IA : Algorithme Génétique

Wim Poignon
Computer Engineering
Pôle Léonard de Vinci
Paris, La Défense 92400
Email : wim.poignon@edu.devinci.fr
Projet informatique

***Résumé*—Des astronomes ont observés une nouvelle étoile inconnue. Leurs premières observations laissent penser que la température à sa surface varie de façon très régulière avec plusieurs périodicités. Les astronomes supposent que l'étoile suit une fonction dite de Weierstrass avec plusieurs périodicités imbriquées au sens fractale.**

A l'aide d'un algorithme génétique, illustré dans ce rapport, on arrive à approximer cette fonction comportant un ensemble de paramètres en utilisant uniquement une liste d'observations. Ici, chaque observation est composée de l'instant de l'observation ainsi que de la température observée.

I. POSER LE PROBLÈME

A. Espace de recherche

L'objectif de notre algorithme est de déterminer la combinaison (a, b, c) avec $a \in]0, 1[$, $(b, c) \in [\![1, 20]\!]^2$. Dans notre cas, on va arrondir a à 10^{-2} près pour éviter d'avoir une taille d'espace de recherche infini. On calcule ainsi le cardinale :

$$\mathrm{Card}(\mathcal{A}^*, \mathcal{B}, \mathcal{C}) = 99 \times 20 \times 20 = 39600,$$

où $\mathcal{A}^*$ est l'ensemble dénombrable $]0, 1[$ arrondi à 10^{-2}.

Ainsi la taille de l'espace de recherche est égale à 39 600.

B. Fonction Fitness

Pour approximer une fonction où on a recueilli une liste d'observations, il est judicieux de comparer la distance séparant la valeur de l'observation avec la valeur générée par l'algorithme génétique. Ainsi notre fitness se résume à la somme des distances euclidiennes des observations et des résultats d'un individu :

$$\mathrm{Fitness} = \sum_{i=1}^{n} |t(x_i) - t_g(x_i)|$$

où t_g est la fonction de Weierstrass avec les paramètres générés par l'algorithme génétique.

C. Opérateurs mis en œuvre

Il y a trois opérateurs d'évolution dans les algorithmes génétiques (pour diversifier la population au cours des générations) présentés ci-dessous :

1) Mutation: L'opérateur de mutation apporte aux algorithmes une certaine garantie de l'exploration de l'espace d'états. Le principe est simple pour des problèmes discrets : généralement on tire aléatoirement un gène dans le chromosome et on le remplace par une valeur aléatoire. Dans notre algorithme génétique, sachant qu'un individu est constitué du couple (a, b, c), on peut facilement choisir un paramètre aléatoirement et lui attribuer une nouvelle valeur aléatoire (compris dans son ensemble).

2) Croisement: L'opérateur de croisement a pour but d'enrichir la diversité de la population en manipulant la structure des chromosomes. Le principe est alors de « découper » les chromosomes. Ainsi, les croisements sont envisagés avec deux parents et génèrent deux enfants. Dans notre algorithme, on a fait le choix d'échanger les paramètres b et c ou juste uniquement c.

3) Sélection: La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer les mauvais. Il existe plusieurs méthodes de sélection :

- Sélection par rang
- Probabilité de sélection proportionnelle à l'adaptation
- Sélection par tournoi
- Sélection uniforme

Pour notre algorithme génétique, nous avons opté pour la méthode la plus simple (mais peut être pas la plus efficace) : la sélection par rang. Cette technique choisit toujours les individus possédant les meilleurs scores d'adaptation. On peut également choisir les pires pour une plus grande diversité.

Ainsi, pour notre population, on a fait le choix de prendre les 40% meilleurs de notre population et les 10% pires.

D. Influence des paramètres d'entrée

La taille de la population doit être bien choisi mais nécessite une étude empirique car pas assez d'individus réduit les chances de converger vers une solution stable. Alors que trop d'individus augmente le temps d'exécution. Après plusieurs tests, j'ai opté pour une population de taille 50.

En utilisant tel paramètre, il fallait que je choisis une condition d'arrêt pour que obtenir une combinaison à la fin. J'ai décidé que ma Fitness soit minimum inférieur à 0.02. Cela dit, le nombre d'itération tourne au alentour de 45. Initialement, lors de ma première version, mon algorithme génétique me donnait une valeur après plus de 200 générations. Cela est dû aux opérateurs non optimisés (cf. Section II).

E. Temps d'exécution

Le temps d'exécution de notre algorithme génétique va dépendre de plusieurs paramètres. Notamment la taille de la population influe sur le temps d'exécution vu que les opérateurs mis en œuvre sont appliqués sur l'intégralité de la population. De plus, la taille de l'échantillon peut varier le temps d'exécution car on aura plus ou moins d'opérations à réaliser pour calculer la fitness typiquement.

Dans mon cas, avec 50 individus et choisissant l'échantillon "temperature_sample.csv", le temps d'exécution tourne au alentour de 0.7seconde.

II. OPTIMISATION

Mon algorithme génétique comporte plusieurs versions. La toute première était assez lent avec beaucoup de générations car je ne faisais qu'une seule mutation pour l'ensemble de la population. Puis dans ma deuxième version, j'ai muté tous les individus ce qui implique que je n'obtenais aucune solution stable. Ainsi pour compter ce problème, je mute uniquement une partie de ma population.

De plus, j'ai testé une mutation avec un certain pas (au lieu de générer un nombre aléatoire), sauf que cela a abouti à un solution stable qui n'était pas le minimum. Ainsi, on peut dire que j'étais souvent tombé sur un minimum local.