ALLARD Elsa

29/11/2024

Compte Rendu de TP d'Introduction à l'Intelligence Artificielle

Table des matières

I/ TP 1 : Algorithmes gloutons 3
Exercice1 3
Exercice2 4
TP2 : Initiation à l'algorithme génétique 5
Exercice 1 : Représentation des solutions (codage) et définition de la fonction objective 5
Exercice 2 : Prise en main du framework d'algorithme génétique 5
Exercice 3 : Implémentation 5
TP3 : Initiation à l'algorithmique génétique (suite) 7
Exercice 1 : Opérateurs génétiques 7
conclusion 8

TP 1 : Algorithmes gloutons

Exercice1

L'objectif de ce programme est de gérer un ensemble d'épreuves, dont la lecture d'un fichier, leur manipulation (ajout, suppression, tri), ainsi que la résolution des conflits d'horaire entre épreuves. Ce genre de gestion est particulièrement bénéfique dans des situations où il est nécessaire d'aménager de manière efficace des ressources limitées (telles qu'une salle d'examen).

Le programme est constitué d'une classe principale appelée "ListeEpreuves", qui offre des techniques pour charger les données, repérer et résoudre les conflits, et classer les épreuves en fonction de divers critères. Les épreuves sont manipulées en utilisant une classe auxiliaire appelée "Epreuve" (non fournie dans ce code mais souvent supposée exister).

Pour la classe « ListeEpreuves » Le nom du fichier contenant la liste des épreuves. Ce fichier est donc lu au moment de l'instanciation. De plus une collection de type ArrayList<Epreuve> qui contient toutes les épreuves. L'utilisation d'une ArrayList permet de manipuler dynamiquement les épreuves, de les parcourir ou de les trier facilement.

Le programme s'attend à recevoir un fichier texte contenant une liste d'épreuves, chaque ligne représentant une épreuve avec les informations suivantes (Nom de l'épreuve, Horaire de début, Horaire de fin).

La méthode lireFichierEpreuves(String fichier) permets de lire les épreuves depuis le fichier spécifié.

- Les données de chaque ligne sont séparées à l'aide de la classe `StringTokenizer`.
- Ces informations sont ensuite utilisées pour créer une instance de la classe `Epreuve`.
- Chaque épreuve est ajoutée à l'attribut `liste`.

La méthode eliminerConflits(Epreuve) élimine toutes les épreuves en conflit horaire avec une épreuve donnée. Une épreuve est considérée en conflit si ses horaires se chevauchent avec ceux de l'épreuve fournie.

- Parcourt la liste des épreuves.
- Supprime les épreuves répondant aux conditions suivantes :
- L'épreuve actuelle est différente de l'épreuve donnée.
- Son horaire de début est avant la fin de l'épreuve donnée.
-Son horaire de fin est après le début de l'épreuve donnée.

La méthode triParHeureFin() trie la liste des épreuves en fonction de leur horaire de fin, en utilisant `Collections.sort`. Ce tri est particulièrement utile pour implémenter un algorithme glouton, comme maximiser le nombre d'épreuves qui peuvent être organisées dans une même salle.

Pour que cette dernière fonctionne, la classe Epreuve doit implémenter l'interface et fournir une méthode compareTo qui compare les horaires de fin.

La classe propose plusieurs méthodes utilitaires pour manipuler directement la liste des épreuves :

- get(int i): Récupère une épreuve à une position donnée.

- set (int i, Epreuve): Ajoute une épreuve à une position donnée.

- replace (int i, Epreuve): Remplace une épreuve existante à une position donnée.

- remove(int i): Supprime une épreuve par son indice.

Par la suite chaque épreuve est convertie en texte via sa méthode `toString` et ajoutée à une chaîne de caractères avec des sauts de ligne.

Exercice2

Ce code présente deux méthodes pour diviser une liste d'entiers en deux sous-listes équilibrées en termes de somme des éléments. Voici une explication détaillée des différentes parties du programme :

- Deux listes, E et F, contenant des entiers.
- La liste utilisée pour la division peut être sélectionnée en modifiant la variable liste dans la vfméthode main.
- sListe1 et sListe2 : stockent les deux partitions générées.

Méthode 1 : Approche gloutonne

La méthode completerSousEnsembleLePlusPetit procède de la manière suivante. Premièrement il trie la liste initiale en ordre décroissant puis parcourt chaque élément et l'ajoute à la sous-liste dont la somme est actuellement la plus petite.

L'approche gloutonne possèdes quelques avantages car elle est plus rapide à implémenter ce qui permet une approche simple pour équilibrer les sous-listes. Cependant cette méthode possède des limites effectivement nous nous sommes rendu compte qu'elle ne peut pas produire la partition la plus optimale.

Méthode 2 : Utilisation de la moitié de la somme comme repère

La méthode utiliserMotieSommeCommeRepere quant à elle calcule la somme totale des éléments de la liste. Dans un premier temps elle définit un seuil (moitié = somme totale / 2) pour guider la répartition, puis elle trie la liste en ordre décroissant. Pour finir elle va ajouter les éléments à la première sous-liste tant que sa somme ne dépasse pas « moitié ». Les autres éléments sont ajoutés à la seconde sous-liste. Cette méthode est mieux adaptée pour minimiser l'écart entre les sommes des sous-listes. Cependant elle peut échouer si la liste contient des valeurs où aucune combinaison ne correspond exactement à la moitié de la somme.

Ainsi, nous pouvons dire que les résultats de la bipartition dépendent fortement de l'ordre des éléments et de l'algorithme utilisé. La méthode basée sur la moitié de la somme tend à minimiser plus efficacement les écarts entre les deux sous-listes, tandis que l'approche gloutonne reste plus simple à implémenter mais moins précise. Le tri préalable en ordre décroissant, assuré par Collections.sort avec Collections.reverseOrder(), est essentiel pour maximiser l'efficacité de ces deux méthodes.

TP2 : Initiation à l'algorithme génétique

Exercice 1 : Représentation des solutions (codage) et définition de la fonction objective

1) a) Nos variables de décision (gènes) sont les bits de la solution.
   b) Les valeurs possibles pour chaque gène (valeur min, valeur max) sont 0 ou 1.
   Notre chromosome correspond à la solution
2) Chacun des n bits peuvent 2 valeurs (0 ou 1), la taille de l'espace de recherche est donc 2n
3) Ici, la fonction objective est la fonction permettant de convertir un nombre binaire en nombre décimal

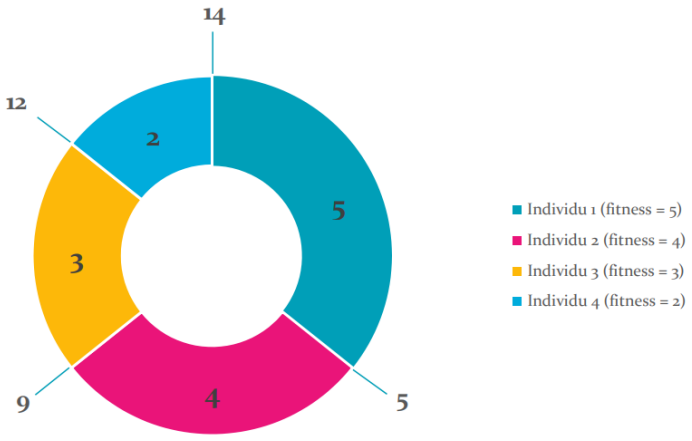
$$\sum_{i=0}^{n-1} 2\text{^}i * valeur\ du\ gène\ i$$

Exercice 2 : Prise en main du framework d'algorithme génétique

En changeant certains paramètres du code donner, nous pouvons remarquer que celui-ci implémente une méthode de sélection par roulette pour un algorithme génétique. La classe Selection_roulette hérite d'une classe Selection et prend une population de solutions en paramètre. La méthode selectionner calcule la somme des valeurs de fitness de la population, génère une valeur aléatoire basée sur cette somme, puis sélectionne une solution en fonction de la roulette virtuelle, où la probabilité de sélection est proportionnelle à la fitness de chaque solution.

Exercice 3 : Implémentation

1,2) La roulette consiste à choisir aléatoirement des individus parmi une population. Chaque personne reçoit une valeur de fitness, qui correspond à ses probabilités d'être choisies au hasard. L'augmentation de la valeur de fitness augmente les chances d'être sélectionné pour l'individu. La dimension de la roulette est donc la somme des valeurs de fitness de chaque personne.



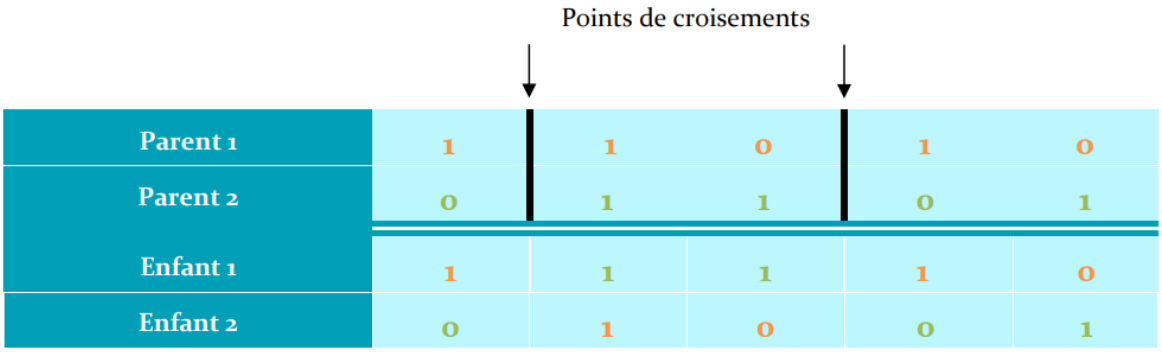
Il s'agit donc du principe suivant : on sélectionne une valeur aléatoire entre 0 et 14, par exemple le 7, 7 se situe entre 5 et 9, c'est donc l'individu 2.

Afin de programmer une sélection par roulette en partant des individus et de leur valeur de fitness, il est nécessaire de trouver d'abord la liste de la somme cumulée croissante des valeurs de fitness (dans notre cas, la liste sera donc [5, 9 , 12, 14]).

Une valeur aléatoire entre 0 et 14 (entre 0 et liste[-1]) est sélectionnée, puis elle est comparée à chaque valeur de la liste à l'aide d'une boucle. À chaque fois, la condition aléa > liste[i] est vérifiée avec l'itération.

Dès que la condition n'est plus respectée, on s'arrête et l'individu sélectionné correspondra à la dernière itération. En remplaçant la sélection aléatoire par une sélection basée sur une roulette biaisée, on remarque une très légère amélioration du résultat moyen.

Pour obtenir un croisement 2-points on tire aléatoirement 2 nombres entiers a1 et a2, qui seront nos deux points de croisements, entre 1 et la longueur L du chromosome - 1.



On fait en sorte que a1 corresponde au plus petit des nombres tirés aléatoirement et a2 le plus grand. Dans une boucle « pour i allant de 0 à L – 1 », on définit les conditions suivantes :

- Si i < a1 ou si i > a2, alors l'enfant 1 hérite de l'allèle i du parent 1, et l'enfant 2 hérite de celui du parent 2

- Si a1 < i < a2, alors l'enfant 1 hérite de l'allèle i du parent 2, et l'enfant 2 hérite de l'allèle i du parent 1.

En remplaçant le croisement 1-point par celui de 2-point, on constate une amélioration. Le résultat moyen a encore augmenté, et il semble être plus constant et moins varier.

TP3 : Initiation à l'algorithmique génétique (suite)

Exercice 1 : Opérateurs génétiques

1) Une sélection basée sur un tournoi de taille k implique de choisir le meilleur élément parmi k éléments en organisant un tournoi La population est divisée en groupes de k, puis les éléments de chaque groupe sont comparés pour choisir le meilleur. Cette procédure se répète jusqu'à ce qu'il ne reste qu'un élément, qui est alors le vainqueur de la manifestation. Cette sélection est donc mise en œuvre pour être testée. Cette méthode de sélection est plus performante que la méthode de sélection basée sur une roulette biaisée, avec une moyenne de deux gènes à 0 et un résultat satisfaisant.

2) On tire aléatoirement k nombres entiers, qui seront nos points de croisements, entre 1 et la longueur L du chromosome -1. Il est nécessaire de définir les conditions de changement dans la boucle de la façon suivante cette fois-ci :

   - Si j%2 == 0, c'est-à-dire si i est pair, alors l'enfant 1 hérite de l'allèle i du parent 1, et l'enfant 2 hérite de celui du parent 2
   - Si j%2 == 1, alors l'enfant 1 hérite de l'allèle i du parent 2, et l'enfant 2 hérite de l'allèle i du parent 1 Avec j, l'indice du point de croisement actuel. On incrémente j si i > points_de_croisements[j].

conclusion

Pour finir avec l'introduction à l'algorithmique génétique, deux concepts essentiels sont mis en évidence : la roulette biaisée pour la sélection et le croisement en deux points pour la transcription. L'utilisation de la roulette biaisée permet de choisir des individus performants tout en préservant la diversité génétique. Les deux points de croisement permettent de générer de nouvelles solutions en échangeant des portions génétiques entre deux parents. Ces idées, qui tirent leur inspiration de la sélection naturelle, jouent un rôle essentiel dans la création d'algorithmes génétiques, créant ainsi une approche heuristique efficace pour résoudre des problèmes d'optimisation et de recherche.