Université
جامعة ابن طفيل
Ibn Tofaïl
Faculté des Sciences

Programmation Génétique pour la régression symbolique

Mini-projet – Master Ingénierie des Systèmes Intelligents. Module : Métaheuristique – S1

Ce mini-projet explore l'application de la Programmation Génétique (GP) comme méthode de métaheuristique pour résoudre des problèmes de régression symbolique. L'objectif est de découvrir automatiquement des solutions sous forme d'expressions mathématiques qui approchent une fonction cible, démontrant ainsi l'efficacité de cette approche évolutive dans des problématiques d'optimisation complexes.

Réalisé par : Khadija JAMAI

Encadré par : Samir ANTER

Résumé

L'optimisation est un enjeu central dans de nombreux domaines scientifiques et industriels. Ce projet s'inscrit dans ce cadre et vise à résoudre un problème de régression symbolique à l'aide de la programmation génétique (Genetic Programming), une technique évolutive inspirée des mécanismes de l'évolution naturelle. L'objectif est de retrouver une expression mathématique approchant au mieux un ensemble de données générées à partir d'une fonction cible. L'algorithme mis en œuvre a permis de découvrir automatiquement des expressions proches de la fonction attendue, illustrant l'efficacité de cette approche pour explorer des espaces de solutions complexes.

Introduction

Dans le domaine de l'intelligence artificielle et de l'optimisation, de nombreux problèmes complexes ne peuvent être résolus efficacement par des méthodes classiques déterministes, en raison de la taille de l'espace de recherche ou de la non-linéarité des fonctions à optimiser.

Les méta-heuristiques, inspirées de phénomènes naturels, biologiques ou sociaux, offrent des alternatives puissantes. Elles permettent d'explorer l'espace des solutions de manière intelligente, souvent en combinant recherche globale et amélioration locale.

Ce travail s'inscrit dans le cadre d'un mini-projet visant à étudier un algorithme de méta-heuristique et à l'appliquer à un problème d'optimisation concret. L'algorithme étudié ici est la Programmation Génétique (Genetic Programming, GP), une approche évolutive dérivée des algorithmes génétiques, permettant de générer automatiquement des programmes ou des expressions mathématiques sous forme d'arbres syntaxiques.

Le problème choisi pour illustrer cette méthode est celui de la régression symbolique, qui consiste à retrouver automatiquement une fonction mathématique à partir d'un ensemble de points d'apprentissage. Ce type de problème est courant en machine Learning, en modélisation scientifique et en ingénierie.

L'objectif de ce projet est donc double : d'une part, comprendre et implémenter la Programmation Génétique, et d'autre part, analyser ses performances sur un cas pratique de régression symbolique.

1. Programmation Génétique : Concepts Théoriques

1.1 C'est la programmation Génétique

La **Programmation Génétique** (**GP**, pour *Genetic Programming*) est une technique d'optimisation inspirée du processus d'évolution biologique, introduite par **John Koza** dans les années 1990. Elle est considérée comme une extension des algorithmes génétiques (GA), mais au lieu d'optimiser des vecteurs de paramètres, elle évolue des structures de programmes, généralement représentées sous forme d'arbres syntaxiques.

Ces arbres représentent des formules ou programmes composés d'opérateurs (comme +, -, *, /, sin, log, etc.) et de terminaux (variables ou constantes numériques). La GP cherche ainsi à faire émerger automatiquement une solution sous forme d'expression mathématique ou de programme qui résout un problème donné.

1.2 Application de la programmation génétique

En plus de la résolution de problèmes de régression symbolique, la Programmation Génétique a montré son efficacité dans plusieurs domaines variés, où elle permet de découvrir des solutions innovantes et adaptées à des contextes complexes. Voici quelques exemples :

Robotique : Dans le domaine de la robotique, la PG est utilisée pour l'évolution de contrôleurs de robots. Par exemple, des robots peuvent être entraînés à réaliser des tâches complexes (comme la marche, la manipulation d'objets ou la navigation dans un environnement) en faisant évoluer des programmes qui contrôlent leur comportement. La PG permet ainsi de concevoir des algorithmes de contrôle robustes, souvent sans connaissance préalable précise des dynamiques du robot.

Finance : En finance, la Programmation Génétique est utilisée pour la création de modèles prédictifs et la modélisation de marchés financiers. Elle peut aider à générer des stratégies de trading automatiques ou à prévoir l'évolution de certains indices économiques. Grâce à sa capacité à explorer des espaces de solutions non linéaires, la PG peut trouver des modèles qui seraient difficiles à découvrir avec des méthodes classiques.

Biologie Computationnelle : Dans le domaine de la biologie computationnelle, la PG a été appliquée à la recherche de relations biologiques complexes, comme l'analyse de séquences génétiques ou la modélisation de réseaux de régulation génique. Elle est utilisée pour faire évoluer des algorithmes capables de simuler des processus biologiques ou d'identifier des structures cachées dans les données biologiques, facilitant ainsi des découvertes importantes.

Optimisation de conception : La Programmation Génétique est également utilisée dans l'optimisation de conception dans des domaines comme l'ingénierie aérospatiale ou la conception de produits industriels. Elle permet de générer des structures ou des designs qui répondent à des contraintes multiples, tout en explorant une grande variété de solutions possibles.

Évolution de logiciels : La PG est aussi utilisée dans le domaine du développement logiciel, pour générer automatiquement du code ou des programmes qui satisfont à certaines spécifications. Cela peut inclure des applications dans les domaines de la recherche en intelligence artificielle, où la génération automatique de code permet de concevoir des solutions innovantes à des problèmes de programmation complexes.

2. Domaine d'Application

2.1 Définition du problème

Dans le cadre de ce mini-projet, j'ai choisi d'appliquer la **programmation génétique** (GP) à un problème de **régression symbolique**. Ce type de problème appartient à la famille de l'apprentissage supervisé, mais contrairement aux méthodes classiques comme la régression linéaire ou polynomiale, la régression symbolique ne suppose aucune forme fonctionnelle prédéfinie. L'objectif est de découvrir automatiquement, à partir des données, une expression mathématique qui approxime au mieux la relation entre les variables d'entrée et de sortie.

Ce problème est bien adapté à la programmation génétique, car cette dernière permet de représenter des formules sous forme d'**arbres syntaxiques**, et d'explorer de manière évolutive un grand nombre de combinaisons possibles entre opérateurs (comme +, -, *, /), variables et constantes.

Pour cette expérimentation, j'ai décidé de travailler avec une fonction cible simple et bien connue, afin de faciliter l'évaluation de l'efficacité de l'algorithme. La fonction choisie est la suivante :

$$f(x) = x^2 + 2x + 1$$

Elle est intéressante car elle présente une structure polynomiale non triviale tout en restant accessible pour une première implémentation de GP.

L'objectif principal est donc de vérifier si, à partir de données générées avec cette fonction, l'algorithme de programmation génétique est capable de retrouver une expression mathématique équivalente ou très proche, sans connaître à l'avance la forme exacte de la fonction.
Cette approche permet notamment d'évaluer :

- La capacité de la GP à modéliser des relations non linéaires ;
- Son aptitude à explorer un espace symbolique complexe ;

- Sa faculté à fournir des solutions interprétables, ce qui est un avantage par rapport aux modèles boîte noire.

2 .2 Données utilisées

Pour générer les données d'entrée, j'ai utilisé des valeurs de xxx réparties uniformément dans l'intervalle [−10,10][-10, 10][−10,10], avec un pas de 0.5. La sortie yyy est calculée via la formule $\boldsymbol{f(x) = x^2 + 2x + 1}$.
Aucun bruit n'a été ajouté aux données afin de se concentrer uniquement sur la capacité de la programmation génétique à retrouver la forme exacte de la fonction.

3. Méthodologie

3.1 Description de l'algorithme de Programmation Génétique (GP)

L'algorithme de Programmation Génétique (GP) utilisé dans ce projet suit les étapes classiques d'un algorithme génétique, adaptées à la recherche de fonctions symboliques qui approchent une fonction cible en utilisant des arbres binaires. Les principales étapes de l'algorithme sont les suivantes :

Initialisation de la population : Génération aléatoire d'une population initiale d'arbres d'expressions mathématiques (individus), où chaque arbre représente une solution potentielle (une fonction symbolique).

Évaluation du fitness : Pour chaque individu de la population, on évalue sa capacité à approximer la fonction cible à l'aide d'un critère de fitness (par exemple, l'erreur quadratique entre la fonction calculée par l'arbre et la fonction réelle).

Sélection : Sélection des individus les plus adaptés pour la reproduction, en fonction de leur valeur de fitness (meilleurs fitness signifie plus de chances de reproduction).

Opérations génétiques :

- **Croisement** : Deux individus sélectionnés (parents) s'unissent pour produire un ou plusieurs descendants en échangeant une partie de leurs arbres respectifs.
- **Mutation** : Des modifications aléatoires sont effectuées sur certains individus pour introduire de la diversité et explorer de nouvelles solutions.

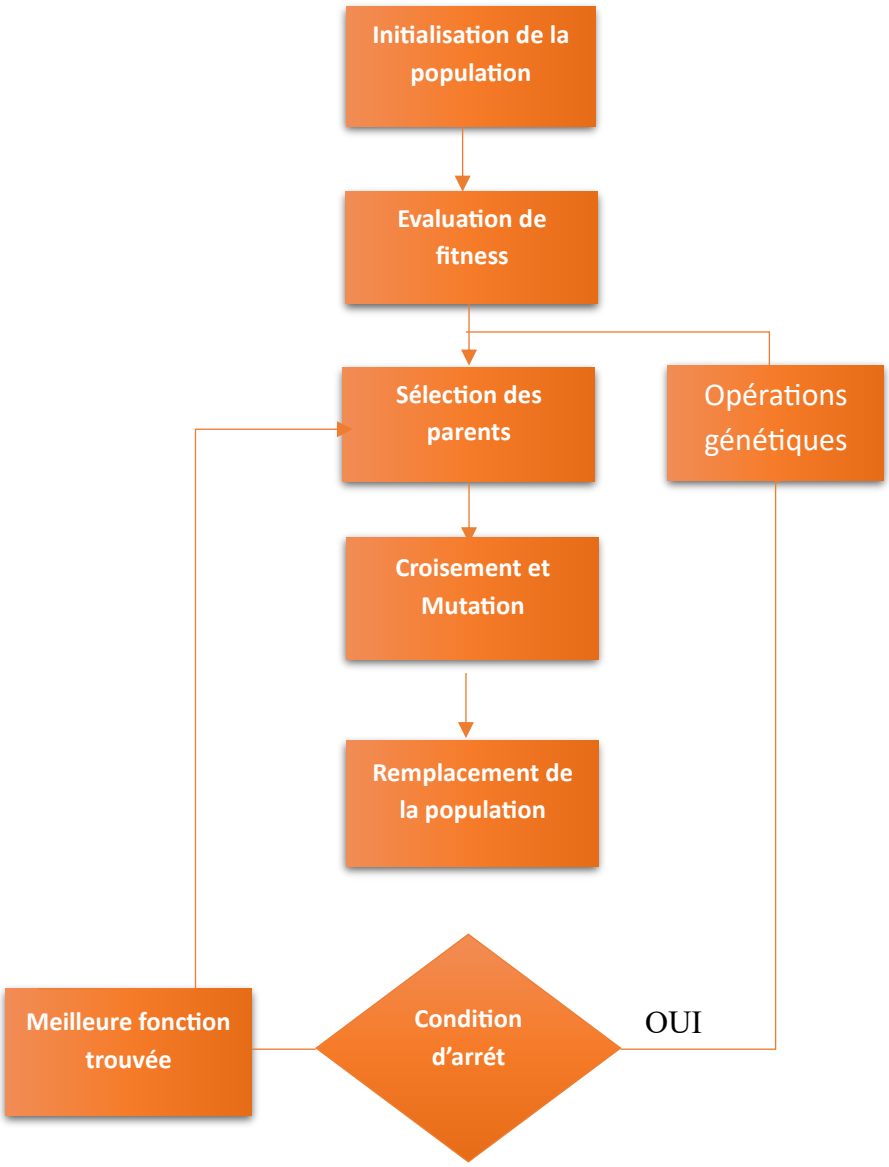
Remplacement : Les individus de la population sont remplacés par les nouveaux individus générés (enfants). La nouvelle génération entre dans la phase suivante de l'algorithme.

Vérification de la condition d'arrêt : L'algorithme se répète pour un nombre donné de générations ou jusqu'à ce qu'un critère de convergence soit atteint (par exemple, un seuil de fitness spécifié).

Explication des choix

1. **Sélection** : La sélection se fait par **tournoi** ou **sélection par roulette**. Les deux parents sont choisis à partir de la population actuelle en fonction de leur fitness. Plus un individu est adapté à la fonction cible, plus il a de chances d'être sélectionné.
    - **Tournoi** : On sélectionne un petit sous-ensemble d'individus au hasard et on choisit le meilleur de ce groupe pour la reproduction. Cela permet de préserver la diversité génétique tout en favorisant les bons individus.
    - **Roulette** : Chaque individu a une probabilité proportionnelle à sa fitness d'être sélectionné, ce qui favorise les individus plus adaptés, tout en laissant une chance aux autres de participer à la reproduction.
2. **Mutation** : La mutation consiste à apporter des modifications aléatoires dans un arbre de manière à explorer de nouvelles solutions.
    - **Types de mutation** : On peut modifier un sous-arbre entier, remplacer un opérateur par un autre (par exemple, remplacer une addition par une multiplication), ou remplacer une constante par une nouvelle valeur générée aléatoirement.
    - **Taux de mutation** : Le taux de mutation est généralement faible (par exemple, 1-5 %), car une mutation trop fréquente peut perturber l'évolution de la population.
3. **Croisement** : Le croisement (ou "crossover") consiste à échanger des sous-arbres entre deux individus (parents) pour produire des descendants.
    - **Type de croisement** : Un sous-arbre d'un parent est échangé avec un sous-arbre d'un autre parent à un point de croisement aléatoire.
    - **Taux de croisement** : Le taux de croisement est généralement plus élevé que celui de la mutation (par exemple, 60-90 %) pour favoriser la recombinaison de bonnes solutions.
4. **Paramètres de l'algorithme** :
    - **Taille de la population** : La taille de la population a un impact important sur la diversité génétique et la capacité de l'algorithme à explorer l'espace de recherche. Une taille typique de population varie entre 50 et 500 individus.
    - **Nombre de générations** : Le nombre de générations (généralement entre 100 et 500) définit la durée de l'algorithme. Plus le nombre de générations est élevé, plus l'algorithme peut explorer l'espace de recherche en profondeur.
    - **Profondeur maximale des arbres** : Limiter la profondeur des arbres permet de maintenir des solutions simples, ce qui aide à éviter le sur-apprentissage (overfitting) et améliore la généralisation.
    - **Critère d'arrêt** : L'algorithme peut s'arrêter après un nombre fixe de générations ou lorsque la meilleure solution atteint un seuil de fitness prédéfini.

Diagramme de fonctionnement de l'algorithme



3.2 Algorithme de base de la Programmation Génétique

Entrées :

- $f(x)$: Fonction cible à approximer
- P_0 : Population initiale d'arbres (programmes)
- G : Nombre maximal de générations
- ***Fitness*** : Fonction de coût (ex : erreur quadratique moyenne)
- ***Opérations génétiques*** : croisement, mutation
- ***Taille population*** : constante

1. Initialisation

- Générer aléatoirement une population initiale $\boldsymbol{P_0}$ d'arbres de programmes.
- Évaluer chaque individu avec la fonction de fitness.
- Identifier la meilleure solution $\mathbf{S^*}$ dans $\boldsymbol{P_0}$
- Fixer $G = 0$ (numéro de génération).

2. Boucle principale (jusqu'à convergence ou maximum atteint)

Tant que $\mathbf{G < G_{max}}$:

a. Évaluer tous les individus de la population P_G avec la fonction de fitness.
b. Sélectionner les meilleurs individus (par tournoi ou roulette) pour reproduction.
c. Appliquer les opérateurs génétiques (croisement et mutation) pour créer une nouvelle population P_{G+1}
d. Remplacer l'ancienne population par la nouvelle : $P_G \leftarrow P_{G+1}$
e. Mettre à jour la meilleure solution S^* si un individu meilleur est trouvé.
f. Incrémenter G←G+1

3. Fin de l'algorithme

Retourner la meilleure solution S^* c'est-à-dire l'arbre/programme qui approxime au mieux la fonction $\boldsymbol{f(x)}$.

4 Complexité de l'algorithme

Pour mieux comprendre les performances de l'algorithme de Programmation Génétique appliqué à la régression symbolique, j'ai analysé sa complexité à deux niveaux : temporelle et spatiale.

Complexité temporelle

Le temps que prend l'algorithme dépend principalement de trois éléments :

- Le nombre de générations (noté g) que j'ai fixé dans mes paramètres,
- La taille de la population (n) à chaque génération,
- La taille moyenne des arbres générés (h), qui représente en quelque sorte la complexité de chaque individu.

Chaque individu doit être évalué à chaque génération, et pour cela, l'algorithme doit parcourir l'ensemble de son arbre (fonction) pour estimer son erreur. En prenant tout cela en compte, on peut dire que la complexité temporelle est proportionnelle à :

$$\mathbf{O(g \times n \times h)}$$

Cela signifie que si je décide d'augmenter le nombre de générations, la taille de la population ou la complexité des fonctions générées, l'algorithme prendra plus de temps à s'exécuter. Il faut donc trouver un bon compromis entre précision et performance.

Complexité spatiale

En ce qui concerne la mémoire utilisée, elle dépend surtout du nombre total d'individus stockés (donc de la population) et de la taille de chaque arbre. En général, on peut estimer la complexité spatiale de cette façon :

$$O(n \times h)$$

5. Implémentation en Python

L'implémentation a été réalisée en Python, en utilisant principalement :

- Numpy pour la manipulation de données numériques,
- Random pour les opérations stochastiques (croisement, mutation, etc.),
- Matplotlib (facultatif) pour la visualisation des résultats. **Voir l'implémentation en Python**

Paramètre	Valeur utilisée	Remarques
Taille de la population	50	Taille réduite pour des raisons de performance.
Nombre de générations	30	Permet une convergence suffisante pour ce test.
Profondeur maximale des arbres	3	Limite la complexité des individus.
Taux de mutation	≈ 20 % (implémenté dans la fonction)	Favorise l'exploration de nouvelles solutions.
Méthode de sélection	Sélection par tournoi (top 10)	Simplicité et efficacité
Taux de croisement	100 % (avec sélection simple)	Approche simplifiée dans ce prototype.

6. Avantages et limites de PG

Avantages de la Programmation Génétique :

- Capacité à découvrir des solutions originales et complexes.
- Modèles interprétables (notamment en régression symbolique).
- Non dépendante d'une forme fonctionnelle prédéfinie.

Limites de la méthode :

- Très coûteuse en temps de calcul.
- Risque de surapprentissage ou de génération d'arbres trop profonds (phénomène de "bloat").
- Moins efficace que d'autres heuristiques sur certains types de problèmes numériques purs.

Différences avec les Algorithmes Génétiques (GA) classiques :

Critère	GA	GP
Représentation	Vecteurs de réels/binaires	Arbres syntaxiques
Objectif	Optimiser un ensemble de paramètres	Générer un programme ou une expression
Complexité des solutions	Limitées à la taille du vecteur	Potentiellement très variées

Conclusion générale

Ce projet a permis de mettre en œuvre une approche de régression symbolique à l'aide de la programmation génétique pour approximer une fonction cible. L'utilisation d'arbres syntaxiques pour représenter les individus, associée à des opérateurs génétiques tels que le croisement et la mutation, a démontré l'efficacité de cette méthode dans la génération automatique d'expressions mathématiques proches de la solution attendue.

Malgré la simplicité des choix techniques (mutations et croisements élémentaires, sélection par tournoi), les résultats obtenus sont satisfaisants et illustrent la capacité de la programmation génétique à résoudre ce type de problèmes.

Cette implémentation constitue une base solide pour d'éventuelles améliorations futures, que ce soit en complexifiant les opérateurs, en affinant les stratégies de sélection, ou en intégrant des mécanismes de régulation de la complexité des arbres pour favoriser la généralisation.

En résumé, la programmation génétique s'avère être une méthode flexible et prometteuse pour l'approximation de fonctions, avec de nombreuses perspectives d'optimisation et d'extension.