Cours Contrôle Avancé
Partie Algorithmes Génétiques

Ecole Centrale Paris

Algorithme évolutionnaire pour le problème de voyageur de commerce (Travelling Salesman Problem)

Anne Auger, Dimo Brockhoff
firstname.lastname@inria.fr
http://researchers.lille.inria.fr/~brockhof/advancedcontrol/

Introduction

Pour l'optimisation combinatoire, les algorithmes évolutionnaires ne sont en général pas compétitifs (en terme de qualité de la solution trouvée et vitesse de convergence) face à des algorithmes spécialisés. En revanche, contrairement aux algorithmes spécifiques pour un problème, lourds à mettre en oeuvre car nécéssitant une expertise importante sur le problème, les algorithmes évolutionnaires ont l'avantage d'être faciles à implémenter.

Le but de cette séance est de mettre en oeuvre un algorithme évolutionnaire (AE) pour résoudre un problème d'optimisation combinatoire, le problème du voyageur de commerce définit de la façon suivante: étant donné un ensemble de n villes séparées par des distances données, trouver le plus court chemin qui relie toutes les villes.

Représentation des données du problème

- ⋆ Une ville va être représentée par un nombre entier et n va dénoter le nombre de villes.
- ⋆ Une carte de taille n est une matrice à n lignes et 2 colonnes. La ligne i de la matrice contient l'abscisse et l'ordonnée de la ville i. Pour simplifier le problème on va supposer que les abscisses et ordonnées sont des entiers.

$$\text{Carte: } \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix}^T$$

- ⋆ Les distances entre les villes vont être représentées par une matrice symétrique d de taille $n \times n$ avec des 0 sur la diagonale, i.e.

$$d(i,j) = \text{distance entre la ville i et la ville j}$$

Représentation du problème, espace de recherche et fonction fitness

1. Une solution possible pour le problème est un circuit, c'est à dire un ordre de visite des villes. Comprendre pourquoi permutation circulaire à n élément (ou *n-cycle*) et circuit sont la même chose. En matlab, un circuit pour n villes sera un vecteur de taille n.

2. Quel est la taille de l'espace de recherche ? I.e. quel serait la complexité d'une recherche exhaustive?

3. Ecrire (sur papier) la fitness associée à un circuit.

Instance du problème

Afin de tester notre programme, nous allons créer des problèmes à résoudre.

1. Créer une fonction nommée *CarteAlea* prenant en argument un entier n (le nombre de villes), une abscisse maximale **xmax** (que l'on prendra entier) et une ordonnée maximale **ymax** (que l'on prendra entier) et renvoyant une carte aléatoire. Pour tirer un entier uniformément entre 0 et **xmax** on pourra utiliser l'instruction matlab `randint`.

2. Créer une fonction nommée *DistanceCarte* prenant en argument une carte et retournant la matrice des distances entre chaque ville.

Implémentation de la fonction fitness

1. Ecrire une fonction matlab nommée *fitness* prenant en entrée un circuit, une carte des distances et retournant la fonction fitness.

Interface graphique

1. Ecrire une fonction *DessCarte* qui prend en argument une carte et dessine l'emplacement des villes sur l'écran.

2. Ecrire une fonction *DessCircuit* qui prend en argument une carte et un circuit et dessine le circuit sur la carte ainsi que l'emplacement des villes.

Initialisation

L'étape d'initialisation de l'algorithme va consister à créer une population (de taille μ) aléatoire. Il va donc falloir générer des n-cycles aléatoirement et uniformément dans l'ensemble de tous les n-cycles possibles.

Une façon de faire cela en Matlab consiste à utiliser l'instruction `sort`: Etant donné un vecteur x de taille n, l'instruction `[y,i] = sort(x)` trie les éléments de x et retourne le vecteur trié y, ainsi que le vecteur des index i tel que $x(i) = y$.

Si x contient n nombres tirés uniformément et indépendamment, alors i (retourné par `sort(x)`) est un cycle aléatoire uniforme.

1. Ecrire une fonction Matlab *CycleAlea* prenant en argument un entier n et retournant un circuit aléatoire.

Opérateurs de variation

Croisement

[Rappel sur les croisements: http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)]

On va chercher maintenant à croiser les individus de la population en espérant obtenir une nouvelle génération meilleure. Il faut donc essayer de conserver les bonnes propriétés des parents.

1. Proposer des opérateurs de croisement.

Les opérateurs de croisement "classiques" vont souvent donner des solutions non-admissibles pour le problème. Plusieurs opérateurs spécifiques essayant de préserver l'ordre des villes d'une sous-partie des parents ont été mis au point comme le Partially Mapped Crossover ou le order 1 crossover. Nous détaillons le principe de l'opérateur order 1 crossover.

Order crossover: Etant donné deux parents, on génère deux enfants en prenant un sous-chemin d'un des parents et en finissant le cycle en respectant l'ordre de l'autre parent. Par exemple,

Parent 1 [1 2 | 3 4 5 | 6 7 8]
Parent 2 [2 4 | 6 8 7 | 5 3 1]

On conserve les sous-chemins des parents:

Enfant1 [$\star$ $\star$ | 3 4 5 | $\star$ $\star$ $\star$]
Enfant2 [$\star$ $\star$ | 6 8 7 | $\star$ $\star$ $\star$]

Et on complete en respectant l'ordre de l'autre parent

Enfant1 [8 7 | 3 4 5 | 1 2 6]
Enfant2 [4 5 | 6 8 7 | 1 2 3]

Les bornes du sous-chemin sont tirées au hasard.

1. Ecrire la fonction croisement qui prend en argument deux parents et retourne deux enfants.

Mutation

1. Proposer des opérateurs de mutation. Ecrire les fonctions Matlab correspondantes.

Le schéma d'évolution

Il reste à déterminer le schéma général d'évolution pour l'algorithme. Voici un exemple avec une sélection déterministe:

- $\star$ Taille de la population de parents μ. Création de $\lambda = 2 * \mu$ enfants par application successive de:
  - choix aléatoire de 2 parents parmi la population de parents, croisement des deux parents
  - application aux enfants obtenus d'un opérateur de mutation avec une probabilité p_m (typiquement faible). Choix uniforme de l'opérateur de mutation parmi l'ensemble des opérateurs.
  - évaluation des solutions
- $\star$ sélection des μ meilleurs parmi les λ enfants et μ parents. Ces μ meilleurs individus deviennent les parents de la génération suivante.

1. Ecrire une fonction matlab pour votre algorithme évolutionnaire qui prend en entrée une carte et ressort le meilleur circuit trouvé, ainsi que la longueur du circuit. Utiliser comme critère d'arrêt pour l'algorithme un nombre max d'appel à la fonction fitness. Faire afficher à chaque itération la meilleure valeur de fonction fitness pour la population.
2. Etudier l'influence des différents paramètres de l'algorithme.