REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : IVA 1/M2/2019

Mémoire

présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **IVA**

Résoudre le problème de régression symbolique par la programmation génétique.

Par :

ACHACH MOHAMMED ELGHAZALI

Soutenu le 6 juillet 2019, devant le jury composé de :

DJEDI NOUREDDINE	Pr	Président
ABABSA TAREK	MAA	Rapporteur
BEN CHABAN MOUFIDA	MAA	Examinateur

Remerciements

Le messager d'Allah la paix sait sur lui

Celui qui ne remercie pas les gens ne remercie pas dieu ». Louange à dieu,qui nous aider à compléter cette mémoire. Un grand merci à maman .

A mes frères taha yacine ,belegacem,rayan.

*Nous remercie mes encadreur '**Ababsa Tarek**'.*

Un grand merci à tous ceux qui de prés ou de loin nous ont aidés dans l' élaboration de notre mémoire.

Dédicace

Je dédie cet ouvrage

Je remercie le dieu tout puissant qui m'a guide pour accomplir ce travail. A ma maman et parent qui m'a soutenu et encouragé durant ces années d'études. Qu'elle trouve ici le témoignage de ma profonde reconnaissance.

A mes frères et ceux qui on partagé avec moi tous les moments d'émotion lors de la réalisation de ce travail, ils m'ont chaleureusement supporté et encouragé tout au long de mon parcours.

A ma famille achach , mes proches et à ceux qui me donnent de l'amour et de la vivacité.

A tous mes amis qui m'ont toujours encouragé, et à qui souhaite plus de succès.

A tous ceux que j'aime.

Merci

Table des matières

Table des figures

Introduction générale

Au cours du dernier siècle, la technologie a évolué de manière extrêmement rapide, aussi la complexité des problèmes grandisse, et ça nécessite des grandes ressources matérielles et logicielles pour les résoudre. On citons quelques problèmes tell que : le traitement d'image, le contrôle de systèmes industriels, résoudres un problème de régression symbolique avec la programmation génétique , et ce dernier qui nous intéresse. Pour résoudre ces grands problèmes, nous avons besoin à des algorithmes d'optimisation telle que les algorithmes évolutionnaires.

Dans ce travaille à partir d'un point données {(x,f(x)),. . . . . ,(xn,f (xn))} en cherche la fonction approximer (fonction fitness) par la programmation génétique , cette fonction Sous forme symbolique.

Pour résoudre ce problème nous choisissons la programmation génétique, c'est une partie des algorithmes évolutionnaires. Parce que cette méthode s'inspirant de la théorie de l'évolution, elle utilise des mécanismes de la sélection naturelle. Elle permet de création des chromosomes, et puis avec l'application des opérations génétiques « reproduction des nouveaux programmes » et avec la croisement et la mutation.

Ce travail est constitué des 4 partie suivants :

Chapitre I : Il présente des généralités sur les algorithmes évolutionnaires, l'historique, la famille de ces algorithmes et les domaines d'application. Aussi le mode de fonctionnement, les étapes préparatoires et en fin les phases de la programmation génétique.

Chapitre II : Dans ce chapitre on va faire un vu sur la programmation génétique linéaire (définition, Les atomes de la PGL, et donne un programmes LGP, Exactitude sémantique des programmes LGP, (Évaluation de chromosomes LGP), codage de LGP, Opérateurs évolutifs chez LGP), en suite la problème de régression symbolique.

Chapitre III : dans ce chapitre on va détailler techniquement la problématique et la solution proposée, ainsi les structures de données utilisées, les classe déclarer avec ces différentes méthodes, et comment calculer la fitness.

Chapitre IV : dans ce chapitre on va parler sur le langage de programmation et le logiciel utilisé, en suite on va analyser les résultats obtenus sur le programme génétique.

Enfin, on termine notre travail par une conclusion générale.

Chapitre 1

Programmation génétique

1 Introduction

Grâce aux phénomènes physiques ou biologiques de nombreux algorithmes s'inspirant. Ainsi les réseaux de neurones artificiels s'inspirent du fonctionnement du cerveau humain, et les algorithmes évolutionnaires sont inspirés du concept de sélection naturelle élaboré par Charles Darwin.

2 Les algorithmes évolutionnaires

2.1 Définition

Les algorithmes évolutionnistes sont une famille d'algorithmes dits bio-inspirés, car s'inspirant de la théorie de l'évolution pour résoudre des problèmes divers qui sont pour le moment hors d'atteinte des méthodes déterministes plus classiques. Ils font ainsi évoluer un ensemble de solutions à un problème donné, dans l'optique de trouver les meilleurs résultats.

Ce sont des algorithmes stochastiques d'optimisation globale, car ils utilisent itérativement des processus aléatoires. Ces algorithmes sont définis sur des espaces de recherche non-standard 'listes, graphes... ', voir le cycle de base d'un algorithme évolutionnaire dans la figure 1.1. [1]

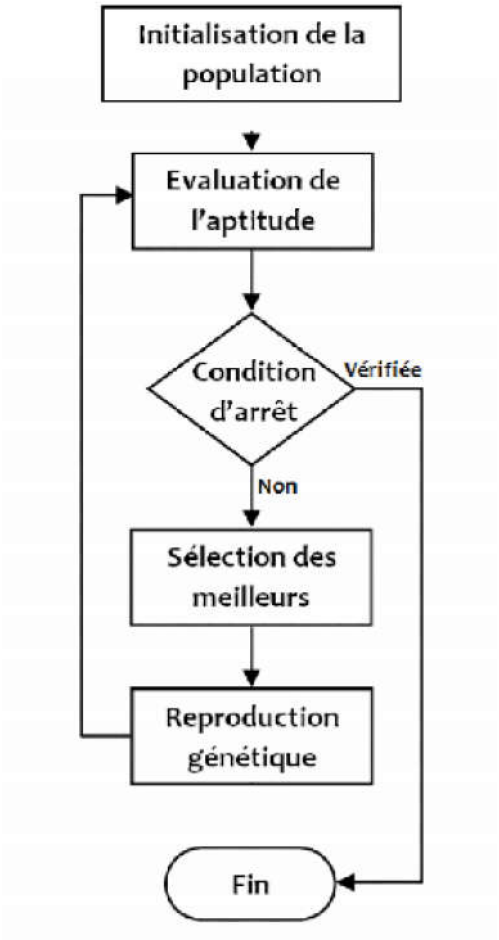


FIGURE 1.1: Cycle de base d'un algorithme évolutionnaire.

2.2 Historique

— 1952 : premiers travaux sur l'utilisation de méthodes stochastiques pour l'optimisation.

— 1954 : 'Barricelli' effectue les premières simulations du processus d'évolution et les utilise sur des problèmes d'optimisation généraux.

— 1965 : 'Rechenberg' conçoit le premier algorithme utilisant des stratégies d'évolution.

— 1966 : 'Fogel', Owens et Walsh proposent la programmation évolutionnaire.

— 1970 : 'John Horton Conway' conçoit le jeu de la vie, l'automate cellulaire le plus connu à ce jour.

— 1975 : travaillant sur les automates cellulaires, 'Holland' propose les premiers algorithmes génétiques.

— 1980 : 'Smith' utilise la programmation génétique.

— 1986 : 'Farmer', 'Packard' et 'Perelson' travaillent sur les systèmes immunitaire artificiels.

— 1988 : la première conférence sur les algorithmes génétiques est organisée à l'université de l'Illinois à Urbana-Champaign.

— 1988 : 'Koza' dépose son premier brevet sur la programmation génétique.

— 1989 : 'Goldberg' publie un des livres les plus connus sur les algorithmes génétiques.

— 1989 : « Evolver », le premier logiciel d'optimisation par algorithmes génétiques est publié par la société 'Axcelis'.

— 1989 : le terme algorithme mémétique apparait.

— 1993 : le terme « calcul évolutionnaire » se répand, avec la parution de la revue éponyme, publiée par le Massachusetts Institute of Technology.

— 1996 : 'Mühlenbein' et 'Paaß' proposent les algorithmes à estimation de distribution.

— 1997 : 'Storn' et 'Price' proposent un algorithme à évolution différentielle.

— 2000 : premiers algorithmes génétiques interactifs. [2]

2.3 L'inspiration darwinienne

Selon la génétique et la théorie de l'évolution

— La constitution génétique d'un enfant est le résultat d'une combinaison entre les gènes des parents.

— Les enfants ne sont pas identiques car des altérations des gènes peuvent se produire (mutations).

— Parmi les mutations, certaines peuvent être favorables et d'autres défavorables.

— Il naît beaucoup de descendants : mais seuls les individus les mieux adaptés pourront survivre et transmettre leurs gènes à leur descendance.

Dans ce modèle, on observe que

— Le hasard joue un rôle de générateur de nouveaux individus différents de leurs parents.

— La sélection naturelle effectue le tri entre les variations favorables et les autres. [3]

2.4 Les principales familles des algorithmes évolutionnaires

2.4.1 Algorithmes génétiques (GA)

Les algorithmes génétiques sont les plus populaires parmi les algorithmes évolutionnaires. Ils ont été imaginés comme outils de modélisation de l'adaptation. Les GA travaillent dans l'espace des chaînes de bits [0,1] (inspirée des chaînes d'ADN. Ils sont utilisés dans le but de découvrir une solution, généralement numérique, résolvant un problème donné. [4][5].

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle.

Les Algorithmes Génétiques sont dans la famille des algorithmes métaheuristiques dont le but est d'obtenir une solution convenable dans un temps acceptable.

2.4.2 Stratégies d'évolution (ES)

« Inventées par I. Rechenberg et H.P. Schwefel, 1965, Berlin »

Les stratégies s'évolution ont été mises au point par deux jeunes ingénieurs travaillant sur des problèmes numériques (l'optimisation paramétrique). Les ES représentent les individus comme un ensemble de caractéristiques de la solution potentielle. En général, cet ensemble prend la forme d'un nombres réels. Les stratégies dévolution s'appliquent à une population de parents à partir de laquelle des individus sont sélectionnés aléatoirement afin de générer une population de descendants. Ceux-ci sont ensuite modifiés par des mutations qui consistent à ajouter une valeur générée aléatoirement selon une fonction de densité de probabilité paramétrée.

2.4.3 Programmation évolutionnaire

« Imaginée par L.J. Fogel et ses coauteurs dans les années 60 également, et reprise par son fils D.B. Fogel dans les années 90, en Californie, USA »

La programmation évolutionnaire a été initialement conçue dans le but de faire évoluer des machines à états finis, ils n'utilisaient que des opérateurs de mutation et de remplacement pas d'opérateur de croisement. Pour effectuer de la PE, une population de solutions potentielles au problème est d'abord générée aléatoirement. Chaque individu de la population produit un ensemble de descendants résultant de mutations. Une opération de sélection naturelle est alors appliquée afin de former une nouvelle population. Le processus de mutation-sélection est répété itérativement jusqu'à ce qu'une solution acceptable soit trouvée.

2.5 Les schémas d'évolution (moteurs d'évolution)

On regroupe sous ce nom les ensemble sélection/remplacement, Un schéma d'évolution est donc la réunion d'une procédure de sélection et d'une procédure de remplacement.

2.5.1 Schéma algorithme génétique générationnel (GGA)

Ce schéma utilise une sélection stochastique (roulette ou tournoi), pour sélectionner exactement P parents. Ces P parents donnent ensuite P enfants par application des opérateurs génétiques (avec une probabilité). Enfin, ces P enfants remplacent purement et simplement les P parents pour la génération suivante. [4][3]

2.5.2 Schéma algorithme génétique stationnaire (Steady-state GA – SSGA)

Dans ce schéma, un individu est sélectionné, généralement par tournoi, et chaque parent donne un seul enfant d'après le croisement et la mutation, cet enfant est réinséré dans la population en remplacement d'un "vieux" parent sélectionné par un tournoi inversé (le moins performant "gagne"). [4]

2.5.3 Schémas stratégies d'évolution (ES)

Deux schémas sont regroupés sous ce titre. Dans les deux cas, l'étape de sélection est un tirage uniforme. À partir d'une population de taille 'P', 'E' enfants sont générés par application des opérateurs génétiques et l'étape de remplacement est totalement déterministe.

Dans le 1er schéma « (P ,E)-ES » : les meilleurs 'E' enfants deviennent les parents de la génération suivante.

Dans le 2éme schéma « (P +E)-ES » : les meilleurs des P + E (Parents + Enfants) sont les parents de la génération suivante. [4][3]

2.6 Les domaines d'application

Les applications des AEs sont multiples : optimisation de fonctions numériques difficiles (discontinues, multimodales, bruitées...), traitement d'image (alignement de photos satellites, reconnaissance de suspects...), optimisation d'emplois du temps, optimisation de design, contrôle de systèmes industriels, apprentissage des réseaux de neurones, etc. Les AEs peuvent être utilisés

pour contrôler un système évoluant dans le temps (chaîne de production, centrale nucléaire...) car la population peut s'adapter à des conditions changeantes.

Les AG sont également utilisés pour optimiser des réseaux (câbles, fibres optiques, mais aussi eau, gaz...), des circuits VLSI, des antennes... Ils peuvent être utilisés pour trouver les paramètres d'un modèle petit-signal à partir des mesures expérimentales. [6].

3 l'objectif de travaille

Notre travail se situe dans le cadre de l'implémentation d'une plateforme pour un moteur de programmation génétique. Pour cela nous avons considéré le problème suivant :

On s'intéresse au problème de la régression symbolique. Le problème consiste à trouver la relation entre des ''inputs" et des ''outputs" en appliquant une sorte d'apprentissage supervisé pour trouver la formule ''rapprochée" qui maitrise le mieux cette relation.

4 Programmation génétique (GP)

« Amenée à maturité par J. Koza, en Californie, USA » La programmation génétique apparue initialement comme sous-domaine des GAs, c'est un paradigme permettant la programmation automatique d'ordinateurs par des opérations de variation génétique et des opérations de sélection.

La différence entre la programmation génétique et les algorithmes génétique réside essentiellement dans la représentation des individus. En effet, la programmation génétique consiste à faire évoluer des individus dont la structure est similaire à des programmes informatiques.

La programmation génétique représente les individus sous forme d'arbres, c'est-à-dire des graphes orientés et sans cycle, dans lesquels chaque nœud est associé à une opération élémentaire relative au domaine du problème. Plusieurs autres représentations comme des programmes linéaires et des graphes cycliques, ont été utilisées depuis. [4][5].

4.1 Définition

La programmation génétique (GP, pour Genetic Programming en anglais) est une méthode inspirée par la théorie de l'Evolution telle qu'elle a été définie par Darwin et notamment ses mécanismes biologiques. Elle fait partie des algorithmes évolutionnaires.

Une technique permettant à un programme informatique d'apprendre, par un algorithme évolutionniste, à optimiser peu à peu une population d'autres programmes pour augmenter leur degré d'adaptation (fitness) à réaliser une tâche demandée par un utilisateur.

— Génération automatique de programmes (J. Koza)

— Génération automatique de comportements représentés par des programmes exécutables (P. Angeline).

Pourquoi une programmation génétique :

— Utilise pour obtenir assurément une solution optimale exacte.

— Espace de recherche important(minimiser l'espace de recherche).

La programmation génétique peut être vue comme l'évolution artificielle des "programmes", elle a pour but de trouver des programmes qui répondent au mieux à une tache définie. Pour ce faire, elle permet à la machine d'apprendre, en utilisant un algorithme évolutionniste afin d'optimiser la population de programmes, ces programmes étant représentés sous forme d'arbres, voir la figure 1.2. [1][7][8]

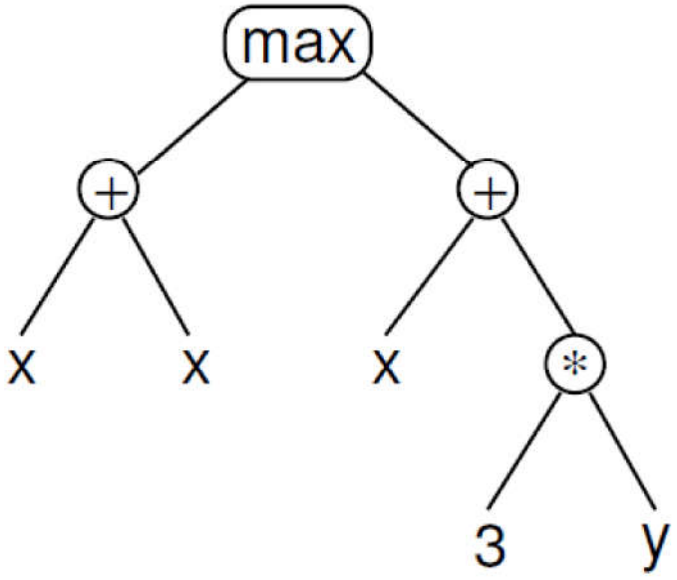


FIGURE 1.2: Syntaxe du programme max(x+x , x+3*y)

4.2 Historique

— 1958 – Friedberg : Mutation aléatoire d'instruction dans un programme génétique.

— 1963 – Samuel : Utilisation du terme « machine learning » dans le sens de programmation automatique.

— 1966 – Fogel, Owen Walsh : Automates à états finis pour des tâches de prédiction, obtenus par sélection de parents efficaces auxquels on applique des mutations : « evolutionary programming ».

— 1985 – Cramer : Utilisation d'expression sous forme d'arbre. Cross-over entre sous-arbres.

— 1986 – Hicklin : Évolution de programmes de jeux en LISP. Sélection des parents efficaces, combinaisons des sous-arbres communs ou présents dans un des parents et de sous-arbres aléatoires.

— 1989/1992 – Koza : Systématisation et démonstration de l'intérêt de cette approche pour de nombreux problèmes. Définitions d'un paradigme standard dans le livre « Genetic programming. On the programming of computers by means of natural selection » [?, ?]. Ce paradigme inclut plusieurs concepts : programmation structurée en expression arborescentes, définition d'une grammaire de langage, type de retour unique pour chaque fonction, définition des proportions de mutation et de cross-over pour chaque génération, etc.

4.3 Principe

— La programmation génétique travaille directement sur un codage de la solution du problème

— Les chromosomes deviennent alors une composition hiérarchique d'opérateurs et d'arguments codant le comportement des individus.

— Il y a le même type d'opérateurs sur cette hiérarchie que sur les chaînes de bits, mais de nouveaux opérateurs peuvent être ajoutés aux opérateurs de **sélection**, **croisement**,**mutation**.

4.4 Principe de base

— L'algorithme consiste à faire évoluer une population constituée d'un grand nombre de programmes (100 à 500).

— Au départ, la population est constituée de programmes créés aléatoirement. Chaque programme est évalué selon une méthode propre au problème posé.

— A chaque itération (génération) on classe les programmes en fonction des notes qu'ils ont obtenues, et on crée une nouvelle population, où les meilleurs programmes auront une plus grande chance de survivre ou d'avoir des enfants que les autres. Ce principe est

le même qu'en algorithmique génétique classique, mais les opérateurs de croisement et de mutation sont un peu différents. En effet, ils travaillent directement sur la structure d'arbre du programme.

— L'espace de recherche de l'algorithme génétique sera ainsi l'ensemble de tous les programmes pouvant être ainsi définis

— Le comportement des individus étant contrôlé par le programme résultant de l'évaluation de l'arbre

— L'individu étudié dans le cadre de la programmation génétique est un programme

— L'évaluation d'un individu doit tenir compte de la taille du programme, son temps d'exécution, son efficacité

— La taille du programme peut être connue simplement en observant le chromosome.

— Pour juger de l'efficacité, comme avec les algorithmes génétiques classiques, une fonction donne une note permettant de classer cet individu par rapport aux autres.

4.5 Les étapes préparatoires de la programmation génétique

Il existe six étapes préliminaires pour résoudre un problème en utilisant la programmation génétique. Ces étapes sont : 1- le choix des terminaux, 2- le choix des fonctions, 3- la fonction de performance, 4- les paramètres du contrôle, 5- le critère de terminaison et en fin déterminer la structure du programme cette étape est optionnel.

* **Les terminaux fonctions :** Sont les composants d'un programme informatique, ou les terminaux sont les nœuds externe (les feuille) dans l'arbre, et les fonctions sont les nœuds interne. Par exemple dans l'expression « (x+y)*z » les terminaux sont (x,y,z) et les fonction sont (+,*).

* **La fonction de performance :** dans cette étape on va déterminer la fonction d'adaptation (la fonction objectif pour calculer la fitness de chaque programme), la détermination de cette fonction dépend à la nature du problème.

* **Les paramètres du contrôle :** Il existe plusieurs paramètres du contrôle mais le plus important c'est déterminé le nombre de la population initiale.

* **Le critère de terminaison :** Simplement c'est la condition d'arrêt du programme génétique, déterminer le nombre de générations ou bien le

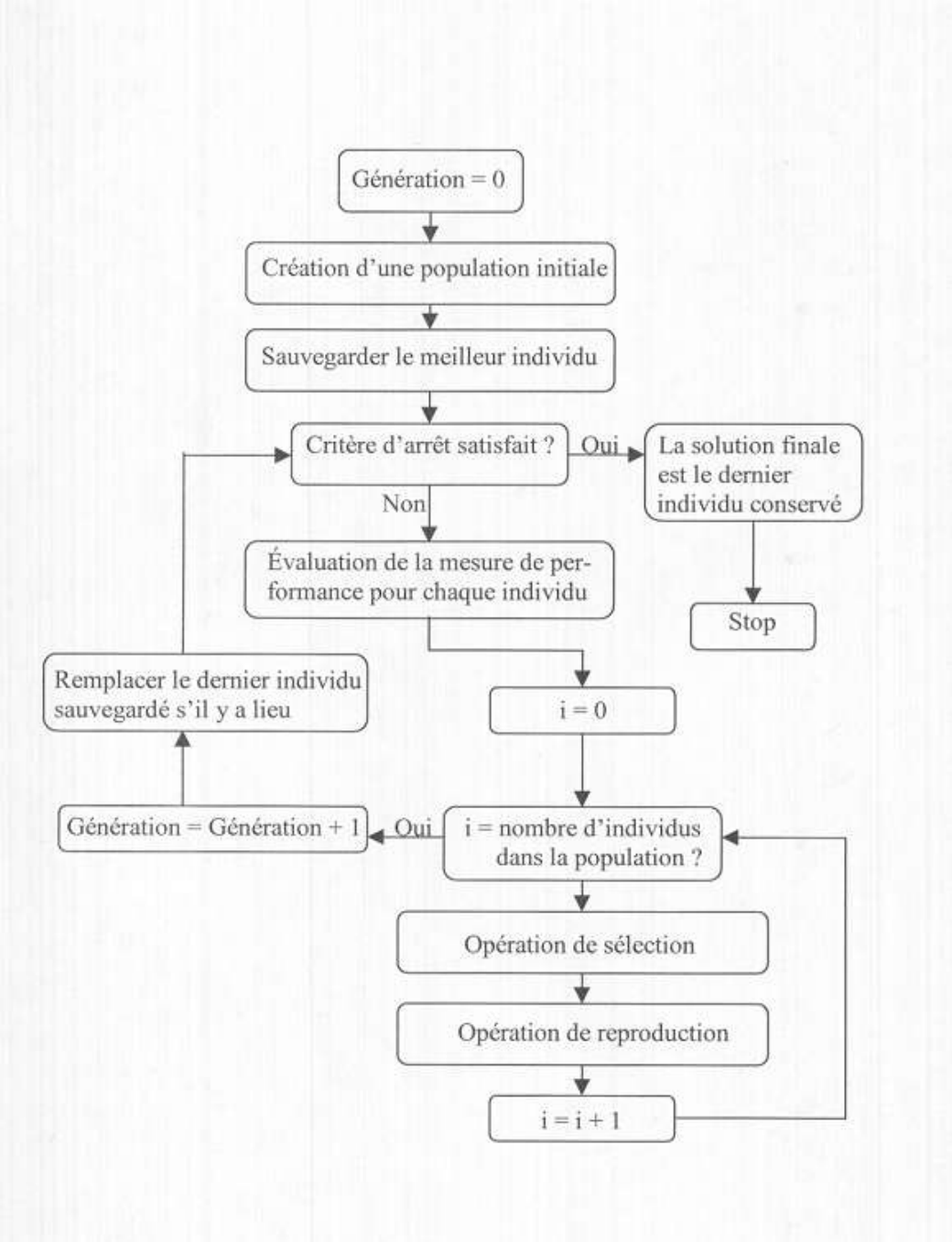


FIGURE 1.3: schéma qui représente les étapes de programmation génétique.

4.6 Terminologie et notations

— La fonction objectif est appelée 'fonction de performance', ou 'fonction d'adaptation' (fitness en anglais).

— Les points de l'espace de recherche sont appelés des 'individus'.

— Les tuples d'individus sont appelés des 'populations'.

— On parlera d'une 'génération' pour la boucle principale de l'algorithme. [1]

4.7 Les phases de la programmation génétique

1. Génération aléatoire de la population initiale (1 individu = 1 programme)
2. Évaluation du fitness de chacun des individus de la population (évaluation de l'adaptation des programmes au problème à résoudre) avec la fonction d'adaptation.
3. Sélection des individus les mieux adaptés à leur environnement au sens de la fonction d'adaptation.
4. Application des opérateurs de croisement, mutation sur la population afin de créer une nouvelle population.
5. Répéter les étapes 2, 3 et 4 un certain nombre de fois, le processus d'évolution s'arrête quand le niveau de performance souhaité est atteint, ou qu'un nombre fixé de générations s'est écoulé sans améliorer l'individu le plus performant.

4.7.1 Schéma représente les phases de programmation génétique

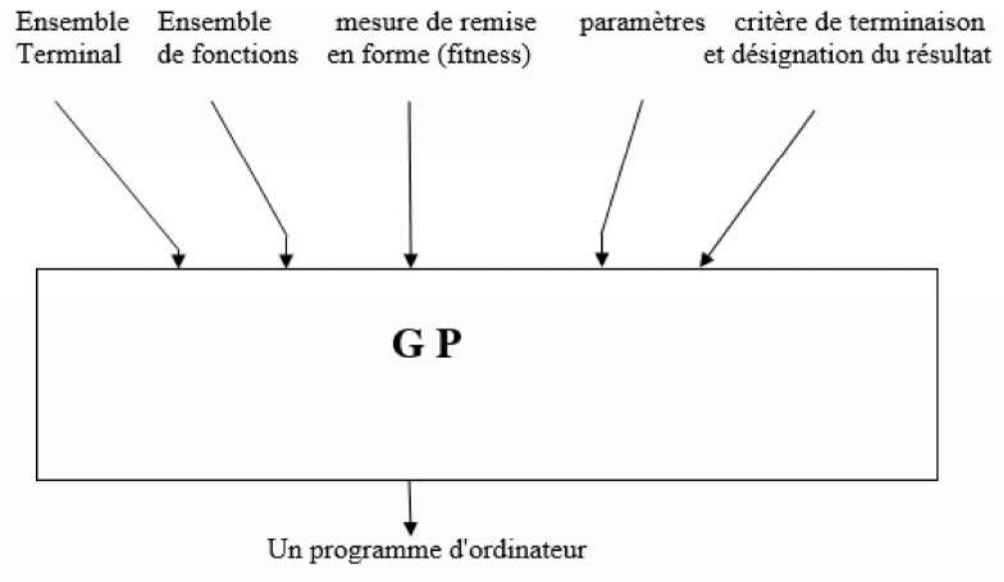


FIGURE 1.4: les cinq grandes étapes préparatoires de la PG.

4.8 Création de la population initiale

La première étape de la programmation génétique est la création d'une population initiale. Cette population est généralement générée d'une manière aléatoire. Il existe trois principaux

types de génération aléatoire des programmes lors de l'initialisation : la méthode croissante (grow), la méthode complète (full) et la méthode « ramped half half ». Les trois méthodes ont une taille maximale de profondeur à ne pas franchir que l'on doit spécifier à l'avance.

Ensuite, un certain nombre d'individus doit être généré. Dans tous les cas, la génération d'un individu se déroulera selon un algorithme simple :

1. Une fonction f est choisie parmi l'ensemble des fonctions, elle sera la racine de l'arbre.
2. Pour chaque fonction, chacun de ses arguments est choisi :
   - Soit parmi l'ensemble des fonctions, auquel cas, on recommence la boucle avec ses arguments.
   - Soit parmi l'ensemble des symboles terminaux. Dans ce cas, c'est une feuille de l'arbre, il n'y a pas à continuer pour cette branche.

* **La méthode croissante :**

A chaque niveau, le prochain symbole est pris aléatoirement soit dans les symboles terminaux, soit dans les fonctions (sauf au niveau de profondeur maximum, où seuls des symboles terminaux sont autorisés). Cette méthode créer des arbres de taille irrégulière et avec des formes différentes, voir la figure 1.5. [7]

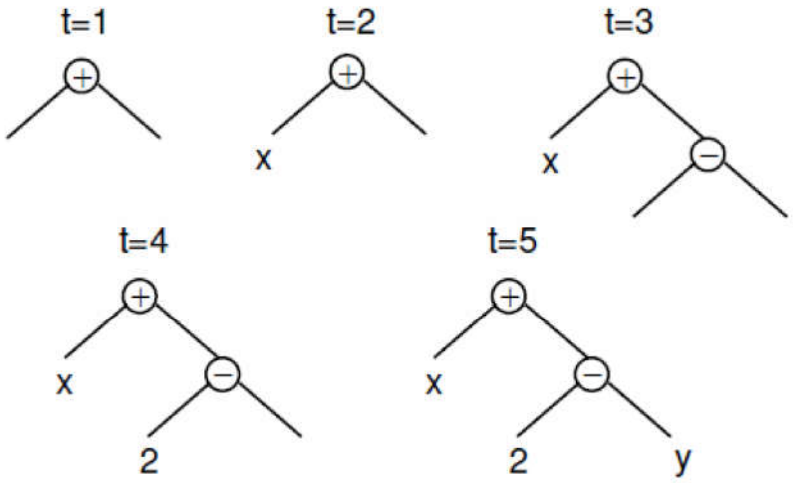


FIGURE 1.5: La création d'un arbre de profondeur 2, avec l'algorithme grow

* **La méthode complète :**

Les symboles sont tous choisis dans l'ensemble des fonctions, sauf au niveau de profondeur maximal. Ainsi, tous les individus sont équilibrés, complets et ont la même taille, correspondant à la taille maximale autorisée, voir la figure 1.6. [7]

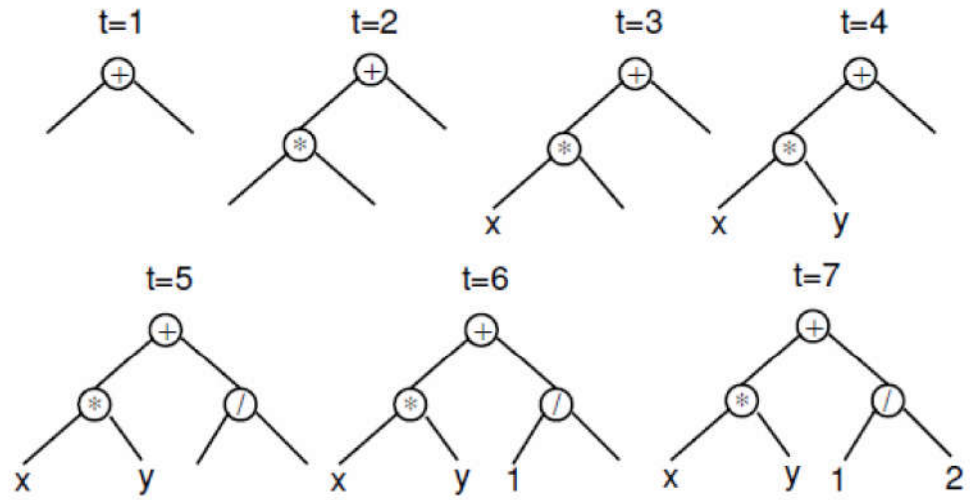


FIGURE 1.6: La Création d'un arbre plein avec la méthode full.

* **La méthode « ramped half half » :**
  Cette méthode est une combinaison des deux méthodes « grow » et « full ». [9]

4.9 Fitness

Lorsqu'une solution est générée, il faut évaluer sa qualité (sa fitness). La façon de la déterminer dépond du problème. Ainsi, pour un problème de comparaison d 'images le nombre de pixels semblables est le variable de la fonction de performance, dans un problème de contrôle d 'un robot le nombre de chocs contre les murs ou le temps totale pour faire la tâche et le variable de la fonction de performance ... etc. Il existe plusieurs représentations de la fonction d'évaluation :

* **Fitness standardisé** Cette mesure est obtenue en modifiant la mesure de performance brute de façon à ce que la plus petite valeur numérique soit toujours la meilleure valeur. Dans un problème de minimisation, la mesure de performance standardisées est égale à la mesure de performance brute, c'est-à-dire s (i , t) = r (i ,t).l'orsqu'il S'agit plutôt d'un problème de maximisation, la mesure de performance standardisée équivaut à la mesure de performance brute maximale moins la valeur courante, c'est-à-dire s (i , t) = r max - r (i , t). La meilleure valeur possible est 0 et toutes les valeurs sont positives.
* **Fitness normalisé** Cette mesure est calculée de la façon suivante :$n(i,t) = \frac{a(i,t)}{\sum a(k,t)}$ Ou k varie entre 1 et la taille de la population. Les avantages de cette approche sont que la mesure de performance se situe toujours entre 0 et 1, la somme de toutes les mesures de

performance est égale à 1 et la valeur est proche de 1,meilleur est l'individu. Similaire à la représentation standardisée.

* **Fitness ajusté**

  La mesure de performance ajustée est calculée de façon suivante : $a(i,t) = \frac{1}{(1+s(i,t))}$. La valeur se situe toujours entre 0et 1 . L'avantage d'utiliser cette mesure est qu'elle accentue les petites différences lorsque la mesure de performance standardisée se rapproche de 0. Ainsi, cela permet de mieux percevoir les différences entre les meilleurs individus .

  Dans le cas ou la mesure de performance standardisée ne pourrait être calculée ,il est possible d'utiliser la mesure de performance brute pour le calcul de la mesure de performance ajustée.

  Similaire à la représentation normalisée mais où le meilleur score possible c'est 1. [9]

4.10 Les opérateurs

4.10.1 L'opérateur de sélection

On distingue deux catégories de sélection : la sélection déterministe et la sélection stochastique.

* **La sélection déterministe** On sélectionne les meilleurs individus (au sens de la fonction performance), mais cela pose un problème de temps de calcul dans les grosses tailles de population. Les individus les moins performants sont totalement éliminés de la population, et le meilleur individu est toujours sélectionné. [1]

* **La sélection Stochastique** Il s'agit toujours de favoriser les meilleurs individus, ce qui laisse une chance aux individus moins performants. Il peut arriver que le meilleur individu ne soit pas sélectionné, et qu'aucun des enfants n'atteigne une performance aussi bonne que celle du meilleur parent. [1]

* **Les différentes méthodes de sélection** On distingue plusieurs méthodes de sélection : Sélection par rang, Sélection « steady-state », Elitisme. Mais les trois les plus utilisées sont la sélection par roulette, la sélection par tournoi et la sélection uniforme.

  — Sélection par tournoi : Cette méthode n'utilise que des comparaisons entre les individus par rapport au résultat de la fonction d'évaluation pour chaque individu et ne nécessite même pas de tri de la population. Elle possède une taille du tournoi K

(K >= 2). Pour sélectionner un individu, on tire K individu uniformément dans la population, et on sélectionne le meilleur de ces K individus.

— Sélection par roulette : Il s'agit de la méthode la plus courante. Les individus parents sont sélectionnés proportionnellement à leur performance. L'individu le plus performent à une grande probabilité d'être sélectionné. Le nombre de fois qu'un individu sera sélectionné est égal à son évaluation divisée par la moyenne de l'évaluation de la population totale.

On peut comparer cette méthode de sélection a une roulette de casino sur laquelle sont placés tous les individus de la population, la largeur allouée à chacun des individus étant en relation avec leur valeur d'évaluation. Ensuite, la bille est lancée et s'arrête sur un individu, voir la figure 1.7.

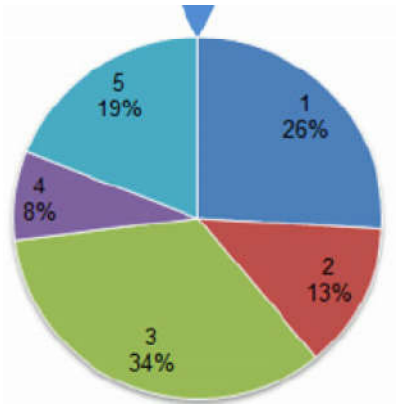


FIGURE 1.7: L'espace réserver par chaque programme dans la roulette

— Sélection uniforme : La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation. Chaque individu a donc une probabilité 1/P d'être sélectionné, où P est le nombre total d'individus dans la population. [5][1][7]

4.10.2 Les opérateurs génétiques

A l'aide de ces opérateurs génétiques on peut créent de nouveaux individus au cours de l'évolution. On retrouve généralement deux types d'opérateurs : les opérateurs de croisement et les opérateurs de mutation. Les opérateurs de croisement combinent deux individus parents pour créer un ou plusieurs nouveaux individus enfants avec des caractéristiques proches de celles des parents. Quant aux opérateurs de mutation, ils modifient un individu parent pour créer un enfant.

* **Croisement** L'opérateur de croisement est généralement un opérateur binaire. Il associe deux individus appelés parents pour en donner deux nouveaux individus appelés enfants. Cet opérateur est adapté à la programmation génétique, car il utilise une recombinaison de sous-arbres pour combiner le patrimoine génétique des programmes. Un point de croisement (un nœud), est choisi aléatoirement dans les deux programmes de manière indépendante.

  Le programme enfant sera créé à partir d'une copie d'un des parents, mais on remplace au point de croisement le sous-arbre par une copie du sous-arbre du deuxième parent, voir la figure 1.8. [9]

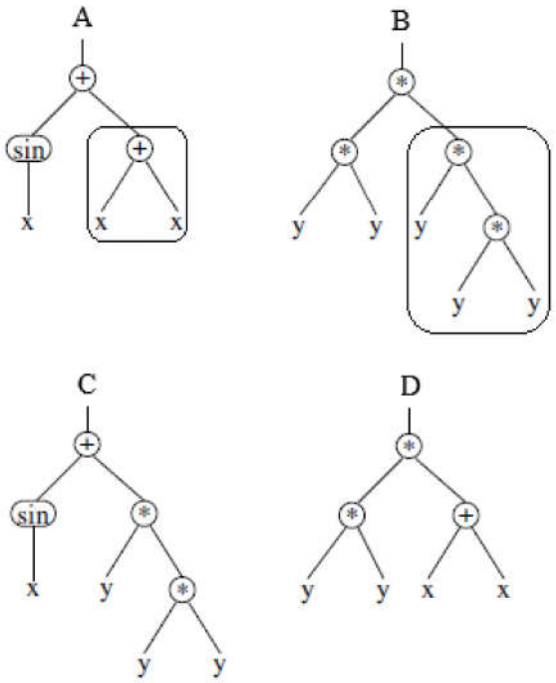


FIGURE 1.8: Le croisement des deux sous arbres (A) et (B) pour obtenir (C) et (D) arbres enfants.

* **Mutation** La forme la plus utilisée de cet opérateur consiste à tirer un point au hasard puis remplacer ce nœud par un nouveau sous-arbre. Il y a d'autre forme comme point-mutation qui remplace plusieurs nœuds avec d'autre nœuds.

  La mutation de sous arbre implique la modification d'un seul sous arbre du programme, mais la mutation par point quant à elle conserve la structure de l'arbre, mais remplace souvent plusieurs nœuds d'un seul programme. Cette mutation est appliquée selon une probabilité par nœud, indépendamment de la

taille de l'arbre, voir la figure 1.9. [9]

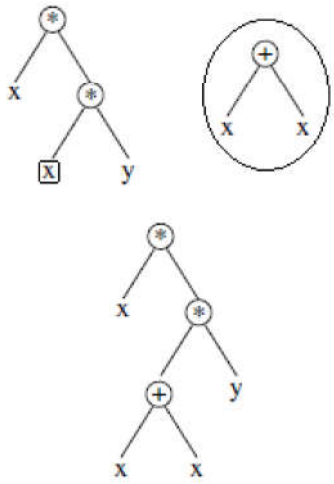


FIGURE 1.9: Le nœud X est remplacé par une sous-arbre généré aléatoirement dans le cercle.

4.10.3 L'opérateur de remplacement

On distingue deux méthodes de remplacement :

— Après l'application des opérateurs de variation sur les individus parents, on obtient une nouvelle génération d'individus, les meilleurs enfants de cette génération deviennent les parents de la génération suivante.

— Après l'application des opérateurs de variation sur les individus parents, on obtient une nouvelle génération d'individus, les meilleurs individus de la population (enfants + parents) deviennent les parents de la génération suivante.

4.11 Avantages et inconvénients de la programmation génétique

* les avantages de la programmation génétique englobent tous ceux énumérés dans la section des algorithmes génétiques.

* Il n'est pas nécessaire de spécifier à l'avance la forme et la taille de solution recherchée, permettant ainsi d'explorer un plus vaste espace des solutions.

* résolution de jeux, tris, recherches.

* aujourd'hui, en plus du logiciel, la programmation génétique est aussi appliquée à l'évolution du matériel.

* calcul quantique.

— les désavantages de la programmation génétique sont moins nombreux que ceux des algorithmes génétiques.

— la facilité de la représentation de certains problèmes sous forme de structures hiérarchiques sous avoir des problèmes de convergence prématurée de la population.

— La solution générée par le processus de recherche constitue la solution optimale ; l'espace des solutions est immense.

— la programmation génétique est couteuse en temps de calcul machine.

5 Application de la programmation génétique

* Il ya de nombreuses applications de la programmation génétique :

* « problèmes de « magie noire », tels que la synthèse automatisée des circuits électriques analogues, des contrôleurs, des antennes, des réseaux des réactions chimiques, et d'autres secteurs de conception,

* la « programmation de l'improgrammable » comportant la création automatique des programmes machine pour les dispositifs de calcul peu usuels tels que les automates cellulaires, systèmes de multi-agent, systèmes parallèles, rangées de porte field-programmable, rangées field-programmable d'analogue, colonies de fourmi, l'intelligence d'essaim, à distribué des systèmes, et des semblables,

* « nouvelles inventions commercialement utilisables » (CUNI) comportant l'utilisation de la programmation génétique comme « machine d'invention » automatisée pour créer de nouvelles inventions commercialement utilisables.

* Reconnaissance d'images (Robinson et al.), classification d'images (Zao), traitement d'images satellites (Daïda), ...

* Prédiction de séries temporelles (Lee), génération d'arbres de décisions (Koza), datamining (Freitas), ...

* Classification de segments d'ADN (Handley), de protéines (Koza et al.), ...

6 Conclusion

Dans ce projet, un algorithme de programmation génétique a été mis en œuvre pour résoudre un problème de régression symbolique. L'object est de trouver une fonction correspondant aux

données d'entrée-sortie données au plus près comme possible.

Nous avons présenté dans ce premier chapitre une vue générale sur la programmation génétique, comment elle fonctionne, ces phases, les différents opérateurs génétiques et le calcul de la qualité de performance pour créer des programmes optimaux capable à résoudre des problèmes quelconque, ensuit la problème de régression symbolique.

Chapitre 2

Programmation génétique linéaire

1 Introduction

Dans ce chapitre il donne un vu sur la programmation génétique linéaire (définition, Les atomes de la PGL, et donne un programmes LGP (Exemples d'opérateurs LGP typiques), Exactitude sémantique des programmes LGP, Chromosomes LGP (Évaluation de chromosomes LGP), codage de LGP, Opérateurs évolutifs chez LGP), en suite en parle sur la problème de régression symbolique.

2 La programmation génétique linéaire

2.1 Définition

La programmation génétique linéaire (LGP) est une variante de GP qui représente les individus linéairement sous forme de séquences d'instructions [6].

Le résultat de chaque instruction est stocké dans un registre.

L'Instruction Se compose d'un opérateur qui agit sur des opérandes, qui peuvent être entrés Données, constantes ou registres. Un exemple d'un tel programme pour le La tâche SR est illustrée à la exemple 1.

0 : r[1] = x * 1

1 : r[2] = x * r[1]

2 : r[0] = r[2] + 3

3 : r[4] = r[2] + r[1]

Exemple 1 : Exemple de programme LGP, représentant la formule f (x) = x2 + x. Le premier numéro de chaque ligne est l'identifiant de l'instruction

— le valeur de dernier registre (instruction 3) est la sortie du programme.

— La représentation linéaire introduit deux caractéristiques : non efficace code et réutilisation de code. Le code non-effectif sont des instructions attribuées à des registres qui ne sont pas utilisés pour calculer le résultat final.

— L'instruction 2 de la exemple1, par exemple, n'est pas efficace car elle est pas utilisé après. Ces instructions aident à augmenter le nombre variations neutres (variations qui ne changent pas le résultat de programme) et peut rendre les individus plus fexibles – un une opération sur une instruction non efficace peut la rendre efficace.

— l'autre caractéristique - la réutilisation du code - peut être illustrée par l'instruction 0 de la exemple 1. Il est utilisé par l'instruction 1 et encore par l'instruction 3. Cette fonctionnalité est utile si le même résultat doit être utilisé plus d'une fois dans le même programme, contribuant à évoluer plus facilement personnes.

— Le croisement dans le LGP traditionnel est identique à celui dans GA (échange de bloc entre parents), tandis que la mutation peut être de deux types : macro et micromutations. Les macro-mutations changent une instruction complète soit supprimer, insérer ou remplacer une instruction aléatoire.

  Sur le D'autre part, les micro-mutations changent un élément d'une instruction comme le registre de destination, l'opérateur ou un argument. Compte tenu du l'existence de codes non efficaces et de variations neutres, les opérateurs peuvent être efficace, c'est-à-dire que, dans la mesure du possible, ne changer que le Code de l'individu.

— l'implémentation LGP utilisée dans ce travail, appelée effmut, utilise seules Les mutations effectives, car les rapports montrent que cette configuration Effectue le meilleur [3][6]. C'est un algorithme d'état stable, comme expliqué en détail dans [6]. A chaque génération, deux tournois sont organisés, donnant deux gagnants et deux perdants. Une copie du gagnant remplace le perdant dans la population, tandis que le gagnant d'origine subit mutation en fonction des taux de mutation. Plus de détails sur le les opérateurs sont trouvés dans [6].

2.2 Les atomes de la programmation génétique linéaire

les individus sont représentés par des structures linéaires constituées de registres et instructions :

* **Registres :**Registres variables ; (noté ri) sont utilisés pour la saisie, la manipulation ,Données, et stocker le résultat résultant d'un calcul.

  Constant enregistre ; (notés ci) sont protégés en écriture une fois qu'ils ont été initialisé avec une valeur.

* **Instructions**Une instruction comprend un opérateur (par exemple +, -, ×, ÷) et un ou plusieurs opérandes. (c'est-à-dire les arguments) : r3 : = c2 + r1

  Je remarque que LGP facilite la possibilité d'utiliser plusieurs programmes sorties, alors que le GP basé sur une arborescence ne fournit qu'une sortie de programme.[?]

2.3 Programmes linéaire génétique programme (LGP)

Exemple : Un programme LGP simple composé de cinq instructions :

1 : r2 := r1 + c1

2 : r3 := r2 + r2

3 : r1 := r2 × r3

4 : if(r1 > c2)

5 : r1 := r1 + c1

L'exécution du programme LGP commence par la plus haute instruction, et continue en bas de la liste. Il s'arrête quand les dernières instructions ont été réalisé.

Branchement conditionnel (instruction 4) : Si vrai, les valeurs suivantes l'instruction (C'est-à-dire l'instruction 5) est exécutée, sinon l'instruction est sauté.

2.3.1 Exemples d'opérateurs LGP typiques

Un ensemble d'instructions LGP de base (c'est-à-dire un ensemble d'instructions autorisées) :

Instruction	Description	Instruction	Description
Addition	ri := rj + rk	Sin	ri := sin rj
Subtraction	ri := rj + rk	Cosine	ri := sin rj
Multiplication	ri := rj * rk	Square	ri := rj2
Division1	ri := rj / rk	Square root	ri := $\sqrt{rj}$
Xponentiation	ri := erj	Conditional branch	if ri >rj
Logarithm	ri := ln rj	Conditional branch	if ri rj

les opérandes (rj, rk) peuvent être des registres variables ou des constantes registres, mais la destination (ri) doit être un registre variable.

2.4 Exactitude sémantique des programmes LGP

Une instruction non valide d'un programme LGP peut provoquer un blocage du programme. Des exemples de tels problèmes sont :

— Instruction de saut illégale (peut provenir de ramifications augmentées instructions).

— I Division par zéro.

Dans ce dernier cas, un opérateur de division protégé est utilisé.

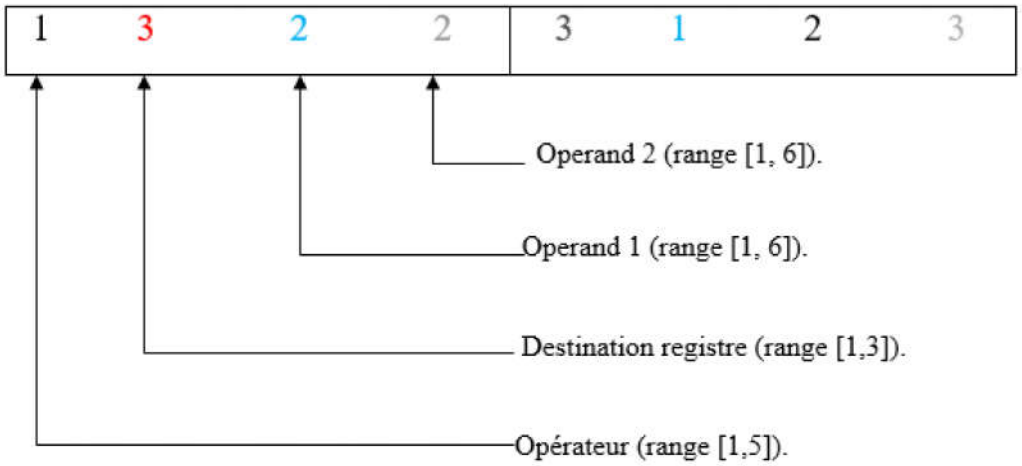
$$ri = \begin{cases} = \frac{rj}{rk} & if \quad rk \neq 0 \\ cmax & otherwise \end{cases}$$

ou Cmax est une grande constante pré-spécifiée.

Les programmes nouvellement générés doivent être soumis à un filtrage afin assurer l'exactitude sémantique.

2.5 Chromosomes LGP

Exemple de schéma de codage LGP. Laissez l'opérateur, les opérandes et destination être représentée par une séquence d'entiers :



Chaque gène est constitué de quatre nombres entiers, par ex. 1 2 1 4.

Dans le cas d'instructions à trois chiffres (par exemple Sine, Cosine), le Le nombre de quatre du gène doit être ignoré !

2.6 Codage LGP

Considérez les registres :

- L'ensemble des registres de variables : R = r1, r2, r3.

- L'ensemble des registres de constantes : C = c1, c2, c3.

-L'union de ces ensembles, R C, est notée A = r1, r2, r3, c1, c2, c3.

Considérons les opérateurs :

I L'ensemble des opérateurs addition, soustraction, multiplication, division, et le branchement conditionnel ri> rj : O = o1, o2, o3, o4, o5.

Considérons le premier gène du chromosome : 1 2 1 4.

1 : opérateur O (addition)

2 : Destination R (registre variable)

1 : Opérande 1 $\in A(toutregistre)$

4 : Opérande 2 $\in A(toutregistre)$

Ainsi, l'instruction décodée est la suivante : r2 : = r1 + c1.

2.7 Évaluation du chromosome LGP

Avant qu'un chromosome puisse être évalué, les registres doivent être initialisé. Supposer que :

Registres constants : c1 = 1, c2 = 3, c3 = 10.

Registres variables : r1 = 1, r2 = r3 = 0.

Les gènes	Instruction	Résultat (r1, r2, r3)
1 2 1 4	r2 := r1 + c1	1 2 0
1 3 2 2	r3 := r2 + r2	1 2 4
3 1 2 3	r1 := r2 × r3	8 2 4
5 1 1 5	if(r1 >c2)	8 2 4
1 1 1 4	r1 := r1 + c1.	9 2 4

De plus, supposons que l'entrée ait été fournie par le registre r1 et que la sortie a Également été prise à partir de r1 : Output = r1 = 9.

Cependant, notez que l'un des registres r1 - r3 pourrait être utilisé comme registre (s) d'entrée ou de sortie !

2.8 Opérateurs évolutifs chez LGP

Il est courant d'utiliser des méthodes à deux points, de longueurs variables (non homologues), croisement dans LGP :

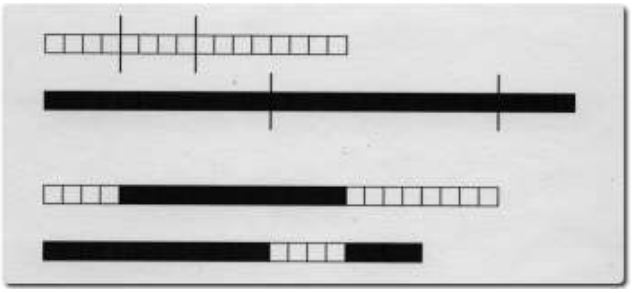


FIGURE 2.1: l'utilisation des méthodes à deux point.

En plus d'être non homologue, le croisement dans LGP fonctionne bien de la même manière que pour les chaînes de bits communément utilisées dans les GA.

La mutation dans LGP se déroule comme suit : sélectionnez au hasard un Instruction de mutation, puis avec une certaine probabilité : Je change (i) le registre de destination, (ii) l'opérateur, et (iii) le opérandes. C'est-à-dire, dans les limites autorisées bien sûr !

On not que la programmation génétique linéaire sous représenté sous forme d'un tableau ou bien arbre, liste. . . etc.

Dans ce travaille on choisir la représentation sous forme tableaux.

3 Le problème de régression symbolique

1. La régression symbolique est un type d'analyse de régression qui parcourt l'espace des expressions mathématiques pour trouver le modèle le mieux adapté à un ensemble de données donné, à la fois en termes de précision et de simplicité.
    * Aucun modèle particulier n'est fourni comme point de départ de l'algorithme.
    * Au lieu de cela, les expressions initiales sont formées en combinant de manière aléatoire des blocs de construction mathématiques tels que des opérateurs mathématiques, des fonctions analytiques, des constantes et des variables d'état. (Généralement, un sous-ensemble de ces primitives sera spécifié par la personne qui l'exploite, mais ce n'est pas une exigence de la technique.) De nouvelles équations sont ensuite formées en recombinant les équations précédentes, à l'aide de la programmation génétique.
    * En n'exigeant pas la spécification d'un modèle spécifique, la régression symbolique n'est pas affectée par les biais humains, ni par les lacunes inconnues dans la connaissance du domaine.
    * Il tente de découvrir les relations intrinsèques de l'ensemble de données en laissant les modèles dans les données elles-mêmes révéler les modèles appropriés, plutôt que d'imposer une structure de modèle réputée mathématiquement exploitable d'un point de vue humain. La fonction de fitness qui pilote l'évolution des modèles prend en compte non seulement les métriques d'erreurs (pour s'assurer que les modèles prédisent avec précision les données), mais également des mesures de complexité particulières, garantissant ainsi que les modèles résultants révèlent la structure sous-jacente des données dans un environnement virtuel. manière qui est compréhensible d'un point de vue humain. Cela facilite le raisonnement et favorise les chances d'obtenir des informations sur le système générateur de données.
2. Est une méthode de modélisation mathématique et d'analyse des données numériques qui nécessite de trouver une approximation d'une fonction mathématique sous une forme symbolique.

* On essaie à partir d'un ensemble de valeurs exprimentale.de trouver la courbe qui reproduit le mieux les variations de la grandeur à étudier et qui passe le plus prés possible des points dont on dispose, de manière à déterminer le comportement général du système.
* Est une méthode de recherche dans l'espace des expressions mathématique qui minimise les métriques d'erreurs.
* La régression symbolique essaie donc de trouver des fonctions dont la sortie possède certaines propriétés désirées, on essaie dans la plupart des cas, de trouver une fonction Qui coïncide avec les points données sans intéresser à la structure de la fonction.
* La régression symbolique s'intéresse simultanément à la recherche des paramètres comme la régression linéaire ou non-linéaire et à la forme et structure de la fonction.

  Cette technique donne automatiquement des objets mathématiques compréhensibles et plus ou moins facile à interpréter par l'utilisateur ce qui est majeur avantage et atout de cette méthode.

  Donc le Problème de régression symbolique : il ya un ensemble des points, on cherche des fonctions a partir de cette fonction choisir la fonction approximer dans cette points on appel la fonction fitness.

4 Conclusion

À partir de la comparaison enter la programmation génétique à base d'arbre et la programmation génétique linéaire(PGL),en choisir la PGL puisque les mieux (l'espace mémoire usager, et le parcoure, le temps). a la fin en peut résoudre le problème de régression symbolique avec la programmation génétique linéaire.

Chapitre 3

Conception

1 Introduction

Dans ce chapitre on va toucher la problématique et la solution proposée, en suite on va voir la conception et analyser le problème ensuit en parle sur les différentes structures de données, types, classes et méthodes utilisées, la formule utilisée dans le calcul du fitness, les étapes principales du programme.

2 Problématique

Notre travail se situe dans le cadre de l'implémentation d'une plateforme pour un moteur de programmation génétique. Pour cela nous avons considéré le problème suivant :

On s'intéresse au problème de la régression symbolique. Le problème consiste à trouver la relation entre des ''inputs" et des ''outputs" en appliquant une sorte d'apprentissage supervisé pour trouver la formule ''rapprochée" qui maitrise le mieux cette relation.

3 Solution proposée

Pour résoudre ce problème dessus cherché la fonction approximait (fonction fitness) sous forme symbolique à partir d'un point donné en utilisons la programmation génétique.

4 Structures de données utilisées

4.1 Les tableaux

Un tableau (array en anglais) est une structure de données qui consiste en un ensemble d'éléments ordonnés accessibles par leur indice (ou index), collecte des données homogènes.

C'est une structure de données de base que l'on retrouve dans chaque langage de programmation, voir la figure 3.1 [10].

Index	0	1	2	3	4	5	6
Valeur	45	154	58	78	31	5	74

FIGURE 3.1: La structure d'un tableau

4.2 les Matrices

Pour travailler sur des matrices, on utilisera la librairie **linalg** qui définit un constructeur matrix et de nombreuses fonctions classiques sur les matrices. Il est important de remarquer que l'objet matrix de Maple n'est qu'un cas particulier de l'objet array : la dimension vaut 2 et les lignes et les colonnes sont toujours numérotées à partir de 1.

4.2.1 Quelques outils concernant les matrices

4.2.1.1 Matrices comme tableaux Comme dit plus haut, les matrices sont des tableaux particuliers. Leur utilisation est donc proche de celle des tableaux. Ainsi, les mêmes difficultés apparaissent pour la copie de la matrice. Il faut dans ce cadre utiliser sans retenue l'opérateur evalm. D'une façon générale. il faut être vigilant lors des affectations. Ainsi, pour effectuer une opération de réaffectation comme[> M :=3*M ;, il faut taper[> M := evalm (3*M) ;.

4.2.1.2 Matrices avec linalg De nombreux outils matriciels d'algèbre linéaire sont disponibles dans le package linalg qui se charge en tapant l'instruction [> with (linalg) ;. On ira donc voir avec intérêt l'aide sur ce package.

Calculs matriciels

Vous avez appris en cours que Mn(K)est une algèbre, ce qui veut dire que concrètement, vous pouvez faire la somme et le produit de deux éléments - rappelez ces définitions -, et qu'il existe un produit extérieur entre un élément Ket une matrice M $\in Mn(K).Ainsileproduitdedeuxmatricesse faite$

5 La représentation d'un chromosome

— Pour représenter le chromosome En utilise la structure de données tableau.

— La taille de tableau 4*n.

— codage binaire de chromosome (que des 0 et des 1).

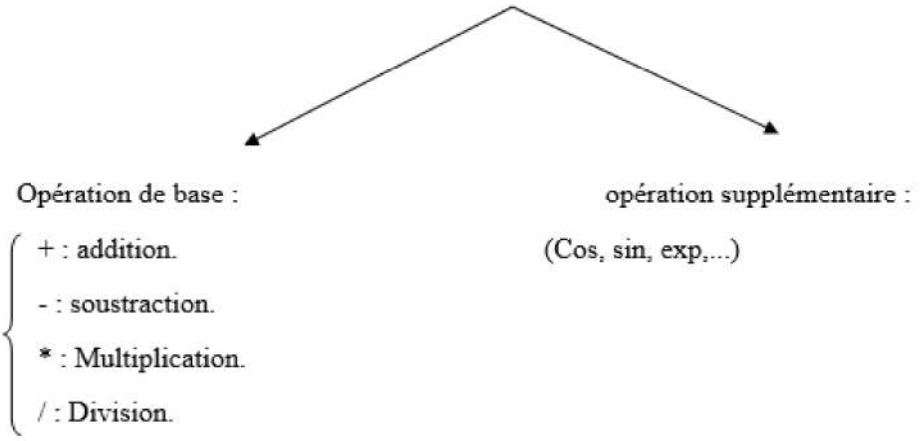
— le tableau contient des ensembles des cases.

— chaque quatre case présenter une instruction.

— chaque instruction contient :

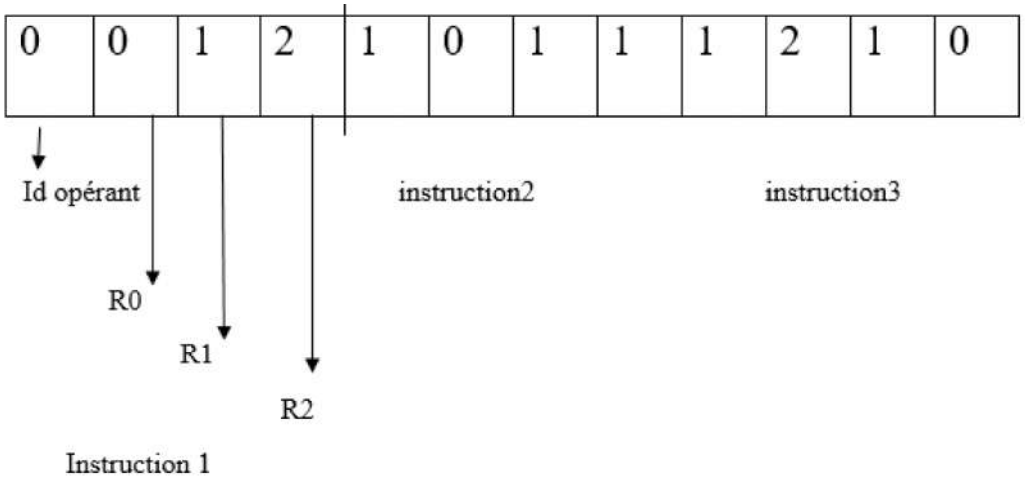
1. id opération.
2. registre destination (résultats).
3. premier opérande.
4. deuxième opérande.

1. **id opération :** L'intervalle [1.5], il ya deux type d'opération.



2. **le registre de destination (résultats) :** l'intervalle [1.3].
3. **le premier opérande :** l'intervalle [1.6].
4. **le deuxième opérande :** l'intervalle [1.6].

Exemple



Premier instruction : R0=R1+R2

Deuxième instruction : R0=R1-R1.

Troisième instruction : R2=R1-R0

Dans ce travaille Consiste à trouver la relation entre des ''inputs" et des ''outputs" pour trouver la formule ''rapprochée" qui maitrise le mieux.

1. à partir d'un point données (x,f(x),. . . . . ,(xn,f (xn) en cherche la fonction approximer (fonction fitness) par la programmation génétique , cette fonction Sous forme symbolique.
2. en crier un ensemble des chromosomes.
3. chaque chromosome représenté par un tableau de dimension 4*n.
4. en suite en applique une méthode pour calculer la fitness sur chaque chromosome .cette méthode comme suite :
   - Ona un tableau c1 qui représente le chromosome.
   - il ya un ensemble des points données (X0,f(x),(X1 ,f(x1),. . . (Xn,f(Xn) pour remplir un tableau de cette points.
   - l'ensemble de c'est points représente une fonction mathématique mais il n'est pas connus cette fonction.
   - au début en initialiser le registre r0 .
   - Le registre r0 c'est l'entrée et la sortie de programme.
   - en exécute c1 pour toute valeur de registre r0(chaque fois r0 prendre une valeur des x des points données).
   - calculer la différence enter les f(x) de point de table erreur et les f(x) des points données.

- les valeurs de cette différence représenté sous forme d'un table (tableau d'erreur) .
- calculer la somme de valeur de table erreur.
- les étapes appliquent pour un seul chromosome, on répéter cette étape pour tous les chromosomes.
- à prés en parcoure les chromosomes.
- nous choisissons les plus petites différences totales.
- Choisir les meilleurs chromosomes.
- en suite en applique l'opération de sélection et mutation et croisement sur les meilleurs chromosomes choisir.
- quand on postule la sélection et mutation et croisement sur les chromosomes nous obtenons des individus nouveaux.
- les individus nouveaux aussi on applique la sélection et mutation et croisement et calculer la fitness, et comparais les mieux fitness.
- après un nombre des générations on obtient les mieux individus.
- a la fin le minimum différence qui représente une fonction sous forme symbolique (fonction fitness).

6 Les classes utilisées

Pour notre simulation nous avons utilisé plusieurs class, et chacune de ces classes à sa propre tâche tell que :

6.1 Class sélection

La class sélection permet d'applique l'opération génétique « sélection ». La sélection sous forme aléatoirement (en sélectionner l'instruction aléatoirement)

- déclarer un tableau qui représente un chromosome ensuit en sélectionner une point aléatoirement . Ces points celons l'intervalle de la taille de table chromosome [4*n].
- dl'emplacement de point de sélection soit :
- dsélection le point dans la case de id opération.
- dsélection le point dans la case de registre destination.

— dsélection le point dans la case de premier opérande.

— d sélection le point dans la case de deuxième opérande.

```
//affichage du chromosome 2
     System.out.println("\n"+"chromosome 2");
     for(int i=0;i<4*n2;i+=4) {
        System.out.print(tab_chromosome_2[i] + ":" + tab_chromosome_2[i+1] + ":" +
            tab_chromosome_2[i+2] + ":" + tab_chromosome_2[i+3]+"/");
     }
```

6.2 Class mutation

La class mutation a la responsabilité de faire l'opération génétique «mutation »dans un programme. Cette opération permet de changer le comportement d'un instruction (changement dans l'id opération ,changement de la registre destination(résultat) , changement de la premier opérande, changement de la deuxième opérande),ou bien changent l'instruction complètement.

Remplacer une instruction avec une instruction autre aléatoirement ou bien supprimer complètement l'instruction choisie.

L'opération génétique mutation soit modification ou bien supprimer. Dans ce cas en donne 50Cette class contient Les étapes suivant :

— la déclaration de chromosome.

— la remplissage de notre chromosome.

— l'affichage le chromosome.

— en suite la sélectionner d'un point pour la mutation :

```
int k = 1+rnd.nextInt(n1*4);
 System.out.println("\n"+"le point de la mutation choisissez est: "+k);
int inst_ch_mutation;
inst_ch_mutation=k/4;
 System.out.print("le numro d'unstruction: "+inst_ch_mutation+"\n");
     inst_ch_mutation=inst_ch_mutation*4;
 system.out.print("le numro de la case: "+inst_ch_mutation);
```

— la suppression le chromosome après la modification :

```
int choix_sup = rnd.nextInt(2);
        if(choix_sup == 0) {

        //supression
           System.out.println("\n"+"Suprission");
        }
```

— ensuite en traite les cas de choix de type mutation (modification) :

```
  //le choix de type de la mutation
  int chois_mutation= rnd.nextInt(5);
  //System.out.println("\n"+"le chois de type de la mutation est:
      "+chois_mutation);

int chois;
switch(chois_mutation){
```

1. changement de type d'opération :

```
    case 0: {
          System.out.println("\n"+"changement de type d'opration");
      //pour assurer que le contenu de la case est changer, parce que
         l'alatoire peut me donne le mme nombre
           do {chois= rnd.nextInt(4);
              }while(chois == tab_chromosome[inst_ch_mutation]);

             tab_chromosome[inst_ch_mutation]= chois;
        //afficher le chromosome apres la modification
          for(int i=0;i<4*n1;i+=4) {
          System.out.print(tab_chromosome[i] + ":" + tab_chromosome[i+1] + ":"
              + tab_chromosome[i+2] + ":" + tab_chromosome[i+3]+"/");
       }
     }break;
```

2. changement de destination

```
case 1:
     System.out.println("\n"+"changement de la destination");
        do  {chois= rnd.nextInt(4);
        }while(chois == tab_chromosome[inst_ch_mutation+1]);
         tab_chromosome[inst_ch_mutation+1]= chois;

      }break;
```

3. changement de premier opérande

```
case 1:
     System.out.println("\n"+"changement de la 1 er oprand");
        do  {chois= rnd.nextInt(4);
       }while(chois == tab_chromosome[inst_ch_mutation+2]);
         tab_chromosome[inst_ch_mutation+2]= chois;

      }break;
```

4. changement de deuxième opérande

```
case 3: {
   System.out.println("\n"+"changement de la 2eme oprand");
       do  {chois= rnd.nextInt(7);
       }while(chois == tab_chromosome[inst_ch_mutation+3]);

        tab_chromosome [inst_ch_mutation+3]= chois;
       } break;
```

5. changement de l'instruction complètement

```
case 4:
   System.out.println("\n"+"changement d'instruction");

        int tab_mutation[] = new int[4];
       tab_mutation[0]= rnd.nextInt(4);
```

```
tab_mutation[1]= rnd.nextInt(4);
tab_mutation[2]= rnd.nextInt(7);
tab_mutation[3]= rnd.nextInt(7);

System.out.println("\n"+"le tableau de la mutation: ");
System.out.print(tab_mutation[0] + ":" + tab_mutation[1] + ":" +
    tab_mutation [2] + ":" + tab_mutation [3]+"/");


                tab_chromosome[inst_ch_mutation]=tab_mutation[0];
        tab_chromosome[inst_ch_mutation+1]=tab_mutation[1];
        tab_chromosome[inst_ch_mutation+2]=tab_mutation[2];
        tab_chromosome[inst_ch_mutation+3]=tab_mutation[3];

}break;
```

l'affichage du chromosome.

6.3 Class croisement

La class 'Croisement' permet d'appliquer l'un des opérations génétiques qui est le croisement entre deux chromosome pour reproduire des nouveaux individus. Chaque chromosome en sélectionner deux point aléatoirement ensuit enter chaque deux chromosome en fait la croisement de façon aléatoirement. on applique les étapes suivantes :

— la déclaration des deux chromosomes.

— la remplissage des deux chromosomes. Chaque chromosome remplir comme suit :

```
//remplir notre chromosome
   for(int i=0;i<4*n1;i+=4) {
      tab_chromosome_1[i]= rnd.nextInt(4);
      }

   for(int i=1;i<4*n1;i+=4) {
        tab_chromosome_1[i]= rnd.nextInt(nbrVariables);
      }
```

```
for(int i=2;i<4*n1;i+=4) {
tab_chromosome_1[i]= rnd.nextInt(nbrVariables + nbrConstantes);
  }
for(int i=2;i<4*n1;i+=4) {
tab_chromosome_1[i]= rnd.nextInt(nbrVariables + nbrConstantes);
  }

for(int i=0;i<4*n2;i+=4) {
tab_chromosome_2[i]= rnd.nextInt(4);
  }

for(int i=1;i<4*n2;i+=4) {
tab_chromosome_2[i]= rnd.nextInt(nbrVariables);
  }

for(int i=2;i<4*n2;i+=4) {
tab_chromosome_2[i]= rnd.nextInt(nbrVariables + nbrConstantes);
  }
for(int i=2;i<4*n2;i+=4) {
tab_chromosome_2[i]= rnd.nextInt(nbrVariables + nbrConstantes);
  }
```

— l'affichage d'un chromosome 1

```
//affichage du chromosome 1
System.out.println("\n"+"chromosome 1");
for(int i=0;i<4*n1;i+=4) {
System.out.print(tab_chromosome_1[i] + ":" + tab_chromosome_1[i+1] + ":" +
    tab_chromosome_1[i+2] + ":" + tab_chromosome_1[i+3]+"/");
  }
```

— l'affichage d'un chromosome 2

```
//affichage du chromosome 2
```

```
System.out.println("\n"+"chromosome 2");
for(int i=0;i<4*n2;i+=4) {
System.out.print(tab_chromosome_2[i] + ":" + tab_chromosome_2[i+1] + ":" +
    tab_chromosome_2[i+2] + ":" + tab_chromosome_2[i+3]+"/");
     }
```

sélection des points pour le croisement

```
//la selection des points pour le croisement
int k1 = 1+rnd.nextInt(n1*4)-1;//pour eviter le cas du chois de la derniere
    case "-1"s
int k2 = 1+rnd.nextInt(n1*4)-1;
int k3 = 1+rnd.nextInt(n2*4)-1;
int k4 = 1+rnd.nextInt(n2*4)-1;

System.out.println("\n"+"les point du croisement pour le chromosome 1 est:
    "+k1+" , "+k2);
System.out.println("\n"+"le point du croisement pour le chromosome 2 est:
    "+k3+" , "+k4);

      int inst_ch_croisement_1=k1/4;
      int inst_ch_croisement_2=k2/4;
      int inst_ch_croisement_3=k3/4;
      int inst_ch_croisement_4=k4/4;

System.out.print("le numro d'unstruction: "+inst_ch_croisement_1+" ,
    "+inst_ch_croisement_2+"\n");
System.out.print("le numro d'unstruction: "+inst_ch_croisement_3+" ,
    "+inst_ch_croisement_4+"\n");

      inst_ch_croisement_1=inst_ch_croisement_1*4;
      inst_ch_croisement_2=inst_ch_croisement_2*4;
      inst_ch_croisement_3=inst_ch_croisement_3*4;
      inst_ch_croisement_4=inst_ch_croisement_4*4;
      //System.out.print("le numro de la case: "+inst_ch_mutation);
```

— la création du tableau du croisement

```
        // cration  du tableau du croisement
     int tab_croisement[] = new int[4];
```

— copier a partir chromosome 1 vers le tableau.

```
        //le croisement au premier point
//copier a partir chromosome 1 vers le tableau
tab_croisement[0]=tab_chromosome_1[inst_ch_croisement_1];
tab_croisement[1]=tab_chromosome_1[inst_ch_croisement_1+1];
tab_croisement[2]=tab_chromosome_1[inst_ch_croisement_1+2];
tab_croisement[3]=tab_chromosome_1[inst_ch_croisement_1+3];
```

— copier a partir chromosome 2 vers chromosome 1

```
            //copier a partir chromosome 2 vers chromosome 1
     tab_chromosome_1[inst_ch_croisement_1]=tab_chromosome_2[inst_ch_croisement_3];
     tab_chromosome_1[inst_ch_croisement_1+1]=tab_chromosome_2[inst_ch_croisement_3+1];
     tab_chromosome_1[inst_ch_croisement_1+2]=tab_chromosome_2[inst_ch_croisement_3+2];
     tab_chromosome_1[inst_ch_croisement_1+3]=tab_chromosome_2[inst_ch_croisement_3+3];
```

— copier a partir le tableau vers chromosome 2

```
            //copier a partir le tableau vers chromosome 2
     tab_chromosome_2[inst_ch_croisement_3]=tab_croisement[0];
     tab_chromosome_2[inst_ch_croisement_3+1]=tab_croisement[1];
     tab_chromosome_2[inst_ch_croisement_3+2]=tab_croisement[2];
     tab_chromosome_2[inst_ch_croisement_3+3]=tab_croisement[3];
```

— affichage du chromosome 1

```
//affichage du chromosome 1
System.out.println("\n"+"chromosome 1");
for(int i=0;i<4*n1;i+=4) {
System.out.print(tab_chromosome_1[i] + ":" + tab_chromosome_1[i+1] + ":" +
    tab_chromosome_1[i+2] + ":" + tab_chromosome_1[i+3]+"/");
     }
```

— affichage du chromosome 2

```
//affichage du chromosome 2
System.out.println("\n"+"chromosome 2");
for(int i=0;i<4*n2;i+=4) {
System.out.print(tab_chromosome_2[i] + ":" + tab_chromosome_2[i+1] + ":" +
    tab_chromosome_2[i+2] + ":" + tab_chromosome_2[i+3]+"/");
     }
```

— le croisement au deuxième point.(le même méthode pour le croisement de premier point).

— copier a partir chromosome 1 vers le tableau.

— copier a partir chromosome 2 vers chromosome 1.

— copier a partir le tableau vers chromosome 2.

— affichage du chromosome 1.

— affichage du chromosome 2.

6.4 Class chromosome

-la class de chromosome c'est la class principale de programme.

1. au début :
   — déclarer la table de chromosome.
   — déclarer un tableau pour trier les chromosomes.
   — initialiser la valeur de fitness.
   — déclarer un nombre de population.
   — déclarer le nombre de la génération.
   — déclaration une matrice.
   — déclaration de tableau pour garder les meilleur fitness (tab meilleur$_f itness$).
2. ensuit en fait l'initialisation des points (x,f(x))pour les fonction suivant : $f(x) = x^2$; $f(x) = x + 1$; $f(x) = cos(x)$.
   — après initialiser le tableau des résultats d'exécution du chromosome.
   — la création de la population initiale par matrice .

— afficher le chromosome.

— en fait une boucle des ensembles des générations.

— initialiser le registre de destination.

— affecter le résultat au table de comparaison.

— la comparaison entre les valeurs.

— garder la meilleur fitness du génération.

— calculer la fitness du chaque chromosome.

Chapitre 4

Résultats

1 Introduction

Dans ce chapitre on va faire des études expérimentales de notre système avec la programme génétique. En fait une comparaison entre les fonction de ce programme (comparaison de la meilleur fitness pour chaque fonction), en suite on peut analyser les résultats obtenus.

2 Le langage utilisé

Nous avons choisi le langage de programmation JAVA. C'est un langage orienté objet créé par James Gosling et Patrick Naughton de Sun Microsystems avec le soutien de Bill Joy (co-fondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld. [11][10]

Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitations (Windows, Mac, Linux...) C'est un langage permettant de créer des applications de manière simple et facile. Il possède plusieurs avantages, nous pouvons cités :

* JAVA est un langage orienté objet. Chaque class a ces propre méthodes et variables mais on peut accéder à ces méthodes facilement.
* Java offre plusieurs bibliothèques pour la création des interfaces graphiques, afin de visualiser les résultats de l'application.

3 Le logiciel utilisé

L'environnement de développement que nous choisissons c'est l'environnement Eclipse. Eclipse est un IDE (Integrated Development Environment) développé par IBM, c'est un logiciel gratuit et disponible pour la plupart des systèmes d'exploitation, voir l'interface du logiciel éclipse dans la figure 4.1. [12]

Eclipse possède plusieurs avantages dans l'objectif de simplifié la programmation tell que :

* La compilation automatique du code que vous écrivez, en soulignant en rouge ou jaune les problèmes pendant l'écriture du code.
* Eclipse peut vous aider à compléter ce que vous écrivez, en utilisant le raccourci clavier Control+Espace.
* Eclipse possède l'autocorrection de sort que pour un certain nombre de problèmes, il propose de modifier le code pour corriger l'erreur, par exemple dans le cas d'importer des bibliothèques.

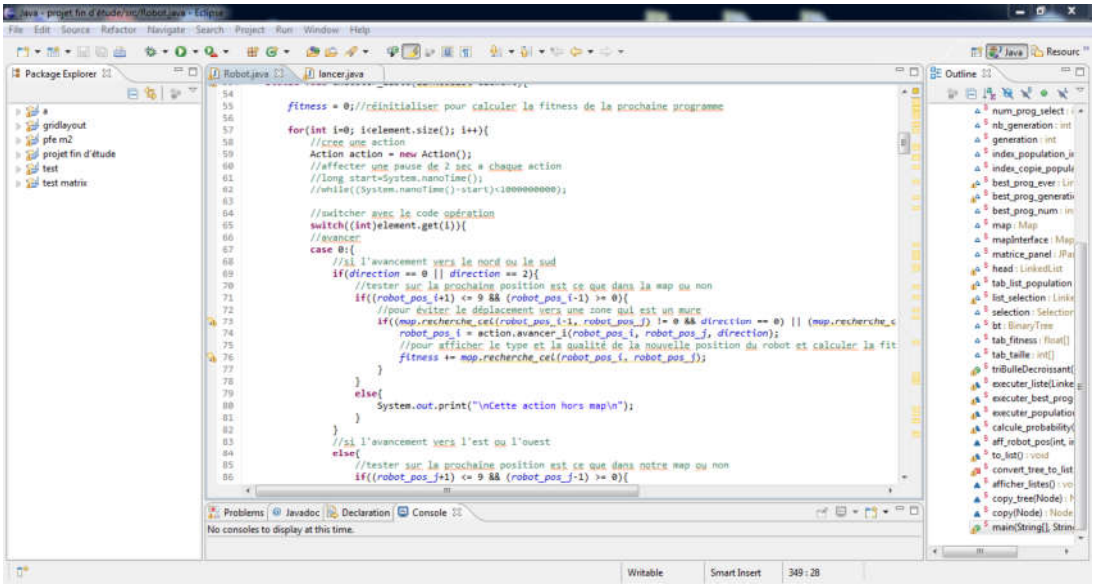


FIGURE 4.1: L'interface graphique du logiciel eclipse.

4 Résultat

— Les résultats suivant obtenue après l'exécution de programme

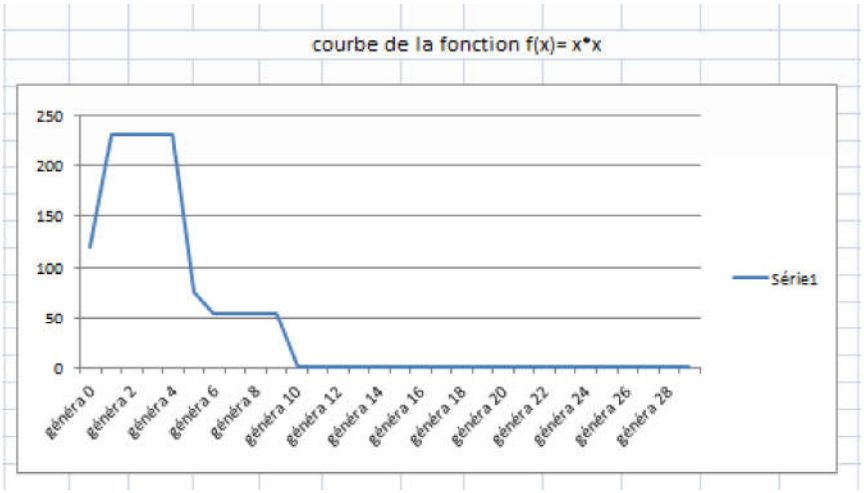
— Le nombre de génération 100.

— Les résultats pour la fonction f(x)=x*x

Tableau1 :représenter la finesse en fonction de générations

	génér a 0	généra 1	généra 2	généra 3	généra 4	génér a 5	généra 6	généra 7	généra 8	généra 9	généra 10	généra 11	généra 12	généra 13	généra 14
val de fitness	120	231	231	231	231	75	55	55	55	55	1	1	1	1	1

généra 15	généra 16	généra 17	généra 18	généra 19	généra 20	généra 21	généra 22	généra 23	généra 24	généra 25	généra 26	généra 27	généra 28	généra 29
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

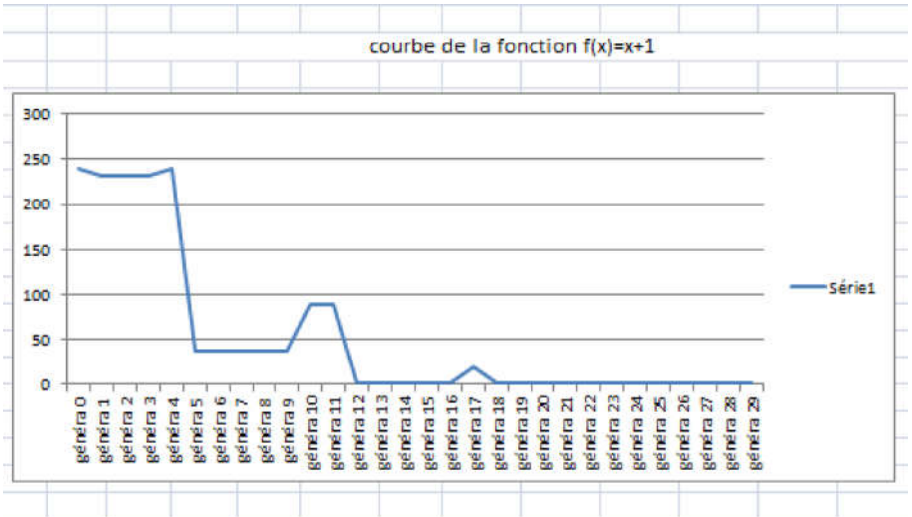


— Les résultats pour la fonction f(x)=x+1.

Tableau2 :représenter la finesse en fonction de générations

	génér a 0	généra 1	généra 2	généra 3	généra 4	génér a 5	généra 6	généra 7	généra 8	généra 9	généra 10	généra 11	généra 12	généra 13	généra 14
val de fitness	239	231	231	231	239	37	37	37	37	37	89	89	1	1	1

généra 15	généra 16	généra 17	généra 18	généra 19	généra 20	généra 21	généra 22	généra 23	généra 24	généra 25	généra 26	généra 27	généra 28	généra 29
1	1	19	1	1	1	1	1	1	1	1	1	1	1	1

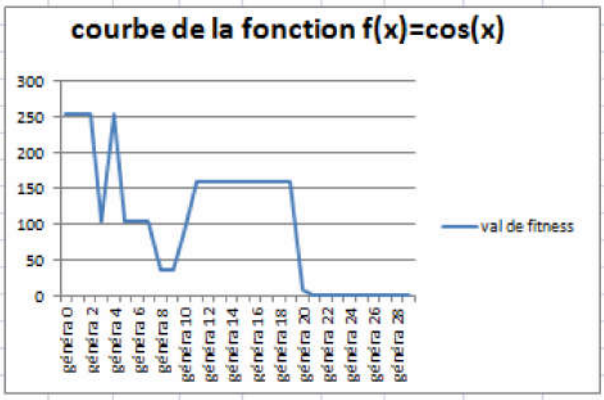


— -Les résultats pour la fonction f(x)= cos(x).

Tableau3 :représenter la finesse en fonction de générations

	génér a 0	généra 1	généra 2	généra 3	généra 4	génér a 5	généra 6	généra 7	généra 8	généra 9	généra 10	généra 11	généra 12	généra 13	généra 14
val de fitness	255	255	255	105	255	105	105	105	37	37	89	159	159	159	159

généra 15	généra 16	généra 17	généra 18	généra 19	généra 20	généra 21	généra 22	généra 23	généra 24	généra 25	généra 26	généra 27	généra 28	généra 29
159	159	159	159	159	9	1	1	1	1	1	1	1	1	1



5 Conclusion

Après l'étude expérimentale sur notre programme génétique avec l'analyse des résultats obtenus, on trouve qu'il est possible d'utiliser la programmation génétique Il est intéressant de constater que la population initiale ne contient que des programmes aléatoires, mais grâce au paradigme de la programmation génétique ces programmes aléatoires s'évoluent au cours du temps afin de s'adapter à son environnement « c'est le rôle de la fonction objectif ».

Ainsi, il est intéressant de constater que dans la programmation génétique il y'a l'aspect de l'aléatoire, ce qui fait que par fois le résultat trouvé ne répond pas parfaitement au problème à résoudre.

Conclusion générale

Notre projet c'est utiliser la programmation génétique pour résoudre un problème de régression symbolique

La programmation génétique est un paradigme de programmation automatique motivé par l'évolution biologique des espèces. Cette méthode génère une succession des hypothèses par une répétition de croisement et de mutation sur des segments de codes aléatoires pour le but de trouver la meilleure solution possible pour le problème visé. Dans ce travail on s'intéresse au problème de la régression symbolique. Le problème consiste à trouver la relation entre des ''inputs" et des ''outputs" en appliquant une sorte d'apprentissage supervisé pour trouver la formule ''rapprochée" qui maitrise le mieux cette relation.

Dans le premier chapitre nous avons présenté des généralités sur les algorithmes évolutionnaires, le cycle de vie d'un algorithme évolutionnaire, les domaines d'applications, la famille des algorithmes évolutionnaires. Après, nous nous sommes focalisés sur la programmation génétique, les opérations génétiques et les phases d'un programme génétique.

Dans le deuxième chapitre-nous il donne un vu sur la programmation génétique linéaire, en suite en parle sur la problème de régression symbolique.

Dans la troisième chapitre avons présenté les détaille de conception avec les techniques pour l'implémentation de la programmation génétique pour résoudre le problème.

Dans le troisième chapitre nous avons fait des études expérimentales sur notre programme génétique , ensuit analyser les résultat obtenue.

Après avoir faire le déférent résultat, on tire quelques remarques :

— La programmation génétique minimise le temps de programmation, parce qu'il s'agit de la programmation automatique qui base sur l'évolution d'une population de programmes aléatoires.

— Pour trouver la solution optimale il faut donner un grand nombre de générations.

— Nécessite un peu de temps pour trouver la solution optimale.

— Dans notre problématique, il est difficile de trouver la meilleur fitness qui donne toujours des bons résultats. Nos perspectives sont :

— Trouver une les meilleur fitness plus efficace pour augmenter de trouver la solution optimale. trouver la meilleure solution possible pour le problème de régression symbolique.

— Données les résultats obtenus .

Bibliographie

[1] Cours : donné à l´ecole polytechnique,*http ://sdz.tdct.org/sdz/introduction-a-la-vision-par-ordinateur.html*, Consulter le 16/04/2019.

[2] Algorithme évolutionniste,*https ://fr.wikipedia.org/wiki/Algorithme_ évolutionniste*, 2019.

[3] Cours : donné à l'université de bourgogne,*http ://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/CM2009 /Genetiques/Genetique*$_1$*.pdf*, 2019.

[4] Cours : chapitre 8 « algorithmes évolutionnaires et problèmes inverses » donné à l´ecole polytechnique".,*http ://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/poly/*
2019.

[5] Mohamed Oulmahdi. Algorithmes évolutionnaires dans les systèmes de parole. *Master recherche informatique*, page 26, 2011.

[6] http ://magnin.plil.net/spip.php ?article45, 2019.

[7] La programmation génétique,*https ://perso.liris.cnrs.fr/alain.mille/enseignements/master_ ia/rap*

[8] http ://agerodol.pagesperso-orange.fr/gp/gp.html, 2019.

[9] Oussama EL GERARI. *Contributions à l'Amélioration des Techniques de la Programmation Génétique et de la Programmation Évolutionnaire.* PhD thesis, Thèse de doctorat, l'Université du Littoral Côte d'Opale, 2011.

[10] François Morain. *Les bases de l'informatique et de la programmation.* École polytechnique, Département d'informatique, 2002.

[11] http ://ipeti.forumpro.fr/t21-definition/internet-java-java-485/, 2019.

[12] JP Barthélemy and NX Luong. Sur la topologie d'un arbre phylogénétique :
aspects théoriques, algorithmes et applications à l'analyse de données textuelles. *Mathématiques et Sciences humaines*, 100 :57–80, 1987.