Intelligence Bio-Inspirée

TP : Algorithmes génétiques

Bamba William , Hossain Shajjad

30/01/2021

Encadrants ..... Simon Forest – Remy Chaput

1 Objectif

Vous devez utiliser un algorithme genetique pour trouver le mot de passe, ou au moins s'en approcher le plus possible. Vous en avez les informations suivantes :

- Il est compose uniquement de chiffres et de lettres majuscules.
- Il contient entre 12 et 18 caracteres.

Un ≪ individu ≫ dans votre algorithme correspondra donc a une tentative de mot de passe, son phenotype etant une chaine de 12 a 18 chiffres et lettres majuscules.

Vous ne serez pas evalues uniquement sur la decouverte ou non du bon resultat, mais surtout sur la conception de l'algorithme et la justification de vos choix concernant :

- le codage du genotype ;
- la selection (et la presence ou non d'elitisme) ;
- les mutations ;
- le cross-over ;
- les valeurs des hyper-parametres (nombre d'individus, taux de mutation, etc.).

2 Codage du genotype

Notre génotype est codé à l'aide de caractères multiples. Pour coder le génotype nous avons utilisé Numpy qui est une librairie python qui permet un objet de tableau multimensionnel. Pour créer la première génération d'individus, nous avons donc généré N (N étant le nombre d'individus) ndarray de dimension 1, avec une taille aléatoire entre 12 et 18 et des caractères tirés aléatoirement parmi ceux inclus dans le domaine (lettres majuscules et chiffres).

3 Selection

Pour la sélection des individus qui vont être reproduit entre chaque génération, nous avons testé plusieurs méthodes. Chaque méthode était accompagnée d'élitisme , c'est-à-dire que à chaque génération la fitness des individus était calculée grâce à la fonction check qui nous renvoie une valeur compris entre 0 et 1 (1 signifiant que la chaîne de caractères donnée en entrée est exactement identique au mot de passe à trouver) et une fois toutes les fitness calculée on triée es individus de la meilleure à la pire fitness et on garder un pourcentage des premiers individus définis par l'hyper paramètres `GRADED_RETAIN_PERCENT` plus un pourcentage plus petit d'individus tirées aléatoirement . Une fois cette première sélection faite voici les méthodes implémentées :

- La roulette-wheel sélection : Le principe est que chaque individu en fonction de sa fitness à une probabilité de reproduction plus ou moins grandes. On place ensuite les individus sur une "roue biaisée" et on tire un nombre aléatoire entre 0 et 1 qui va déterminer le couple d'individu sélectionner pour la reproduction.

- La sélection uniforme : On tire aléatoirement deux individus avec une probabilité égale entre les individus. Le fait que ça soit aléatoire peu augmenter la diversité génétique mais aussi empêcher les bons phénotypes de se transmettre de génération à génération.

- La sélection par rang : Dans cette technique chaque individu est classé selon une probabilité qui dépend dans leur rang. Le rang est calculé en triant les individus du plus petit fitness à la plus grande et ensuite en appliquant la fonction suivante :

w(r) = (2 /N(N 1))*(r 1)

Cette méthode permet qu'en début d'évolution l'avantage reproductif des meilleurs individus soit modérés surtout avec un N grand. Et en fin d'évolution la sélection accentue les tout petits avantages, les mutations avantageuses sont sélectionnées même si elles ne sont que très faiblement avantageuses.

4 Cross-Over

Pour le cross-over nous avons essayé plusieurs méthodes ayant chacune leur avantages-désavantages. Nous allons citer toutes les méthodes testées mais seulement développé les

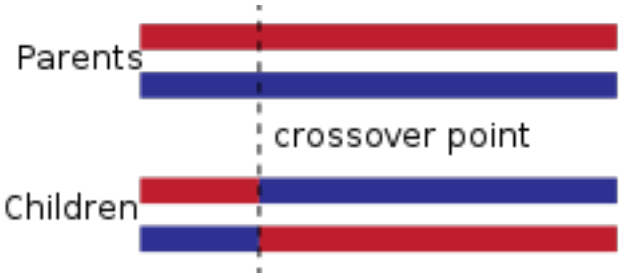
deux retenues.

Les méthodes testées :

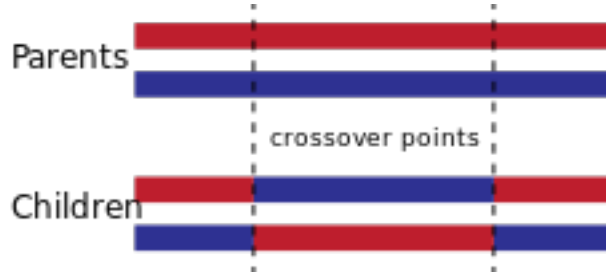
- Uniform cross-over - Cross-over caractères par caractères. - Arithmetic recombination

Les méthodes retenues :

One-Point Cross-over : Un point est choisi parmi la taille des parents. Et on échange les caractères à droite et à gauche du point pour former deux enfants composés de chacune des deux parties de leur parents.



Le multi-point cross-over : Même principe que le précédent même avec plusieurs points.



5 Mutations

Pour les méthodes de mutations elles ont évolué à force d'essayer et de se rendre compte des différents problèmes lorsque l'on tente de trouver la solution. Un des problèmes principaux d'un algorithme génétique est que l'algorithme ne peut pas identifier quand il faut sacrifier des bons résultats afin de pouvoir en avoir des mieux. Par exemple :

Pour une solution qui est : GTY5B L'algorithme peu préférer l'individu XGTY5C à l'individu GTY5 alors que le deuxième est plus proche. Dans ce cas précis l'algorithme ne peut pas identifier qu'il faut qu'il néglige leur fitness respectif pour mieux s'approcher de la solution. Dans ces différents cas que la mutation intervient.

Première mutation implémentée à était le random resetting dans laquelle on choisit un gène parmi l'individus et on le remplace par un caractère choisi aléatoirement dans le domaine. Cette méthode permet d'améliorer les résultats petit à petit en changeant les caractères et en ayant une chance de tomber sur un caractère nous rapprochant de la solution. Mais dans notre cas on peut se retrouver vite bloqué à cause la taille qui ne varie pas ou dans le cas ou deux caractères sont bons mais pas à la bonne position.

Les deux mutations implémentées pour palier à cela sont :

Swap$_mutation$: $Onchoisitdeuxgènesd'unindividuetleséchanges$

Expend$_r etract_m utation$: $Troiscasdiffférents$:

- Si l'individu n'est pas de la taille minimale : On retire un gène à une position aléatoire

- Si l'individu n'est pas de la taille maximale : On ajoute un caractère choisi aléatoirement parmi le domaine à une position aléatoire.

- On retire et puis ajoute un caractère à des positions aléatoires.

6 Hyper-parametres

Les hyper-parametres choisis sont les suivants :

- `init_population_size` : Le nombre d'individus à l'initialisation

- `steps` : Le nombre de génération

- `phenotype_domain` : Le domain de caracteres possibles .

- `GRADED_RETAIN_PERCENT` : Le pourcentage d'individus bien classés qu'on retient (Elitisme)

- `CHANCE_RETAIN_NONGRATED` : Pourcentage de garder des individus peu importe leur classement -¿ Apporte de la diversité génétique.

- `CHANCE_TO_MUTATE_CHAR` : Le pourcentage de chance de faire une mutation sur un gene (Le changer avec un caratere aléatoire)

- `CHANCE_TO_MUTATE_SIZE` : La chance de faire la mutation `expend_retract`

- `CHANCE_TO_SWAP` : La chance de faire la `swap_mutation` .

7 Observations

Nous avons observé et noté au fur à mesure le comportement de l'algorithme et les changements à effectués :

- Au début on atteignait un maximum local entre 0,80 et 0,90 on a donc décider de jouer sur les hyper-paramètres ce qui nous a permis de faire les observations suivantes :

• Élitisme : Un pourcentage de bon élément conservés trop grand baisse la diversité génétique dès le début et on peut se retrouver vite bloqué avec des individus qui sont les meilleurs sur le moment mais qui ne sont pas forcément proche de la solution. Trop petit et on se retrouve avec beaucoup d'éléments faibles et après cross-over la génération suivante baisse en performance. Le choix final à était 0.3 donc sur la population les 30 % meilleurs seront conservés.

• `CHANCE_RETAIN_NONGRATED` : Prendre un pourcentage d'individus parmi la génération permet d'apporter de la diversité génétique. Mais il faut qu'il soit petit et bien inférieur à celui de l'élitisme pour permettre d'assurer une bonne reproduction tous en permettant d'avoir un nombre réduit mais présent d'individus différents. On a choisi 0.05.

• Les hyper-paramètres sur les mutations : Les chances des diverses mutations influence grandement le processus. Pour chaque enfant on tire un nombre aléatoire entre 0 et 1 et on effectue la mutation si ce nombre est inférieur à la chance de mutation. Une valeur trop basse pour la chance va entrainer une stagnation car très peu de changement sera fait. Mais une valeur trop grande peu causé trop de mutations ce qui empêche de s'approcher de la solution lorsque l'on est très proche et que l'on a seulement besoin de changement sur un gène. Surtout pour la mutation sur la taille une fois qu'on a convergée sur une taille, changer la taille à chaque fois rend impossible de trouver la solution. Au contraire changer souvent un gène par un caractère aléatoire est bon et permet de tester des individus très proche des meilleurs individus en espérant avoir un individu meilleur. Mais dans un cas où on à de l'élitsime , pouvoir garder et reproduire les bons individus et ensuite faire beaucoup de mutations sur les enfants s'avère très efficace. Les paramètres retenus sont :

•`CHANCE_TO_MUTATE_CHAR` : 0.8

•`CHANCE_TO_MUTATE_SIZE` : 0.8

•`CHANCE_TO_SWAP` : 0.8

•Le choix sur les paramètres concernant la population et le nombre de génération a été simples : Le but pour nous était de pouvoir trouver la solution avec un nombres d'individus minimum. On s'est rendu compte que quand le nombre est trop bas il y a peu de diversité génétique et on arrive vite vers une convergence avec des éléments semblables et faibles. Quand il est trop haut il y a trop de diversité le temps de l'algorithme augmente et on ne trouve pas une solution plus rapidement. On arrive à trouver des résultats avec des centaines d'individus mais de nombreuses fois on reste bloqué à 0.97 car il manque un seul caractère et la probabilité de faire la mutation nécessaire sur le bon caractère à la bonne position avec le nombre d'enfants est bien trop faible. On a donc choisi 1000 individus. Et pour le nombre de génération 250 suffit mais dans la plupart des cas on trouve une solution

avant 200 voir avant 100 générations.

- Une des idées que nous avons essayées est de rajouter un paramètre de distance. Nous voulions faire en sorte d'avoir des individus très bon mais différent pour augmenter la diversité génétique. Une fonction distance supprimait les individus qui avait un certain nombre de gènes en commun. Le problème était que à partir d'un certains nombres de génération beaucoup d'individus sont censés se ressembler si on s'approche de la solution mais dans le cas où on supprime les individus qui se ressemble on se retrouve avec des individus très faible entre chaque génération ce qui nous empêche de nous améliorer. Il est préférable de faire des mutations sur des individus semblable et proche de la solution.

- On a essayé de lancer l'algorithme sur plusieurs générations et ensuite combiner les meilleurs éléments de chaque génération mais ça n'apporter pas de meilleurs résultats et c'était plus long.

- On a essayé de faire en sorte que à partir d'un certains nombres de générations il n'y es plus de cross-over pour permettre de plus petit changement entres individus lorsque on est très proche de la solution (Par exemple à 0.97 on a souvent seulement un seul caractère manquant). Mais ça n'apporter pas des résultats concluants.

8 Résultats

Sur 20 run de suite nous avons obtenu le mot de passe 19 fois. Soit un % de réussite de 0.95 pourcent. Le seul échec est dû à l'aléatoire et au fait que nous nous arrêtons à 250 générations seulement.

On a testé notre algorithme sur différents numéros d'étudiant et on obtient le mot de passe entre 80 et 200 générations la variation dans ce nombre dépendant seulement de l'aléatoire. En effet, on trouve rapidement une solution supérieure à 0.95 % dans laquelle un ou deux caractères manquent et seul une mutation du bon caractère peut nous apporter la solution. Par exemple s'il manque un Y on peut faire des changements aléatoires qui vont nous apporter le Y directement ou dans 100 générations. Dans très peu de cas des centaines de générations peuvent arriver sans trouver le résultat et finir avec des individus à un caractère près de la solution.

A chaque lancement de l'algorithme on sauvegarde les performances de chaque génération pour l'afficher dans un graphique à la fin de la dernière génération. Voici un exemple de graphique avec les paramètres suivants :

- `init_population_size` : 1000

- `steps` : 250

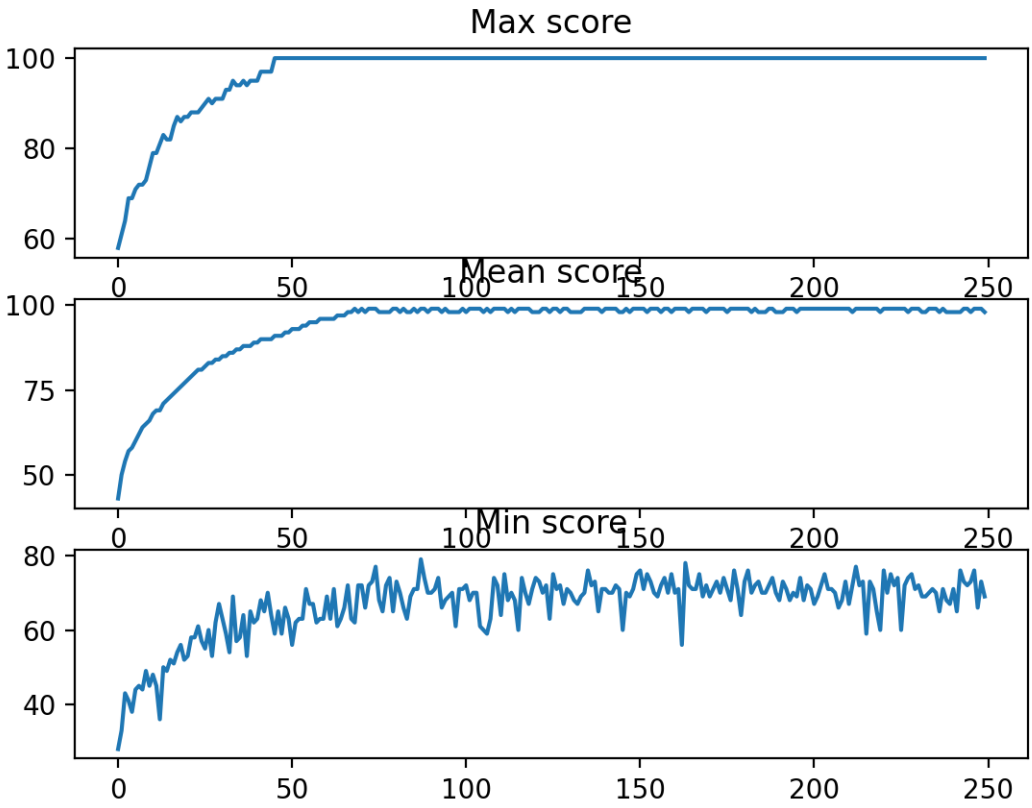
- `GRADED_RETAIN_PERCENT` : 0.3

- `CHANCE_RETAIN_NONGRATED` : 0.05

- `CHANCE_TO_MUTATE_CHAR` : 0.8

- `CHANCE_TO_MUTATE_SIZE` : 0.8

- `CHANCE_TO_SWAP` : 0.8



On observe que le résultat du meilleur élément augmente très vite dès les premières générations et atteint 100 (qui équivaut à 1) avant la 100 ième générations.

Pour la moyenne sur tous les individus c'est une évolution semblable qui montre que les meilleurs éléments sont bien conservés entre chaque génération et que les cross-over fournisse des individus de qualités permettant d'avoir une population très performante avec des individus qui plus on se rapproche de la solution plus ils se ressemblent à quelques mutations près.

Pour le score minimal on observe que même s'il augmente il varie beaucoup entre chaque génération. Cela est dû au fait que l'on introduit des individus de façon aléatoire donc en fonction des individus choisis cela peut-être des bons comme mauvais individus.