();
        }
    }
}
```

```

        //Calculate new fitness value
        demo.population.calculateFitness();

        System.out.println("Generation: " + demo.generationCount
+ " Fittest: " + demo.population.fittest);
    }

    System.out.println("\nSolution found in generation " +
demo.generationCount);
    System.out.println("Fitness:
"+demo.population.getFittest().fitness);
    System.out.print("Genes: ");
    for (int i = 0; i < 5; i++) {
        System.out.print(demo.population.getFittest().genes[i]);
    }

    System.out.println("");
}

//Selection
void selection() {

    //Select the most fittest individual
    fittest = population.getFittest();

    //Select the second most fittest individual
    secondFittest = population.getSecondFittest();
}

//Crossover
void crossover() {
    Random rn = new Random();

    //Select a random crossover point
    int crossOverPoint =
rn.nextInt(population.individuals[0].geneLength);

    //Swap values among parents
    for (int i = 0; i < crossOverPoint; i++) {
        int temp = fittest.genes[i];
        fittest.genes[i] = secondFittest.genes[i];
        secondFittest.genes[i] = temp;} }

//Mutation
void mutation() {
    Random rn = new Random();

```

```

        //Select a random mutation point
        int mutationPoint =
rn.nextInt(population.individuals[0].geneLength);

        //Flip values at the mutation point
        if (fittest.genes[mutationPoint] == 0) {
            fittest.genes[mutationPoint] = 1;
        } else {
            fittest.genes[mutationPoint] = 0;
        }

        mutationPoint =
rn.nextInt(population.individuals[0].geneLength);

        if (secondFittest.genes[mutationPoint] == 0) {
            secondFittest.genes[mutationPoint] = 1;
        } else {
            secondFittest.genes[mutationPoint] = 0;
        }
    }

    //Get fittest offspring
    Individual getFittestOffspring() {
        if (fittest.fitness > secondFittest.fitness) {
            return fittest;
        }
        return secondFittest;
    }

    //Replace least fittest individual from most fittest offspring
    void addFittestOffspring() {

        //Update fitness values of offspring
        fittest.calcFitness();
        secondFittest.calcFitness();

        //Get index of least fit individual
        int leastFittestIndex = population.getLeastFittestIndex();

        //Replace least fittest individual from most fittest
        offspring
            population.individuals[leastFittestIndex] =
getFittestOffspring();
    }
}

```

```

//Individual class
class Individual {

    int fitness = 0;
    int[] genes = new int[5];
    int geneLength = 5;

    public Individual() {
        Random rn = new Random();

        //Set genes randomly for each individual
        for (int i = 0; i < genes.length; i++) {
            genes[i] = Math.abs(rn.nextInt() % 2);
        }

        fitness = 0;
    }

    //Calculate fitness
    public void calcFitness() {

        fitness = 0;
        for (int i = 0; i < 5; i++) {
            if (genes[i] == 1) {
                ++fitness;
            }
        }
    }
}

//Population class
class Population {

    int popSize = 10;
    Individual[] individuals = new Individual[10];
    int fittest = 0;
    //Initialize population
    public void initializePopulation(int size) {
        for (int i = 0; i < individuals.length; i++) {
            individuals[i] = new Individual();
        }
    }
    //Get the fittest individual
    public Individual getFittest() {
        int maxFit = Integer.MIN_VALUE;

```

```

        int maxFitIndex = 0;
        for (int i = 0; i < individuals.length; i++) {
            if (maxFit <= individuals[i].fitness) {
                maxFit = individuals[i].fitness;
                maxFitIndex = i;}
        }
        fittest = individuals[maxFitIndex].fitness;
        return individuals[maxFitIndex];
    }

    //Get the second most fittest individual
    public Individual getSecondFittest() {
        int maxFit1 = 0;
        int maxFit2 = 0;
        for (int i = 0; i < individuals.length; i++) {
            if (individuals[i].fitness >
individuals[maxFit1].fitness) {
                maxFit2 = maxFit1;
                maxFit1 = i;} else if (individuals[i].fitness >
individuals[maxFit2].fitness) {maxFit2 = i;}}
        return individuals[maxFit2];
    }



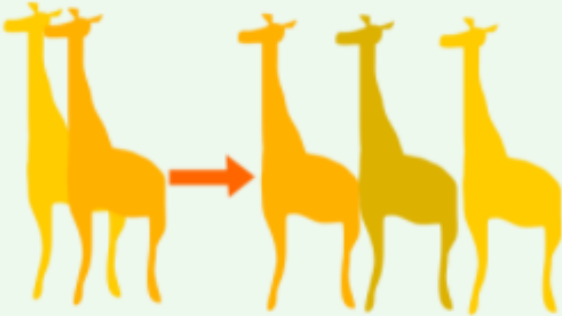
    //Get index of least fittest individual
    public int getLeastFittestIndex() {
        int minFitVal = Integer.MAX_VALUE;
        int minFitIndex = 0;
        for (int i = 0; i < individuals.length; i++) {
            if (minFitVal >= individuals[i].fitness) {
                minFitVal = individuals[i].fitness;
                minFitIndex = i;}}
        return minFitIndex;
    }

    //Calculate fitness of each individual
    public void calculateFitness() {
        for (int i = 0; i < individuals.length; i++)
{individuals[i].calcFitness();}
        getFittest();}

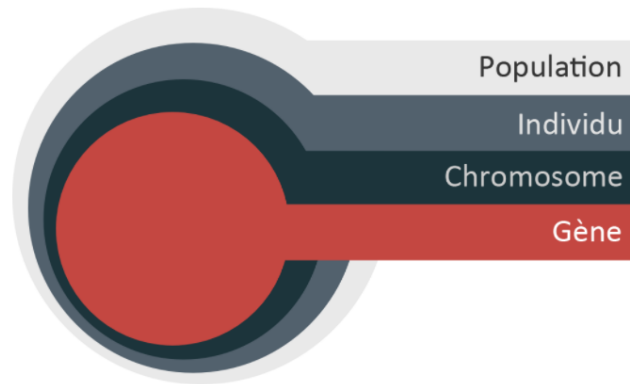
```

II. Paramètres et utilité (Bin Based Representation):

- Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné.
- Les algorithmes génétiques utilisent la théorie de Darwin sur l'évolution des espèces. Elle repose sur trois principes : le principe de variation, le principe d'adaptation et le principe d'hérédité.

<p>Le principe de variation : Chaque individu au sein d'une population est unique. Ces différences, plus ou moins importantes, vont être décisives dans le processus de sélection.</p>	
<p>Le principe d'adaptation : Les individus les plus adaptés à leur environnement atteignent plus facilement l'âge adulte. Ceux ayant une meilleure capacité de survie pourront donc se reproduire davantage.</p>	
<p>Le principe d'hérédité : Les caractéristiques des individus doivent être héréditaires pour pouvoir être transmises à leur descendance. Ce mécanisme permettra de faire évoluer l'espèce pour partager les caractéristiques avantageuse à sa survie.</p>	

- Ce paradigme, associé avec la terminologie de la génétique, nous permet d'exploiter les algorithmes génétiques : Nous retrouvons les notions de Population, d'Individu, de Chromosome et de Gène.



- **La population** : est l'ensemble des solutions envisageables.
- **L'individu** : représente une solution.
- **Chromosome** : Le chromosome contient plusieurs gènes et a une longueur n .
- **Gène** : Un gène représente un bin, la position de chaque gène dans un chromosome, indique alors le numéro de l'objet placé dans l'emplacement représenté par ce gène.

Pour mieux comprendre, voici un exemple :

Le chromosome **2 3 4 1 5** indique que l'**objet 1** est placé dans l'**emplacement 2**, l'**objet 2** est placé dans l'**emplacement 3**, l'**objet 3** est placé dans l'**emplacement 4**, et ainsi de suite.(figure 1).

1	2	3	4	5	← <i>object</i>
2	3	4	1	5	← <i>bin</i>

Figure 1: Bin-based representation of a chromosome

III. Application de l'algorithme :

Pour l'application de l'algorithme nous allons suivre 5 phases :

- 1) Initial Population
- 2) Fitness Function
- 3) Selection
- 4) Crossover
- 5) Mutation

1) Initial Population :

Le processus commence par un ensemble d'individus que l'on appelle une population. Chaque individu est une solution au problème.

Pour cet exemple, on prend tout simplement un exemple de transport de fourniture dans un camion, nous allons prendre par exemple 50 différentes dispositions des fournitures dans le camion.

2) Fitness Function :

La fonction de fitness détermine l'aptitude d'un individu(solution) (sa capacité à rivaliser avec d'autres solutions). Elle donne un score de fitness à chaque individu.

La probabilité qu'un individu soit sélectionné pour la reproduction est basée sur son score de fitness. Donc nous allons choisir lesquels de ces 50 dispositions ont un temps et efficacité positive.

3) Selection :

L'idée de la phase de sélection est de choisir les individus les plus aptes et de les laisser transmettre leurs gènes à la génération suivante.

On choisit 20 dispositions et on les met par paires.

4) CrossOver :

Le croisement est la phase la plus importante d'un algorithme génétique. Pour chaque paire de parents à accoupler, un point de croisement est choisi au hasard dans les gènes.

Exemple :



5) Mutation :

Dans certaines nouvelles descendances formées, certains de leurs gènes peuvent être soumis à une mutation avec une faible probabilité aléatoire. Cela implique que certaines parties de la chaîne peuvent être inversées.

Alors nous avons en finale une nouvelle population de solutions/individus pour notre problème.