

ITI 1521. Introduction à l'informatique II

Hiver 2008

Devoir 1

Échéance : Mercredi 30 janvier 2008, 23 h

[[PDF](#)]

Objectifs

- Bien maîtriser les concepts liés aux tableaux ;
- Savoir manipuler des variables références ;
- Revisiter les concepts de base de la programmation orientée objet ;
- Éditer, compiler et exécuter des programmes Java ;
- Sensibiliser les étudiants face au problème du plagiat, et connaître les règlements de l'université à ce sujet.

Introduction

Pour certains problèmes, il n'y a aucun algorithme **exact** et **efficace** connu. Exact signifie un algorithme qui trouve la « vraie » solution au problème donné. Par exemple, pour un problème d'optimisation (minimisation), l'algorithme doit trouver une solution dont la valeur est inférieure à la valeur de toutes autres solutions. Efficace signifie un algorithme qui peut traiter des jeux de données intéressants, de taille réelle, dans un temps raisonnable (personne ne veut attendre 32,000 ans pour obtenir une solution).

La résolution de tels problèmes est bien souvent nécessaire. Le problème de ce genre le mieux connu est sans doute le problème du commis voyageur («travelling salesman problem»). Étant donné n villes ainsi que les coûts de déplacement d'une ville à l'autre, il s'agit de trouver un chemin de moindre coût visitant chaque ville exactement une fois, puis qui retourne à la ville de départ.

Puisque les solutions de tels problèmes sont nécessaires, on se contente souvent de solutions approchées que l'on peut trouver en un temps raisonnable. Les algorithmes génétiques (GAs) sont une famille d'approches pour trouver des solutions approchées pour un grand nombre de problèmes difficiles à résoudre.

Les algorithmes génétiques s'inspirent du processus de sélection naturelle. En résumé, l'approche consiste à générer une population initiale de solutions. Elles sont généralement générées au hasard. Par conséquent, la qualité des solutions initiales est généralement médiocre. Par la suite, l'algorithme génère de nouvelles populations en sélectionnant (au hasard) des parents qui produiront (par croisement et mutations) une solution enfant. Les solutions mieux adaptées (de meilleure qualité) remplaceront les solutions inférieures afin d'augmenter la performance moyenne de la population. L'algorithme s'arrête après un nombre fixe et prédéterminé de générations (itérations). À la fin, l'individu ayant la meilleure performance est retourné.

Pour ce devoir, vous devez implémenter un algorithme génétique pour résoudre un problème d'affectation. On souhaite placer n objets, chacun ayant un poids spécifique, dans b boîtes. De plus, on cherche une solution telle que le poids des boîtes soit environ le même. Pour y arriver, nous chercherons une solution telle que la différence de poids entre la boîte la plus lourde et la moins lourde soit aussi petite que possible. Ainsi, si toutes les boîtes avaient le même poids, la solution idéale, alors la différence serait 0. De tels problèmes se présentent souvent dans des contextes industriels (placer n objets dans b contenants, placer n contenants dans b avions, etc.). Un exemple pratique consiste à trouver la meilleure stratégie pour faire la sauvegarde

des n fichiers audio de sa bibliothèque iTunes en utilisant b CD, de sorte que la taille des CD (en octets) soit environ la même.

Voici un exemple complet. Il faut placer n objets, donc voici les poids respectifs, 27, 1, 26, 24, 8, 22, 6, 29, 26, 44, 11, 22, 45, 19, 34, 2, 20, 47, 45 et 22, dans 8 boîtes de sorte que la différence de poids entre les boîtes soit aussi petite que possible.

L'énumération exhaustive des solutions est impossible (force brute) puisqu'il y a $b^n = 8^{20} \sim 10^{18}$ solutions distinctes¹.

Voici une solution assez médiocre : [1, 22, 34], [27, 22], [2, 20, 45], [26, 19], [24, 11], [44, 45], [8, 6, 47, 22] et [29, 26]; chaque groupe de nombres représente une boîte, chaque nombre représente le poids d'un objet. La différence de poids entre les boîtes 5 et 6 est 54.

La solution qui suit est de meilleure qualité, [8, 29, 19, 22], [24, 44], [1, 22, 45], [26, 20], [26, 22], [27, 2, 47], [6, 34] et [11, 45]. En effet, le poids de la boîte la plus lourde est $8 + 29 + 19 + 22 = 78$. Le poids de la boîte la plus légère est $6 + 34 = 40$. Ainsi, la différence de poids maximale est de 38 unités. La meilleure solution que j'ai trouvée a une différence de 10 unités entre la boîte la plus lourde et la moins lourde.

Pour ce devoir, vous devez suivre les directives ci-bas et concevoir un algorithme génétique afin de résoudre ce problème d'affectation.

La classe **GeneticAlgorithm** implémente la boucle extérieure de l'algorithme. Sa méthode principale fait la saisie des données sur la ligne de commande. Le premier argument est toujours le nombre de boîtes. Il est suivi du poids des n objets du problème. Il y aura toujours au moins deux arguments, le nombre de boîtes, et le poids d'au moins un objet. Voici un exemple.

```
> java GeneticAlgorithm 8 27 1 26 24 8 22 6 29 26 44 11 22 45 19 34 2 20 47 45 22
```

```
Best solution = [26,47], [44,2], [26,22], [29,45], [1,11,19,22], [45,34], [27,24,8], [22,6,20],  
Fitness = 33
```

Étant donné la nature stochastique de l'algorithme, chaque exécution produira vraisemblablement une nouvelle solution.

Respect des règles et consignes (15 points)

Vous devez préférentiellement faire le travail en équipe de deux, mais vous pouvez aussi faire le travail individuellement. Veuillez suivre les consignes disponibles sur la page [des consignes aux devoirs](#). Tous les devoirs doivent être soumis à l'aide de [maestro](#).

1 Chromosome (40 points)

De façon générale, un **Chromosome** représente une solution candidate. Sa représentation dépend du problème à résoudre. La recombinaison (crossover) de deux chromosomes produit un chromosome enfant. Comme pour les chromosomes naturels, ces derniers subissent aussi des mutations. Chaque chromosome possède une valeur d'adaptation qui sert à mesurer la qualité de cette solution. Une **Population** est une collection de chromosomes. À chaque itération (génération), l'algorithme génétique sélectionne une paire de chromosomes pour se reproduire. La solution enfant est insérée dans la population, l'individu le moins bien adapté sera éliminé. La taille de la population demeure fixe.

Implémentez la classe **Chromosome**.

- La classe **Chromosome** déclare deux constantes. L'une d'elles spécifie le taux de recombinaisons («crossover rate»). L'autre spécifie le taux de mutations. Les valeurs des constantes appartiennent à l'intervalle 0 à 1. Utiliser les valeurs, 0.8 et 0.005, respectivement, comme point de départ pour vos expériences ;
- Cette classe possède un seul constructeur, voici sa signature **public Chromosome(int[] weights, int numberOfContainers)**. Le tableau **weights** définit le poids de chacun des n objets. Le second paramètre spécifie le nombre de boîtes à utiliser (b) pour l'affectation des n objets ;

¹Si on fait l'hypothèse qu'on peut traiter un million de solutions à la seconde, il faudrait près de 32,000 ans pour toutes les évaluer.

- Chaque chromosome mémorise le poids des objets, ainsi que le nombre de boîtes à utiliser ;
- Chaque chromosome représente une solution candidate au problème d'affectation. Une solution affecte chaque objet à l'une des b boîtes. Spécifiquement, le chromosome possède un tableau de taille n . Chaque cellule contient un entier pris dans l'intervalle de valeurs 0 à $(b - 1)$. La valeur de la cellule i est l'affectation de l'objet i à l'une des (b) boîtes. Par analogie avec les chromosomes naturels, les entrées du tableau sont des gènes ;
- Lors de la création d'un nouveau chromosome, chaque objet est affecté à une boîte au hasard ;
- Un chromosome possède une méthode d'instance **public Chromosome copy()** qui retourne un nouveau chromosome dont le contenu est identique à **celui-ci** ;
- Un chromosome possède une méthode d'instance **public Chromosome crossover(Chromosome other)** qui retourne un nouveau **Chromosome** ayant les caractéristiques suivantes. Avec probabilité $(1 - \text{CROSSOVER_RATE})$, le chromosome enfant sera une copie conforme de **ce** parent. Sinon, le chromosome enfant aura les k premiers gènes de **ce** parent et les $n - k$ derniers gènes du second parent (**other**). La valeur k est choisie au hasard ;
- Un chromosome possède une méthode d'instance **public void mutate()**. Avec probabilité $(1 - \text{MUTATION_RATE})$, un gène demeure inchangé. Sinon, il y a mutation et la valeur du gène est changée par une valeur choisie au hasard (dans l'intervalle 0 à $b - 1$). La méthode doit traiter tous les gènes du chromosome de cette façon ;
- La méthode d'instance **public int getFitness()** retourne une valeur indiquant la qualité de la solution (valeur d'adaptation). Spécifiquement, la méthode retourne la différence de poids entre la boîte la plus lourde et la moins lourde que représente cette solution ;
- Finalement, la méthode **public String toString()** retourne une représentation sous forme de chaîne de cette solution. Voici le format attendu. La chaîne présente toutes les (b) boîtes. Pour chaque boîte, elle spécifie le poids des objets qui s'y trouvent. De plus, la chaîne donne la valeur d'adaptation. Voir l'exemple ci-haut.

2 Population (30 points)

Comme nous l'avons vu, une **Population** est une collection de chromosomes (où chaque chromosome est une solution candidate au problème d'affectation). Pour faciliter l'implémentation des méthodes, **les chromosomes sont toujours en ordre croissant de valeur d'adaptation**. Puisque le but de l'exercice est de minimiser la différence de poids, la plus petite valeur d'adaptation est la meilleure.

- La classe **Population** déclare une constante établissant la taille implicite (par défaut) de la population. Utilisez la valeur 100 ;
- Une **Population** comprend un nombre fixe de chromosomes. Le nombre de chromosomes est déterminé lors de la création d'une population ;
- La classe possède deux constructeurs. Les constructeurs créent les populations initiales ;
- Le premier constructeur a la signature qui suit **public Population(int size, int[] weights, int numberOfContainers)**. Le paramètre **size** spécifie la taille de la population, donc le nombre de chromosomes à créer. Le tableau **weights** définit le poids des n objets. Finalement, le troisième paramètre détermine le nombre de boîtes à utiliser (b) ;
- Le second constructeur a la signature suivante : **Population(int[] weights, int numberOfContainers)**. Il crée une nouvelle population ayant la taille par défaut ;
- La classe **Population** possède une méthode **public void evolve()** dont voici les caractéristiques. La méthode choisit deux chromosomes parent. Un chromosome enfant est créé à partir des deux parents, à l'aide de la méthode **crossover**. Le chromosome subit des mutations, puis il est inséré dans la population. En conséquence, le chromosome le moins bien adapté est éliminé de la population. Souvenez-vous que les individus de la population sont toujours en ordre croissant de valeur d'adaptation. Pour la sélection des parents, vous avez plusieurs choix. L'un des choix consiste à sélectionner au hasard deux parents. Un autre choix consiste à sélectionner l'individu le mieux adapté, ainsi qu'un individu au hasard. Finalement, les deux meilleurs individus pourraient être sélectionnés pour la reproduction ;
- La méthode d'instance **public Chromosome getFittest()** retourne le chromosome le mieux adapté, c'est-à-dire, celui ayant la plus petite valeur d'adaptation.

3 GeneticAlgorithm (10 points)

Complétez l'implémentation de la méthode `private static Chromosome solve(int weights[], int numberOfContainers)`. Celle-ci doit créer une nouvelle population, simuler `NUMBER_OF_GENERATIONS` générations, et retourner l'individu ayant la meilleure valeur d'adaptation.

4 Qualité de votre meilleure solution (5 points)

Dans un fichier, nommé `best.txt`, soumettez votre meilleure solution.

- 0 point si le programme ne produit aucune solution ;
- 5 points si la valeur d'adaptation de la solution est 10 ou moins ;
- 4 points si la valeur d'adaptation de la solution est 15 ou moins ;
- 3 points si la valeur d'adaptation de la solution est 20 ou moins ;
- 2 points si la valeur d'adaptation de la solution est 30 ou moins ;
- 1 point pour toute autre solution.

Fraude scolaire

Cette partie du devoir a pour but de sensibiliser les étudiants face au problème de fraude scolaire (plagiat). Lisez les deux documents qui suivent :

- www.uottawa.ca/academic/info/regist/annuaires/reglements/fraude.html
- www.uottawa.ca/plagiat.pdf

Les règlements de l'université seront appliqués pour tout cas de plagiat.

En soumettant ce devoir, 1) vous témoignez avoir lu les documents ci-haut et 2) vous comprenez les conséquences de la fraude scolaire.

Fichiers

Vous devez soumettre les fichiers suivants.

- README.txt
- [StudentInfo.java](#)
- Chromosome.java
- Population.java
- [GeneticAlgorithm.java](#)
- best.txt

A Foire Aux Questions (FAQ)

1. «Aucune»
Pour l'instant...

Modifié le : 16 janvier 2008