ESIR 1-AC, TP n°4 : Algorithmes génétiques

Pierre Maurel, pierre.maurel@irisa.fr

Dans ce TP vous allez implémenter un cadre général d'algorithme génétique (AG) et l'utiliser pour étudier les problèmes du sac à dos et du voyageur de commerce (vus en TD). Sur l'ENT, vous trouverez un fichier `Individu.java` contenant une interface `Individu` et un fichier `Population.java` contenant le squelette (à compléter) d'une classe `Population<Indiv extends Individu>`.

Le principe est le suivant :

- la classe `Population` contiendra les fonctionnalités communes nécessaires à tous les AG (sélection, reproduction, ...), cette classe ne dépendra donc en aucune manière d'un problème spécifique (sac à dos, voyageur de commerce, ...).
- pour chaque problème, il ne reste "plus qu'à" implémenter l'interface `Individu` pour adapter la représentation interne, l'adaptation, la reproduction et la mutation au problème spécifique (sac à dos, voyageur de commerce, ...). Enfin on pourra utiliser les fonctions de la classe `Population` pour résoudre le problème en question.

1 Problème du sac à dos

1.1 Travail demandé

- Vous commencerez par écrire une classe `Individu_SAD` implémentant l'interface `Individu`. Il vous faudra donc (au moins) : choisir une représentation interne et les attributs correspondants, définir un constructeur construisant un individu aléatoire, implémenter les fonctions présentes dans l'interface.
- Vous compléterez ensuite la classe `Population<Indiv extends Individu>` qui représente un ensemble d'individu de type `Indiv` (où `Indiv` est un type générique devant implémenter l'interface `Individu`).
- Ensuite, complétez le client `Client_Sac_A_Dos` fourni pour tester votre algorithme. Vous écrirez donc une fonction `main` qui :
  - crée et remplit un tableau de poids (vous pourrez utilisez la fonction `charge_poids` qui lit les poids dans un fichier texte comme ceux fournis en exemple et renvoie le tableau correspondant)
  - crée un tableau d'`Individu_SAD` d'une taille donnée. Chacun des `Individu_SAD` du tableau sera donc associé au tableau de poids ainsi qu'au poids maximal du problème du sac à dos considéré.
  - construit un objet `Population` à partir de ce tableau
  - produit des populations (ou générations) successives.
  - affiche pour chaque génération l'adaptation moyenne de la population ainsi que l'adaptation maximale.
- Finalement, testez la méthode pour les différents fichiers de poids et pour différentes valeurs des paramètres. En particulier vous pourrez comparer vos résultats avec ceux obtenus pour une probabilité de mutation égale à 0.5 (ce qui revient, comme vu en TD, à une recherche aléatoire de la solution optimale).

1.2 Indications

Vous devez donc (dans un premier temps) implémenter les méthodes étudiées en TD (cf TD n°7). Vous pouvez également tester d'autres manières d'implémenter ces méthodes en vous inspirant par exemple des liens proposés à la fin de ce TP.

1.2.1 `Indiv_SAD.croisement()`

Pour le croisement des deux individus, on tire au hasard une position et les sous-chaînes s'inversent.

1.2.2 `Population.selection()`

Commencez par implémenter la sélection par roulette vu en TD. Vous pourrez éventuellement par la suite comparer par exemple avec la sélection par tournoi.

1.2.3 `Population.reproduction()`

Il s'agit de la fonction principale de l'algorithme. Pour commencer, vous pouvez construire la nouvelle génération à partir de la précédente de la manière suivante : on sélectionne deux individus (grâce à la méthode `selection()`), ils se reproduisent (méthode `croisement()`), les enfants sont alors ajoutés à la nouvelle génération et on recommence tant qu'on n'a pas assez d'individus. Une fois la nouvelle génération créée, on applique l'opérateur de mutation à chaque individu.

Afin d'être sûr que l'adaptation maximale de la population ne diminue jamais au cours du temps, implémentez l'amélioration suivante, qu'on appelle **l'élitisme**. Cela consiste à conserver systématiquement l'individu ayant l'adaptation maximale d'une génération et à l'insérer (sans mutation) dans la suivante. On peut également rajouter plusieurs autres copies de l'individu maximal qui seront, elles, soumises à la mutation.

2 Voyageur de Commerce

Écrivez une classe `Individu_VDC` implémentant l'interface `Individu` dans le but d'utiliser votre classe `Population<Indiv_VDC>` pour résoudre le problème du voyageur de commerce. Il vous faudra donc réfléchir à la représentation interne de cette classe `Individu_VDC` ainsi qu'aux méthodes de reproduction et mutation choisies.

De la même manière que pour le sac à dos, écrivez un client permettant de tester votre algorithme sur le problème du voyageur de commerce. Vous pourrez tester dans un premier temps avec les fichiers coordonnées fournis. Vous pouvez afficher les solutions trouvées grâce à la classe `Display_VDC`[1]. Testez finalement votre méthode sur ce problème : `http://labo.algo.free.fr/defi250/defi_des_250_villes.html`.

Liens utiles

- `http://lsis.univ-tln.fr/~tollari/TER/AlgoGen1/node5.html`
- `http://en.wikipedia.org/wiki/Genetic_algorithm`
- `http://numeratus.net/applets/knapsack/gks.html`
- `http://labo.algo.free.fr/pvc/probleme_du_voyageur_de_commerce.html`

1. librement inspiré de `http://labo.algo.free.fr/code/DisplayTsp/DisplayTsp.html`