N° d'ordre 13/2014-D/inf

République Démocratique et Populaire Algérienne

Ministère de la Recherche et de l'Enseignement Supérieur

Université des Sciences et de la Technologie Houari Boumedienne

Faculté d'Electronique et Informatique

THESE

Présentée pour l'obtention du grade de DOCTEUR

En INFORMATIQUE

Spécialité : Infortmatique

Par : BENKHIDER BABA-ALI SADJIA

Sujet

Metaheuristiques pour l'Extraction de Règles de Classification en Data Mining

Soutenue publiquement le 02/02/2014 devant le jury

Mr Guessoum Ahmed	Professeur USTHB	Président
Mme Drias Habiba	Professeur USTHB	Directrice de Thèse
Mme Soule-Dupuy Chantal	Professeur U.Toulouse France	Examinatrice1
Mr Boukra Abdelmadjid	Professeur USTHB	Examinateur 2
Mr Moussaoui Ahmed	Professeur U.F.A de SETIF	Examinateur 3
Mme Kamel Nadjet	MCA U.F.A de SETIF	Examinatrice 4

Dédicaces

- A la mémoire de mon défunt père, Hadj Mohamed Benkhider, décédé le 08juillet2013, qui aurait tant souhaité voir ce travail s'achever. Que Dieu l'acceuille en son vaste paradis

- A ma petite famille : Riadh, Assia, Lies et Mehdi qui ont beaucoup patienté pendant que j'étais occupée à finaliser cette thèse.

- A la mémoire de ma collègue Ahlem Benchennaf , ravie aux siens à la fleur de l'age : Que Dieu ait son ame en son vaste paradis.

Remerciements

Au terme de ce travail, j'aimerais adresser mes vifs remerciements à :

- Prof. Habiba Drias pour m'avoir acceuillie dans son laboratoire de recherche LRIA, pour son suivi, ses conseils et ses orientations : qu'elle trouve ici l'expression de mon plus profond respect.

- Prof. Ahmed Guessoum pour avoir accepté de présider ce jury et pour ses efforts déployés tous les jours pour que notre département informatique puisse s'améliorer sans cesse.

.

- Prof. Chantal Soule-Dupuy de l'université de Toulouse pour avoir accepté d'examiner mon travail et de faire le voyage à Alger pour ma soutenance, qu'elle soit assurée de ma grande gratitude.

- Prof. A. Moussaoui et DR N. Kamel qui ont accepté de juger ce travail et de se déplacer eux aussi de l'université de Sétif vers Alger pour ma soutenance, je ne saurais jamais les remercier assez.

- Prof. A. Boukra de l'USTHB lui aussi a accepté de juger ce travail, qu'il en soit vivement remercié.

- Dr A.R. Baba-Ali MCA à la FEI de l'USTHB, Je ne saurais jamais le remercier assez pour ses encouragements, ses conseils fructueux et ses orientations très nombreuses.

- Mes collègues et amis du LRIA, leur liste est longue et je ne peux les citer toutes et tous, en particulier Dr N. Aleb, Dr D. Boughaci, Dr H. Necir, Dr Y. Aklouf, Dr S. Kechid et feu Mme A.Benchennaf : nous avons toujours partagé de très agréables moments ensemble dans notre laboratoire : qu'ils trouvent ici l'expression de mes sentiments les plus amicaux.

- Je n'oublie pas de remercier mon étudiante en ingéniorat et en magister et qui est devenue par la suite mon amie et collègue : Melle A. Mahani, pour son aide et les agréables moments passés ensemble.

- Toutes et tous mes collègues du département informatique: la liste est encore plus longue mais j'espère qu'elles et qu'ils trouveront ici l'expression de mes remerciements les plus sincères pour leurs encouragements tout au long de ce travail.

SOMMAIRE

LISTE des FIGURES

Liste des Tableaux

Introduction Générale

Le datamining ou forage de données consiste à extraire automatiquement des connaissances cachées ou des modèles à partir de données du monde réel où la connaissance découverte est idéalement précise, compréhensible et intéressante pour l'utilisateur.
C'est un domaine interdisciplinaire et une zone très large et active de recherche auquel de nombreuses conférences internationales annuelles et plusieurs revues académiques sont entièrement consacrées.
Les méthodes d'extraction de connaissances peuvent être utilisées pour effectuer une variété de tâches, notamment la découverte de l'association, le regroupement, la régression et la classification. Lorsque la classification est envisagée, un algorithme de fouille de données pourrait générer un modèle de classification à partir d'un ensemble de données, et ce modèle pourra ensuite être appliqué à classer des objets dont les classes ne sont pas connues.
D'une manière générale, les modèles de classification peuvent être divisés en deux catégories selon le type de représentation des connaissances utilisées: des modèles compréhensibles de l'homme et des modèles dits "boîte noire". Parmi les exemples de modèles compréhensibles de l'Homme nous pouvons citer les arbres de décision et les règles de classification et parmi les modèles dits "boite noire" sont cités par les réseaux de neurones, les réseaux bayésiens et les SVM : support vector machines.
Dans le cadre de nos travaux, nous focalisons nos efforts sur l'extraction de règles de connaissances et plus précisément de classification. La recherche des règles de classification consiste à extraire un ensemble de formules logiques conditionnelles qui déduisent la valeur d'un attribut but à partir des valeurs d'autres attributs apparaissant simultanément. La combinatoire du choix des attributs à faire figurer dans la règle et leurs différentes valeurs possibles, font que la taille de l'espace de recherche des règles candidates est exponentielle. Ce problème est NP_complet [FRE1998].
Nous avons pour cela développé plusieurs algorithmes d'extraction de règles de classification, avec un souci majeur : la précision, la compréhensibilité du modèle obtenu et d'autre part le temps de classification que nous nous attacherons à réduire. Notre démarche consiste donc à construire des algorithmes fiables, précis, produisant des règles compréhensibles mais aussi rapides et efficaces.

Pour cela, nous définirons tous les concepts et paradigmes que nous avons utilisés.

Au premier chapitre de cette thèse, nous avons défini d'une manière générale le data mining : ce concept relativement nouveau, puis au deuxième chapitre la classification d'une manière plus détaillée puisque c'est la tâche à laquelle nous nous sommes particulièrement intéressés.
Le troisième chapitre présentera en détails tout le formalisme des algorithmes génétiques ainsi que leurs différents schémas parallèles et le quatrième chapitre comportera des notions de mémétisme.

Au cinquième chapitre nous développerons la méthode de classification que nous avons mise au point en premier : à savoir une approche génétique pure pour l'extraction de règles de classification.
Quant au sixième chapitre, nous présenterons une approche génétique parallèle pour l'extraction de règles de classification où un nouveau schéma parallèle asynchrone de l'AG a été proposé pour réduire le temps d'exécution et d'obtention de classificateurs.

Au chapitre suivant, nous détaillerons une approche mémétique toujours pour l'extraction de règles de classification où plusieurs algorithmes de recherche locale sont mis au point afin d'augmenter la précision des classificateurs obtenus.
Toutes nos approches ont été testées et vérifiées sur des bases d'apprentissage de l' Université de Californie Irvine (UCI) qui propose une large bibliothèque de « benchmarks » pour le data mining. Ainsi un chapitre comportera tous les résultats de tests effectués ainsi que des études comparatives par rapport à d'autres approches d'extraction de règles de classification que nous avons trouvées dans la littérature.

Nous terminerons par une conclusion générale portant sur les résultats de nos travaux de recherche et quelques propositions de travaux futurs.

Chapitre I

Définitions et Concepts du Data Mining

- Introduction

- Présentation de notions sur les DataWarehouses et le Data Mining

-Historique

-Domaines d'application du Data Mining

-Tâches et Techniques du Data Mining

-Conclusion

1. Introduction

Dans le monde des sciences et des technologies de l'information sans cesse en évolution, de nouveaux concepts apparaissent au fur et à mesure de besoins exprimés dans la réalité. Souvent ils expriment des concepts anciens n'ayant pas eu la chance de se développer faute de technologies ou de maturité. Dans l'univers du décisionnel, plusieurs concepts émergent grâce à l'évolution des technologies de l'information : le "data warehousing" et le "data mining". Dans ce qui suit, nous allons donner quelques préliminaires concernant ces deux paradigmes dans le but d'en faciliter leur compréhension.

2. Présentation d'un data warehouse

Un Data Warehouse est un entrepôt de données d'une entreprise contenant à la fois :
-ses données opérationnelles enregistrées,
-ses données agrégées selon toute dimension,
-ses données historisées,
-toutes données externes à l'entreprise mais ayant une relation possible avec ses activités.

Le Data Warehouse est caractérisé par l'orientation de l'usage des données vers la décision plutôt que vers le simple archivage, contrairement au système d'information.

Le Data Warehouse est composé de données intégrées, c'est-à-dire qu'un "nettoyage" ou pré-traitement des données est à mener dans le but de rationalisation et de normalisation.

Les données du Data Warehouse sont caractérisées par la non volatilité : une donnée entrée dans l'entrepôt est toujours datée puis sauvegardée et ne doit jamais être supprimée.

L'organisation des données est conçue (voir Figure1.1) dans un souci majeur qu'est la rapidité d'accès à l'information stratégique et sous forme synthétique dont on a besoin pour la prise de décision.

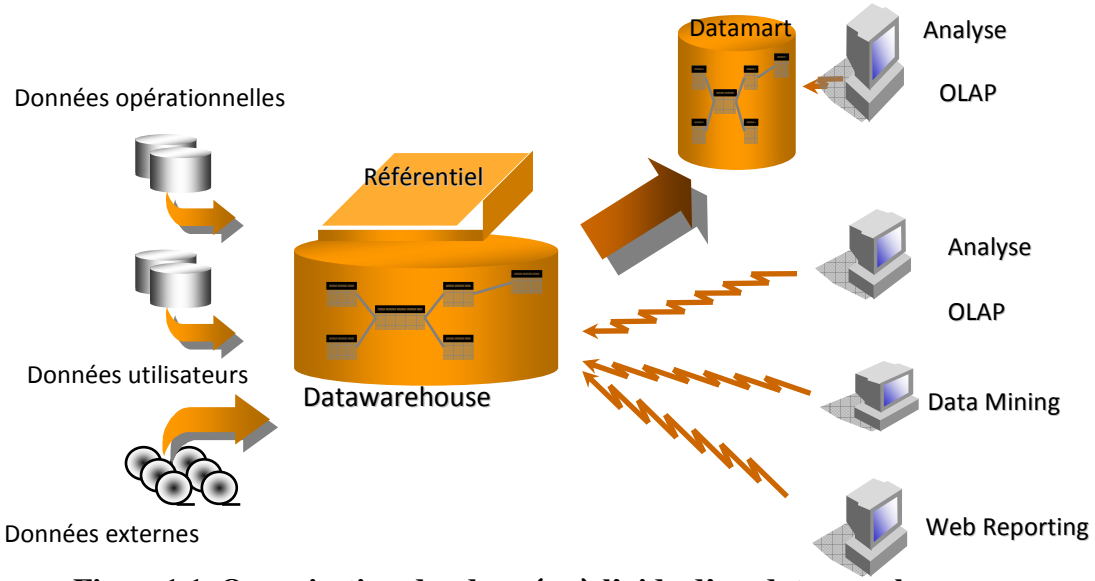


Figure1.1. Organisation des données à l'aide d'un data warehouse

3. Présentation du data mining

3.1 Définitions

Tout d'abord, certaines distinctions entre les notions suivantes : donnée, information et connaissance sont à discerner. D'après [LEF98] :

La donnée : décrit des exemples ou des événements précis. Elle peut être receuillie de manière automatique ou par écrit.

L'information : est tout le signifiant que l'on attache et que l'on peut déduire d'un ensemble de données et de certaines associations entre ces dernières.

La connaissance : décrit une catégorie abstraite. Chaque catégorie peut couvrir plusieurs exemples. Des experts sont nécessaires pour recueillir et formaliser la connaissance.

Les données sont ainsi transformées en information qui peut être utilisée à des fins commerciales ou industrielles et même dans l'explication de phénomènes scientifiques.
A l'origine se trouvent donc des données brutes non traitées, puis collectées et formatées puis explorées de façon à ce que leurs relations deviennent manifestes. Plus grand est le nombre d'informations révélées, plus grande est l'intelligence du domaine, jusqu'à ce qu'un certain niveau de profonde compréhension soit atteint : la connaissance. Celle-ci est donc le résultat d'un processus complexe.
Connaissance et information sont deux notions souvent confondues dans la perspective de découverte de connaissances dans les bases de données [KOD1999], l'information n'étant qu'un élément de connaissance susceptible d'être codée, mémorisée et traitée. D'après [BER2004] la relation entre ces trois notions peut être illustrée par la Figure 1.2.

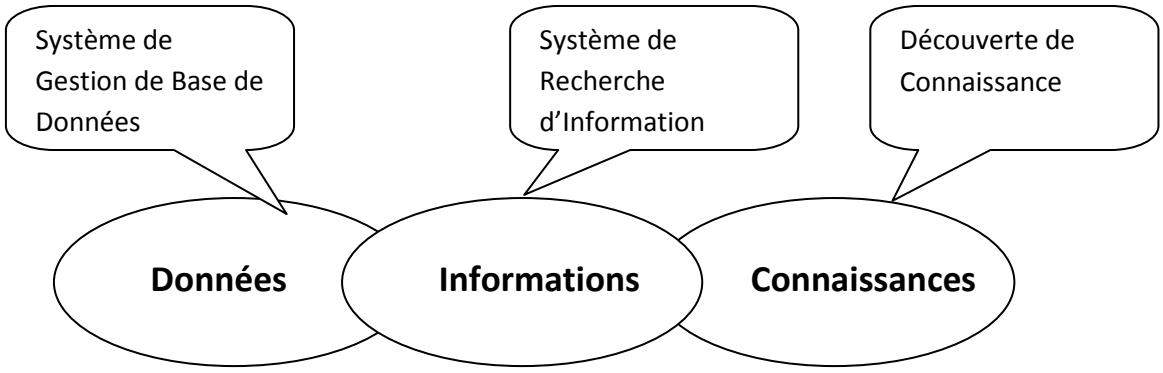


Figure1.2 : Utilisations de Données, Informations et Connaissances

Le Data Mining est un ensemble de techniques ou d'outils du domaine statistique, mathématique et informatique qui permettent à l'utilisateur l'extraction de la connaissance auparavant inconnue, à partir d'un important volume de données.
L'objectif de ces méthodes est de découvrir des ensembles d'associations, des tendances ou des relations qui existent entre ces données, puis de générer une information riche, qui augmente la valeur des données contenues dans le Data Warehouse, bien que le Data

Warehouse ne soit pas indispensable pour entamer des tâches de data mining, comme nous le verrons par la suite.

Le data mining est un domaine de recherche très actif dans lequel il s'agit de concevoir des méthodes de calcul afin d'extraire des connaissances utiles à partir de données du monde réel. Il a été engendré par la rapidité d'amélioration de la technologie de production et de stockage de données au cours de ces deux dernières décennies.

3.2 Historique

L'expression "Data Mining" est apparue vers le début des années 1960. Le courant "Data Analysis" est né avec les progrès et la diffusion des ordinateurs ayant permis la mise en œuvre de techniques descriptives multidimensionnelles sur de grandes masses de données. Des chercheurs ont commencé à traiter sans a priori statistiques des tableaux de données relatifs à des enquêtes ou des expériences disponibles. Comme les résultats constatés étaient prometteurs, ils se trouvèrent confrontés à la systématisation de cette approche opportuniste. Les statisticiens n'approuvaient guère cette démarche et la considéraient comme peu scientifique : ils utilisèrent alors le terme "data mining" et "data fishing" pour les critiquer.

Le succès de cette démarche alors empirique était recueilli chaque jour malgré les critiques. L'analyse de données s'est développée et son intérêt augmentait aussi vite que le volume des données. Vers la fin des années 80, des chercheurs tels que Rakesh Agrawal ont commencé à travailler sur l'exploitation des bases de données volumineuses notamment celles des tickets de caisses de grandes surfaces, convaincus par la possibilité de valoriser ces masses de données dormantes.

Le Data Mining dans sa forme et compréhension actuelle à la fois comme domaine scientifique et industriel est apparu au début des années 90. Cette émergence n'est que le résultat de la combinaison de nombreux facteurs à la fois technologiques, économiques et sociaux-politiques. En effet, la survie d'une entreprise par exemple, ne peut avoir lieu que grace à l'adaptation de sa réponse à la demande du marché. Au départ l'entreprise fabriquait des produits proposés au marché : ce modèle d'économie peut être qualifié "orientée produits". Actuellement la concurrence est beaucoup plus importante et les clients sont de plus en plus exigeants. Cet environnement a mis les entreprises devant des responsabilités plus importantes : celles d'offrir des biens ou des services répondant au mieux aux besoins plus spécifiques du client. Ainsi l'économie est devenue plutôt "orientée client".

Le data mining est actuellement appelé "fouille de données" ou "forage des données". C'est un processus itératif et/ou interactif selon les objectifs à atteindre. Cela consiste en un ensemble de tâches et techniques permettant de procéder à une analyse de données provenant d'un data warehouse ou tout simplement de bases de données volumineuses, celles-ci pouvant se présenter sous forme de tables. Les modèles générés par les outils de data mining peuvent être sous forme de fonctions, sous forme de conditions « if…then » ou définies dans un langage propre à l'intelligence artificielle, tels que LISP, Prolog, etc...

3.3 Domaines d'application [TUF2005]

Le datamining a envahi aujourd'hui de nombreux domaines qui vont de l'infiniment petit (génomique) à l'infiniment grand (astrophysique), du plus quotidien (gestion de la relation client) au moins quotidien (aide au pilotage automatique), du plus ouvert (e-commerce) au plus sécuritaire (prévention du terrorisme, détection automatique de la fraude dans la

téléphonie mobile ou l'utilisation de cartes bancaires), du plus industriel (control qualité, pilotage de la production) au plus théorique (enquêtes en sciences humaines, études biologiques, médicales et pharmaceutiques) et mêmes des domaines tels que la prévision d'audience TV. A cette simple énumération l'on peut deviner que le spectre des applications du datamining est très large. Les plus concernées sont les secteurs où d'importants volumes de données doivent être analysés, parfois en vue de prendre une décision rapide. L'aide à la décision devient une finalité du datamining dont on n'attend plus seulement qu'il aide à comprendre le réel en le modélisant. Le data mining permet de limiter la subjectivité humaine dans les processus de décision et aussi grâce à la puissance grandissante des outils informatiques, de traiter de plus en plus rapidement de grands nombres de dossiers. A titre d'indication, en juin 2002 un sondage a été effectué sur le portail web "kdnuggets.com". Ce sondage avait révélé les principaux secteurs utilisant le datamining : la banque(13%), les télécommunications(9%), le e-commerce(9%), la détection de fraudes(8%), le marketing direct(7%), l'assurance(6%), la distribution(6%), la biologie(5%), et l'industrie pharmaceutique(5%), le reste pour divers autres applications. Actuellement, ces chiffres ont certainement dû être multipliés par un facteur impressionnant.

4. Tâches et Techniques du datamining

Le datamining est constitué d'un ensemble de tâches capables de mener des études sur des données très volumineuses, dans le but d'extraire de la connaissance cachée sous forme de modèles. Parmi ces tâches, l'on peut citer :

-l'association ou le regroupement par similitudes.
-la classification :
- supervisée,
- non supervisée,

-l'estimation,
-la prédiction.

4.1 Présentation de l'association ou regroupement par similitude

Elle consiste à rechercher les relations ou les dépendances, existant entre plusieurs caractéristiques. Le cas typique qui relève d'une démarche d'association est "l'analyse du panier de la ménagère". Dans ce problème, les observations considérées sont les paniers, contenant plusieurs produits.

L'association consiste à trouver parmi la masse d'informations disponibles contenues dans les tickets de caisse, les articles qui vont ensemble lors d'une transaction.

D'une manière générale, on dispose en entrée d'un ensemble de transactions où chacune d'entre elles est un ensemble de littéraux appelées Items. Une règle d'association peut être la suivante : "30% des transactions qui contiennent le produit X et le produit Y contiennent aussi le produit Z ; 2% de toutes les transactions contiennent ces trois produits". Ainsi la valeur 30% représente la confiance de la règle et 2% le support.

Un autre exemple simple peut être formulé comme suit :

Age(20-29) ^ revenu(40K-49K) alors achat(Laptop) avec un support =2% et confiance=60%

Cette règle d'association précise que parmis les clients, 2% sont agés entre 20 et 29ans et ont 40K à 49K comme revenus ont acheté des "laptops" et il y a 60% de probabilité que un client de cette tranche d'age et de revenus achètent aussi des "laptops" .

La formulation du problème est une implication sous la forme A=>B. La partie gauche est l'antécédent de la règle et la droite le conséquent. La façon intuitive d'interpréter une règle est que les transactions qui contiennent A tendent à contenir B, en reprenant le 1er exemple, la plupart des fois où X et Y ont été achetés, Z l'a été également.

La recherche de règles d'association consiste à extraire un ensemble de formules logiques conditionnelles qui déduisent la valeur d'un attribut but à partir des valeurs d'autres attributs apparaissant simultanément.

Une règle d'association peut être définie formellement par:

- **Définition** : Soit I = $\{i_1,i_2,\ldots,i_m\}$un ensemble d'items. Soit T= $\{t_1,t_2,\ldots,t_n\}$ un ensemble de transactions, telles que t_i soit un sous-ensemble de I. Une règle d'association s'exprime sous la forme :

$$X \longrightarrow Y \text{ , où } X \in I \text{ et } Y \in I \text{ et } X \cap Y = \emptyset$$

- D'une manière généralisée, X est un sous-ensemble de I appelé itemset.

  Le recouvrement ou la couverture de X dans T noté $K_T(X)$ est définie par :
  $\{k = 1, 2, ..n | X \subseteq t_k\}$.

  $$K_T(X \cup Y) = K_T(X) \cap K_T(Y)$$

  Remarques :

- Le compteur de support de X dans T est le nombre de transactions de T qui contiennent X. On le note Compteur(X) = Card(K_T(X))
- Le support est alors la proportion des transactions de T contenant X, soit Supp(X)=Compteur(X)/Card(T). Celui-ci peut être vu comme une estimation de la probabilité P(X).
- La qualité d'une règle d'association est mesurée par son indice de support et son indice de confiance.
  - L'indice de support d'une règle est défini par la proportion de transactions de T qui contiennent XUY (à la fois X et Y et non pas X ou Y), soit Supp(XUY).

    Il s'agit donc d'une estimation de la probabilité P(XUY).

  - L'indice de confiance (*confidence*) d'une règle X $\longrightarrow$ Y est défini par la proportion de transactions de T contenant X qui contiennent aussi Y, soit Compteur(XUY)/Compteur(X). Il peut être vu comme une estimation de la probabilité conditionnelle P(Y/X). Remarque :

Conf(X→Y) = Supp(XUY)/Supp(X)

- Le "*lift* " d'une règle X→Y mesure l'amélioration apportée par la règle d'association par rapport à un jeu de transactions aléatoire (où X et Y seraient indépendants). Il est défini par :

  Supp(XUY)/(Supp(X).Supp(Y)) .

  Un "*lift* " supérieur à 1 traduit une corrélation positive de X et Y, et donc le caractère significatif de l'association.

  Parmis les algorithmes de génération de règles d'association, nous pouvons citer :

Apriori : c'est l'algorithme le plus courant de génération de règles d'association. Il s'execute en deux étapes :

- Soient min_Supp l'indice de support minimum donné, et min_Conf l'indice de confiance donné.
- Génération de tous les itemsets fréquents, tels que :

$$IF = \{X_i \subseteq T | supp(X_i) = X_i.count \geq minsupp, i = 1, 2, ..n\}$$

  Où X_i.count= nombre d'instances de la base d'apprentissag contenant X.

- Génération de toutes les règles d'associations de confiance à partir des itemsets fréquents, c'est-à-dire

$$\{X_i, Y_j \subseteq IF | Conf(X_i \rightarrow Y_j) \geq minconf \ i = 1, 2, ..p \ j = 1, 2, ..q\}$$

D'autres algoithmes ont été récemment mis au point, tels que FP-Growth [HPY2000], Eclat [ZAK2000] basé sur la recherche d'associations utilisant les intersections d'ensembles et Closed [PBTL99]. Tous ces algorithmes sont détaillés dans [HAN2011].

4.2 La classification

La classification se fait depuis très longtemps pour comprendre et communiquer notre vision du monde (par exemple les espèces animales, minérales ou végétales).

"La classification consiste à examiner des caractéristiques d'un élément nouvellement présenté afin de l'affecter à une classe d'un ensemble prédéfini."

La classification consiste à regrouper les individus en classes, en se basant sur la ressemblance entre ces individus, chaque processus de classification comprend les éléments suivants :

♦ *Un critère de ressemblance* : entre deux individus, un individu et une classe ou bien aussi entre deux classes.

♦ *Un critère de décision* **:** permettant l'attribution d'un nouvel individu à une classe.

Elle se présente sous deux formes : supervisée et non supervisée. La classification est notre pôle d'intérêts dans nos travaux, nous lui consacrons le chapitre II.

4.3 L'estimation

Contrairement à la classification, le résultat d'une estimation permet d'obtenir une variable continue. Celle-ci est obtenue par une ou plusieurs fonctions combinant les données en entrée. Le résultat d'une estimation permet de procéder aux classifications grâce à un barème. Par exemple, on peut estimer le revenu d'un ménage selon divers critères (type de véhicule et nombre, profession ou catégorie socioprofessionnelle, type d'habitation, etc.).

Il sera ensuite possible de définir des tranches de revenus pour classifier les individus.

Un des intérêts de l'estimation est de pouvoir ordonner les résultats dans un but précis, par exemple ne retenir que les n meilleures valeurs. Cette technique sera souvent utilisée en marketing, combinée à d'autres, pour proposer des offres aux meilleurs clients potentiels.

Une des techniques les plus appropriées à l'estimation est : le réseau de neurones. Le problème d'estimation est une généralisation du problème de classification, étant donné que la variable à modéliser devient continue.

L'estimation comporte les étapes suivantes :

- Identifier la variable objet et les autres variables qui pourraient permettre de la prévoir.

- Regrouper un certain nombre de cas pour lesquels à la fois les variables objectives et les variables explicatives sont disponibles.

- Construire un modèle qui relie de façon satisfaisante les variables explicatives à la variable objective.

4.4 La prédiction

La prédiction ressemble à la classification et à l'estimation mais dans une échelle temporelle différente. Tout comme les tâches précédentes, elle s'appuie sur le passé et le présent mais son résultat se situe dans un futur généralement précisé. La seule méthode pour mesurer la qualité de la prédiction est d'attendre !

Les techniques les plus appropriées à la prédiction sont :

-Le raisonnement basé sur la mémoire,

-Les arbres de décision,

-Les réseaux de neurones.

4.5 L'analyse des clusters

L'analyse des clusters consiste à segmenter une population hétérogène en sous populations homogènes. Contrairement à la classification, les sous populations ne sont pas préétablies. Plusieurs techniques existent et qui seront étudiées dans le chapitre II.

4.6 La description

C'est souvent l'une des premières tâches demandées à un outil de Datamining. On lui demande de décrire les données d'une base complexe. Cela engendre souvent une exploitation supplémentaire en vue de fournir des explications.

4.7 L'optimisation

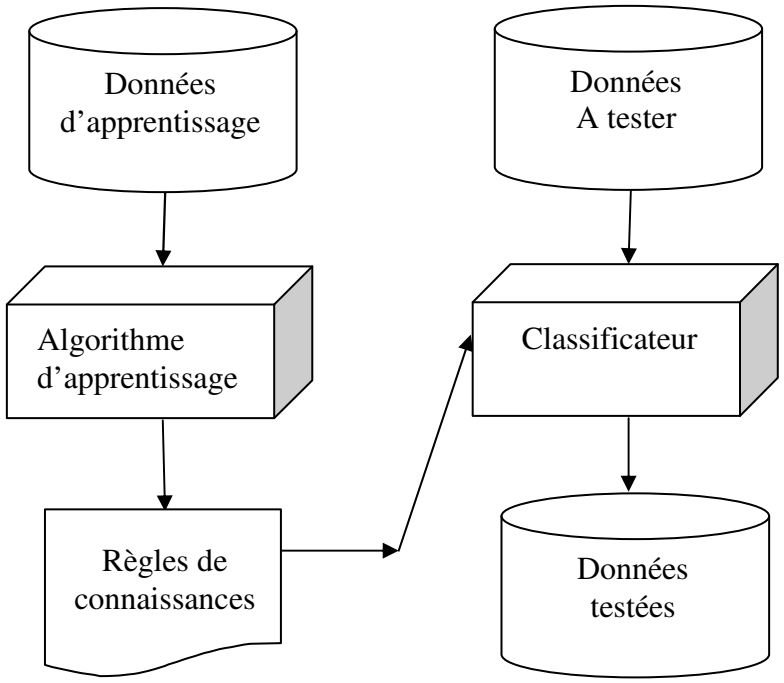
Pour résoudre de nombreux problèmes, il est courant pour chaque solution potentielle d'y associer une fonction d'évaluation. Le but de l'optimisation est de maximiser ou minimiser cette fonction. Quelques spécialistes considèrent que ce type de problème ne relève pas du datamining.

5. Conclusion

Dans ce chapitre nous avons regroupé quelques définitions de notions telles que le data waraehouse et le data mining. Pour cela, nous avons défini quelques tâches de data mining en indiquant quelques techniques relatives à ces tâches, dans le but de montrer dans le chapitre suivant la tâche de classification ainsi que toutes ses techniques classiques. Tout cela a été développé dans le but de montrer nos contributions dans ce domaine.

1. Introduction [PAR 2002][HOR2007][WAI2003]

La fouille de données est définie comme un processus d'extraction de connaissances à partir de bases de données. Parmi les fonctions de fouille de données, on distingue la classification. Dans cette fonction, le but est d'assigner à chaque cas (enregistrement de données), une classe choisie à partir d'un ensemble de classes prédéfinies, en fonction de la valeur de certains attributs. En d'autres termes, étant donné un ensemble d'enregistrements de données appartenant chacun à une classe parmi un nombre de classes prédéfinies, le problème de la classification peut être celui de la découverte de règles qui permettent à des enregistrements de classes indéterminée, d'être correctement classés (voir *figure 2.1*).

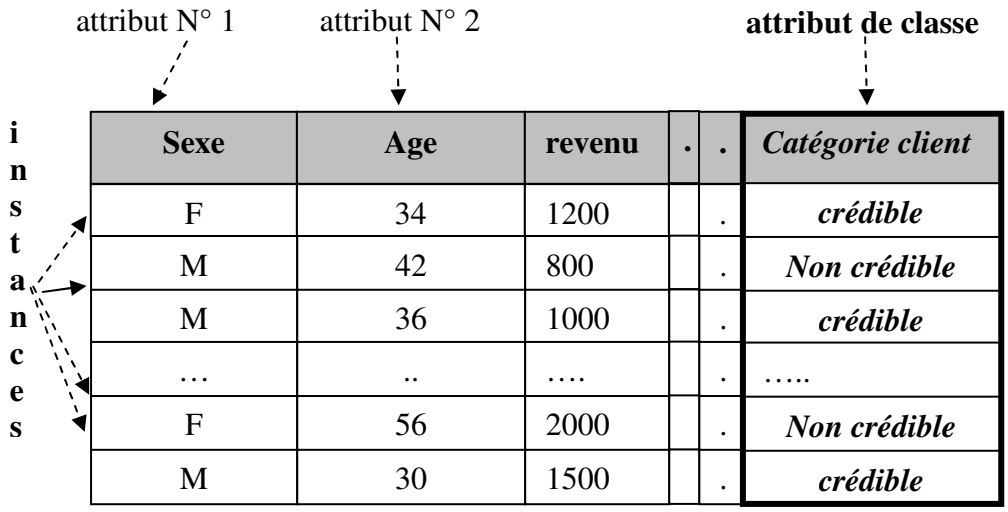


(a) **(b)**

Figure 2.1 (a) Illustration du processus d'apprentissage pour la fouille de données
(b) Illustration du processus de classification

2. Notions de données :

Dans le domaine de la fouille de données, les données sont généralement organisées en ***tables*** (voir ***tableau 01***) qui contiennent un ensemble ***d'instances***. Une table peut être vue comme une matrice.



Sexe	Age	revenu	.	.	*Catégorie client*
F	34	1200		.	***crédible***
M	42	800		.	***Non crédible***
M	36	1000		.	***crédible***
…	..	….		.	…..
F	56	2000		.	***Non crédible***
M	30	1500		.	***crédible***

Tableau 01 : **Exemple de données d'entrée pour la fouille de données**

Chaque ligne de la matrice correspond à une instance qui est associée à un cas expérimental à analyser. Un cas peut être par exemple un client particulier, alors que l'ensemble des instances peut être une population de clients à analyser.

Chaque colonne de la matrice représente toutes les valeurs possibles associées à un ***attribut***. Un attribut est une donnée descriptive d'un individu. Pour une base de données Clients, cela peut être le statut marital, le sexe, l'âge, le revenu annuel, etc.

Un attribut peut être numérique ou nominal. Un attribut numérique prend ses valeurs d'un domaine de valeurs continu alors qu'un attribut nominal prend ses valeurs à partir d'un ensemble de choix discret. Enfin, un attribut particulier appelé ***l'attribut de classe***, est utilisé comme attribut de classification. L'ensemble des valeurs de l'attribut de classe représente l'ensemble des catégories qu'on attribue à chaque instance. Dans notre cas, cela peut être par exemple des classes telles que : (clients crédibles) ou (non crédibles), ou bien (fraudeur).

On parle de ***données d'apprentissage*** si toutes les données ont déjà été classées ie : elles contiennent toutes une valeur pour l'attribut de classe (voir Tableau 01). On parle de ***données de test*** si ces dernières ne possèdent pas encore une donnée pour l'attribut de classe.

3. Notions de modèle : [PAR 2002]

Dans le contexte de la fouille de données et plus particulièrement de la classification, la connaissance extraite ou ***modèle*** se présente généralement sous la forme d'un ensemble de ***règles***. Ces règles ont généralement la forme :

SI *antécédent* **ALORS** *conséquent*

L'***antécédent*** est constitué de conditions reliées par l'opérateur logique ET. Chaque condition appelée un ***terme*** ou ***prémisse*** est formée par un triplet ***< attribut, opérateur, valeur >.***

- **Exemple de terme** : Sexe=F

Où "*Sexe*" est un attribut de la table clients (voir Tableau 1 également), le signe "=" est l'opérateur de comparaison et "F" est la valeur de comparaison.

- **Exemple d'antécédent** : Sexe=*M* ***ET*** *Age>30* ***ET*** *Revenu>1000*

Le ***conséquent*** est constitué de la classe prédite par la règle qui peut être dans notre cas, la classe crédible.

- **Exemple de règle** : **SI** *Sexe=M ET Age>30 ET Revenu>1000* **ALORS** *Crédible*

Cette règle prédit que si une personne vérifie chaque condition (terme) de l'antécédent, alors il est probable que cette personne soit crédible.

L'ensemble des règles forme un modèle de classification [WIT2005].

- **Exemple de modèle :**

SI *Sexe=M ET Age<30 ET Revenu>1000* **ALORS** *Crédible*
Sinon SI *Sexe=M ET Age>50 ET Revenu>2500* **ALORS** *Crédible*
Sinon SI *Sexe=F ET Age>50 ET Revenu>1500* **ALORS** *Non crédible*

La classification est l'une des tâches les plus étudiées en data mining. C'est un processus qui consiste à identifier la classe à laquelle appartient un individu, selon certains attributs qui le caractérisent [BER 2004], [FRE 1998]. La classification est appliquée dans un grand nombre d'activités humaines, tels que l'analyse de risques dans un système bancaire, l'aide à la décision, la détection de fraudes, la détection d'intrus dans un réseau, l'aide au diagnostique dans le domaine médical, …
La tâche de classification est scindée en deux types :

- la classification supervisée : consiste à grouper des éléments selon certaines caractéristiques dans des classes prédéfinies,
- la classification non supervisée : consiste à regrouper des éléments similaires en groupes inconnus d'avance. Ce second type est aussi appelé segmentation ("clustering").

Dans nos travaux notre pôle d'intérêt a été la classification supervisée.

4. La Classification Supervisée

La tâche de classification supervisée est elle même subdivisée en deux groupes: la classification basée sur des règles de classification, où le modèle extrait est un ensemble de règles, et celle basée sur d'autres méthodes telles que par support de vecteurs, par réseaux artificiel de neuronnes et réseaux bayésiens..

L'exemple illustré par la figure 2.2 fait apparaître la classification comme une tâche qui consiste à ranger des ***formes*** ou ***individus*** décrits par un ensemble de variables descriptives dans un certain nombre de catégories ou classes définies à priori.

Pour extraire la connaissance enfouie dans les données, nous avons besoin d'extraire une connaissance compréhensible, faute de quoi elle n'est pas utilisable dans un domaine où la clarté et la facilité d'interprétation sont requises. La connaissance compréhensible est importante dans différents domaines d'application, où l'expert humain joue un grand rôle dans le processus de la résolution de problèmes [TAN 2006].

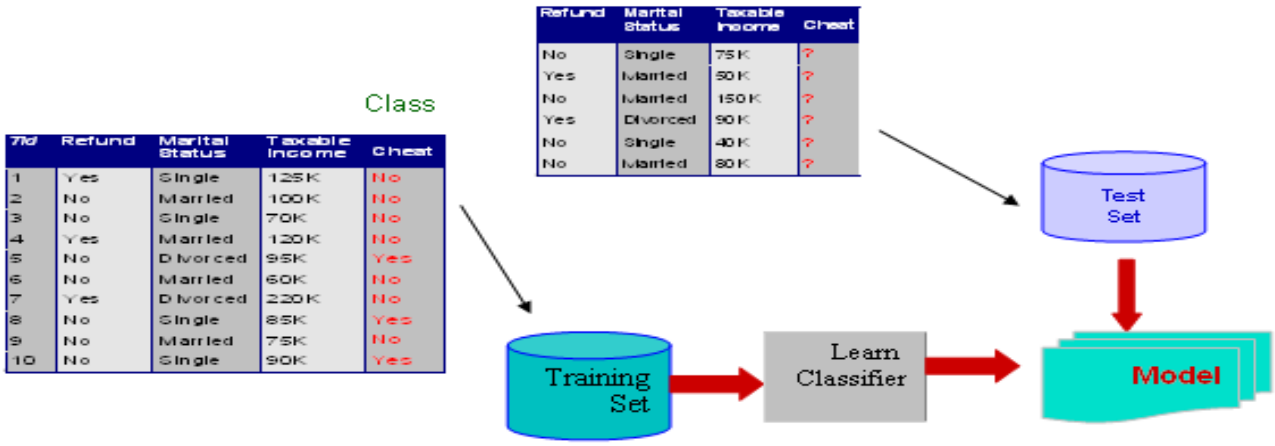


Figure2.2 Illustration du processus de classification

Nous étudions dans ce qui suit, quelques techniques ou méthodes de classification supervisée connues.

4.1. Classification par Heuristiques

Les heuristiques d'extraction de règles sont des algorithmes évolutifs, dont le but est la découverte de règles justes et précises, aboutissant par la suite à certaines informations et connaissances concernant l'analyse d'une base de données.

Il existe plusieurs algorithmes d'extraction de règles, nous avons étudié trois d'entres eux : X2R, OneR et les Tables de Décision.

Chacun d'eux possède ses propres méthodes d'évolution, ses propres mesures de qualités, pour extraire les meilleures règles, et cela par rapport à la compréhensibilité et la facilité ainsi que la simplicité de ces dernières.

4.1.1 L'algorithme X2R [HUA 1995]

L'algorithme X2R est un algorithme d'extraction de règles, il comporte trois étapes majeures :

- Étape 1 : Génération de la règle

Elle choisit l'instance qui revient le plus fréquemment dans le jeu d'instances pour générer la règle, puis la suivante jusqu'à épuisement des instances. De cette façon, X2R peut manipuler des bruits dans les données, pouvant être des instances sans valeur pour l'attribut de classe, ou bien des attributs qui ne possèdent pas de valeurs.

Le cœur de cette étape est un algorithme avide (glouton) qui trouve la plus petite règle basée sur un premier ordre d'information (une seule condition).

Il génère itérativement les règles et élimine les instances couvertes par chaque règle générée jusqu'à ce que toutes les instances soient couvertes par des règles.

- Etape 2 : Regroupement ou segmentation de règle

Les règles générées lors de la 1ère étape sont regroupées par rapport à leurs labels de classe pour les prochaines procédures.

- Etape 3 : Nettoyage des données

A chaque regroupement de règle, les règles redondantes sont éliminées, puis les règles spécifiques sont remplacées par des règles plus générales.

Cet algorithme produit des règles courtes, ce qui implique plus de compréhensibilité et leur nombre est réduit. Cependant, elles sont de 1er ordre et le taux de précision du modèle extrait varie entre 80% et 100%. Le taux d'erreur peut être élevé lorsqu'il s'agit d'applications sensibles et ne tolérant pas plus de 1% d'erreurs. Par ailleurs, plusieurs contraintes peuvent être éliminées, ce qui induit une perte d'informations utiles.

4.1.2. L'algorithme OneR [BUD 1992 , HOL 1993]

C'est un algorithme de classification simple. Il produit des règles simples basées seulement sur un attribut. Elles sont de la forme :

Si attribut=valeur alors classe

Son principe est donné par l'algorithme suivant :

Début
Pour chaque attribut
Faire
Pour chaque valeur de cet attribut
Faire choisir une règle comme suit:
-Calculer le nombre d'apparitions pour chaque classe;
-Trouver la classe la plus fréquente;
-Construire la règle qui associe cet attribut à la classe la plus fréquente trouvée;
Fait;
Fait;
Fin.

Dans des domaines d'application simples, il donne de bons taux de précision.

4.1.3. Les tables de décision : [KOH 1995]

Les tables de décision sont utilisées pour décrire sous forme tabulaire toutes les conditions possibles d'une décision, les actions qui devraient en résulter et leurs relations. Elles permettent de représenter clairement et succinctement les processus de décision complexes et donc de les clarifier, ces tables pouvant être automatiquement converties en programmes. Elles sont maintenant utilisées dans différents domaines, dans l'analyse de systèmes et de processus de décisions. Elles sont particulièrement utiles pour la construction de systèmes à base de connaissances.

Les tables de décision s'appliquent cependant difficilement aux décisions découlant d'un nombre important de conditions, puisque les possibilités de combinaisons connaissent alors une croissance exponentielle.

Pour appliquer la technique, on doit :

1. **Identifier** les différentes conditions influant sur la décision, les conditions sont en fait tous les "Si" reliés à la décision, ces conditions découleront des attributs pertinents des données (par exemple, le sexe ou l'âge d'un individu).
2. **Identifier** toutes les **action**s qui peuvent découler des combinaisons de conditions "Si" c'est-à-dire tous les "Alors" correspondant aux "Si";

3. **Identifier** toutes les **combinaisons possibles** de conditions et le nombre de règles en découlant. Le nombre de règles résulte de la multiplication du nombre de valeurs de chacun des attributs. Ainsi, si nous avons des individus des deux sexes, regroupés en trois groupes d'âge et en trois niveaux de revenus, 18 règles (2 x 3 x 3) sont théoriquement possibles. Au besoin, on peut construire un tableau pour examiner les combinaisons; par exemple, pour l'âge et le sexe, on a six combinaisons possibles et donc 6 règles différentes.
5. **Vérifier** la table avec les experts du processus pour s'assurer qu'elle est complète, sans répétition ou contradiction.
6. **Simplifier** la table de décision en éliminant les règles impossibles et en combinant celles qui ont traits à des conditions qui n'affectent pas vraiment la décision.
7. Au besoin, il peut être nécessaire de construire des **tables secondaires**, si la complexité de la situation le justifie.

4.2 Classification par Arbres de Décision

Un arbre de décision est un outil d'aide à la décision et à l'exploration de données. Il permet de modéliser simplement, graphiquement et rapidement un phénomène mesuré plus ou moins complexe. Sa lisibilité, sa rapidité d'exécution et le peu d'hypothèses nécessaires a priori expliquent sa popularité actuelle. Certains algorithmes sont utilisés pour répartir une population d'individus (de clients par exemple) en groupes homogènes, selon un ensemble de variables discriminantes (l'âge, la situation familiale, la catégorie socio-professionnelle, le revenu annuel, ...) en fonction d'un objectif fixé et connu (chiffres d'affaires, réponse à un mailing, ...). Il s'agit de prédire avec le plus de précision possible les valeurs prises par la variable à prédire (objectif, variable cible, attribut classe, ...) à partir d'un ensemble de descripteurs (variables prédictives, variables discriminantes, variables d'entrées,...) [TUF2005]. Voir figure 2.3 .

Avant d'entamer les algorithmes de classification par arbres de décision, il convient d'introduire les notations suivantes. Soient :

- **S** un échantillon.
- **{1,2,…, *c*}** un ensemble de classes.
- **t** un arbre.

A chaque position **p** de t correspond un sous ensemble de l'échantillon qui est l'ensemble des exemples qui satisfont les tests de la racine jusqu'à cette position. Donc, nous pouvons définir les quantités suivantes :

- **N(p)** est le cardinal de l'ensemble des exemples associés à **p**.
- **N (k/p)** est le cardinal de l'ensemble des exemples associé à **p** qui sont de classe **k**.
- **P (k/p) = [N (k/p) / N(p)]** la proportion d'éléments de classe **k** à la position **p**.

On commence par choisir un attribut parmi les attributs non sélectionnés (les attributs qui ne sont pas encore associés aux nœuds) et on crée un nœud portant un test sur cet attribut. Pour chaque branche de test, on opère le traitement suivant :

1. Si tous les exemples de cette branche sont de la même classe, alors on crée une feuille en lui attribuant la classe correspondante.
2. Dans le cas contraire, on réitère le processus en éliminant l'attribut précédemment considéré, des attributs à sélectionner.

.

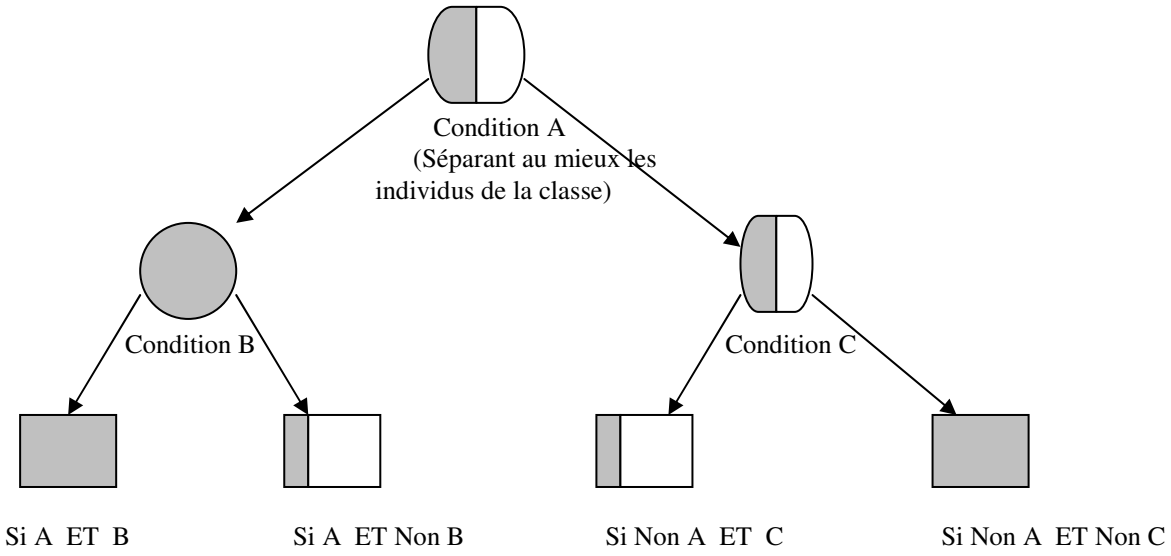


Figure2.3 Illustration d'un arbre de décision

L'ordre dans lequel ces attributs sont sélectionnés est primordial pour la construction de l'arbre. On peut avoir différents arbres résultants de l'apprentissage sur la même base d'exemple, cependant on choisit le plus simple à interpréter.

Le problème de la taille de l'arbre généré dépend du choix de l'attribut racine. Quinlan a proposé une méthode pour traiter ce problème. Il utilise une technique fondée sur la théorie de l'information de Shannon [QUI1993] . L'idée consiste à appliquer une méthode qui permet de maximiser **le gain d'information** apporté par chaque test, donné par :

$$Gain(p,test) = i(p) - \sum_{j=1}^{n} (p_j \ \ i(p_j))$$

Où :
- test : l'attribut choisi.

- **n** : est le nombre des valeurs possibles que peut prendre cet attribut.

- P_j : la proportion d'éléments qui satisfont la $j^{ième}$ branche du test.

- i(p) : est une fonction permettant de mesurer le "degré de mélange" des exemples entre les différentes classes. Une telle fonction doit vérifier la propriété suivante : elle doit prendre son minimum lorsque tous les exemples sont dans une même classe (le nœud est pur) et son maximum lorsque les exemples sont équi-répartis. Il existe différentes fonctions qui satisfont ces propriétés, nous en citerons deux : la fonction entropie et la fonction de Gini [QUIN 1993].

- **Entropie(p) = $-\sum_{k=1}^{c} p(k/p) * \log(p(k/p))$**
- **Gini (p) = $1 - \sum_{k=1}^{c} p(k/p)^2$**

Remarque :
En général, la fonction utilisée pour calculer le gain est la fonction d'entropie.

Ce processus de calcul sera appliqué pour chaque position. Nous allons, dans le paragraphe suivant, parler de la règle majoritaire de classement et présenter le schéma général des algorithmes, puis présenter deux algorithmes particuliers **CART** et **C4.5**.

Règle majoritaire : *Cmaj* est associée à une feuille de classe k de {1,2,…, c} telle que P(k) soit maximale.
Pour toutes les méthodes, nous trouvons les trois opérateurs de construction suivants :

1. **Décider si un nœud est terminal :** c'est-à-dire décider si un nœud doit être étiqueté comme une feuille. Par exemple : tous les exemples sont dans la même classe et il y a moins d'un certain nombre d'erreurs.
2. **Sélectionner un test à associer à un nœud.**
3. **Affecter une classe à une feuille :** on attribue la classe majoritaire sauf dans le cas où l'on utilise des fonctions coût ou risque.

Les méthodes vont différer par les choix effectués pour ces différents opérateurs, c'est-à-dire sur le choix d'un test (par exemple, utilisation du gain et de la fonction entropie) et le critère d'arrêt (quand décider si un nœud est terminal). Le schéma général des algorithmes est donné par le pseudo code suivant.

Algorithme d'apprentissage générique :

```
Entrée : langage de description ; échantillon S ;
Début
    Initialiser l'arbre vide ; la racine est le nœud courant
    Répéter
        Décider si le nœud courant est terminal ;
        Si le nœud est terminal
            Alors
                Affecter une classe ;
            Sinon
                Sélectionner un test et créer le sous-arbre ;
        FinSi
        Passer au nœud suivant non exploré s'il en existe ;
    Jusqu'à obtenir un arbre de décision
Fin.
```

L'arbre construit avec cet algorithme possède une erreur apparente faible, voire nulle car l'algorithme procède de façon descendante sans jamais remettre en question les choix effectués et les feuilles sont étiquetées de telle manière qu'il y ait peu d'erreurs. Mais, l'arbre risque d'avoir un pouvoir de prédiction faible.

L'idée serait de trouver un critère permettant d'arrêter la croissance de l'arbre au "bon moment". Malheureusement, dans l'état actuel des recherches, un tel critère n'a pu être mis au point. De plus, le risque d'arrêter trop tôt la croissance de l'arbre est plus important que de l'arrêter trop tard. Par conséquent, les méthodes utilisées précédemment sont divisées souvent en deux phases :

1. **Expansion :** consiste à appliquer l'algorithme précédent.
2. **Elagage :** consiste à supprimer certains sous arbres pour diminuer l'erreur réelle et la taille de l'arbre.

Un premier algorithme :
CHAID : [KAS1980]

CHAID (**CH**i-squared **A**utomatic **I**nteraction **D**etector) est une technique de type ***arbre de décision***. Elle a été publiée, en 1980, par Gordon V. Kass. Elle peut être utilisée pour la prédiction ou pour la détection d'interaction entre variables. En pratique, elle est souvent utilisée en marketing direct pour sélectionner un groupe de consommateurs et prédire leurs réponses à certaines variables et comment ils affectent d'autres variables. Comme avec les autres arbres de décision, ces avantages sont un résultat essentiellement visuel et facilement interprétable. À cause de la segmentation de la population lors de l'analyse, l'échantillonnage doit être suffisamment large de manière à ce que la taille de chaque groupe ne devienne pas trop petite, ce qui rendrait l'analyse peu fiable.

Un Deuxième algorithme :
CART [BRE1984]

Cette méthode permet d'inférer des arbres de décision binaires. Nous supposons prédéfini un ensemble de tests binaires. Pour définir l'algorithme, nous allons définir les trois opérateurs utilisés par la méthode CART pour calculer un bon arbre de décision (phase d'expansion), puis nous verrons la phase d'élagage. Nous nous plaçons dans le cas d'un échantillon S "assez grand" qui peut être découpé en un ensemble d'apprentissage A et un ensemble de tests T.

Phase d'expansion : en entrée, on dispose l'ensemble d'apprentissage A. La fonction utilisée est la fonction de Gini.

1. **Décider si un nœud est terminal :** Un nœud p est terminal si $\mathbf{Gini(p) \leq i_o}$ ou $\mathbf{n(p) \leq n_0}$, où $\mathbf{i_0}$ et $\mathbf{n_0}$ sont des paramètres à fixer.
2. **Sélectionner un test t à associer à un nœud :** Soit **p** une position et soit ***test*** un test. Si ce test devient l'étiquette du nœud à la position p, alors on appelle $\mathbf{P_{gauche}}$ (respectivement $\mathbf{P_{droite}}$) la proportion d'éléments de l'ensemble des exemples associés à **p** qui vont sur le nœud en position $\mathbf{p_1}$ (respectivement $\mathbf{p_2}$). La réduction d'impureté définie par le test ***t*** est identique au gain est définie par :

$$Gain\ (p, t) = Gini(p) - (P_{gauche} * Gini\ (p_1) + P_{droite} * Gini\ (p_2))$$

3. **Affecter une classe à une feuille :** on attribue la classe majoritaire.

Phase d'élagage :

Soit t l'arbre obtenu en sortie. Pour l'élaguer, on utilise l'ensemble test T. on suppose, en effet, que l'erreur apparente sur T est une bonne estimation de l'erreur réelle. Un élagué de t est obtenu en remplaçant un sous-arbre de t par une feuille. Cette méthode est trop coûteuse en temps de calcul. La phase d'élagage est décrite comme suit :

1. Construction de la suite des arbres : on construit une suite $t_0=t_1,\ldots,t_p$ telle que t_0 soit l'arbre obtenu à la fin de cette phase. Pour tout i, t_{i+1} est un élagué de t_i et le dernier arbre de la suite t_p est réduit à une feuille. Il nous faut définir le procédé de construction de t_{i+1} à partir de t_i. Pour toute position p de t_i, on note $\mathbf{u_p}$ le sous-arbre de t_i en position p. on calcule la quantité :

$$G(p) = \frac{\Delta_{app}(p)}{| \; | \; u_p \| \; | - 1}$$

Où $\Delta_{app}(p)$ est la variation d'erreur apparente mesurée sur l'ensemble d'apprentissage *A* lorsqu'on élague *t* en position *p* et $|u_p|$ est la taille de u_p. On peut remarquer que :

$$\Delta_{app}(p) = \frac{MC(p) - MC(u_p)}{N(p)}$$

Où :
- ***N*(*p*)** est le cardinal de l'ensemble des exemples de *A* associé à la position *p* de t_i.
- ***MC*(*p*)** est le nombre d'éléments de *A* mal classés à la position *p* lorsqu'on élague t_i en position *p*.
- ***MC*(u_p)** est le nombre d'éléments de *A* associés à la position ***p*** de t_i mal classés par u_p.

2. Choix final : on calcule pour chaque arbre t_i de la suite construite, l'erreur apparente sur l'ensemble test T. Cette valeur est prise comme estimation de l'erreur réelle. On retourne donc l'arbre qui minimise l'erreur apparente de T.

<u>Un troisième algorithme</u> : C4.5 [QUI 1993]
1. Décider si un nœud est terminal : Un nœud p est terminal si tous les éléments associés à ce nœud sont dans une même classe ou si aucun test n'a pu être sélectionné.

2. Sélectionner un test à associer à un nœud : à chaque étape, dans l'ensemble des tests disponibles, ne peuvent être envisagés que les tests pour lesquels il existe au moins deux branches ayant au moins deux éléments (cette valeur par défaut peut être modifiée). Si aucun test ne satisfait cette condition alors le nœud est terminal. Soit *p* une position, on choisit alors le test ***test*** qui maximise le gain en utilisant la fonction entropie. La fonction Gain, ainsi définie, privilégie les attributs ayant un grand nombre de valeurs. Elle est donc pondérée par une fonction qui pénalise les tests qui répartissent les éléments en un trop grand nombre de sous-classes. Cette mesure de la répartition est nommée ***SplitInfo*** et est définie par :

$$\text{SplitInfo}(p, \text{test}) = \sum_{j=1}^{n} P'(j/p) \times \log(P'(j/p))$$

Où : - n : nombre de valeurs possibles que peut prendre l'attribut test.

- P'(j/p) : est la proportion des éléments présents à la position *p* prenant la *j*-ème valeur de *test*.

Remarques :

a/ Contrairement à l'entropie, la définition précédente est indépendante de la répartition des exemples à l'intérieur des différentes classes. La valeur de ***SplitInfo*** ne dépend que de la répartition entre les différentes valeurs possibles pour le test.
b/ SplitInfo a des valeurs grandes lorsque le test a un grand nombre de valeurs possibles avec peu d'éléments pour chacune des valeurs. En effet, considérons le cas extrême d'un attribut *n*-aire avec un exemple par classe, la fonction vaut alors log *n*. À l'inverse, considérons le cas d'un attribut binaire pour lequel les exemples sont répartis uniformément entre ces deux valeurs, la fonction vaut alors 1. La nouvelle fonction de gain, appelée **ratio de gain** et notée ***GainRatio***, est alors définie par :

$$\text{GainRatio}(p,T) = \frac{\text{Gain}(p,T)}{\text{SplitInfo}(p,T)}$$

En position *p* (non maximale), on choisit le test qui maximise le GainRatio.

3. Affecter une classe à une feuille : on attribue la classe majoritaire. S'il n'y a aucun exemple on attribue la classe majoritaire du nœud père.

Phase d'élagage

C4.5 utilise l'ensemble d'apprentissage pour élaguer l'arbre obtenu. Le critère d'élagage est basé sur une heuristique permettant d'estimer l'erreur réelle sur un sous-arbre donné. Bien qu'il semble peu pertinent d'estimer l'erreur réelle sur l'ensemble d'apprentissage, il semble que la méthode donne des résultats corrects.

4.3. Classification par Les réseaux de neurones : [HON1996] [HAN2011]

4.3.1 Définition d'un neurone biologique :

Un neurone est composé d'un corps qui contient le noyau ; il est aussi généralement doté d'un axone et d'une dendrite, qui est une structure spécialisée dans la communication avec d'autres neurones. Le corps est responsable de la sommation des informations reçues, la dendrite reçoit les informations et l'axone transite le résultat si la somme dépasse un certain seuil.

4.3.2 Définition d'un neurone formel :

Le neurone formel est une modélisation mathématique qui reprend les principes du fonctionnement du neurone biologique, en particulier la sommation des entrées. Un neurone

est une fonction non linéaire paramétrée. Les variables sur lesquelles opère le neurone sont appelées les entrées du neurone, la sortie de neurone n'est qu'une valeur de la fonction. On peut le définir par cinq éléments :

1. La nature de ses entrées $x_1, x_2, \ldots, x_n$.
2. La fonction d'entrée totale V qui définit le prétraitement effectué sur les entrées :

$$V = W_0 + \sum_{i=1}^{n} w_i x_i$$

3. La fonction d'activation ou état du neurone ***f***.
4. La fonction de sortie **y**, qui calcule la sortie en fonction de son état d'activation : $y = f(V)$.
5. La nature de la sortie.

Une telle définition peut être illustrée par la figure figure 2.4

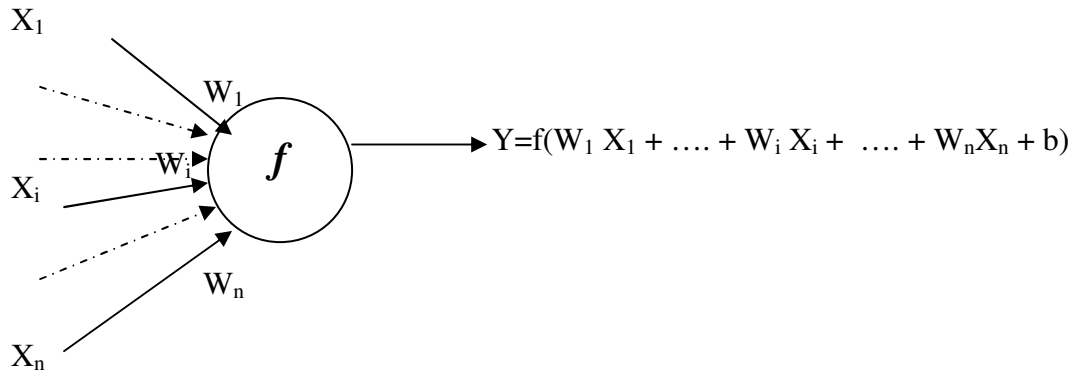


Figure 2.4 Représentation graphique d'un neurone formel.

Où :
- f est une fonction d'activation ou de transfert,
- Xi les variables d'entrée et Wi les poids associés.

4.3.3 Définition d'un réseau de neurones formel

Un réseau de neurones est une composition de la fonction élémentaire constituée par les neurones individuels. Le réseau de neurones est un calcul (algorithme) dont le résultat reproduit ou prévoit de façon plus ou moins précise le comportement d'un processus réel en fonction des entrées qui le déterminent**.**

4.3.4 Les réseaux de neurones dans l'apprentissage supervisé :

Dans ce cas, on s'intéresse aux résultats obtenus en fonction des entrées. Pour cela, on initialise les poids à des valeurs quelconques et on calcule la sortie puis on effectue une comparaison entre les résultats obtenus et les résultats désirés. Cette comparaison détermine donc ***l'erreur de réseau*.** Il s'agit alors de répartir cette erreur sur chaque poids de réseau de manière à réduire l'erreur. La procédure s'arrête lorsque l'erreur est nulle ou très faible, ou bien après un nombre d 'itérations maximum.

Algorithme : "Back Propagation" [HAN2011]

- D : un ensemble de données contenant les tuples d'apprentissage et leurs valeurs cibles associées.
- l : un taux d'apprentissage

- Output : un réseau de neurones ayant acquis l'apprentissage.

(1) Initialiser tous les poids et biais du réseau.
(2) Tantque Critère d'arrêt non atteint
{ Pour chaque Tuple X dans D
{ Pour chaque unité d'entrée j d'un neurone
{ Oj = lj
Pour chaque couche cachée ou sortie d'un neurone j
{ $lj = \sum_i w_{ij}\ o_i + \theta_j$ //Calculer l'entrée du neurone j à partir de son
//prédécesseur i
$Oj = \frac{1}{1 + e^{-l_j}}$ //Calculer la sortie de chaque unité j

// Propager en arrière les erreurs
Pour chaque Unité j dans la couche de sortie
$Errj = Oj\ (1 - Oj)*(Tj - Oj)$
Pour chaque Unité j dans les couches cachées allant de la dernière vers la première
$Err_j = O_j * (1-O_j) * \sum_k Err_k * w_{jk}$ // calculer l'erreur à partir de la
//couche supérieure k

Pour chaque poids w_{ij} dans le réseau
{ $\Delta_{wij} = l * Err_j * O_i$
$w_{ij} = w_{ij} + \Delta_{wij}$ //Incrémenter le poids
}

Pour chaque biais θ_j dans le réseau
{ $\Delta \theta_j = l * Err_j$
$\theta_j = \theta_j + \Delta \theta_j$ // Incrémenter le biais
}

}
}

Les poids du réseau sont initialisés avec des nombres aléatoires petits, par exemple appartenant à l'intervalle [-1, +1] ou bien [-0.5 , +0.5] .Les biais sont également choisis petits, aléatoirement.

En application à la classification, soient :
$X_i = <P_1, P_2, \ldots P_m> : C_j$
Xi est un individu caractérisé par plusieurs attributs, chacun pouvant prendre m valeurs, sa classe est supposée être C_j.
Le but étant d'apprendre la fonction $F(X_i , P) = C_j$ pour chaque X_i appartenant à la base de données d'apprentissage.
La couche d'entrée : Ensemble des caractéristiques des individus déja classés (données d'apprentissage)
La couche de sortie : un ou plusieurs neurones indiquant la classe affectée à l'individu.
Les couches cachées : Pour chaque tuple de données d'apprentissage on propage le tuple dans le réseau (de couche en couche). On rajoute ensuite les poids des connexions afin de minimiser l'erreur d'apprentissage.

Utilisés pour la classification, la complexité de cet algorithme se présente en O(K*n*maxIterations), où K est le nombre d'attributs caractérisant un individu à classer, n est le nombre d'individus à classer et maxIterations le nombre maximum d'itérations à déterminer empiriquement.

Les avantages

- Méthode robuste au bruit,
- Classification ou estimation rapide une fois le réseau construit,
- Disponible dans tous les logiciels de fouille de données.

Les Inconvénients

- Boîte noire: difficile d'interpréter la classification d'un individu,
- Temps d'apprentissage important,
- Difficulté de choix des paramètres.

4.4 Classification par les machines à vecteurs de support [HAN2011] (Support Vector Machines : SVM)

Une tâche de classification implique généralement la séparation des données en jeux d'apprentissage et de test. Chaque instance de l'ensemble d'apprentissage contient une valeur cible, ie : les étiquettes de classe) et plusieurs attributs, ie : les fonctions ou variables observées.

Cette méthode a été proposée par Vladimir Vapnik et ses collègues B. Boser et I. Guyon en 1992 [VBG 1992]. Elle vise à faire la distinction entre deux ou plusieurs ensembles de points par un hyperplan séparateur appelé frontière de décision. Elle repose sur l'existence d'un classificateur linéaire dans un espace approprié.
Cette méthode est très prometteuse pour la classification de données linéairement et non linéairement séparables. Ceci est dù à sa grande capacité à modéliser les données. Lorsque ces données sont linéairement séparables, elle vise à déterminer une droite qui sépare les données, sinon elle a recours à une transformation non linéaire des données d'apprentissage originales pour transformer ces données dans un autre espace de manière à ce qu'elles deviennent linéairement séparables.

4.4.1 Principe de fonctionnement

L'objectif des SVMs est de trouver un classificateur qui sépare les données entre classes en maximisant la marge entre elles. En effet , il existe une infinité d'hyperplans séparateurs mais l'objectif des SVMs est de trouver l'hyperplan optimal.
L'hyperplan optimal doit non seulement passer entre les points des classes mais il faut que la distance minimale entre l'hyperplan et les points qui lui sont les plus proches (marge) soit maximale, d'où l'appelation "Séparateurs à Vaste Marge".

La figure fig. 2.5 suivante illustre deux exemples de classification, celui de gauche a une faible marge entrainant une erreur de classification et celle de droite possède une marge optimale.

4.4.2 Cas des données séparables linéairement

Dans un espace à deux dimensions, les données sont dites linéairement séparables s'il existe une droite pouvant les séparer de part et d'autre. En fait il y a une infinité de droites que l'on peut tracer mais le grand but est de trouver une droite "optimale" pouvant séparer au mieux les tuples de la classe 1 des tuples de la classe 2, ie : sans erreurs ou avec une erreur de classification minimale. Il est à noter que la notion de généralités n'est pas perdue en parlant de deux classes, pour trois classes il faudra trouver le meilleur plan et à N classes il s'agit de trouver le meilleur hyperplan.
Dans l'exemple de la figure 2.5**,** à gauche on montre une marge séparant les tuples des deux classes qui n'est pas large alors que à droite la marge est plus large.
Le but est donc de trouver l'hyperplan avec la marge la plus large possible ("Max Marginal Hyperplan" ou MMH).

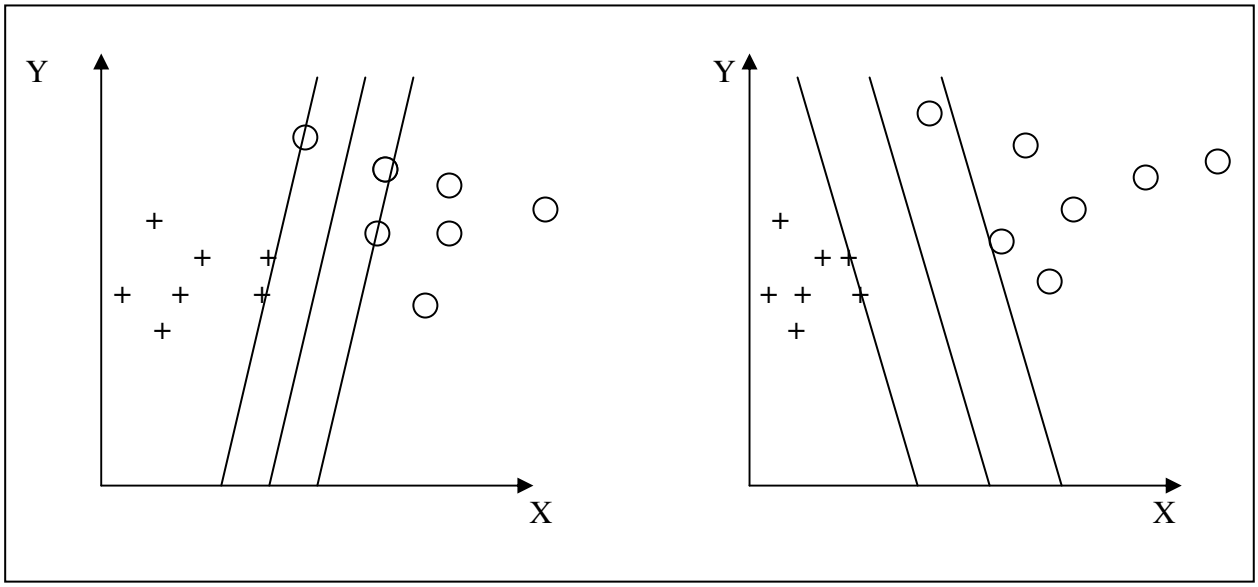


Figure 2.5 Exemple de Séparations des Classes avec différents Hyperplans

Etant donné un ensemble d'apprentissage $(x_i , y_i)^{l}_{i=1}$ où chaque x_i appartient à R^m et y_i appartient à {-1, +1}, ces derniers définissent la classe de chaque exemple d'apprentissage donné.
Pour construire l'hyperplan optimal, considérons l'ensemble suivant :

$$D=\{(x_i,y_i) \in R^m \times \{-1, +1\} \text{ pour } i= 1,\ldots,m\}$$

Soit $(H = w . x + b)$ l'hyperplan qui peut satisfaire les conditions suivantes :

$$\begin{cases} H_1 : W . x_i + b \geq 1 & \text{si } y_i=+1 \\ H_2 : W . x_i + b \leq 1 & \text{si } y_i =-1 \end{cases}$$

W représente un vecteur de poids à ajuster et b est une constante qui peut être assimilée à un poids w_0 .
Ce qui est équivalent à :

$y_i . (w . x_i + w_0) \geq 1$ pour i=1,….,m

Tout tuple se trouvant dans la zone de H1 appartient à la classe +1 et tout autre tuple dans la zone de H2 appartient à la classe -1

Maximiser la marge M revient à maximiser la distance entre les hyperplans canoniques H_1 et H_2.

$$M = \min_{xi/yi=1} \frac{w.x + b}{\|w\|} - \max_{xi/yi=-1} \frac{w.x + b}{\|w\|}$$

$$M = \frac{1}{\|w\|} - \frac{-1}{\|w\|} \quad \text{d'où} \quad M = \frac{2}{\|w\|}$$

Où $\|w\|$ est la norme euclidienne du vecteur w
$\|w\| = (w_1^{\,2} + w_2^{\,2} + \ldots + w_n^{\,2})^{1/2}$

Déterminer l'hyperplan optimal revient à maximiser M ou bien à minimiser le rapport $\|w\|^2/2$
Ce problème est le problème primal, pour le résoudre on peut passer au dual.
Pour cela il suffit de résoudre le Lagrangien correspondant.

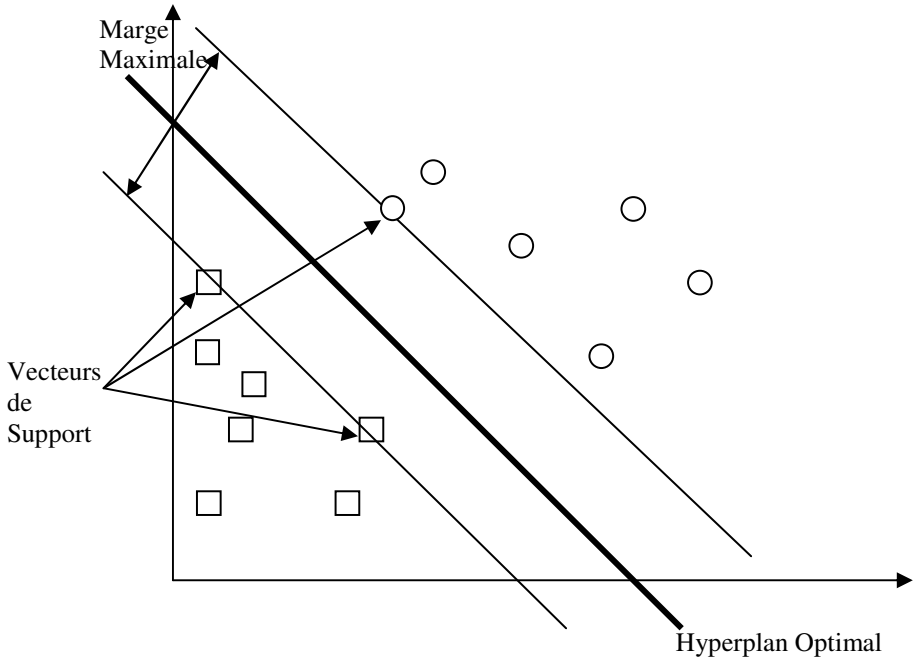


Figure 2.6 Hyperplan optimal et Vecteurs de Support

Les vecteurs de support sont les points qui appartiennent aux hyperplans canoniques. Leur nombre est généralement très petit devant le nombre des données d'apprentissage.

Remarques :

1- D'un point de vue mathématique, les vecteurs de support sont les points qui annulent le Lagrangien.

2- Soit :

$$f(x) = \langle w, x_i \rangle + b$$

$$= \Sigma^2_{i=1} \langle w, x_i \rangle + b$$

Où : $w \in IR^2$;

$b \in IR$ sont des paramètres ;

$x \in IR$ est une variable.

Pour décider à quelle catégorie un exemple x* appartient , il suffit de prendre le signe de la fonction de décision :

$$y = signe(f(x^*))$$

En orientant l'hyperplan (en fixant un côté pour lequel les exemples sont classés positivement), la règle de décision revient à observer de quel coté de l'hyperplan se trouve l'exemple x*. Le vecteur w définit la pente de l'hyperplan : w est perpendiculaire à l'hyperplan et le terme b quant à lui permet de translater l'hyperplan parallèlement à lui-même.

4.4.3 Cas des données non séparables linéairement

L'idée retenue dans les SVM est de trouver un "mapping" ou "transformation" de l'espace linéairement séparables. La figure 2.7 donne une représentation imagée de ce genre de "mapping". Nous pouvons alors appliquer une méthode à marge maximale dans l'espace transformé et garder le contrôle (qui est indépendant de la dimension de l'espace dans lequel nous travaillons) que nous avions sur la capacité. La dimension de l'espace transformé est généralement très élevée. Cela ne pose pas de problème pour notre classificateur à marge maximale vu que sa formulation duale fixe le nombre des variables à déterminer en fonction de la taille de l'ensemble d'apprentissage. Nous noterons l'espace transformé F, et la fonction de "mapping" vers cet espace ϕ, définie de X vers F.

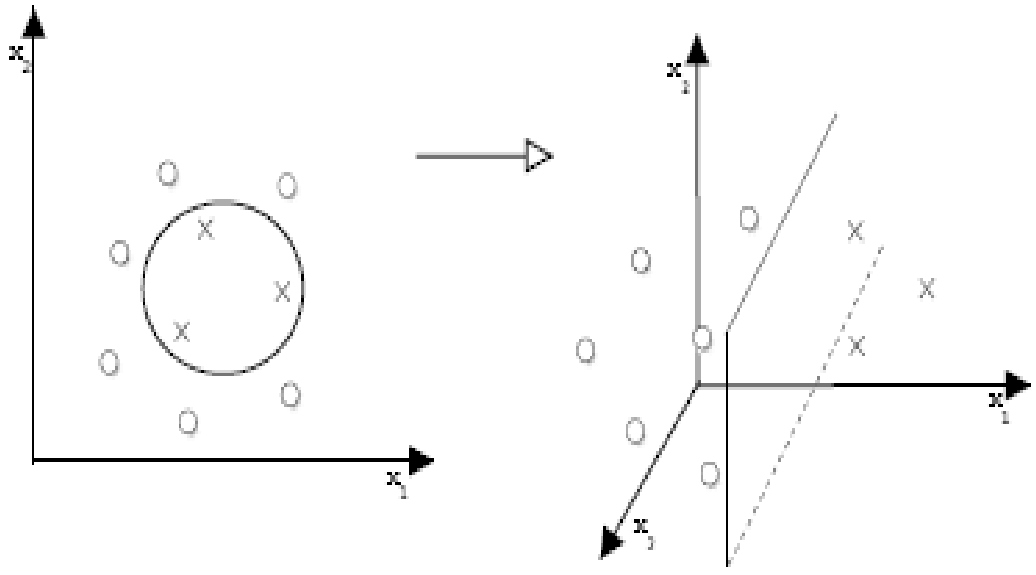


Figure 2.7 Un "mapping " ϕ rendant les exemples linéairement séparables.

Remarque :

Comme on peut le remarquer à la **figure 2.7** l'espace transformé est de dimension supérieure à celle de l'espace de départ. Les nouveaux axes contiennent une surgénération d'informations par rapport aux précédents, ce qui permet idéalement d'effectuer une discrimination linéaire là où auparavant ce n'´etait pas possible.

On peut étendre l'approche linéaire au cas non linéaire.
Considérons l'exemple suivant :

$X = (x_1, x_2, x_3)$ un vecteur à 3 dimensions qui peut être transformé (ou mappé) dans un espace à 6 dimensions Z par :

$$\phi_1(x) = x_1 \qquad \phi_2(x) = x_2 \qquad \phi_3(x) = x_3$$
$$\phi_4(x) = (x_1)^2 \qquad \phi_5(x) = (x_1 . x_2) \qquad \phi_6(x) = (x_1 . x_3)$$

Un hyperplan de décision Z dans le nouvel espace est :

$$d(Z) = w. Z + b$$

w et Z étant des vecteurs, cela devient donc linéaire. On résout cette équation de manière à ce que l'espace linéaire Z corresponde à un polynome non-linéaire du second ordre correspondant à l'espace 3D d'origine.
Les transformations linéaires sont décrites par :

$$d(Z) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 (x_1)^2 + w_5 (x_1 x_2) + w_6 (x_1 x_3) + b$$
$$= w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_5 + w_6 x_6 + b$$

Où w_i représentent des poids affectés aux dimensions de l'espace de départ.

On parle alors de fonctions "Kernel" définies par :
$K(X_i , X_j) = \phi(X_i) . \phi(X_j)$
On peut remplacer $\phi(X_i) . \phi(X_j)$ par $K(X_i , X_j)$ partout dans l'algorithme d'apprentissage.
Trois fonctions Kernel admissibles sont données par :

1- Polynomiale de degré h :

$$K(X_i , X_j) = (X_i + X_j + 1)^h$$

2- Gaussienne ou RBF(Radial Basis Function):

$$K(X_i , X_i) = e^{- \|Xi - Xj\|^2 / 2\sigma^2}$$

3- Sigmoide :

$$K(X_i , X_j) = \tanh(k . X_i . X_j - \delta)$$

k et δ sont des constantes.

4- Composée de noyaux, à définir selon le domaine d'application:

$$K(X_1 , X_2) = K_1(X_1 , X_2) + K_2(X_1 , X_2)$$
$$K(X_1 , X_2) = \alpha . K_1(X_1 , X_2) \text{ avec } \alpha \in R^+$$
$$K(X_1 , X_2) = K_1(X_1 , X_2) . K_2(X_1 , X_2)$$

La classification supervisée par SVM donne des résultats précis lorsque le nombre de données est relativement faible (moins de 2000 données).
Le modèle reste cependant incompréhensible.

4.5 Classification par Réseaux Bayésiens [HAN2011]

Ce type de classification est basé sur les statistiques. Les classificateurs bayésiens peuvent prédire l'appartenance d'un tuple donné à une classe particulière.
La classification bayésienne est basée sur le théorème de Bayes. Elle a montré de hautes performances telles que la précision et la rapidité de classification de bases de données de grandes tailles, comparables aux résultats obtenus par arbres de décision et réseaux de neurones.
Les classificateurs bayésiens naifs supposent que l'effet de la valeur d'un attribut sur une classe donnée est indépendant de la valeur des autres attributs. Cette hypothèse est appelée "indépendance conditionnelle" de classe. Ceci a été fait dans le but de simplifier les calculs et pour cela on l'a appelée " naïve".

Les réseaux Bayésiens représentent des modèles graphiques qui permettent la représentation des dépendances entre des sous ensembles d'attributs, contrairement aux réseaux naifs bayésiens. Ils sont basés sur le théorème de Bayes :

$$P(H/X) = \frac{P(X/H) * P(H)}{P(X)}$$

H est l'hypothèse ie : la classe à laquelle pourrait appartenir X.

4.5.1 Principe de Fonctionnement d'un réseau bayésien naif pour la classification

Un réseau bayésien naif ou simple pour la classification fonctionne de la manière suivante :

1- Soit D un ensemble de tuples d'apprentissage contenant le label de leur classe chacun. Chaque tuple est de la forme : $X = (x_1 , x_2 ,….., x_n)$, les x_i représentent n mesures des n attributs $A_1 , A_2 , …, A_n$.
2- Supposons que pour cet esemple, il existe m classes $C_1 ,C_2 , …..,C_m$.
Pour un tuple X donné, le classificateur bayésien va prédire l'appartenance de X à la classe qui a la plus haute probabilité conditionnelle de X. Pour cela, le classificateur bayésien naif prédira X appartenant à la classe C_i si et seulement si :

$$P(C_i /X) \geq P(C_j/X) \quad \text{pour tout} \quad 1 \leq j \leq m \quad \text{avec} \quad j \neq i$$

La classe C_i pour laquelle $P(C_i /X)$ est maximale est appelée "Maximale Hypothèse a postériori" . Utilisant le théorème de Bayes on a :

$$P(C_i / X) = \frac{P(X/ C_i) * P(C_i)}{P(X)}$$

3- Sachant que P(X) est constante pour toutes les classes, il suffit de maximiser le produit $P(X/ C_i) * P(C_i)$.

On suppose que toutes les probabilités des classes sont égales :

$P(C_1) = P(C_2) =\ldots\ldots= P(C_m)$

et que nous voulons maximiser le produit $P(X/ C_i) * P(C_i)$.

En fait :

$$P(C_i) = | C_{i,D} | / | D|$$

Avec : $| C_{i,D} |$ = nombre de tuples X appartenant à la classe C_i dans la base D

$| D |$ = nombre total de tuples dans la base D.

4- Pour les bases de données apprentissage contenant un grand nombre d'attributs décrivant les tuples X, les temps de calcul de $P(X/C_i)$ deviennent très grands. Dans le but de réduire ces temps de calcul, nous prenons la formule de Bayes concernant l'indépendance conditionnelle des classes :

$$P(X/C_i) = \prod_{k=1}^{n} P(X_k / C_i) = P(X_1 / C_i) * P(X_2 / C_i) *\ldots.. * P(X_n / C_i)$$

Nous pouvons facilement estimer les probabilités $P(X_1 / C_i) * P(X_2 / C_i) *\ldots.. * P(X_n / C_i)$ en comptant les tuples vérifiant X appartenant à C_i dans la base d'apprentissage.

Dans le cas continu, nous pouvons estimer ces probabilités par le calcul de leur distribution, assimilée à une loi gaussienne :

$$G(x, \mu, \sigma) = \frac{1}{(2 \Pi \sigma)^{1/2}} * e^{\frac{-(X - \mu)^2}{2 * \sigma^2}}$$

Où : μ est la moyenne de X et σ est la variance de X.

Dans ce cas :

$$P(X_k / C_i) = g(X_k , \mu_{Ci} , \sigma_{Ci})$$

5. La classification non supervisée

Elle tend à découvrir des populations homogènes à l'intérieur d'une population hétérogène. Le but est de découvrir ces populations homogènes, appelées couramment "Segments" ou "Clusters".

5.1 Les réseaux de neurones dans l'apprentissage non supervisé :

Dans ce cas, une entrée au réseau est présentée et on le laisse évoluer librement jusqu'à ce qu'il se stabilise. Donc, la règle d'apprentissage n'est pas en fonction du comportement de la sortie du réseau, mais plutôt du comportement local des neurones. Cela simplifie considérablement le problème du choix des poids synaptiques appropriés. Par conséquent,

l'apprentissage non supervisé modifie les poids du réseau en fonction d'un critère interne, indépendant de l'adaptation entre le comportement du réseau et la tâche qu'il doit effectuer.

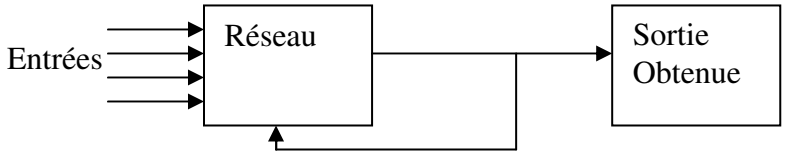


Figure 2.8 Illustration de l'apprentissage non supervisé par les réseaux de neurones.

5.2 La Segmentation (clustering) : Les techniques de clustering cherchent à décomposer un ensemble d'individus en plusieurs sous ensembles les plus homogènes possibles. On ne connaît pas la classe des exemples (nombre, forme, taille). Les techniques utilisées sont :

1. Les méthodes de partitionnement.
2. Les méthodes hiérarchiques.

5.2.1 Les méthodes de partitionnement :

Construire une partition en classes d'un ensemble d'objets ou d'individus.

Principe :

- ✓ Création d'une partition initiale de K classes,
- ✓ Puis itération d'un processus qui optimise le partitionnement en déplaçant les objets d'une classe à une autre.

Elles contiennent plusieurs types de méthodes. Nous citons les méthodes approchées/heuristiques qui contiennent aussi d'autres méthodes.

5.2.2 La méthode de K-Moyennes (centres mobiles) :

Principe :

- Une classe est représentée par son centre de gravité.
- Un objet appartient à la classe dont le centre de gravité est le plus proche.

Cette méthode est basée sur le calcul de distances entre un individu et le centre de gravité de chaque classe.

Algorithme :

- Entrée : un échantillon de m objets $x_1, \ldots, x_m$
- 1. Choisir k centres initiaux $c_1, \ldots, c_k$
- 2. Répartir chacun des m objets dans le groupe i dont le centre c_i est le plus proche.
- 3. Si aucun élément ne change de groupe alors arrêt et sortir les groupes.
- 4. Calculer les nouveaux centres : pour tout i, c_i est la moyenne des éléments du groupe i.
- Aller à 2.

Cette méthode est simple à implémenter et à comprendre. Sa complexité est en O(T.K.N) où T est le nombre d'itérations , K le nombre de centres et N le nombre d'exemples. Il est à noter que généralement K et T sont nettement inférieurs à N.
Cependant, elle nécessite la spécification du nombre de clusters à l'avance. D'autre part, elle est basé sur le calcul de moyennes donc elle est utilisable seulement quand la notion de moyenne existe (donc pas sur des données nominales) et elle reste sensible aux conditions d'initialisation (on n'obtiendra pas forcément les mêmes classes avec des partitionnements initiaux différents). Les points isolés sont aussi mal gérés : doivent-ils appartenir obligatoirement à un cluster ?
Une variante de la K moyenne est la méthode de K-Medoids qui possède le même principe que la précédente mais on utilise à la place de la moyenne le medoid. Un medoid est l'objet le plus central dans la classe.

5.2.3 Les méthodes hiérarchiques

Les clusters sont organisés sous la forme d'une structure d'arbre. Il existe deux familles d'approches hiérarchiques :

a/ Approche ascendante (ou agglomération) :

• Commencer avec un objet dans chaque classe,
• Puis fusionner successivement les 2 classes les plus proches,
• S'arrêter quand il n'y a plus qu'une classe contenant tous les exemples.

Algorithme général de la classification ascendante hiérarchique :

Etablir la table TD des valeurs de D(x, y), x et y parcourant S (S est l'ensemble d'exemples).
Tant que la table TD a plus d'une colonne
Faire
Choisir les deux sous-ensembles h_i, h_j de S tels que D (h_i, h_j) est le plus petit nombre réel dans la table TD;
Supprimer h_j de la table, remplacer hi par (h_i Ụ h_j)
Calculer les mesures de similarité D entre (h_i Ụ h_j) et les autres éléments de la table.
Fait.

Remarque :
Les mesures de similarité pour calculer D (h_i, h_j) sont nombreuses :
1) **Lien simple :** distance des deux objets les plus proches des clusters.
2) **Lien complet :** distance des deux objets les plus éloignés dans les deux clusters.
3) **Lien des moyennes :** la distance entre deux objets est donnée par la distance des centres.
4) **Lien moyen:** distance moyenne de toutes les distances entre un membre d'un cluster et un membre de l'autre.

b/ Approche descendante (ou par division)
• Tous les objets sont initialement dans la même classe,

• Puis division de la classe en sous classes jusqu'à ce qu'il y ait suffisamment de niveaux ou que les classes ne contiennent plus qu'un seul exemple.

Les méthodes hiérarchiques sont faciles à implémenter : elles fournissent une structure facile et compréhensible pour une analyse détaillée. Cependant, leur complexité est évaluée en $O(N^2)$. Il est donc à noter que ces méthodes ne sont pas extensibles pour des ensembles de données de grande taille.

6. Evaluation d'une règle de classification : [AGR 1994]

La qualité d'une règle, une fois établie, est à évaluer avec grand soin, car le plus important pour les utilisateurs de ces règles est qu'elles soient pertinentes, intéressantes et plus lisibles par rapport à son langage et plus précises par rapport à son domaine. Dans [FRE2000], on définit 4 mesures permettant de formuler d'autres mesures, et qui sont :

- TP (True Positive) : nombre d'exemples qui satisfont A et C.
- FP (False Positive) : nombre d'exemples qui satisfont A et non C.
- FN (False Negative): nombre d'exemples qui ne satisfont pas A mais satisfont C.
- TN (True Negative) : le nombre d'exemples qui ne satisfont ni A ni C.

A partir de ces quatre mesures, nous pouvons formuler toutes sortes de mesures de qualités de règles extraites.
Dans la littérature, nous avons trouvé de nombreux critères d'évaluation de qualité dont nous citerons quelques uns. Un panorama de critères pour mesurer la qualité d'une règle de classification peut être trouvé dans [FRA2003] et dans [KHA2006].

Nous nous contentons ici de citer les plus importants.

Support : le support d'une règle R, noté : ***Supp* (R)** représente la proportion d'entrées de la base B contenant R.

$$\text{Supp}(R) = \frac{\text{Nombre d'instances de B qui vérifient au moins une partie de R}}{\text{Nombre total d'instances de B}}$$

Confiance: la confiance d'une règle R, notée : ***Conf* (R)** représente la probabilité que la règle R prédise la classe C sachant A, l'antécédent de R, est vérifié.

$$\text{Conf}(R) = \frac{\text{Supp}(R)}{\text{Supp}(A)}$$

Couverture : la couverture (qui est la contrainte de la confiance) de la règle R, notée ***couv* (R)** représente le nombre d'instances dans la base qui vérifient la partie condition (A) (peut importe la classe).

Spécifité : la spécifité représente le taux de précision d'une règle et cela par le calcul du taux suivant :

$$\text{Spé}(R) = \frac{TN}{(TN + FP)}$$

- TN est aussi le nombre d'exemples n'appartenant pas à C et non prédits dans C par le modèle.
- FP est aussi le nombre d'exemples n'appartenant pas à C et prédits dans C par le modèle.

La précision : la précision de la règle R, notée ***PR* (R)** représente le nombre d'instances dans la base qui vérifie la partie condition (A) et qui ont comme classe C.

$$Pr(R) = \frac{TP}{(TP + FP)}$$

- TP (True Positive) : le nombre d'exemples appartenant à C et prédits dans C par le modèle

Les méthodes de classification utilisent en général des approches basées sur l'apprentissage. Ceci consiste à utiliser un ensemble de données constituées à partir d'expériences réelles pour pouvoir soit extraire un modèle de classification ou bien classer un nouvel individu dans la classe jugée la plus pertinente. La technique utilisée consiste à partager l'ensemble d'apprentissage en deux : l'un pour l'apprentissage proprement dit et l'autre pour les tests. En effet, dans les méthodes qui produisent un modèle de classification, la construction du classificateur a toujours lieu en deux phases : la première consiste à extraire les règles et la seconde à évaluer la pertinence de ces règles sur le deuxième sous ensemble de données auquel on a supprimé la classe. Si la classe prédite par le modèle est la même que la classe réelle des instances de test alors le modèle est précis et validé, sinon tout dépend du taux d'instances de test correctement classées par le modèle.

7. Conclusion

En résumé, l'extraction de règles de classification se fait de manière approchée car il n'existe pas une méthode précise qui nous donne des règles pertinentes, utiles, intéressantes et plus lisibles par rapport à son langage, mais l'utilisation de critères favorise un critère par rapport à l'autre par exemple la confiance d'une règle doit être élevée. La prise en considération de plusieurs critères en même temps est nécessaire. Par conséquent, c'est un problème combinatoire auquel nous devons trouver une solution satisfaisante en un temps raisonnable et qu'on pourra améliorer en second lieu en greffant par exemple des méthodes de recherche locale.

Il existe différentes méthodes d'apprentissage dont la complexité diffère. Le choix de la méthode la plus appropriée se fait selon la taille des données et la disponibilité de l'attribution de classification. Cependant, des améliorations doivent être faites pour diminuer le temps de calcul.

Dans les chapitres suivants, nous présentons des définitions et détails sur les algorithmes évolutionnaires comme méthodes de résolution de problèmes, étant donné que nos contributions pour la résolution du problème de la classification sont toutes basées sur ces approches.

1-Introduction :

En 1860 Charles Darwin publie son livre intitulé "*L'origine des espèces au moyen de la sélection naturelle" ou "la lutte pour l'existence dans la nature"*. Dans cet ouvrage, Darwin rejette l'existence "de systèmes naturels figés", déjà adaptés pour toujours à toutes les conditions extérieures et expose sa théorie de l'évolution des espèces : sous l'influence des contraintes extérieurs, les êtres vivants se sont graduellement adaptés à leur milieu naturel au travers de processus de reproductions. Darwin proposa alors une théorie qui clarifie l'évolution des espèces.

Presque simultanément, en 1866, Mendel publie l'article retraçant dix années d'expériences d'hybridation chez les végétaux (recombinaison des gênes) et l'adresse aux sociétés scientifiques des quatre coins du monde. Les réactions sont mitigées, voire inexistantes. Le monde scientifique n'est pas prêt à reconnaître la qualité de ses résultats. Ce n'est qu'en 1900 que la publication de trois nouveaux articles signés Hugo de Vries, Carl Correns et Erich Von Tschermark, révèle des résultats similaires à ceux de Mendel et feront que ces premiers seront reconnus.

Dans les années 1960, John Holland étudia les systèmes évolutifs et en 1975 [HOL1975], il introduisit le premier modèle formel des algorithmes génétiques (*the canonical genetic algorithm AGC*) dans son livre *Adaptation in Natural and Artificial Systems*. Il expliqua comment ajouter de l'intelligence dans un programme informatique avec les croisements (échangeant le matériel génétique) et la mutation (source de la diversité génétique). Ce modèle servira de base aux recherches ultérieures et sera plus particulièrement repris par Goldberg qui publiera en 1989 [GOL1989], un ouvrage de vulgarisation des algorithmes génétiques.

Trois types d'algorithmes évolutionnaires ont été développés isolément et à peu prés simultanément, par différents scientifiques : la programmation évolutionniste [FOG 1966], les Stratégies d'évolution [REC 1973] et les Algorithmes Génétiques [HOL 1975].

Dans les années 90, ces trois champs ont commencé à sortir de leur isolement et ont été regroupés sous le terme anglo-saxon "*Evolutionnary Computation"*.

Les algorithmes génétiques sont des algorithmes d'optimisation qui s'appuient sur des techniques engendrées par la génétique et l'évolution naturelle. Ainsi, le vocabulaire employé est directement calqué sur celui de la théorie de l'évolution et de la génétique. Nous parlons donc *d'individus* (solutions potentielles), de *population* (ensemble d'individus donc de solutions), de *gènes* (variables), de *chromosomes* (génotypes), *d'adaptations* (phénotypes), *de parents,* de descendants, de reproduction, de *croisement,* de *mutations,* etc …

2- Définitons

2.1 Eléments de base

Les éléments de base permettant l'élaboration d'un algorithme génétique sont décrits ainsi (voir Figure 3.1):

A- Un principe de codage de l'élément de population. Durant cette étape, il faut choisir une manière adéquate pour représenter les individus de la population, les individus étant en fait chacun une solution possible au problème à résoudre. Cette phase est donc une phase de modélisation mathématique du problème traité, appelée "codage" des données et elle conditionne le succès des algorithmes génétiques. Les codages binaires ont été très utilisés à l'origine puis les codages réels sont apparus et largement utilisés

de nos jours, notamment dans des domaines d'application pour l'optimisation de problèmes à variables réelles.

B- Un mécanisme de génération de la population initiale. Ce mécanisme est appelé à générer une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale suffisamment diversifiée est important car il peut mener à la convergence vers l'optimum global.

C- Une fonction à optimiser. Celle-ci calcule une valeur de R^+ appelée *fitness* ou fonction d'évaluation ou d'adaptation de l'individu, ce dernier étant un N-tuple appelé couramment chromosome. Un chromosome est constitué de N gènes, chaque gène représente un codage d'une des variables du problème à résoudre.

D- Des opérateurs afin de diversifier la population au cours des générations et d'explorer l'espace d'état. L'opérateur de croisement recombine des individus existant dans la population, l'opérateur de mutation garantit l'exploration de l'espace d'états.

E- Des paramètres de dimensionnement : la taille de la population, le nombre total de générations ou le critère d'arrêt, les probabilités d'application des opérateurs de croisement et de mutation.

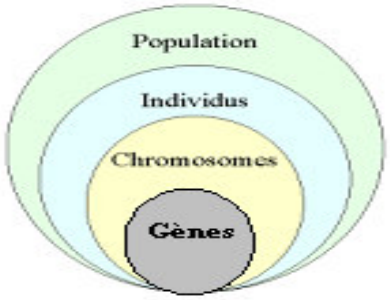


Fig. 3.1 Les quatre niveaux d'organisation de l'algorithme génétique.

2.2 Opérateurs Génétiques

2.2.1- La sélection :

La sélection se base sur la fonction d'adaptation des individus. Elle permet d'identifier les meilleurs individus de la population et d'éliminer partiellement les moins adaptés sans cependant confondre sélection et élitisme : dans la vie un individu correctement adapté à son milieu ne survivra pas nécessairement (les aléas de la vie) et de même un individu non adapté peut ne pas disparaître (la chance aide également à survivre).

On distingue deux catégories de procédures de sélection: les procédures déterministes et les procédures stochastiques.

A- Sélection déterministe

Dans ce type de sélection, les meilleurs individus au sens de la fonction d'adaptation sont sélectionnés. Pour cela, on procède d'abord à un tri de l'ensemble de la population, ce qui risque de poser un problème de temps de calcul pour de très grosses tailles de population. Les individus les moins adaptés sont complètement éliminés de la population et le meilleur individu est toujours sélectionné ; on parle alors d'*élitisme*.

B- Sélection stochastique

Dans ce type de sélection, les meilleurs individus sont favorisés de manière stochastique, ce qui laisse une chance aux individus moins performants. Par contre, il peut arriver que le meilleur individu ne soit pas sélectionné, et que les nouveaux individus peuvent ne pas atteindre une adaptation aussi bonne que celle du meilleur parent.

Il existe différents principes de sélection stochastique, les plus classiques sont cités ci-dessous.

-La sélection par roulette de casino *(wheel selection)* :

Cette sélection est très connue : La roue est divisée en autant de secteurs que d'individus dans la population. La taille de ces secteurs est proportionnelle à l'adaptation de chaque individu. En faisant tourner la roue, l'individu pointé à l'arrêt de la boule est sélectionné. Les individus les mieux adaptés ont donc plus de chance d'être tirés au sort lors du déroulement du jeu.

Pour qu'un individu soit sélectionné, nous calculons d'abord sa probabilité cumulée :

$$Q_j = \sum_{i=1}^{j} p_i$$

La probabilité p_i de sélection d'un individu i est donnée par :

$$p_i = f_i / \sum_{i=1}^{n} f_i$$

n : le nombre d'individus, fi : l'adaptation de l'individu i.

L'individu i est sélectionné si **Q**i-1 **<** r **<= Q**i , r étant un réel tiré aléatoirement dans l'intervalle [0,1].

-Une variante dite *"stochastic remainder without replacement selection"* de la sélection précédente existe et dans laquelle:

- Pour chaque élément i, on calcule le rapport ri de sa fitness sur la moyenne des fitness.
- Soit E (ri) la partie entière de ri, chaque élément est reproduit exactement E(ri) fois.
- La *roulette "Wheel sélection"* précédemment décrite est appliquée sur les individus affectés des fitness (ri – E (ri)).

- La sélection par rang

La sélection par rang est une variante du système de roulette également. Il s'agit la aussi d'implémenter une roulette, mais cette fois les secteurs de la roue sont proportionnels au rang des individus dans la population triée en fonction de la qualité des individus.

D'une manière plus explicite, la population est triée en fonction de la qualité des individus puis on attribue à chacun un rang. Les individus les moins adaptés se "voient" octroyés d'un rang faible (à partir de 1). Chaque individu reçoit donc un rang et on finit par attribuer le rang N au meilleur individu (où N est la taille de la population).

La suite de la méthode consiste uniquement en l'implémentation d'une roulette basée sur les rangs des individus. L'angle de chaque secteur de la roue sera proportionnel au rang de l'individu qu'il représente.

- la sélection par tournoi :

Comme son nom l'indique, cette méthode fait affronter deux ou plusieurs individus afin que le meilleur gagne. Plusieurs variantes existent :

- On peut par exemple faire varier le nombre d'individus qui doivent s'affronter au départ,
- On peut également permettre ou non que le même individu soit éligible plusieurs fois lors d'un même tournoi.

2.2.2 Améliorations classiques dans la sélection :

Les différentes méthodes de sélection présentées ci-dessus étant très sensibles aux écarts des valeurs prises par la fonction d'adaptation (fitness), dans certains cas, un très bon individu risque d'être reproduit maintes fois et peut même provoquer l'élimination complète de ses congénères.

Une population homogène prend forme au bout de quelques itérations, ne contenant qu'un seul type d'individu. Un optimum local risque d'être le seul reproduit pour la génération suivante, voir Figure3.2. La mutation pourra ensuite aider à trouver l'objectif global, mais au prix de nombreuses générations supplémentaires.

Pour éviter ce comportement pénalisant, il a été développé d'autres méthodes de sélection (le *ranking*) et des principes (*scaling*, *sharing*) qui empêchent les chromosomes de haute valeur de fitness d'éliminer complètement ceux de faibles valeurs de fitness.

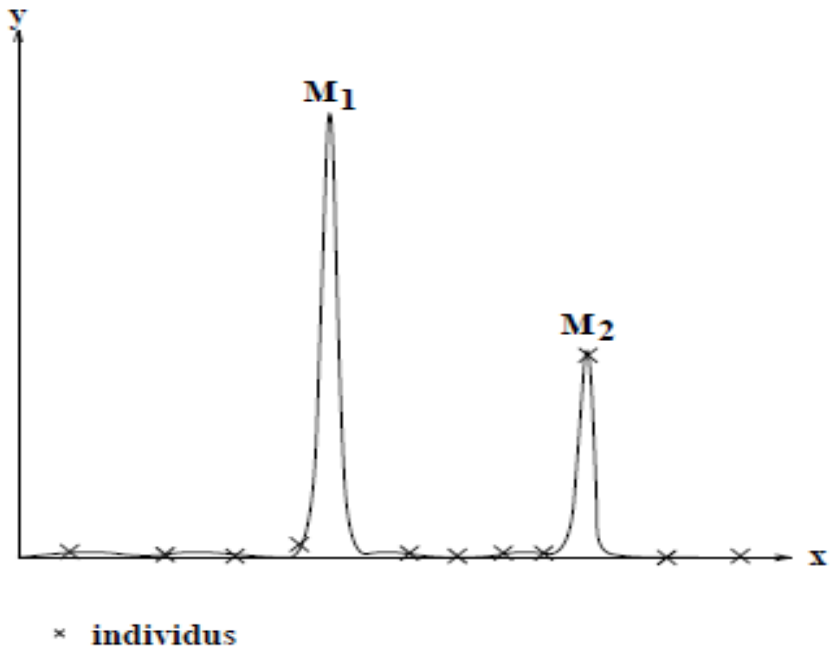


Figure3.2: exemple de population ou les sélections classiques risquent de ne reproduire qu'un individu

Où x sont les individus de la population et y leurs valeurs de fitness. Un individu x de fitness y=M1 risque d'empêcher l'évolution des autres individus car il possède un grand écart de fitness par rapport à ces autres individus de la population.

A- Le scaling :

Les individus à forte valeur de fitness se reproduisent mieux que les moins adaptés et risquent de dominer la population et d'empêcher une évolution équilibrée. Ceci induit une convergence prématurée de l'algorithme qui risque de "tomber" dans un piège local : c'est le cas dans une fonction à fort gradient (à fortes variations). Ceci est illustré par la Figure 3.3.

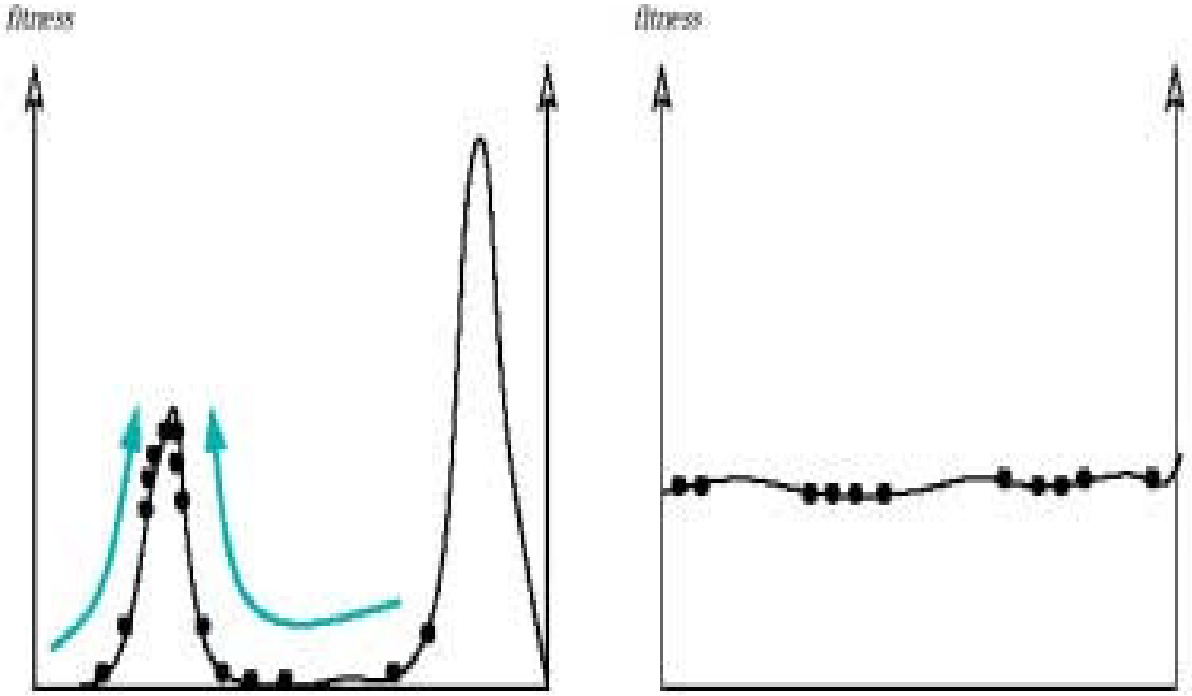


Figure 3.3 Objectif du scaling

Le "*scaling*" ou mise à l'échelle, modifie les valeurs de la fonction fitness afin de réduire ou d'amplifier artificiellement les écarts entre les individus. La sélection n'utilise plus la fitness réelle mais plutot son image aprés scaling. Parmi les fonctions de scaling, on peut envisager le "scaling" lineaire et le "scaling" exponentiel. Soit **fr** la fitness avant scaling et **fs** la fitness modifiée par le scaling.

-scaling linéaire :
Dans ce cas, la fonction de scaling est définie de la façon suivante :

fs = a * **fr** + b

a= (max'- min')/ (max-min)

b= [(min'*max)-(min*max')] / (max-min)

où :

- max et min sont les valeurs maximale et minimale de la fonction de fitness avant scaling ,
- max' et min' sont les valeurs maximale et minimale de la fonction de fitness après scaling.

En règle générale, le coefficient 'a' est inferieur à 1, (a<1) ce qui permet de réduire les écarts de fitness et donc de favoriser l'exploration de l'espace. Ce scaling est statique par rapport au numéro de génération.

-scaling exponentiel :

Il est défini de la façon suivante :

$$fs = (fr)^{k(n)}$$ Où n est la génération courante.

• Pour k proche de zéro, on réduit fortement les écarts de fitness ; aucun individu n'est vraiment favorisé et l'algorithme génétique se comporte comme un algorithme de recherche aléatoire et permet d'explorer l'espace.
• Pour k proche de 1 : le scaling n'est pas fonctionnel.
• Pour k > 1 les écarts sont exagérés et seuls les bons individus sont sélectionnés.

Dans la pratique, on fait généralement varier k des faibles valeurs vers les fortes valeurs au cours des générations. Pour cela on peut utiliser la formule suivante :

$$K = (\tan [(n/(N+1))(\pi/2)])^p$$

n étant la génération courante, N le nombre total de générations prévues, p un paramètre a choisir. Le choix de p = 0.1 s'est avéré pertinent dans les applications. L'évolution de K en fonction de la génération n est donnée par la figure 3.4.

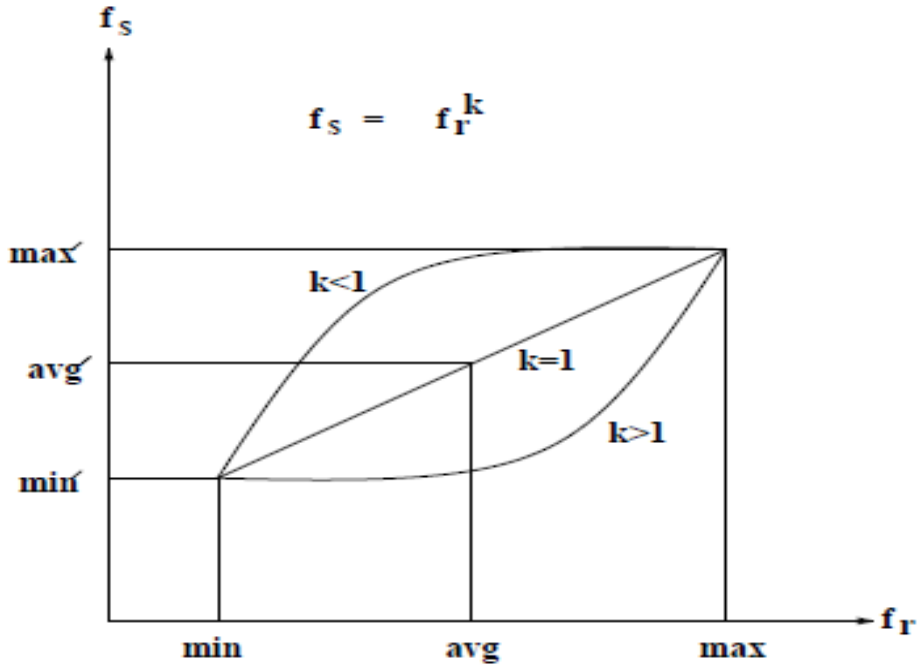


Figure 3.4 Fonction de scaling exponentiel

B - Le sharing :

L'objectif du sharing est de répartir sur chaque sommet de la fonction à optimiser un nombre d'individus proportionnel à la fitness associée à ce sommet. La figure 3.5 présente deux exemples de répartitions de populations dans le cas d'une fonction a cinq sommets : le premier sans sharing, le second avec sharing.

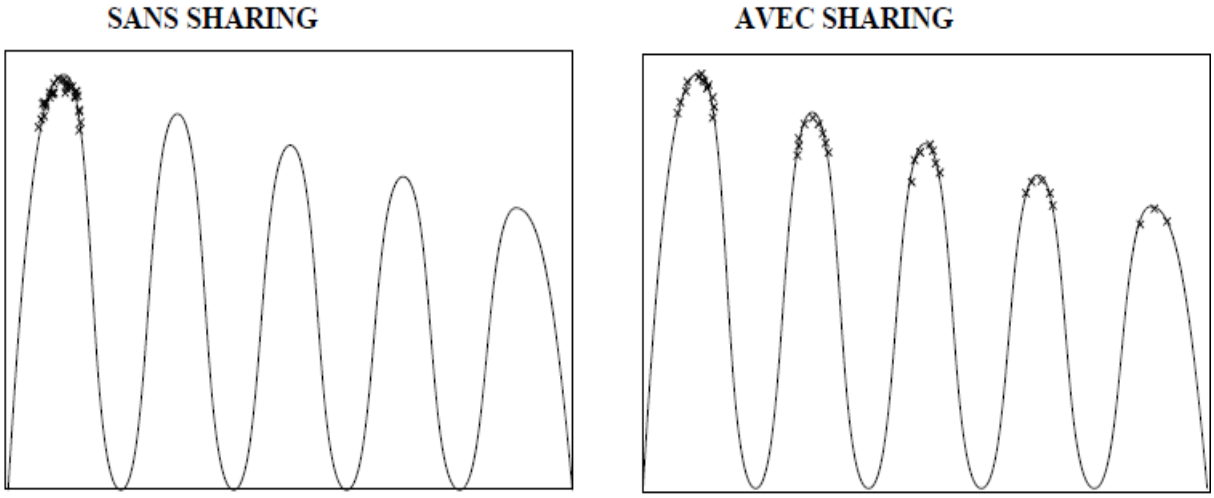


Figure 3.5 Objectif du sharing

Le sharing consiste à modifier la fitness utilisée par le processus de sélection. Pour éviter le rassemblement des individus autour d'un sommet dominant, le sharing impose l'introduction d'une distance. En pratique, on définit une distance de dissimilarité entre deux individus qui sera utilisée pour calculer la nouvelle fitness comme suit :

$$f_i' = \frac{f_i}{m_i'}$$

$$m_i' = \sum_{j=1}^{N} S(d(x_i, x_j))$$

$$S(d) = 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} \text{ si } d < \sigma_{share}$$

$$S(d) = 0 \text{ si } d \geq \sigma_{share}$$

Le paramètre "**σ share**" permet de délimiter le voisinage d'un point et ***dépend du problème traité.*** Suivant la valeur donnée à α empiriquement, le sharing sera plus ou moins efficace. Pour α < 1, on dit que les groupes très agglomérés sont pénalisés.

2.2.3 le croisement ou "crossover" :

Le croisement a pour but de produire de nouveaux individus dans la population en manipulant la structure des chromosomes. Classiquement, les croisements ou reproductions sont faisables à l'aide de deux parents et génèrent deux enfants. Cela consiste à échanger une ou plusieurs parties de gènes des parents afin de donner des enfants qui cumulent des propriétés combinées. Bien qu'il soit aléatoire, cet échange d'informations offre aux algorithmes génétiques une part de leur puissance : quelque fois, de "bons" gènes d'un parent viennent remplacer les "mauvais" gènes d'un autre et créent des enfants mieux adaptés que leurs parents.

On peut noter que le nombre de points de croisements ainsi que la probabilité de croisement permettent d'introduire plus ou moins de diversité.

2.2.4 La mutation

La mutation modifie des individus déjà existants pour créer de nouveaux individus. Une fois de plus, le hasard est très utile dans la mesure où l'on modifie aléatoirement seulement quelques individus de la population, en altérant un ou plusieurs gènes. L'opérateur de mutation modifie de manière complètement aléatoire les caractéristiques d'un individu ce qui permet d'introduire et de maintenir la diversité au sein de la population.

Cet opérateur possède 4 grands avantages :
- garantit la diversité de la population,

- permet d'éviter la ***dérive génétique***. On parle de dérive génétique lorsque quelques gènes sont présents par hasard au même endroit sur tous les chromosomes [GOL1989] et se répandent ainsi au détriment des autres. L'opérateur de mutation entraîne de manière aléatoire des changements au niveau de n'importe quel locus permettant d'éviter ainsi l'installation de cette situation défavorable.

- Dans le cas d'une convergence prématurée, la population est constituée par des individus identiques mais ne sont que des optimums locaux. Le croisement ne pourra rien changer à la situation. L'évolution se retrouve bloquée ou n'atteindra jamais l'optimum global.
La mutation, entrainant des inversions de bits ou des modifications de gènes de manière aléatoire, permet de réintroduire des différences entre les individus.

- La mutation permet de garantir la propriété d'ergodicité qui fait que chaque point de l'espace de recherche soit atteint.

La mutation consiste en une inversion d'un bit si le codage utilisé est un codage binaire, ou bien en un tirage aléatoire d'une valeur du gène concerné dans son domaine de valeurs.

2.2.5 Le remplacement :

Cet opérateur consiste à réintroduire les descendants obtenus par application successive des opérateurs de sélection, de croisement et de mutation dans la population.

On trouve essentiellement 2 méthodes de remplacement différentes:

- Le ***remplacement stationnaire :*** dans ce cas, les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives.
  Cette méthode peut être mise en œuvre de deux manières différentes :
  1- Le ***remplacement générationnel*** se contente de remplacer la totalité de la population par la nouvelle population.
  2- La deuxième méthode consiste à choisir une certaine proportion d'individus de la nouvelle population qui remplaceront leurs parents.

- Le ***remplacement élitiste*** : dans ce cas, on garde au moins l'individu possédant les meilleures performances d'une génération à la suivante. En général, on procède au remplacement de l'élément le moins adapté dans la population courante par l'élément qui vient d'être créé. Ainsi, les enfants d'une génération ne remplaceront pas nécessairement leurs parents comme dans le remplacement stationnaire. Ce type de stratégie améliore les performances des algorithmes évolutionnaires dans certains cas, mais présente aussi l'inconvénient d'augmentation du taux de convergence prématuré.

Néanmoins, des implémentations plus fines procèdent de manière différente. Dans ce cas là, le taux de remplacement n'est pas de 100 %, la taille de la population augmente donc au cours des générations successives, on dit qu'il y a "***overcrowding".*** Il faut donc trouver un moyen pour sélectionner les parents qui seront supprimés de la population. De Jong a proposé la solution suivante : imaginons qu'on veuille remplacer 30 % des parents, soit *np* le nombre de parents correspondants à ce pourcentage, on remplacera les *np* parents les plus proches de leurs descendants de P'. Cette méthode permet donc premièrement de maintenir la diversité et deuxièmement d'améliorer la fitness globale de la population.

2.3 Paramètres des algorithmes génétiques

Les opérateurs de l'algorithme génétique sont guidés par un certain nombre de paramètres fixés à l'avance. Les valeurs données à ces paramètres influencent la réussite ou non d'un algorithme génétique. Ces paramètres sont les suivants :

-La taille de la population N,

-La probabilité de croisement,

-La probabilité de mutation,

-Le nombre maximum de générations ou d'itérations.

Généralement, ces paramètres sont déterminés expérimentalement.

Les algorithmes génétiques ne possèdent pas de problèmes dédiés dans l'ensemble des problèmes d'optimisation. Souvent, des problèmes complexes peuvent entrainer des difficultés de modélisation sans compter la mise en équation de la fonction objectif. Cependant, une fois cette étape effectuée et les operateurs adaptés au problème, sa résolution peut avoir lieu d'une manière satisfaisante et au bout d'un temps adéquat.

En plus de leur robustesse , les AGs possèdent un autre grand avantage : Ils traitent une population de solutions, ce qui introduit du parallélisme.

3- Schémas Parallèles des Algorithmes Génétiques

3.1 Introduction

Les ordinateurs parallèles sont des machines qui comportent une architecture parallèle, constituée de plusieurs processeurs identiques, ou non, qui concourent au traitement d'une application.

La performance d'une architecture parallèle est la combinaison des performances de ses ressources et de leur agencement (Latence, Débit).

L'objectif du parallélisme est d'obtenir de meilleures performances par rapport aux calculateurs classiques séquentiels :

- Pas de limite de mémoire,
- Pas de limite de processeurs,
- Accélération des calculs complexes ou coûteux en temps d'occupation CPU (calcul matriciel, simulation numérique, transformée de fourier...),
- Calcul répétitif sur un large ensemble de données structurées,
- Traitement indépendant.

3.2 Modèles du calcul parallèle : [MOR1996]

Un algorithme consiste en un flot d'instructions à exécuter sur un flot de données. Il existe quatre Modèles de calculs, selon qu'il y a un ou plusieurs flots.

SISD *(Single Instruction Single Data) :*

C'est le modèle séquentiel classique (Von Neumann)

MISD *(Multiple Instruction Single Data)*

Il y a plusieurs flots d'instructions agissant sur un seul flot de données. On a ***n > 1*** processeurs partageant une unique mémoire.

A chaque étape de manière synchrone, une donnée est traitée parallèlement par les ***n*** processeurs qui exécutent chacun une instruction spécifique.

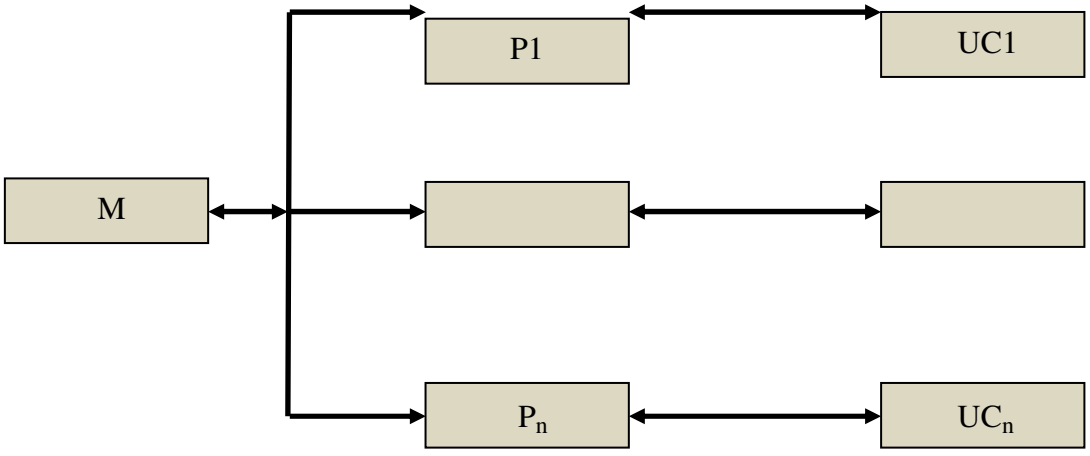


Figure 3.7 Le modèle MISD

***SIMD** (Single Instruction Multiple Data):*

On a *n* processeurs tous identiques ayant chacun une mémoire locale, où l'on peut stocker des données et des instructions.

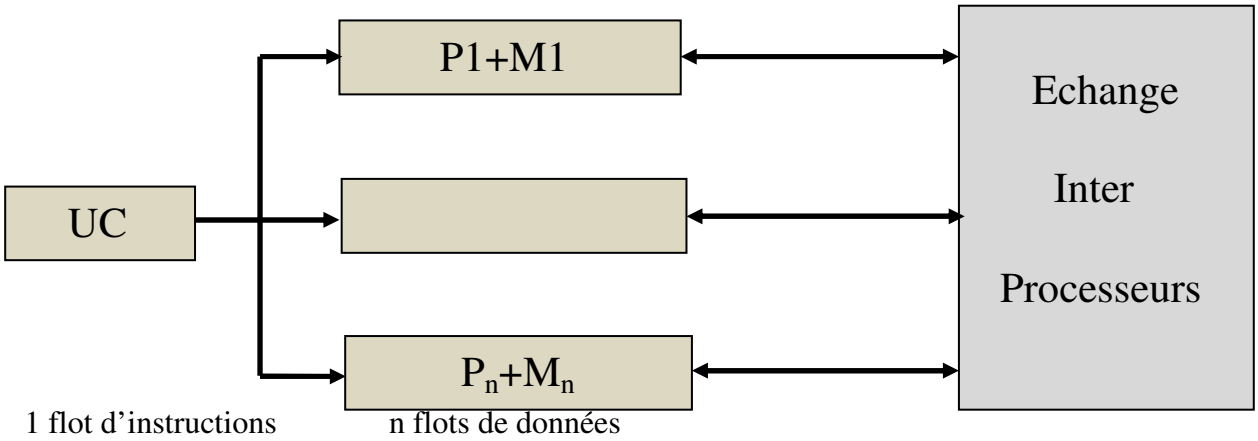


Figure 3.8 Le modèle SIMD

Les processeurs exécutent la même instruction ou le même programme (SPMD) stocké en mémoire locale sur des données différentes. Les processeurs fonctionnent de manière synchrone ; les instructions peuvent être simples ou complexes ; seul un sous-ensemble des processeurs peut avoir à exécuter une instruction.
Ce modèle est efficace surtout si on a un flot régulier sur des données régulières (auxquelles on accède de manière prévisible).

***MIMD**(multiple instruction multiple data) :*

C'est le modèle le plus général. Il y a *n* processeurs avec *m* flots d'instructions distincts et k flots de données distincts. Les échanges inter-processeurs se font par une mémoire partagée ou par un réseau d'interconnexion.

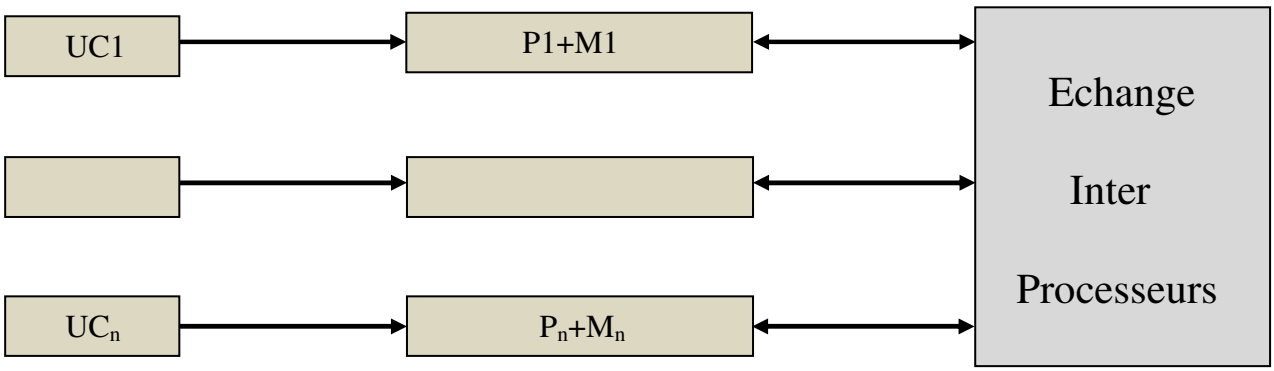


Figure 3.8 Le modèle MIMD

Chaque processeur exécute son propre programme sur ses propres données : chacun est amené à résoudre un sous-problème différent du problème original. Le fonctionnement est asynchrone, sans horloge globale et la coordination des activités se fait via une mémoire partagée ou via un réseau couplé.

3.3 Les architectures de la mémoire :

Les types d'architectures parallèles se basent essentiellement sur les architectures de la mémoire. On en distingue deux types :
- les machines à mémoire partagée
- les machines à mémoire distribuée

3.3.1 Les machines à mémoire partagée :

Les machines à mémoire partagée permettent de réaliser le parallélisme de données et de contrôle.

Le programmeur n'a pas besoin de spécifier la distribution des données sur chaque processeur. Il définit seulement la partie du programme qui doit être parallélisée (directives) et doit gérer les synchronisations. Les principales caractéristiques de cette architecture sont :
- plusieurs processeurs avec des horloges indépendantes,
- une seule mémoire commune à tous les processeurs, Voir la figure 3.9 .

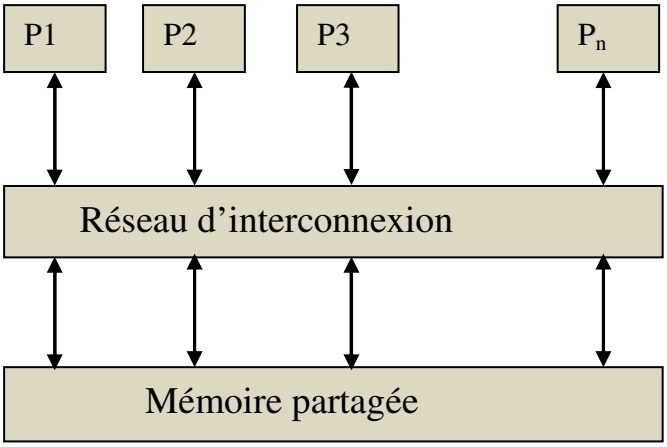


Fig 3.9 Architecture des machines paralleles à mémoire partagée

Avantages :
-simplicité d'utilisation
-portabilité au SMP (architecture homogène à mémoire partagée)
-parallélisation de haut niveau

Inconvénients :
-coût
-limitation du nombre de processeurs (conflit d'accès au niveau matériel)
-la bande passante du réseau est le facteur limitant de ces architectures

3.3.2 Les machines à mémoire distribuée:

-Interconnexion (réseau local rapide) de systèmes indépendants (noeuds)
-Mémoire propre locale à chaque processeur
-chaque processeur exécute des instructions identiques ou non, sur des données identiques ou non.
-Parallélisme par échange de message, voir la figure **3.10**

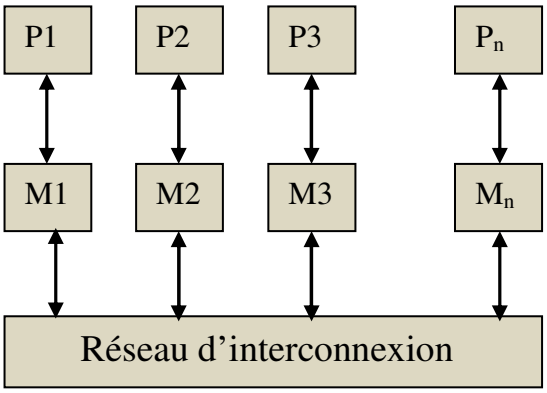


Figure 3.10 Architecture des machines à mémoire distribuée

Avantages :
-non limité théoriquement en nombre de processeurs :
Grande puissance de calcul
-réalisation aisée par la conception de cluster :
Mise en réseau d'un certain nombre de machines
-partage transparent des ressources

Inconvénients :
-plus difficile à programmer que les machines à mémoire partagée
-efficacité limitée
-administration du réseau en plus.

3.4 Analyse des performances des algorithmes parallèles :

Pour un problème de taille ***n***, on note :
- ***t*(*n*)**, le temps d'exécution parallèle au pire des cas pour un problème de taille ***n***.
- ***w*(*n*)**, le temps d'exécution séquentielle (au pire) du meilleur algorithme séquentiel pour le problème de taille ***n***.
- ***p*(*n*)**, le nombre de processeurs pour résoudre en parallèle le problème de taille ***n***.
On introduit le ***SpeedUp*** ou l'accélération de l'algorithme parallèle.

$$SpeedUp\ (n)=w\ (n)\ /t\ (n)$$

On introduit le *coût* de l'algorithme parallèle.

$$c\ (n)= p\ (n)\ \times t\ (n)$$

Si tous les processeurs exécutent exactement le même nombre d'opérations alors ***c(n)*** est égal au nombre total d'opérations. Sinon, ***c(n)*** est un majorant de ce nombre. La différence entre ***c*(*n*)** et ***w*(*n*)** représente ce qu'on appelle ***"overhead"*** dû à la gestion du parallélisme.

$$t_0\ (n) = c\ (n) - w\ (n)\ \geq 0$$

Si ***c*(*n*)** est asymptotiquement équivalent à ***w*(*n*)**, on dira que l'algorithme parallèle est *à coût optimal*. En d'autres termes, un algorithme parallèle n'est pas à coût optimal s'il existe un algorithme séquentiel dont la complexité est asymptotiquement inférieure à son coût.

On introduit également l'efficacité du schéma parallèle, qui est un rendement par :

$$Eff\ (n) = SpeedUp\ (n)/p\ (n) \leq 1$$

Si ***p(n)*** est constant, alors ***t*0(*n*)** est constant. Si ***p(n)*** est constant et *n* croissant, alors ***Eff (n)*** est croissante. On a une granularité plus grosse et des processeurs mieux utilisés. Si *n* est constant et ***p*(*n*)** croissant, alors ***Eff(n)*** est décroissante. Paradoxalement, le meilleur rendement, et non pas le meilleur temps, est obtenu en séquentiel.

3.5 Parallélisation des algorithmes génétiques :

Les algorithmes génétiques parallèles (AGP) consistent simplement à la distribution des tâches d'un algorithme génétique basique sur différents processeurs.
Comme ces tâches sont implicitement parallèles, peu de temps sera consacré à la communication et, par conséquent, l'algorithme se lance beaucoup plus rapidement et trouve des résultats plus précis.

Il a été établi que l'efficacité des AG est de trouver une solution optimale et déterminée en grande partie par la taille de la population.

Avec une plus grande taille de population, la diversité génétique augmente, l'algorithme est plus susceptible de trouver un optimum. Cependant, une grande population demande plus de mémoire pour être traitée. Il a également été prouvé que cela prend plus de temps à converger.

De nos jours, l'emploi d'ordinateurs parallèles offre non seulement plus d'espace de stockage mais permet également l'usage de plusieurs processeurs pour produire et évaluer d'autres solutions en un temps relativement faible.
Par la parallélisation de l'algorithme, il est possible d'augmenter la taille de la population, de réduire le coût de calcul et d'améliorer ainsi la performance de l'AG.

3.5.1 Les modèles de parallélisation d'un algorithme génétique :

Dans la littérature nous avons pu trouver plusieurs travaux concernant la parallélisation de l'AG classique. Parmis ces travaux nous citons ceux de Muhleinben [MUH 1989], Cantu Pazz [CAN 1995], [CHI 1996] et [CAN 2000]. Leurs travaux au début et milieu des années 90, ont consisté à proposer des modèles en ilot de l'AG classique. En effet, toutes les approches développées dans ces travaux montrent l'efficacité des modèles en ilot à explorer plus l'espace de recherche en recherchant l'optimum global tout en minimisant les temps de calculs.

Les modèles de parallélisation d'un AG reflètent différentes manières d'exploiter au mieux le parallélisme de ce dernier. Dans la littérature nous avons recensé trois modèles :

-le modèle global (maître/esclave),
-le modèle en ilots (migration),
-le modèle en grain (diffusion).

3.5.2 Le modèle global :

La première tentative de parallélisation des AGs consiste simplement à paralléliser à l'échelle globale.
Cette approche, plus communément appelée "Approche Maître/Esclave", tente de paralléliser explicitement les tâches parallèles qui sont implicites dans un AG séquentiel. La nature des problèmes reste inchangée.
L'idée de base est que les différents processeurs peuvent créer de nouveaux individus et calculer leur aptitude en parallèle, pratiquement sans aucune communication entre eux.
Chaque processeur est associé à un sous-ensemble d'individus.
Les figures 3.11 et 3.12 illustrent ce mode de parallélisation de l'AG.

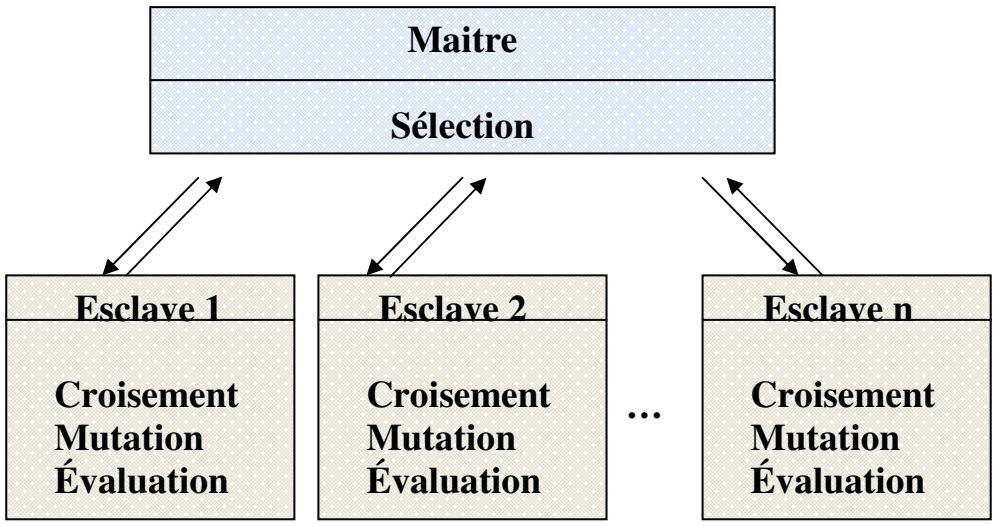


Figure 3.11 : Méthode de parallélisation globale ou Maître/Esclave

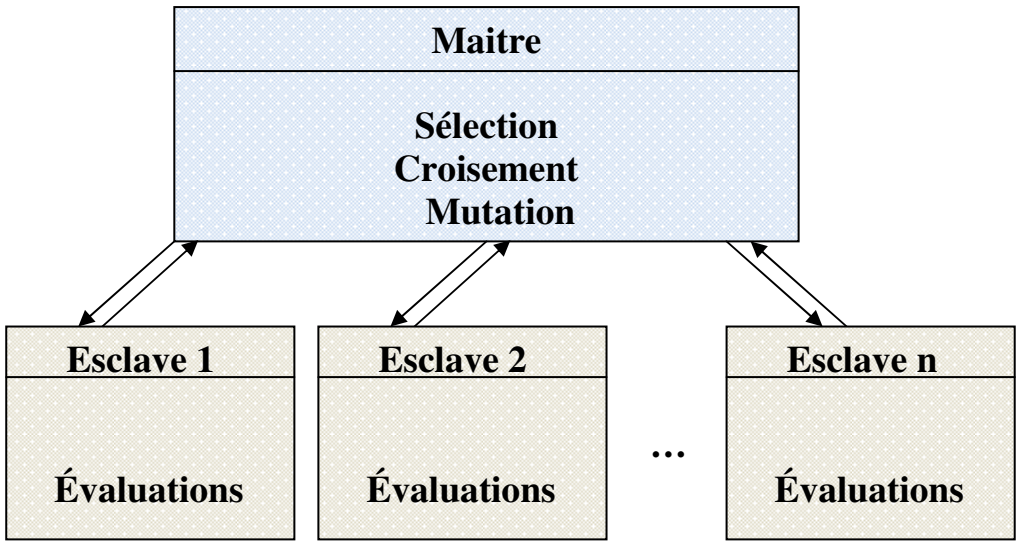


Figure 3.12 Méthode de parallélisation globale avec parallélisation des évaluations seulement

Sur un ordinateur à mémoire partagée: Les individus pourraient être stockés dans la mémoire partagée, de sorte que chaque processeur ait accès aux chromosomes qui lui sont affectés et peut en retour écrire le résultat du calcul de remise en forme.
Cette méthode suppose seulement que les AGs travaillent avec une mise à jour de générations de la population. Certaines synchronisations cependant sont nécessaires entre les générations. En général, la plupart du temps de calcul dans un algorithme génétique est passé dans l'appel de la fonction d'évaluation (au moins 80% du temps total), le temps passé à manipuler les chromosomes lors de la sélection ou de la phase de recombinaison étant généralement négligeable.
Il est cependant impératif d'équilibrer le partage de la population entre les processeurs.

Sur un ordinateur à mémoire distribuée: La population est stockée dans un processeur 'Maitre' responsable de l'envoi des individus aux autres processeurs, 'Esclaves'. Le processeur Maître est également responsable de la collecte des résultats des évaluations. Une nouvelle étape pourrait consister dans l'application des opérateurs génétiques en parallèle. En fait, l'interaction à l'intérieur de la population ne se produit que lors de la sélection.
La reproduction, impliquant seulement deux individus pour produire N/2 paires d'individus, pourrait facilement être effectuée simultanément.
Le croisement est généralement très simple et ne prend pas beaucoup de temps. Par contre la plus grande partie du temps sera perdue au cours de la communication, mais ce temps sera presque négligeable par rapport à l'effort produit pour le calcul des fonctions d'adaptation.

3.5.2 Le modèle en ilots:

La deuxième catégorie de l'AGP est une fois de plus inspirée par la nature. La population est maintenant divisée en plusieurs sous-populations, et chacune de ces sous-populations est relativement importante et évolue séparément sur différents processeurs.
L'échange entre les sous-populations est possible par l'intermédiaire d'un opérateur de migration. Le terme île est facilement compréhensible; l'AG se comporte comme si le monde était constitué d'îles où les populations évoluent de manière isolée, les unes des autres sur chaque île. La population est libre de converger vers les différents Optima.
L'opérateur de migration permet le "métissage" des différentes sous–populations. Il est censé mélanger les bonnes caractéristiques qui se dégagent au niveau local dans les différentes sous/populations. Dans ce cas, il est à noter que la nature de l'algorithme change : Un individu ne peut plus se reproduire avec tous les individus de la population, mais seulement avec ceux du même îlot. Cet algorithme est capable de donner différentes solutions sous optimales et, dans de nombreux problèmes, c'est un avantage dans le cas de la détermination d'un type de paysage de l'espace de recherche pour savoir où sont réparties les bonnes solutions.
Un autre grand avantage du modèle en ilots est que la population de chaque île peut évoluer avec des règles différentes. Cela peut être utilisé pour l'optimisation multicritère. Sur chaque île, la sélection peut être faite selon différentes fonctions de remise en forme, représentant différents critères. Par exemple, il peut être utile d'avoir autant d'îles que de critères, plus une autre île, où la sélection est réalisé avec une fonction de remise en forme multicritère. L'opérateur de migration permet aux individus de se déplacer entre les îles, et par conséquent de mélanger des critères.
Dans la littérature, ce modèle est également appelé "modèle à gros grains". Nous pouvons également trouver le terme "AG distribué", car ils sont généralement mis en œuvre en utilisant une mémoire distribuée (ordinateurs MIMD).

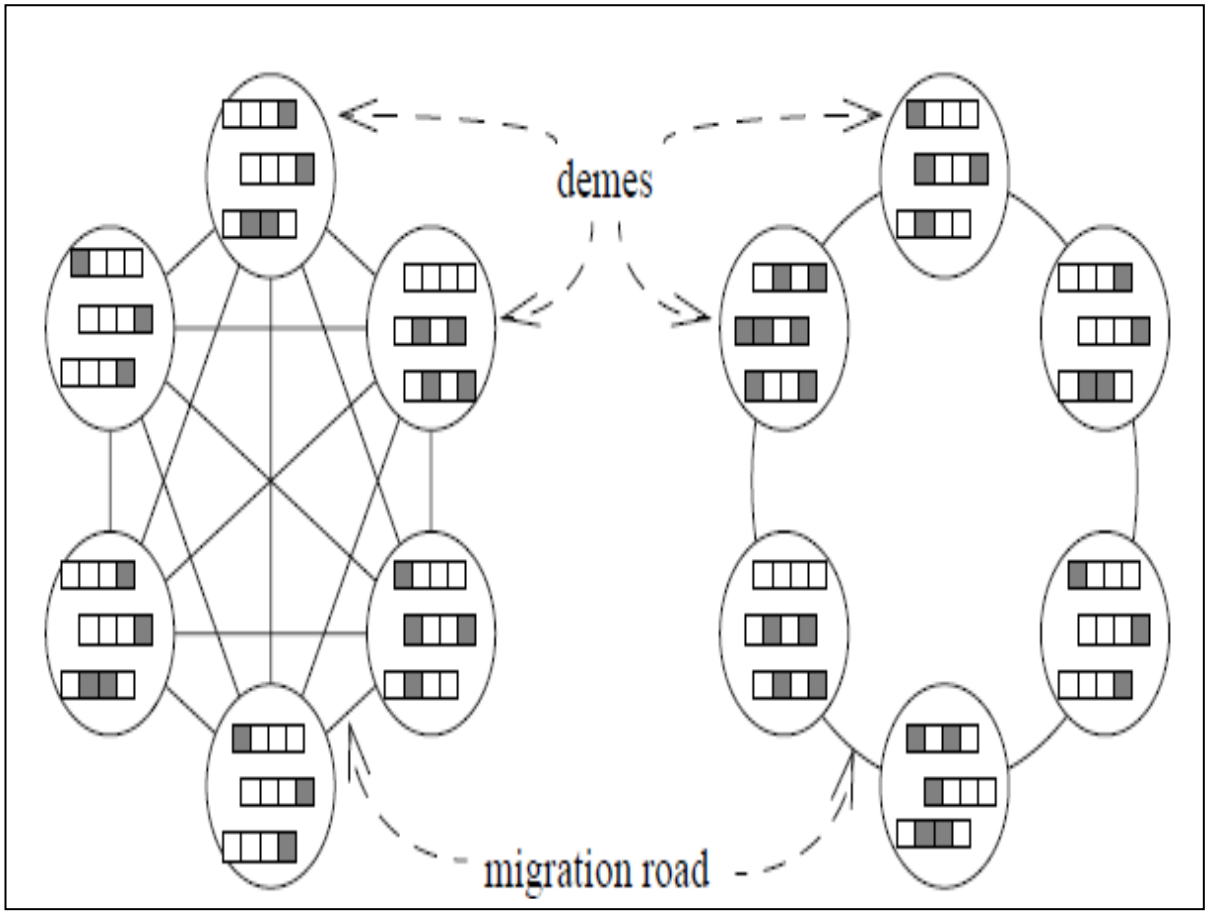
Techniquement, il existe trois éléments importants dans la parallélisation d'AG à gros grains:

- la topologie : qui définit les connexions entre les sous-populations,
- le taux de migration : qui contrôle la manière dont les individus émigrent,
- les intervalles de migration : qui affectent la façon dont la migration se produit.

Beaucoup de topologies peuvent être définies pour connecter les sous-populations, il existe principalement trois topologies :

-la migration en anneau,
-la migration par voisinage,
-la migration non restrictive.

Différents travaux ont montré que la topologie de l'espace n'est pas si importante tant qu'il existe une haute connectivité pour assurer le mélange du produit. Par contre, la difficulté réside dans le choix du moment opportun et des individus adéquats pour la migration. Dans la pratique, la migration se produit après un nombre fixe d'itérations dans chaque sous-population ou à des périodes de temps bien déterminées. Les migrants sont habituellement choisis au hasard parmi les meilleurs individus de la population. En fait, le hasard est encore principalement utilisé pour fixer le taux de migration et de l'intervalle de migration.



Migration non restrictive **Migration par voisinage**

Figure 3.13 Topologies de migration

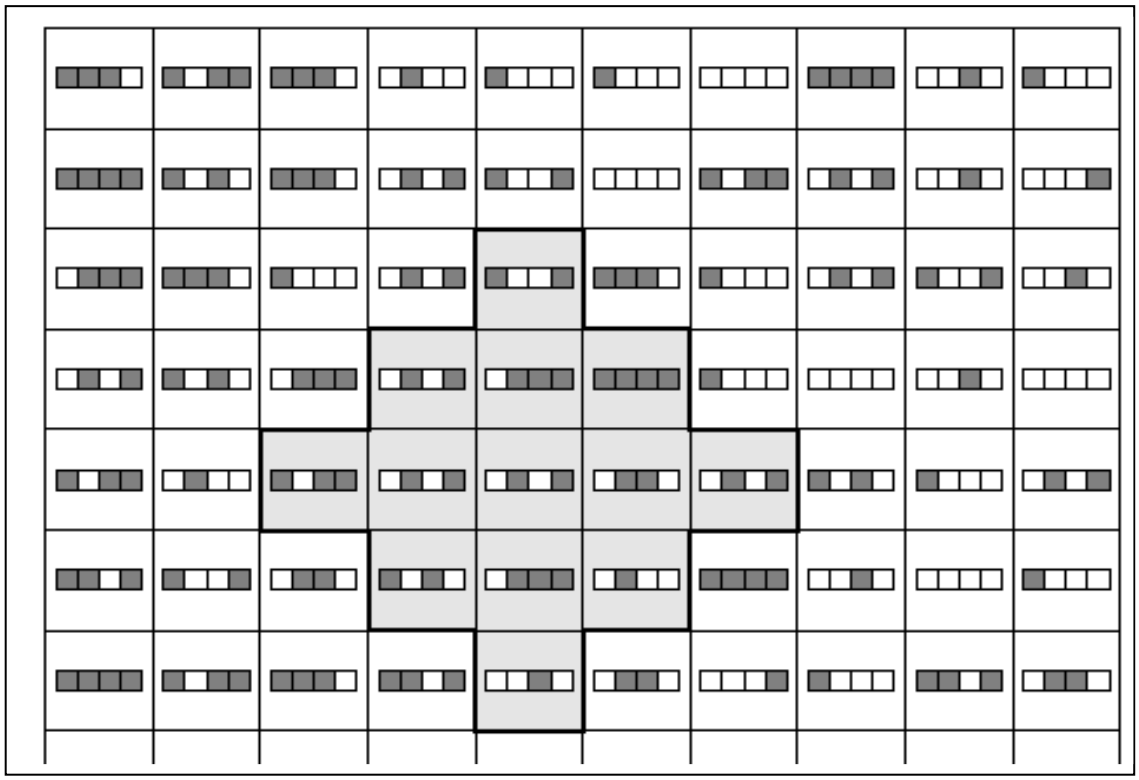


Figure 3.14 Le modèle en grain fin

.5.3 Le modèle en grain (diffusion) :

Le développement des ordinateurs massivement parallèles a déclenché une nouvelle approche de la parallélisation des AGs. Le modèle en grains a été développé avec un grand nombre de processeurs.

Dans ce modèle, la population est divisée en un grand nombre de très petites sous-populations. L'idéal est d'avoir un seul individu pour chaque élément de traitement disponible.

Dans ce modèle, la reproduction est seulement possible entre les individus voisins, c'est-à-dire les individus stockés sur des processeurs voisins. La sélection se fait aussi dans le voisinage de chaque individu et dépend seulement de cela. Ce schéma est illustré par la figure 3.14.

4 Conclusion :

Les algorithmes génétiques ont l'énorme avantage de pouvoir être appliqués dans un grand nombre de domaines de recherche de solutions. Dés lors qu'il est important de déterminer la solution optimale et que cela pourrait induire des temps de calculs et des ressources énormes pour être calculée, il est recommandé de les paralléliser. Pour cela toutes ces méthodes décrites peuvent servir de modèles. Il reste à opter pour le modèle le mieux adapté selon la nature du problème posé et des ressources disponibles. Il est également possible d'imaginer d'autres modèles ou d'hybrider ceux déjà existants pour de meilleures performances, ce qui sera développé dans la suite de nos travaux.

1- Introduction :

Un **mème** (de l'anglais *meme*) est un élément culturel reconnaissable (par exemple : un concept, une habitude, une information, un phénomène, une attitude, etc.), répliqué et transmis par l'imitation du comportement d'un individu par d'autres individus. L'OED (Oxford English Dictionnary) définit le mème comme "un élément d'une culture (prise ici au sens de civilisation) pouvant être considéré comme transmis par des moyens non génétiques, en particulier par l'imitation". Le terme de *mème* a été proposé pour la première fois par Richard Dawkins dans "Le Gène Egoiste" en 1976 et provient d'une association entre gène et *mimesis* (du grec « imitation »).

Les mèmes ont été présentés par Dawkins comme des réplicateurs, comparables à ce titre aux gènes, mais responsables de l'évolution de certains comportements des animaux et des cultures.

L'étude des mèmes a donné naissance à une nouvelle science : la mémétique.

La définition que donne R. Dawkins du terme "même" correspond à une « unité d'information contenue dans un cerveau, échangeable au sein d'une société ». Elle résulte d'une hypothèse selon laquelle les cultures évolueraient comme les êtres vivants, par variations et sélection naturelle. À l'instar du gène, le mème serait l'unité de base dans cette évolution [DAW1989].

Mème et Mémétique sont analogues à Gène et Génétique appliqués aux éléments des cultures et non aux individus biologiques.

En 1989, Pablo Moscatto [MOS1989] a introduit cette notion de "même" dans la vie artificielle.

2 Définition du mémétisme [MOS 2000] [KRA 2002] [MER 2000]

Les algorithmes mémétiques (AMs) sont des algorithmes évolutionnaires hybridés avec des méthodes de recherche locale. Cette hybridation est destinée à accélérer la découverte de bonnes solutions pour laquelle la seule évolution serait trop longue à découvrir. Elle est également destinée à trouver des solutions qui, autrement, seraient inaccessibles par une évolution ou une seule recherche locale. La recherche évolutionnaire effectue une large exploration de l'espace de solutions tandis que la recherche locale permet en quelque sorte de "zoomer" sur des régions prometteuses de l'espace de recherche.

3 Définition Formelle d'un AG et d'un AM

Comme déjà spécifié, les AMs sont des instances d'AGs généralisés dans lesquelles une recherche locale est introduite. Dans [RAB1996] les auteurs décrivent la classe des AMs comme étant non seulement basée-AG mais une nouvelle classe à part, appelée PMA : Polynomial Merger Algorithm, mais dans [KRA2002] les auteus pensent qu'avec la définition de PMA donnée dans leur travail, les auteurs ont fortement limité et ignoré de nombreux travaux qui ne peuvent pas être prouvés comme faisant partie de cette classe. Ceci est dù au fait que les PMA ne reconnaissent pas l'importance des opérateurs de mutation, sélection et même la recherche locale comme étant des composants importants de la meta-heuristique.
Dans [RAD1996] Radcliffe et Surry présentent un formalisme algébrique des AMs. Dans cette approche, les auteurs définissent un AM comme un AG très spécifique où une période comportant une Recherche Locale (RL) est rajoutée. Leur formalisme d'un AM est dit

"apriori" car il limite sérieusement l'analyse complète et la généralisation du comportement des AMs. Krasnogor a repris un modèle dit "a-posteriori", proposé par J.Smith [SMI1998] où des algorithmes sont considérés eux-mêmes comme données. Il a défini un modèle syntactique en se basant sur des AMs qui ont été prouvés performants dans la résolution de certains problèmes connus dans la littérature.
Dans ce modèle, un algorithme génétique (AG ou GA) peut être formalisé par :
GA = $(P^\circ, \Delta^\circ, \lambda, \mu, F, G, V)$

Où :

$P^\circ = (a_1^\circ, a_2^\circ, \ldots\ldots, a_\mu^\circ)$ appartient à I^μ la population initiale.
$I = \{a_1, a_2, \ldots.., a_n\}^l$: Représentation discrète et finie du Problème.

$\Delta^\circ \subseteq R$ Ensemble Initial des paramètres

μ appartient à N : Taille de la population
λ appartient à N: nombre d'enfants créés appelés Offspring
l appartient à N : longueur d'une représentation (longueur du chromosome)
$F : I \longrightarrow R^+$ la fonction d'adaptation ou "fitness"

$G : I^\mu \longrightarrow I^\lambda$ Fonction de Génération (Generating Function)

$V : I^\mu \times I^\lambda \longrightarrow I^\lambda$ Fonction de remplacement (Upadating function)

Comme exemples de fonctions de génération nous citons le croisement et la mutation.
Un opérateur de recombinaison possède une signature telle que :

$R : I^\mu \times \Delta \longrightarrow I$

Et un opérateur de mutation possède la signature :

$M : I \times \Delta \longrightarrow I$

Si O_{ffsp} appartenant à I^λ représente un ensemble d'enfants créés, alors une itération d'un AG serait :

$O_i^t = M(R(P^t, \Delta^t), \Delta^t)$ quelque soit la valeur de i prise dans $\{1, \ldots., \lambda\}$

$P^{t+1} = U(O^t \cup P^t)$ où t= un pas de l'algorithme.

A présent, nous devons étendre cette définition formelle de l'AG pour inclure les opérateurs de recherche locale comme de nouvelles fonctions de génération.
Les opérateurs de recherche locale sont considérés comme membres d'un ensemble d'opérateurs, on appelle cet ensemble LSS.

$LSS = \{L_1, L_2, \ldots\ldots\ldots., L_m\}$

La signature de chaque membre de l'ensemble LSS est définie par :

$L_j : I \times \xi \longrightarrow I^{c_1}$

Où :
ξ : est un paramètre spécifique à la stratégie de recherche locale dont le rôle équivaut à Δ.
j : est un index dans l'ensemble LSS
c_1 : une constante qui détermine combien de solutions l'opérateur de recherche locale utilise comme arguments et combien de solutions il doit retourner.

Comme spécifié auparavant, l'étape de recherche locale peut être intégrée à n'importe quel niveau de la boucle génétique, avant ou après : croisement, mutation, séléction ou dans n'importe quelle combinaison de ces opérateurs.
Pour cela on a défini deux autres entités appelées :

fS pour "fine grain Scheduler"

cS pour "coarse grain Scheduler"

Ce sont des fonctions de plus haut niveau

$$fS : (I^{c}_{1} \times \Delta \longrightarrow I) \times LSS \times I^{c}_{2} \times \Delta \times \xi \longrightarrow I$$

fS reçoit en entrée deux fonctions génératrices, l'une avec la signature:

$(I^{c1} \times \Delta \longrightarrow I)$ qui représente l'une des fonctions standards de l'AG, telle que le croisement ou la mutation.
La deuxième provient de l'ensemble LSS avec la signature :

$$I^{c}_{2} \times \xi \longrightarrow I^{c}_{2}$$

C'est l'opérateur de recherche locale qui a été sélectionné dans cet ensemble.
Un ensemble de solutions représenté par I^{c1} et des opérateurs ainsi que les paramètres de stratégies Δ et ξ sont aussi des arguments d'entrée.
Il est à noter que la valeur de c^2 doit être 1 ou c^1 car l'opérateur de recherche locale traite les productions des fonctions génératrices.

Dans le cas le plus simple, on aura une mutation (fS_M) et une recombinaison (fS_R) avec les signatures :

$$(fS_M) : (I \times \Delta \longrightarrow I) \times LSS \times I \times \Delta \times \xi \longrightarrow I$$

$$(fS_R) : (I^{\mu} \times \Delta \longrightarrow I) \times LSS \times I^{\mu} \times \Delta \times \xi \longrightarrow I$$

Comme exemple, nous pouvons définir la fonction "fine grain Scheduler" comme :
fS_M (M,L_1, i, Δ, ξ_1) où i est un individu , M : opérateur de mutation, L_1 est un opérateur de recherche locale sélectionné dans LSS et ξ_1 ensemble de paramètres pour L_1 .
fS_R (R , L_2, P, Δ, ξ_2) où R est un opérateur de recombinaison (ou croisement) et P un ensemble de parents avec en général |P| =2.
En résumé, "fine grain Scheduller" et "coarse grain Scheduler" ont pour objectif de décider quand, où et avec quels paramètres les opérateurs de recherche locale de LSS seront appliqués pendant la mutation et le croisement respectivement.
Il est à noter que souvent les résultats obtenus par l'AG et l'AM sont comparés mais la synergie de la recherche locale avec le croisement ou avec la mutation ne le sont pas. L'étude de la synergie entre la recherche locale et l'un des opérateurs génétiques peut montrer l'efficacité de l'algorithme mémétique.
Dans le cas de la mutation, le critère de passage d'un point à un autre de l'espace de recherche serait de maximiser la variance de la fonction "fitness" obtenue quand le point sélectionné est l'objet d'un nombre de mutations réduit. Une plus grande variance de la "fitness" aide à assurer la diversité de la population De cette façon les opérateurs de recherche locale et les opérateurs de mutation et de croisement travaillent de manière synergétique par essence.

En plus, il est suggéré de mener le calcul de la distance de corrélation entre la fitness et les opérateurs génétiques afin d'affiner l'analyse de leur synergie, comme cela a été fait dans [MAN1991, MER2000, MER1999] .

4 Les raisons de l'hybridation

1. Tout problème complexe peut être décomposé en sous problèmes dans une certaine mesure.
2. Les algorithmes évolutionnaires ou les méthodes de recherche locale peuvent être utilisés comme pré/post méthode de traitement de solution. On utilise souvent une méthode de recherche locale en hybridation avec un algorithme évolutionnaire pour améliorer la meilleure solution trouvée.
3. On peut disposer de méthodes de recherche locale très puissantes mais on aimerait pouvoir diversifier la recherche. Dans ce cas, nous pouvons recourir à une méthode de recherche locale pour produire une population initiale puis exécuter un algorithme évolutionnaire sur cette population initiale.
4. L'information d'un problème spécifique peut être dissipée par la variation des opérateurs, comme les opérateurs de croisement ou de mutation, ou bien par l'algorithme de recherche locale. L'hybridation des deux méthodes peut éviter cela efficacement en dirigeant la recherche vers des régions prometteuses de l'espace de recherche.
5. Dans certains cas, il existe des méthodes exactes ou approximatives pour les sous problèmes du problème traité, capables de prendre en charge certaines spécificités du problème. Lorsque celles-ci sont disponibles, elles peuvent être incorporées dans l'algorithme évolutionnaire afin de produire de meilleures solutions.
6. Les problèmes associent des contraintes à des solutions et les heuristiques de recherche locale sont utilisées pour améliorer les solutions trouvées par l'algorithme évolutionnaire. Si les stratégies de recherche locale dans les algorithmes mémétiques sont considérées comme première préoccupation du concepteur alors une définition plus riche des métaheuristiques hybrides d'adaptation est possible : les stratégies de recherche locale pourraient être générées en même temps avec les solutions qu'elles envisagent d'améliorer. Cependant, la forme la plus populaire de l'hybridation est d'appliquer une ou plusieurs phases de recherche locale, sur la base de certains paramètres de probabilité à des membres d'individus de la population dans chaque génération.

5 Les décisions de conception : [KRA2005]

Quelques importantes décisions de conception doivent être prises lors de la conception d'un algorithme mémétique, telles que:

- ✓ La stratégie de remplacement des individus : LAMARCKIANISM ou BALDWINIAN.
- ✓ La préservation de la diversité.
- ✓ Le choix des voisinages pour la recherche locale.
- ✓ L'utilisation des profils de performance.

Nous allons décrire chacune d'elles dans ce qui suit.

5.1 LAMARCKIAN / BALDWINIAN :

Lors de l'intégration de la recherche locale dans l'algorithme évolutionnaire (AE), nous sommes confrontés à ce qu'il faut faire avec la solution améliorée produite par la recherche locale.

En d'autres termes, supposons:

i: un individu de la population P de la génération t.

f (i): La valeur de sa fonction d'adaptation.

En outre, supposons que la recherche locale produit un individu i' avec :

f (i) < f (i') (dans le cas d'un problème de maximisation).

Pour décider de remplacer i' par i dans la population, deux approches existent:

a-<u>Approche de LAMARCK</u> :

Elle consiste à remplacer i par i', donc:

P = P – {i} + {i'}

Et f (i)= f (i')

En remplaçant i par i' , l'information contenue dans i est perdue.

2-<u>Approche de BALDWIN</u> :

L'information génétique de i est conservée mais la fonction de fitness est celle de i' :

f(i) :=f (i').

P reste la même.

La question est de savoir si l'évolution naturelle de l'approche 1 ou 2 est meilleure:

Il est à priori difficile de décider quelle est la meilleure méthode et probablement aucune n'est bonne dans tous les cas.

5.2 Préservation de la diversité :

Nous avons mentionné ci-dessus les difficultés lors de l'utilisation d'une méthode très agressive de recherche locale. Au premier rang de ces difficultés la nécessité de préserver la diversité.

Cette diversité pourrait être perdue :

- Par exemple, dans le cas de recherche locale partielle, si l'espace de recherche possède une forme (ou topologie) à très larges bassins, le croisement et la mutation ne peuvent pas facilement éviter d'être piégés dans des optima locaux.
- Divers mécanismes ont été étudiés comme un moyen d'éviter la convergence prématurée dans les algorithmes mémétiques. Par exemple, si une petite population initiale est générée, seule une petite proportion d'individus devrait subir la recherche locale et non pas tous les individus de la population.

Une stratégie très commune visant à traiter la préservation de la diversité a été l'introduction d'opérateurs de croisement très spécifiques qui aident à préserver la variété génétique toujours au-dessus d'un seuil minimum.

Une autre stratégie privilégiée consiste à modifier l'opérateur de sélection pour empêcher les copies multiples des individus.

Plus récemment, trois méthodes distinctes et puissantes ont été introduites qui non seulement préservent la diversité mais aussi permettent de renforcer la performance algorithmique :

1. Des recherches locales multiples : où chacune introduit un espace de recherche différent avec des optimums locaux différents afin d'éviter les pièges locaux **[KRA2005].**
2. La mise en œuvre des ensembles et des systèmes flous pour contrôler explicitement la diversité au sein d'une règle de décision dans la phase de recherche tel que cela a été fait dans **[KRA2002].**
3. La possibilité de modifier l'opérateur de sélection ou des critères d'acceptation de recherche locale pour utiliser la méthode adaptative de Boltzmann comme cela été fait

dans **[KRA2000]**. Cela a été fait de manière similaire à la méthode du recuit simulé où les mouvements peuvent être acceptés avec une probabilité non nulle qui aide à s'échapper de l'optimum local. Cette méthode est prometteuse durant la recherche locale au moins un voisinage peut être accepté avec une probabilité qui augmente exponentiellement avec un facteur k de normalisation.

$$\text{ProbaAccept} = \begin{cases} 1 & \Leftrightarrow \quad \Delta f > 0 \\ e^{K^* (\Delta f/(fmax - fmoy)} & \text{sinon} \end{cases}$$

Où :

- **k** : un facteur de normalisation,
- **Δf=** f (i')-f(i) (problème de maximisation).
- Fmax et fmoy sont les valeurs maximale et moyenne de la fonction d'adaptation respectivement.

Ce mécanisme permet à l'algorithme mémétique d'osciller entre les périodes d'exploitation et les périodes d'exploration en fonction de ce que (f_{max}-f_{moy}) est large ou bien limitée à un intervalle étroit.

5.3 Voisinage à utiliser pour la recherche locale

Dans le cas de recherche locale, cela est fait en définissant la structure de voisinage d'une solution, c'est-à-dire quelles sont les solutions envisageables accessibles à partir des points de l'espace de solutions.
Si l'espace de solutions est trop petit ou trop restrictif, il est probable que l'optimum local sera déterminé directement par la recherche locale. D'autres parts, si le voisinage est complet, alors il est possible de construire un chemin de tout autre point, mais dans ce cas, l'analyse exhaustive est payée par une longue recherche qui peut être exponentielle.

5.4 Utilisation des profils de performance

Les profils de performance peuvent être utilisés pour suivre le progrès de la recherche vis-à-vis des différentes composantes algorithmiques de l'algorithme mémétique. Nous pouvons prendre en considération l'apprentissage acquis par l'algorithme et réutiliser cette connaissance lors de l'exploration future à travers le processus d'optimisation. Une hybridation possible utilise explicitement les connaissances sur les solutions précédemment vues comme un moyen pour guider l'optimisation, comme cela est fait dans la recherche taboue qui n'autorise pas de revisiter des solutions contenues dans la liste taboue.

5.5 Considération particulière pour les domaines continus : [RAD 1996]

Les problèmes de conception que nous avons mentionnés ci–dessus portent principalement sur l'optimisation combinatoire, c'est-à-dire les problèmes discrets. Dans le cas de l'optimisation continue, pour générer l'algorithme mémétique nous devons réfléchir sur les différents facteurs de conception. L'optimisation dans les domaines continus exige que les échelles de recherche appropriée soient identifiées pour la recherche globale et locale. En outre, il est souvent difficile de déterminer si une solution possible représente ou non un optimum local. Si l'information du gradient n'est pas disponible alors de très longues recherches pourraient être nécessaires pour assurer la convergence avec précision.

L'incertitude "quelle méthode de recherche locale est meilleure pour un problème donné?" est plus aigue dans le cas de l'optimisation continue.

Plusieurs méthodes de recherche locale ont été proposées, mais comme ce sont des méthodes générales, il n'est pas clair de savoir si telle ou telle méthode de recherche locale est efficace pour une tâche d'optimisation continue.

Ainsi, la conception compétente des algorithmes mémétiques pour les domaines continus peut être tout à fait différente de celle des problèmes combinatoires.

Un simple exemple permettra de clarifier ce que nous voulons dire : il est commun dans les algorithmes mémétiques pour la recherche combinatoire d'appliquer la recherche locale jusqu'à l'identification des optimums locaux, cependant, en général, on ne peut pas assumer que la méthode de recherche locale puisse rapidement converger vers l'optimum local dans un domaine continu.

Deux stratégies communes sont normalement utilisées pour traiter cette difficulté qui compte essentiellement sur l'équilibre prudent entre la recherche globale et la recherche locale. Cet équilibre a souvent été mis en œuvre que ce soit par une recherche locale tronquée ou une recherche locale sélective.

6 Les algorithmes mémétiques adaptatifs : [COW 2000] [KEN 2001] [BUR 2003]

6.1 Définition :

Les algorithmes mémétiques adaptatifs sont caractérisés par l'utilisation de plusieurs mèmes dans la recherche et la décision du choix du mème à appliquer à un individu est prise dynamiquement. Cette forme d'algorithmes mémétiques adaptatifs favorise à la fois la coopération et la concurrence et favorise les structures de voisinage des solutions de haute qualité qui peuvent être obtenues avec de faibles efforts de calcul. Dans la première étape, la population de l'algorithme génétique peut être initialisée au hasard. Par la suite, pour chaque individu dans la population, un mème est sélectionné à partir d'un ensemble de mèmes considérés dans la recherche pour conduire les améliorations locales. Différentes stratégies pourraient être utilisées pour faciliter le processus de prise de décision. Par exemple, une sorte de "récompense" pourrait être affectée à un mème, basée sur sa capacité à effectuer les améliorations locales. Cette "récompense" pourrait être utilisée comme indicateur dans le processus de sélection du mème à appliquer lors d'une itération. Après les améliorations locales, les génotypes et/ou phénotypes dans la population d'origine sont remplacés par la solution améliorée selon le mécanisme d'apprentissage, c'est-à-dire, l'apprentissage de Lamarck ou Baldwin. Ensuite les opérateurs standards des algorithmes génétiques sont utilisés pour former la prochaine population.

6.2. Les catégories de choix d'un mème :

Dans le cadre de l'optimisation combinatoire, Cowling et al [COW2000] ont introduit le terme "hyperheuristique" comme une stratégie qui gère le choix du mème qui doit être appliqué à tout moment selon les caractéristiques des mèmes et la région de l'espace de solutions actuel dans l'exploration. Avec les hyperheuristiques, plusieurs mèmes ont été pris en compte dans l'évolution des recherches. Dans leurs travaux, trois catégories ont été proposées pour les problèmes de planification qui sont : aléatoire, gourmand et fonction de choix.

Aléatoire (Random):

Dans cette catégorie, il y a trois stratégies :

a. Stratégie aléatoire simple (Simplerandom) : dans ce cas, un mème est sélectionné aléatoirement à chaque point de décision. Elle est purement de nature stochastique, la probabilité de choisir chaque mème est maintenue constante tout au long de la recherche.

Cette stratégie peut être considérée comme une référence avec laquelle d'autres stratégies de sélection peuvent être comparées.

b. Stratégie de la descente aléatoire (Randomdescent): initialement, le choix de mèmes est décidé aléatoirement. Par la suite, ce mème choisi est utilisé de façon répétitive jusqu'à ce qu'il n'y ait plus d'améliorations locales identifiées. Ce même processus peut être répété en considérant tous les autres mèmes.

c. Stratégie de la descente aléatoire avec permutation (Randompermdescent) :
cette stratégie est similaire à la deuxième, sauf que la permutation aléatoire des mèmes M_1, M_2,…, M_n est fixée à l'avance, et quand l'application d'un mème n'engendre aucune amélioration, le prochain mème qui existe dans la permutation sera utilisé.

d. Gourmande (Greedy) :
Cette catégorie ressemble à la technique de la force brutale qui fait des expériences sur tous les mèmes de chaque individu et choisit le mème qui engendre la plus grande amélioration. Comme il s'agit d'une méthode de force brutale, l'inconvénient de cette stratégie est le coût élevé de calcul.

e. Fonction de choix (Choice function) :
Cette stratégie intègre plusieurs paramètres, elle est utilisée pour évaluer le degré d'efficacité d'un mème, en se fondant sur l'état actuel des connaissances sur la région de l'espace de solution dans l'exploration.
La fonction de choix contient trois composants :

a. f_1 qui reflète les améliorations récentes produites par un mème et exprime l'idée que, si un mème a récemment obtenu de bons résultats, il est susceptible de continuer à être efficace.
b. f_2 décrit les améliorations contribuées par les paires consécutives de mèmes.
c. f_3 enregistre la période écoulée depuis qu'un mème a été utilisé pour la dernière fois.

7. Conclusion :
Nous avons vu que les algorithmes mémétiques utilisent les méthodes de recherche locale afin de renforcer les algorithmes génétiques dans le processus exploratoire et ceci dans le but de trouver des solutions qui sont inaccessibles par un AG.
Pour cela, il faut prendre en considération ces différents points pour intégrer les méthodes de recherche locale dans un AG ainsi que les stratégies utilisées, afin d'obtenir de bonnes performances de l'AM.
Ceci sera développé dans les prochains chapitres.

1. Introduction

Dans cette partie, nous présentons tout le formalisme utilisé pour construire un classificateur basé sur une approche génétique pure. Nous proposons une formulation mathématique du problème de l'extraction de règles de classification dans un premier temps. Nous décrirons par la suite quelques travaux récents qui proposent des résolutions de ce problème par des métaheuristiques. Nous présenterons ensuite notre démarche génétique pour extraire des règles de classification. Pour cela, nous détaillerons le codage des individus qui sont dans notre cas des règles de classification, les opérateurs spécifiques que nous avons mis au point et enfin la fonction fitness que nous avons modélisée pour évaluer les qualités des règles de classification extraites ainsi que celles du classificateur obtenu. Enfin, un algorithme d'extraction de règles sera décrit.

2. Formulation Mathématique du problème de l'extraction de règles de classification

Globalement, le problème de la classification est caractérisé par les éléments suivants :

- Une population de N individus,
- Un nombre P de variables descriptives, appelés Attributs décrivant les individus,
- Un nombre C de classes.

Une règle de classification se présente sous la forme suivante :

$$A \longrightarrow C$$

- A est l'antécédent ou la précondition de la règle,
- C est la classe prédite par la règle.

2.1 L'antécédent de la règle:

La partie antécédent A de la règle est un ensemble de conjonctions de la forme suivante:
$T_1 \wedge T_2 \wedge \ldots T_k \wedge \ldots T_m$,
où T_k est le Kème terme de la règle, representé par une condition appliquée à un attribut, avec la syntaxe suivante :

(Attribut Att_k Opérateur Valeur Vk_j)

Où : - Attribut Att_k est le nom de la variable sur laquelle porte le test,

- Opérateur est l'opérateur de comparaison, tels que '=', '<=', '>=' pour les attributs pouvant prendre des valeurs numériques ou bien '=' pour des attributs ayant des valeurs nominales.

- Valeur Vk_j est l'une des valeurs à laquelle l'attribut est comparé, sachant que Att_k peut prendre différentes valeurs pouvant être nominales ou bien appartenir à un domaine continu de valeurs réelles si cet attribut est numérique.

Considérons, par exemple, les instances de la base de données apprentissage appelée "Weather" du répertoire de bases de données "benchmark" de l'UCI [UCI 1998]. Ces instances comportent les attributs suivants: Outlook, Temperature, Humidity, Windy et Play.

Certains de ces attributs sont nominaux et prennent par conséquent des valeurs discrètes :

- {overcast, rainy, sunny} pour Outlook
- {True, False} pour Windy
- {Yes, No} pour Play.

Les attributs Temperature et Humidity sont numériques et prennent leurs valeurs dans les intervalles [64, 85] et [65, 96] respectivement.

2.2 La classe prédite

La classe prédite représente une action à entreprendre. Elle est aussi appelée la variable de décision. Il est important de noter que pour la tâche de classification la variable de décision est nominale, sinon si la variable de décision est numérique alors la tâche de data mining envisagée est plutôt une estimation.
Dans l'exemple précédent, l'attribut "Play" est la variable de décision. Elle représente l'action à prendre: Jouer ou Non, dépendant des conditions météorologiques.
Citons un exemple, la règle suivante:

If Outlook = Overcast AND Humidity <=80 THEN Play = Yes

Elle possède uniquement deux prémisses

- (Outlook = Overcast)
- (Humidity<=80)

conduisant à une décision positive de jouer (Play=Yes)
.
En fait la partie antécédent est interprétée comme la partie indépendante alors que la partie prédiction est vue comme une variable dépendante.

3. Etat de l'art

Le problème de la classification, tel que formulé, est un problème combinatoire, classé NP-Difficile. De nombreux travaux ont été menés pour résoudre ce problème. Citons les travaux de J. Bacardit [BAC 2004] qui a utilisé l'approche de Pittsburgh [SMI 1980] pour la résolution de ce problème.
Dans ce qui suit, nous allons décrire sommairement les travaux de J. Bacardit puis deux **travaux basés sur les metaheuristiques et sur l'approche de Michigan [HOL 1993] pour la résolution du problème de la classification.**

3.1 Systèmes de classificateur basés sur des méthodes évolutionnaires [BAC2004]

Ces systèmes sont basés sur l'approche de Pitssburgh qui consiste à extraire le classificateur durant une seule application de l'algorithme évolutionnaire.
L'approche de Pittsburgh propose un système qui utilise le cycle traditionnel d'un AG où chaque individu est une solution complète au problème de classification : c'est une disjonction d'un ensemble de règles de classification. Chaque règle a une longeur fixe mais le nombe de règles de l'ensemble est variable. Les individus se concurrencent entre eux pour classifier correctement le maximum d'instances et la population converge vers de bons ensembles de règles.
Nous avons recencé dans la littérature deux systèmes de représentation pour cette famille :
Gabil (de DeJong & Spears en 1991) [DEJ 1991],

- **GIL : de Janikow en 1991, [JAN1991].**

3.1.1 La représentation GAbil

1 / Chaque individu est un ensemble de règles de longueur variable

I = (R1 V R2 V …. V Rn)

Chaque règle de classification possède une représentation binaire, une longueur fixe et codifie un prédicat. Ce système utilise le concept d'apprentissage à partir d'instances positives et négatives. Les règles couvrent seulement les instances positives ainsi il n'y a aucune classe associée à une règle. La représentation sémantique de la règle est :

$(A_1 = V_1^1 \text{ v } \dots \text{ v } A_1 = V_m^1) \Lambda \dots \Lambda (A_n = V_1^1 \text{ v } \dots \text{ v } A_n = Vn^m)$

Où :

A_i est le ième attribut de l'ensemble de donnée et qui peut peut prendre plusieurs valeurs dans son domaine de valeurs .

V_i^j est la jieme valeur de l'attribut A_i .

Les predicats peuvent être exprimés à l'aide de chaines binaires de la manière suivante :

Soient un nsemble de données décrites par 4 attributs A_1 A_2 A_3 A_4 et soient les valeurs prises par ces 4 attributs respectivement (A,B,C ,D) , (E,F,G), (H,I,J,K,L) et (M,N).

Le prédicat (A_1 est B ou C) (A_2 est E ou F ou G) et (A_3 est H ou K) et (A_4 est M) est représenté comme suit :

A_1	A_2	A_3	A_4
0 1 1 0	1 1 1	1 0 0 1 0	1 0

En observant ces valeurs nous constatons que tous les bits de A_2 sont à 1 : ce mécanisme représente un attribut non pertinent.

2 / La fonction d'adaptation (fitness) est calculée après classification de toutes les instances de l'ensemble :

F (individu) = (Nombre d'exemples bien classés / Nombre total d'exemples)^2

3/ L'opérateur de croisement : cet opérateur doit garantir que le résultat soit correct sémantiquement : le point de coupure peut être à n'importe quel endroit dans une règle du classificateur mais il doit être différents pour les deux parents tout en étant la même position à l'interieur d'une règle.

Variantes

GABL : (GA Batch Concept Learner) c'est le système décrit

GABIL : (GA Batch Incremental Learner) C'est une amélioration de GABL avec un processus d'apprentissage incrémental :

- Le système commence l'apprentissage avec une seule instance. Un ensemble de règles qui couvrent cette instance est généré.
- Après avoir généré un ensemble de règles initial, le système essaye de classer une deuxième instance avec cet ensemble.
- Si la nouvelle instance es correctement classée, le même essai est répété avec d'autres instances.
- Sinon GABL est ré-exécuté en utilisant toutes les instances dejà testées.

3.1.2 La représentation GIL

1/ Représentation des connaissances

Même représentation binaire que celle de GABIL

2/ Fonction Fitness : elle a pour but d'équilibrer entre la précision et la complexité des individus au moyen de deux termes liés à ces mesures :

$$F(\text{individu}) = \text{correctness} \leq (1 + w \leq (1+\text{COST}))^1$$

F se développe très lentement dans l'interval [0,1] comme âges de population et correctness est définie par :

$$\text{Correctness} = \frac{w_1 * \mathcal{E}^+ / E^+ + w_2 * (1 - \mathcal{E}^- / E^-)}{w_1 + w_2}$$

où $\mathcal{E}^+$ / $\mathcal{E}$ - : c'est le nombre d'instances positives / négatives dans l'ensemble de données. Cost est définie par :

cost = 2≤#rules + #conditions

Les valeurs de w_1 et w_2 sont définies experimentalement = 0.5

3/ opérateurs génétiques :

Ce système est très riche par des opérateurs génétiques de haut niveau qui modofient le chromosome au niveau sémantique contrairement au x opérateurs génétiques classiques définis dans GABIL.

Ils existent 2 Catégories d'opérateurs

1/ Opérateurs des chromosomes :

- Opérateur d'échange de règles :il échange des règles complètes entre parents
- Opérateur de copie de règles : il enlève une règle d'un parent et l'insère dans l'autre
- Opérateur NewPEvent : c'est un opérateur unaire qui ajoute une instance positive non couverte à une règle.
- Opérateur de généralisation de règle : il généralise d'une manière unaire un sous ensemble de règles aléatoire des règles d'un individu.
- Opérateur d'élimination de règles : il élimine un sous ensemble aléatoire de règles d'un individu
- Opérateur de spécialisation de règle : il spécialise un sous ensemble aléatoire de règles d'un individu

2/ Opérateurs des attributs

- Opérateur de changement de reference : change l'état (0 ou 1) d'une des valeurs d'un attribut
- Opérateur d'extension de references : c'est un opérateur qui rajoute des valeurs à un attribut (généralise l'attribut).
- Opérateur de restriction des references : il spécialise l'attribut en enlevant des valeurs à cet attribut.

3.1.3 Les algorithmes

3.1.3.1 GAssist (Genetic Algorithms based claSSIfier sySTem) [BAC 2003] [DEJ 1991] [RIS 1978]

GAssist est un système de classificateur qui utilise l'approche de Pittsburgh pour représenter les règles et la représentation GABIL pour représenter un individu. Son principe consiste à utiliser un AG pour manipuler les individus. Pour évaluer chaque individu, GAssist utilise une fonction d'adaptation spéciale basée sur le principe de **MDL** (**M**inimum **D**escription **L**ength) qui mélange la précision et la complexité d'un individu. La formule du MDL est définie comme suit :

$$MDL = W * TL + EL$$

- **TL** : est la complexité qui prend en considération le nombre de règle et le nombre d'attributs importants dans chaque règle.
- **EL** : mesure l'erreur d'un individu par rapport à l'ensemble d'apprentissage.
- **W** : est une constante qui exprime une relation entre TL et EL.

3.1.3.2. Hider (Hierarchical Decision Rules) [JES 2003] [MIT 1997] [VEN 1993]

Hider est un algorithme qui produit un ensemble de règles hiérarchiques. Il utilise un AG pour la recherche de la meilleure solution dont le but est d'obtenir ces règles. L'évaluation de chaque individu de la population utilise la fonction d'adaptation suivante :

$$f(r) = N - CE(r) + G(r) + Coverage(r)$$

Où :

- **r** : est un individu.
- **N** : est le nombre d'exemples qui doivent être traités.
- **CE(r)** : erreur classe (class error) le nombre d'exemples mal classés par r.
- **G(r)** : le nombre d'exemples bien classés par r.
- **Coverage (r)**: la proportion d'éléments de l'espace de recherche couverts par r.

Le pseudo code de Hider est le suivant :

Procédure Hider

```
Entrée T : l'ensemble d'apprentissage;
Sortie R : l'ensemble de règle ;
Début
    R :=Ø ; Taille initiale=|T| ;
    Tant que (|T|> 0)
    Faire
        r :=AlgEvo(T) ;
        R=R+r ;
        Supprimer les exemples couverts par r ;
    Fait ;
Fin.
```

Procédure AlgEvo(T)

```
Début
    I=0 ; P0 :=Initialiser Population () ; Evaluer(P0) ;
    Tant que (i<Num_Génération)
    Faire
        I++ ;
        Pour j = {1,2, …, |Pi-1|}
        Faire
            x=sélection (Pi-1, i, j) ;
            Pi=Pi+Recombinaison (x, Pi-1, i, j) ;
        Fait ;
        Evaluer(Pi) ;
    Fait ;      Retourner BestOf (Pi) ;
Fin.
```

Remarque 1:
Hider utilise le codage hybride c'est-à-dire le codage binaire pour les attributs discrets et le codage réel pour les attributs continus.

Remarque 2 :
Il existe une autre version de Hider appelée Hider* qui utilise le codage naturel.

3.1.3.2 Par les algorithmes mémétiques: [BAC 2004] [BAC 2006]

Nous allons présenter l'algorithme **Memetic Pittsburgh Learning Classifier System (MPLCS)** qui utilise deux méthodes : **la méthode Rule set-wise et la méthode rule wise.**

1. Méthode Rule set-wise (MPLCS-RS):
Cette méthode utilise un opérateur de recherche locale et qui est intégré dans GAssist, il est défini par 03 phases :

- Génération des règles candidates.
- Sélection des règles qui forment la nouvelle génération**.**
- Génération de l'individu final.

Cet opérateur est intégré dans GAssist en utilisant deux stratégies :

1. **MPLCS-RS(P) :** Dans ce cas, l'opérateur RSW est appelé pour toute la population.
2. **MPLCS-RS(E) :** l'opérateur RSW est appliqué pour les meilleurs individus de la population (elitism fashion).

2. Méthode Rule-wise (MPLCS-R):
Cette méthode utilise trois opérateurs de recherche locale suivants :

1. Règle de nettoyage (**R**ule **C**leaning), noté **RC**.
2. Règle de fractionnement (**R**ule **S**plitting), noté **RS**.
3. Règle de généralisation (**R**ule **G**eneralizing), noté **RG**.

Ces trois opérateurs précédents sont intégrés à l'intérieure de GAssist en utilisant les stratégies suivantes :

1. **MPLCS-R(P):** dans cette stratégie un nouveau stage est ajouté dans le cycle de l'AG après mutation où les trois opérateurs précédents sont appelés pour chaque individu de la population avec une certaine probabilité comme illustre le pseudo code suivant :
2. **MPLCS-R(E):** appeler les opérateurs de recherche locale pour le meilleur individu de la population à la fin de chaque itération.
3. **MPLCS-**RS+R (P où E) : dans ce cas les opérateurs de recherche locale sont appliqués à l'intérieur de RSW après l'insertion de toutes les règles.

3.2 Une approche MultiObjectif Duale pour l'extraction de règles de classification (DOEA : A Dual-Objective Evolutionary Algorithm) [TAN 2006]

A présent, nous allons définir des approches d'extraction de règles par l'approche Michigan. Pour extraire la connaissance enfouie dans les données nous avons besoin d'extraire une connaissance compréhensible, faute de quoi elle n'est pas utilisable dans un domaine où la clarté et la facilité d'interprétation sont requises. La connaissance compréhensible est importante dans différents domaines d'application, où l'expert humain joue un grand rôle dans le processus de la résolution de problèmes. Prenant ces aspects en compte, les auteurs ont incorporé le concept de Pareto dominance, pour faire évoluer un ensemble de règles de décision, chaque liste possédant une précision et un nombre de règles placés au dessus d'un certain niveau spécifié à l'avance. Dans ce travail, les auteurs ont combiné la méthode de Michigan avec celle de Pitssburgh décrites dans la section 4.1. Pour cela, ils ont recours à la première approche pour produire un ensemble de règles suffisamment précises. Ils éliminent les règles qui ne couvrent pas ou très peu d'instances de la base d'apprentissage, puis forment un ensemble de règles présentées en entrée à la deuxième approche, formant un chromosome "liste de règles" (Pool of Rules) pour la deuxième phase. Le schéma de la figure 5.1 illustre bien l'hybridation des deux phases des travaux effectués.

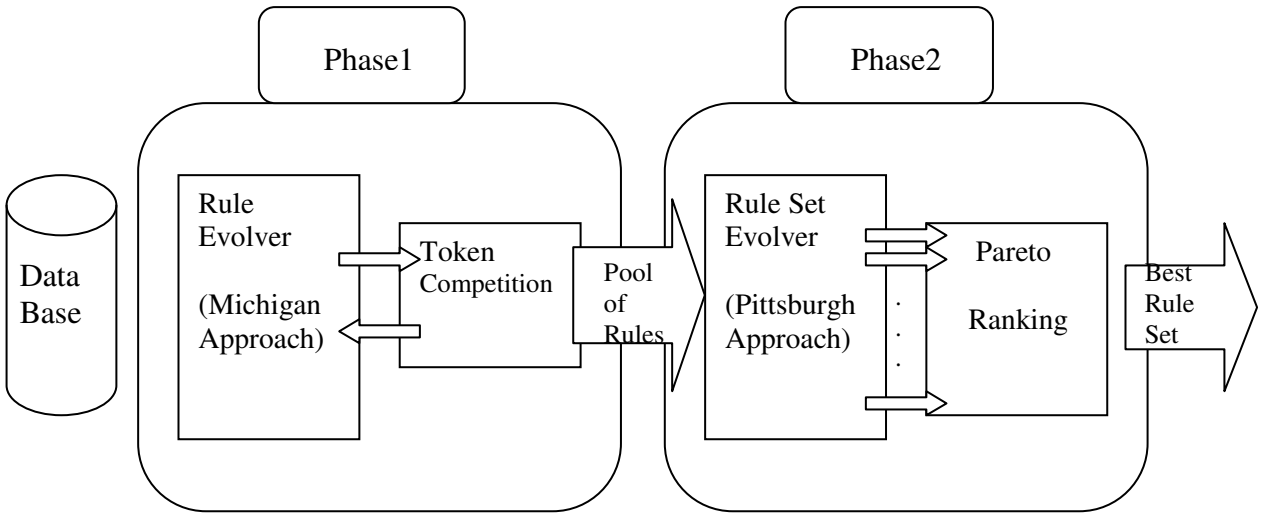


Figure 5.1: Approche DOEA pour l'extraction de règles de classification

La fonction de "ranking" est calculée de la manière suivante :
N1 = MIN + R1(MAX – MIN)
N2 = MIN + R2(MAX – MIN)
Où R1et R2 sont des paramètres choisis, N1 et N2 sont les rangs calculés en fonction de R1 et R2, MIN et MAX sont les bornes minimale et maximale d'un attribut donné des instances de la base considérée.
La fonction de fitness de la 1ere phase est définie par :

$$F = \frac{TP + w1 \cdot TN}{TP + w1 \cdot TN + FP + FN} \cdot \frac{1}{1 + w2 \cdot FP}$$

où : w1 et w2 sont des facteurs de poids ou de pénalité pour les valeurs de TN et FP.

Remarque : les valeurs TP, TN, FP et FN ont été définies dans un chapitre précédent.
"Token Competition" est définie dans [WON 2000].

3.3 Une approche hybride PSO/ACO pour l'extraction de règles de classification [FRE 2008]

Dans ce travail, il est décrit une hybridation de deux metaheuristiques récentes : l'une appelée **PSO** (Particle Swarm Optimisation [KEN 2001], l'autre étant une autre meta heuristique appelée **ACO** (Ant Colony Optimisation [DOR 2004].
Dans un travail antérieur des mêmes auteurs de 2005, il est aussi question de règles de classification où ils se sont inspirés de [SOU2004][FAL2007] pour obtenir un classificateur basé sur l'approche PSO uniquement, puis de [PAR 2002] pour obtenir un classificateur basé sur l'approche ACO qui s'était révélée d'un très bon apport pour l'extraction de règles de classification.

Ils ont proposé alors l'approche hybride PSO/ACO en 2005 qui a été reprise et améliorée en 2008 pour aboutir à PSO/ACO_2 .

Cette méthode utilise l'approche de Michigan et l'approche de couverture séquentielle où une règle extraite couvre un certain ensemble de tuples dans la base d'apprentissage qui sont supprimés avant de relancer l'extraction d'une nouvelle règle.

L'apport de ACO est la prise en charge des attributs nominaux directement sans avoir besoin de les représenter sous forme binaire, ce qui n'est pas le cas pour PSO : un attribut nominal possédant plusieurs valeurs nominales doit être converti (ses valeurs doivent prendre chacune d'elles deux valeurs binaires), pour pouvoir lancer l'extraction d'un modèle de classification par l'approche PSO.

Dans l'approche hybride, PSO/ACO_2 commence par extraire des règles de classification ne contenant que des attributs nominaux, en premier lieu. En second lieu, l'approche PSO est lancée pour finaliser les règles obtenues par ACO et leur rajouter les attributs numériques. Elle se base sur deux valeurs numériques pour chaque attribut continu : sa valeur minimale et sa valeur maximale, ie : les bornes de l'intervalle auquel appartient chaque attribut numérique.

Cet algorithme traite les attributs à valeurs nominales des instances d'apprentissage. Une autre partie est mise en œuvre pour traiter les attributs numériques, purement basée sur l'algorithme PSO. Les résultats obtenus sur 27 datasets de l'UCI sont très satisfaisants, comparés à ceux obtenus par l'algorithme PART [EIB 1998] qui est basé sur C4.5 bien connu.

Algorithme PSO/ACO_2

```
Initialise Population
Repeat for MaxIterations
    For every Particle X
        Set Rule Rx = "If Ø THEN C"
        For each dimension d in X
            Use Roulette Selection to choose whether the state should be set to Off or
          On. If it is on then the corresponding attribute value pair set in the
          initialisation will be added to Rx otherwise nothing will be added.
        Loop
        Calculate the quality Qx of Rx
        //set the past best position .
        P = x's past best state
        Qp = P's quality
        If Qx > Qp then
                Qp = Qx
                P = X
        End IF
    Loop
    For every particle x
        P = x's past best state
        N = the best state ever held by a neighbour of x according to N's quality QN
        For every dimension d in x
            //Pheromone updating procedure
          If Pd = Nd Then
                Pheromone.entry corresponding to,the value of Nd in the current X d is increased by Q p
          Else   If Pd = off and seeding term for xd ≠ Nd Then
                Pheromone.entry for the off state in xd is increasing by Qp .
                Else
                Pheromone.entry corresponding to,the value of Nd in the current x is increased by Qp.
                ENDIF
            Normalize pheromonone.entries
        Loop
    Loop
Loop
Return Best rule discovered
```

Nous avons décrit sommairement ces deux travaux car dans nos expérimentations nous avons comparé nos résultats aux leurs. D'autres travaux récents sur l'extraction des règles de classification basés sur une métaheuristique récente qui est "Honey Bees Mating Optimization" (HBMO)[AFS 2007] peuvent être consultés dans [MAR 2008],

4. Conception d'un Classificateur Génétique

Pour concevoir un classificateur basé sur une approche génétique, nous allons détailler les points suivants :

- Codage des règles
- Population Initiale
- Opérateurs Génétiques mis au point
- La Fonction d'Evaluation des règles extraites
- Le remplacement
- La construction du Classificateur

4.1 Codage des règles de classification

Dans la littérature nous avons trouvé deux représentations différentes de règles de classification, appelées "Approche de Michigan" et "Approche de Pittsburgh".

L'approche de Michigan consiste à coder une règle par un chromosome. Cela consiste à extraire les règles une par une, chacune nécessitant une exécution complète de l'AG. Il faut noter que certains auteurs utilisent le terme «approche de Michigan» dans un sens étroit du terme, pour se référer uniquement aux systèmes classificateurs [HOL1993][FRE1998]. Cependant, nous utilisons le terme «approche Michigan» dans un sens plus large, pour désigner toute approche où chaque individu de l'AG code une règle de prédiction unique. Voir la **figure 5.2**

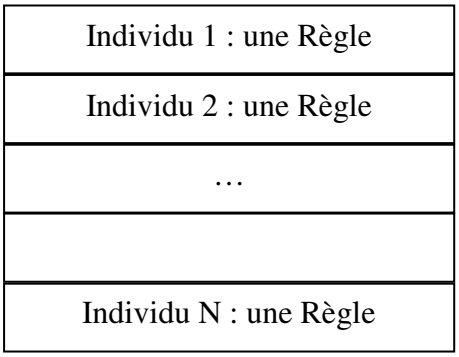


Figure 5.2: Population de l'AG dans l'approche de Michigan

Dans l'approche de Pittsburgh, introduite par Smith [SMI 1980], chaque individu représente un ensemble de règles qui sont générées dans la même itération. Un chromosome est donc un codage d'un ensemble de règles de classification. Tout le classificateur est extrait en une seule exécution de l'AG.

Cependant, à chaque itération de l'AG, le nombre de règles peut être différent avec la contrainte que les individus doivent garder la même taille. Pour cela, on ajoute des variables comportant la valeur -1 pour représenter des règles inexistantes. La représentation d'un individu dans cette approche dans une itération donnée est illustrée par l'exemple de la ***figure5.3***.

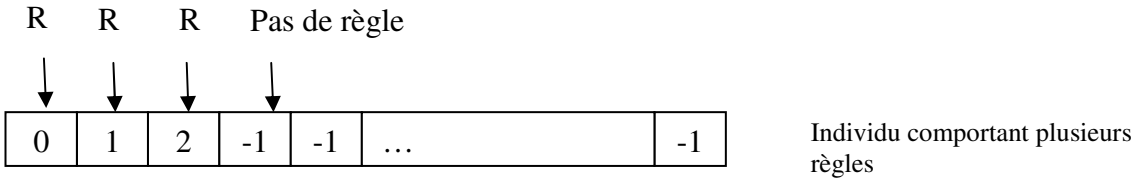


Figure 5.3 Représentation d'un classifieur par l'approche de Pittsburgh.

La population dans cette approche est illustrée par **la figure 5.4**

Règle	Règle	Règle	…..
Règle	Règle	Règle	…..
…..			
Règle	Règle	Règle	…..
Règle	Règle	Règle	…..

Figure 5.4: Population de l'AG dans l'approche de Pittsburgh

Cette approche a été décrite dans la section précédente de ce chapitre.

Des formes hybrides de ces deux approches sont mises en œuvre également dans [BAB2009] [Tan2010] pour la classification en data mining.

Le choix entre ces deux approches dépend fortement de quel type de règle nous voulons découvrir. Ceci est lié à quel type de tâche de data mining : la découverte de règles de classification ou bien de règles d'association. Supposons que la tâche est la classification. Si l'interaction entre les règles est importante, dans ce cas, l'approche de Pittsburgh semble plus appropriée et naturelle. D'autre part, l'approche de Michigan pourrait être plus naturelle dans certains cas. Un exemple est une tâche dont le but est de trouver un petit ensemble de règles de prédiction de haute qualité, et chaque règle est souvent évaluée de façon indépendante des autres règles [FRE 1998]. Un autre exemple est la tâche de détecter des événements rares (Nuggets Discovery) [IGL 2006]. D'une manière générale, on peut dire que si le temps est primordial dans l'extraction du classificateur alors l'approche de Pittsburgh est plus appropriée mais elle reste couteuse en mémoire. Par contre, si la précision de chaque règle extraite est plus importante, alors l'approche de Michigan semble être plus interessante du fait que chaque règle est évaluée indépendament des autres règles du classificateur.

Dans nos travaux nous avons favorisé l'approche de Michigan, elle est moins couteuse en mémoire puisque chaque chromosome code une règle, et elle donne des classificateurs hautement précis. Cette approche consomme plus de temps de calculs, nous verrons l'approche parallèle pour diminuer ce temps d'extraction. Par conséquent, la représentation d'une règle de classification est illustrée par la figure 5.5

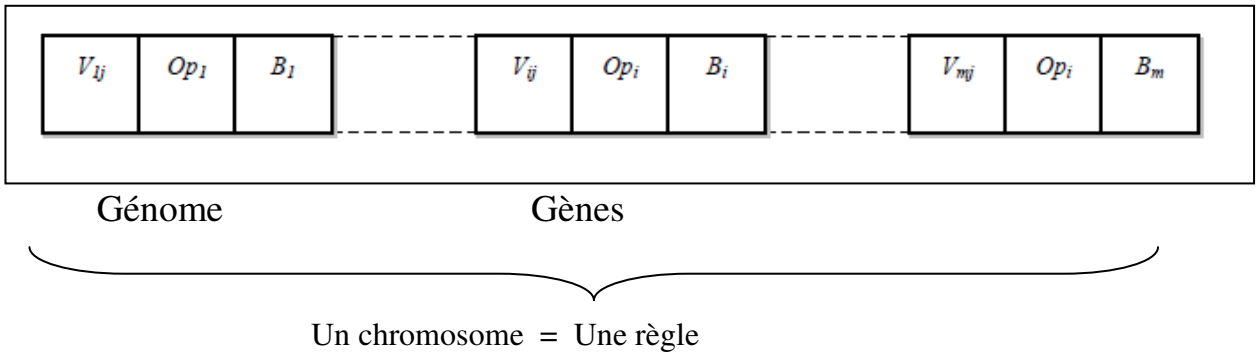


Figure 5.5 : Illustration du codage d'une règle dans notre approche

En fait chaque génome est constitué de 3 gènes :

- Le premier gène est un booléen qui spécifie si le génome (la prémisse) est actif ou désactivé, c'est-à-dire si la $i^{ème}$ condition est présente dans la régle où non. Nous avons utilisé cet artifice dans le but de garder la taille d'un chromosome tout le temps constante, sachant que les règles d'un classificateur n'ont pas toutes le même nombre de prémisses et donc pas la même taille..
- Le second gène correspond à l'opérateur de comparaison de l'attribut par rapport à une valeur que cet attribut peut prendre. Si l'attribut en question est nominal alors l'opérateur ne peut être que "=" ou "≠". Par contre si l'attribut en question est numérique alors l'opérateur peut être "≥", "≤".

- Le 3eme gène correspond à la valeur de l'attribut par rapport à laquelle il est comparé. Si l'attribut est nominal il prend des valeurs dans un ensemble discret,

Exemple : Couleur = { Bleu, Rouge, Vert }

Si l'attribut est numérique il prend des valeurs dans un ensemble continu de définition, exemple : 30≤Temperature≤70

La figure 5.6 représente un codage d'une règle de classification [BEN2011] dans notre modélisation.

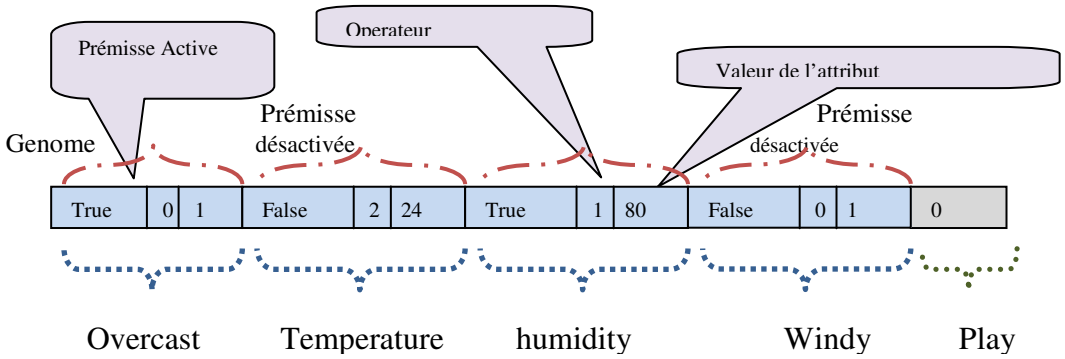


Figure 5.6 Une règle représentée par un chromosome

Ce chromosome ou individu de la population de l'AG correspond à la règle suivante:

If Outlook = Overcast AND Humidity ≤ 80 THEN Play = Yes

En fait, la taille du chromosome est :

Taille_chro= (nb_attributs – 1) *3 + 1

Où : nb_attributs est le nombre d'attributs dans la base d'apprentissage considérée, sans le dernier attribut qui correspond à la variable de décision ou classe de l'individu.

Il est à noter que les gènes sont rangés dans le même ordre que celui de leur représentation dans la base de données, de sorte que la valeur d'index d'un gène, représente la position de l'attribut associé dans cette base de données.

4.2 La population initiale

Un ensemble de chromosomes tels que définis précédemment constitue une population. C'est un ensemble de solutions potentielles au problème. Dans notre cas, nous générons un ensemble de chromosomes aléatoirement qui représentent tous une des règles du classificateur que l'on cherche à construire. Ceci est effectué par tirage aléatoire de chaque gène de chaque génome, les gènes étant tirés dans leurs domaines de valeurs respectifs.
Ensuite nous appliquons un algorithme pour sélectionner un certain nombre d'entre eux qui constituent des règles admissibles. On appelle règle admissible toute règle qui contient au moins une prémisse active. Puis, l'algorithme procède à l'élimination des règles redondantes. L'ensemble de règles final obtenu constitue la population initiale.

Ci-dessous un pseudo-code de création d'un chromosome de la population initiale :

```
Début
    Entier i, j ; i=0 ; j=0 ; entier L=taille du chromosome ;
    Tant que (i<L-1)
    Faire
        Chromosome[i]=fonction aléatoire(0,1) //Attribut actif ou inactif
        i++ ;
         //Voir le type de l'attribut j
        Si (Type de l'attribut(j) est nominal)
           Alors
                Chromosome[i]=0 ; //Opérateur =
           Sinon //Le type de l'attribut j est numérique
                Chromosome[i]= random (1,2) ;//Opérateur ≥ ou ≤
        Fsi ;
        i++ ;
        Chromosome[i]=random(valeur possible de l'attribut j) ;
         i++ ; j++ ;
    Fait ;
          Chromosome[L-1]= random(valeurs possibles de l'attribut classe) ;
Fin
```

Une fonction est appliquée par la suite, pour choisir les chromosomes non redondants et admissibles.

4.3 Les opérateurs Génétiques

Afin de pouvoir illustrer les opérateurs génétiques que nous avons mis au point pour ce problème, considérons les deux règles suivantes codées par :

1 0 0	0 2 80	1 2 75	1 0 1	1

0 0 2	1 2 65	1 2 80	0 0 1	1

Ces deux chromosomes codent les règles suivantes :

If Outlook = sunny AND Humidity ≥75 And Windy = False Then Play = No
If Temperature ≥65 AND Humidity ≥ 80 Then Play = No

4.3.1 Le croisement

L'opérateur de croisement dans sa définition générale effectue un tirage aléatoire du ou des points de coupure puis échange les parties de chromosomes, formant ainsi deux ou plusieurs solutions sans vérification de leurs admissibilités. On appelle solution admissible toute solution qui vérifie les contraintes de départ spécifiées pour un problème donné.
Dans notre cas, nous avons mis au point un opérateur de croisement dédié au problème de la classification dans lequel nous choisissons un point de coupure k d'une manière aléatoire mais tel que k soit un multiple de 3, sachant que les génômes sont de taille 3. En fait, nous avons choisi un point de coupure de manière à créer des individus directement admissibles, c'est à dire qui ne contiennent pas de génomes qui ne respectent pas la structure initiale:

Actif Operateur Valeur

Ainsi, nous n'avons pas besoin de vérifier la structure initiale des solutions produites, ce qui entraine un gain de temps énorme sachant que sans cela il aurait fallu effectuer tous les croisements puis vérifier l'admissibilité de chaque enfant produit puis de supprimer ceux qui ne sont pas conformes et enfin produire encore d'autres enfants, jusqu'à atteindre le nombre d'enfants voulus et spécifié par la probabilité de croisement.

Ensuite, le point de coupure ne doit pas être l'index du dernier gène, c'est à dire la classe prédite par la règle, cette dernière doit rester fixe et hors de manipluations génétiques.

Supposons k=6, les individus créés par croisement sont donc:

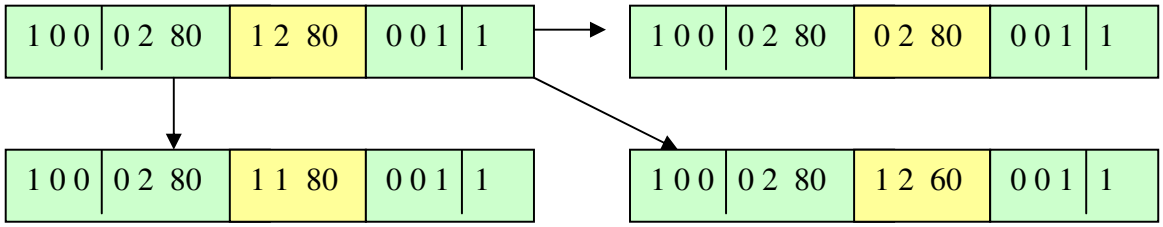
1 0 0	0 2 80	1 2 80	0 0 1	1

0 0 2	1 2 65	1 2 75	1 0 1	1

4.3.2 La mutation

L'opérateur de mutation quant à lui, doit préserver également le gène de la classe.
D'autre part, on procède au tirage aléatoire:

- du génome qui va subir la mutation,
- puis un autre tirage aléatoire du gène qui subira la mutation,
- Enfin, un tirage aléatoire dans le domaine des valeurs du gène concerné par la mutation.



4.4 Evaluation des individus de la population

Chaque individu (ou règle) apporte une solution potentielle au problème à résoudre. Néanmoins, ces solutions n'ont pas toutes le même degré de pertinence. C'est à la fonction de performance (Fitness) de mesurer cette efficacité pour permettre à l'AG de faire évoluer la population dans un sens bénéfique pour la recherche de la meilleure solution. Autrement dit, la fonction de performance (Fitness) doit pouvoir attribuer à chaque individu un indicateur positif représentant sa pertinence pour le problème que l'on cherche à résoudre. Notons qu'il n'y a pas de fonction précise permettant de calculer avec exactitude la performance de chaque individu, pour cela nous avons opté pour les critères suivants :

- Maximiser la couverture de la règle.
- Maximiser le taux de précision de la règle.
- Minimiser la taille de la règle car la compréhensibilité de la règle est mesurée par le nombre d'attributs qui la composent.

Ce problème est donc un problème multiobjectifs. Pour le résoudre nous avons opté pour le ramener à un problème monocritère en utilisant une méthode d'agrégation.
Pour cela, nous avons choisi de définir notre fonction d'adaptation par :

$$\mathbf{Fitness} = \lambda 1 * \frac{\boldsymbol{Couverture}}{\boldsymbol{Nb\_Instances}} + \lambda 2 * \frac{\boldsymbol{TP}}{\mathbf{Couverture}} - \lambda 3 * \frac{\boldsymbol{Nb\_Att(Règle)}}{\mathbf{Nb\_Att\_Total}}$$

Où λ_i est une valeur comprise entre 0 et 1, avec $\sum_{i=1}^{3} \lambda_i = 1$
Cette fonction est évaluée par le calcul de chacune de ses composantes.

4.4.1 Fonction Couverture : Il s'agit d'une fonction qui calcule la couverture d'une règle R de la forme (R : A → C) par rapport à la base d'apprentissage, Nous disons que l'instance ***inst*** est couverte par la règle *R* si et seulement si la partie de l'instance *inst* sans l'attribut de classe, satisfait la partie prémisse A de la règle, peut importe la classe C. Ci-dessous, le pseudo code qui vérifie si une instance de la base d'apprentissage est couverte par la règle R, ainsi que le pseudo code qui calcule la couverture d'une règle.

Boolean Fonction **Est_Couverte**(Instance ***inst*** , Règle ***R***)

Début
Entier Nb = nombre d'attributs de la partie condition de la règle ***R***;
Booléen B = Vrai ;
Pour i = 1 à (Nb-1)
Si (le i^{ème} antécédent de l'instance ***inst*** ne satisfait pas le i^{éme} attribut de la règle ***R***)
Alors B=***Faux*** ;
Fsi ;
Fait ;
Retourner B ;
Fin.

Entier Fonction **Couverture** (Règle ***R***)
Début
Entier N = nombre d'instances de la base de données ;
Entier Couverture = 0 ;
Pour i = 1 à (N)
Faire
// On vérifie si la i^{ème} instance ***inst*** $_i$ est couverte par la règle ***R***
Si (**Est_Couverte** (la i^{ème} instance ***inst*** $_i$, la règle ***R***) = Vrai)
Alors Couverture++ ;
Fsi ;
Fait
Retourner Couverture ;
Fin

4.4.2 Calcul de TP (True Positif)

TP est une fonction qui calcule le nombre d'instances de la base d'apprentissage qui satisfont la partie condition et la partie classe de la règle. En fait, cela représente le nombre d'instances bien classées par la règle.
Ci dessous un pseudo code qui permet de vérifier si la partie prémisse et la partie classe d'une instance de la base d'apprentissage sont bien vérifiées par la règle R et un autre qui permet de calculer TP.

Boolean Fonction **Instance_TP**(Instance ***inst*** , Règle ***R***)
Début
Entier Nb = nombre d'attributs de la partie condition de la règle ***R***;
Boolean couvre =vrai ;;
Pour i = 1 à (Nb-1)
Faire
Si (le i^{ème} antécédent de l'instance ***inst*** $_i$ ne satisfait pas le i^{éme} attribut de la règle ***R***)
Alors couvre= ***Faux*** ;
Fsi ;
Fait
Si (couvre = vrai) ET (la classe de l'instance ***inst*** est égale à la classe de la règle **R**)
Alors Retourner ***Vrai*** ;
Sinon Retourner ***Faux*** ;
Fsi ;
Fin.

Entier Fonction **TP** (Règle ***R***)
Début
Entier N = nombre d'instances de la base de données ;
Entier TP = 0 ;
Pour i = 1 à N
Faire
// On vérifie si la partie antécédente de la i^{ème} instance ***inst*** $_i$ est satisfaite par la partie //condition de la règle ***R*** et si la classe de l'instance est égale à la classe de la règle.
Si (**Instance_TP** (la i^{ème} instance ***inst*** $_i$, la règle ***R***) = Vrai)
Alors TP ++ ; Fsi ;
Fait ; Retourner TP ;
Fin.

Quant au nombre d'attributs d'une règle, il est aisé de le calculer : c'est le nombre de prémisses de la règle.

La fonction d'évaluation est ainsi calculée pour chaque individu de la population de l'AG.

4.5 Le remplacement et la Sélection

Il existe plusieurs stratégies de remplacement dans les algorithmes génétiques. Dans notre application, nous avons choisi de remplacer le plus mauvais individu de la population par l'individu résultant du croisement et de la mutation, au fur et à mesure. Cette stratégie est dite "Steady State".

Pour la sélection, nous avons opté pour la sélection par tournoi qui consiste à choisir n individus aléatoirement dans la population et de choisir les deux meilleurs individus parmi les n, dans le but de procéder au croisement.

4.6 Application du cycle de l'AG :

Le cycle de l'AG peut être résumé par le pseudo code suivant :

Début
Séléctionner deux individus :
Individus 1=**Tournament Selection()** ;
Individus 2=**Tournament Selection()** ;
Enfant=**Croisement(Individu 1, Individu 2)** ;
Enfant_Muté=**Mutation(Enfant) ;**
Remplacement(Enfant_Muté) ;
Fin.

5. Algorithme d'extraction des règles de classification

Dans [FRE1998], A. Freitas a décrit un algorithme de couverture séquentielle ("Sequential Coverage"). Dans cette approche, la base de données apprentissage est divisée en deux parties : une partie servira à construire le modèle (ie : le classificateur) et l'autre partie contenant les instances dont on a supprimé l'attribut de classe. Ceci est effectué dans le but d'utiliser la 1ere partie de la base à extraire le modèle puis la 2eme partie à tester les

performances du modèle extrait. On calcule alors la précision du modèle qui est le taux des instances de la base de tests bien classées par le modèle. D'autre part, une autre technique appelée "Cross Validation" consiste à partager l'ensemble des instances en petits paquets pour former au hasard un ensemble d'apprentissage et un autre pour les tests. On réitère plusieurs fois l'opération pour calculer la performance du classificateur extrait.
L'approche de couverture séquentielle que nous avons adoptée dans tous nos travaux, consiste à extraire une règle de classification puis de rechercher toutes les instances couvertes par cette règle dans la base d'apprentissage et enfin de marquer (ou supprimer) toutes ces instances, avant de relancer l'extraction d'une nouvelle règle. Ce processus est répété tant que des instances non couvertes dans la base existent ou bien on ne trouve aucune nouvelle règle qui couvre au moins une instance ou alors après un certain nombre d'itérations fixé expérimentalement.

Procédure Construction du classificateur

```
Début
        Classifieur={vide};
        Nb_Inst=le nombre d'instances de la base de données d'apprentissage ;
        Booléen : couvre;
        Tant que(le critère d'arrêt n'est pas atteint)
        Faire
              Appliquer le cycle de l'AG ;
              Individu = le meilleur individu obtenu par l'AG ; //la meilleure solution
              Règle= Décoder(Individu) ;
              couvre=faux ;
              Pour i=1 à Nb_Inst  //Chercher toutes les instances couvertes par cette règle
              Faire
                    Si (Est_Couverte(règle, instance))
                       Alors
                             Supprimer cette instance de la base de données ;
                             couvre=vrai ;
                    Fsi ;
              Fait ;
              Si (couvre=vrai)
                 Alors Règle= Décoder(Individu) ;

                       Ajouter règle au classificateur ;
              Fsi ;

        Fait ;
Fin.
```

Les avantages de cet algorithme :

- si une règle extraite ne couvre aucune instance de la base d'apprentissage elle est abandonnée et non récupérée dans l'ensemble de règles qui constituent le classificateur. Une autre variante serait d'imposer un seuil minimum de couverture d'instances pour lequel la règle est acceptée sinon elle est rejetée.

- Le principe de la généralité est pris en charge et la non redondance des règles également.

Les fonctions utilisées dans la procédure de construction :

Nous avons utilisé dans cette procédure deux fonctions :

1. La fonction ***Décoder*** **:** cette fonction permet de transformer un individu en une règle,
2. La fonction Est_Couverte(Règle, Instance) .

Dans ce qui suit le pseudo-code de cette fonction :

Fonction booléen Est_Couverte(Règle_extraite, instance)

```
Début
        Nb_Prm=nombre de prémisse de Règle_extraite ;
        Entier cpt=0 ;
        Pour i=1 à Nb_Prm
        Faire
            Si (ième prémisse de règle_extraite vérifie OU = ième prémisse de l'instance)
               Alors
                      cpt++ ;
            Fsi ;
        Fait ;
        Si (cpt=NB_Prm)
            Alors
                  Retourner (vrai) ;
            Sinon
                  Retourner (faux) ;
        Fsi ;
Fin.
```

Remarque :

Le critère d'arrêt pour la construction de notre classificateur serait :

- On arrête l'exécution de l'algorithme lorsque la base de données est vide : toutes les instances ont été couvertes, étant donné que les instances couvertes par une règle extraite sont supprimées de la base avant de lancer une nouvelle exécution de l'AG pour extraire une nouvelle règle.
- On arrête l'exécution au bout d'un certain nombre d'itérations : l'algorithme risque de se dérouler de manière infinie dans le cas où il existe des instances dans la base de données qui ne peuvent être couvertes par aucune règle. Ce sont par exemple des individus qui représentent un phénomène rare et n'appartiennent à aucune classe, ils peuvent être classés "divers" par exemple.

6. Conclusion

Dans cette étude, nous avons détaillé les concepts utilisés afin de construire un classificateur par une approche génétique pure. Nous avons détaillé les codages utilisés pour extraire une règle de classification à partir d'un ensemble d'apprentissage, l'approche utilisée étant celle

de Michigan. L'AG est ré-exécuté tant que l'on peut encore extraire d'autres règles. Cependant, l'AG est un algorithme qui donne de bonnes solutions à condition qu'il soit bien adapté au problème traité. Il est connu également pour converger assez rapidement dans certains problèmes.

Par ailleurs le calcul de la fonction fitness est le plus grand consommateur du temps total de l'AG. Pour cela, dans les prochains chapitres, tout ce formalisme sera gardé dans les autres contributions. Nous mettrons en œuvre d'autres concepts pour améliorer les temps de calcul par une approche parallèle, ainsi que la précision des classificateurs obtenus par une approche mémétique.

1. Introduction

L'objectif que l'on s'est fixé est de construire un nouvel algorithme génétique parallèle, basé sur un schéma parallèle Maître/Esclave de l'AG, ensuite de l'appliquer au problème de la classification.

Un compromis entre qualité des connaissances extraites, précision, et temps de calcul étant à trouver, il s'agit d'apporter des améliorations à l'AG séquentiel dans une première étape, de manière à pouvoir proposer un nouveau schéma parallèle de l'AG, basé sur le modèle Maître/Esclave.

Globalement, notre conception se divise en deux parties principales :

- Conception du nouvel algorithme génétique, et son application au problème de la classification.
- Parallélisation du nouvel algorithme, et son application au problème de la classification, la minimisation du temps d'exécution étant l'objectif visé.

2. Proposition de nouveaux concepts pour un nouvel AG

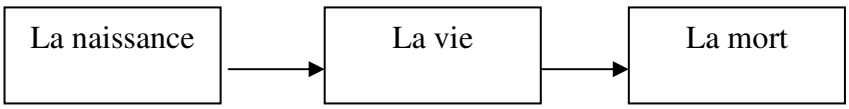
Les algorithmes génétiques sont des outils efficaces pour une classe de problèmes très larges. De plus, ils permettent de traiter des problèmes où la fonction à optimiser ne présente aucune propriété de continuité ou de dérivabilité, par exemple. Cependant, et afin de garantir l'efficacité de ces algorithmes, le calcul **d'un très grand nombre de fitness** (parfois de l'ordre de plusieurs dizaines de milliers) est généralement nécessaire avant l'obtention d'une bonne solution. Ce nombre de calculs important peut s'avérer problématique lorsque le coût de calcul (ressources systèmes ou temporelles) de la fitness est important, lorsqu'on travaille en grande dimension sur des fonctions à complexité importante. Le but principal est de **diminuer le temps d'exécution** de l'AG, tout **en maintenant une bonne qualité de la solution** obtenue.

Dans ce qui suit, l'idée globale du nouvel algorithme génétique qui s'inspire fortement des phénomènes réels propres aux êtres vivants, est explicitée. Notre nouvel algorithme fait évoluer une population d'individus, sur plusieurs itérations, à l'aide de différentes opérations. Pour cela un individu de la population possède un cycle de vie et une durée de vie. Toutes ces nouvelles notions seront détaillées par la suite et nous montrons dans ce qui suit comment l'algorithme aboutit à une solution proche de l'optimale (meilleur individu de la population) d'un problème donné.

2.1 Cycle de vie d'un individu dans l'algorithme :

Avant d'expliciter le cycle de vie d'un individu dans notre population, il est à noter que nous avons introduit un nouveau paramètre dans l'AG : la "Durée de Vie Maximale". C'est un nombre entier positif inférieur au nombre total d'itérations de l'AG et qui représente un nombre d'itérations **maximum** pendant lequel un individu peut participer au processus évolutionnaire. Cette durée de vie maximale est définie à l'initialisation de l'algorithme.

Le cycle d'un individu dans notre population est caractérisé par trois étapes, à savoir :



- **La naissance :**

La naissance signifie la production d'un nouvel individu dans la population.

L'individu produit est considéré comme "NouveauNé" tant que sa fitness (ou fonction d'adaptation) ne possède pas encore de valeur. Il gardera ce statut tant que cette valeur n'a pas été calculée.

Une durée de vie et une date de mort lui sont affectées à cet instant précis, telles que :

- **La durée de vie :**

La durée de vie d'un individu est le nombre d'itérations qu'il peut vivre (exister dans la population) et participer au processus évolutionnaire.

C'est une valeur entière tirée aléatoirement, appartenant à l'intervalle :

[1, durée de vie maximale]

Supposons qu'un individu « ***i*** » est produit à l'itération k de l'algorithme.

On suppose aussi que la durée de vie maximale est égale à l, c'est-à-dire qu'aucun des individus de la population ne peut participer au processus évolutionnaire plus de l itérations.

La durée de vie de l'individu « ***i*** » est un entier e tel que : $1 \leq e \leq l$.

La date de mort de « ***i*** » est égale à k+e, à condition que $(k+e) \leq N$, le nombre d'itérations maximum de l'algorithme. Ainsi l'individu « ***i*** » mourra (disparaîtra de la population) à l'itération (k+e) de l'algorithme.

- **La date de mort :**

C'est la somme entre la durée de vie de l'individu et l'itération courante de l'algorithme.

- **La vie :**

Un individu est dit "Vivant" s'il a déjà été évalué, c'est-à-dire que sa fonction fitness possède une valeur.

- **La mort :**

Un individu disparaît de la population s'il a atteint sa date de mort,

ie : sa date de mort = itération courante de l'algorithme.

2.2 Population de l'algorithme

La population est constituée d'un ensemble d'individus hétérogènes :

- Individus "Nouveaux Nés"
- Individus "Vivants".
- Individus "Morts" ou en "Fin de Vie".

Lors de l'initialisation de la population, les individus sont générés d'une façon aléatoire (ou par une méthode donnée) et considérés tous comme des "Nouveaux Nés" (naissance). Chacun d'entre eux possède sa propre durée de vie, générée aussi d'une façon aléatoire. Pour cela, la durée de vie peut différer d'un individu à un autre.

Pendant l'évolution de la population :

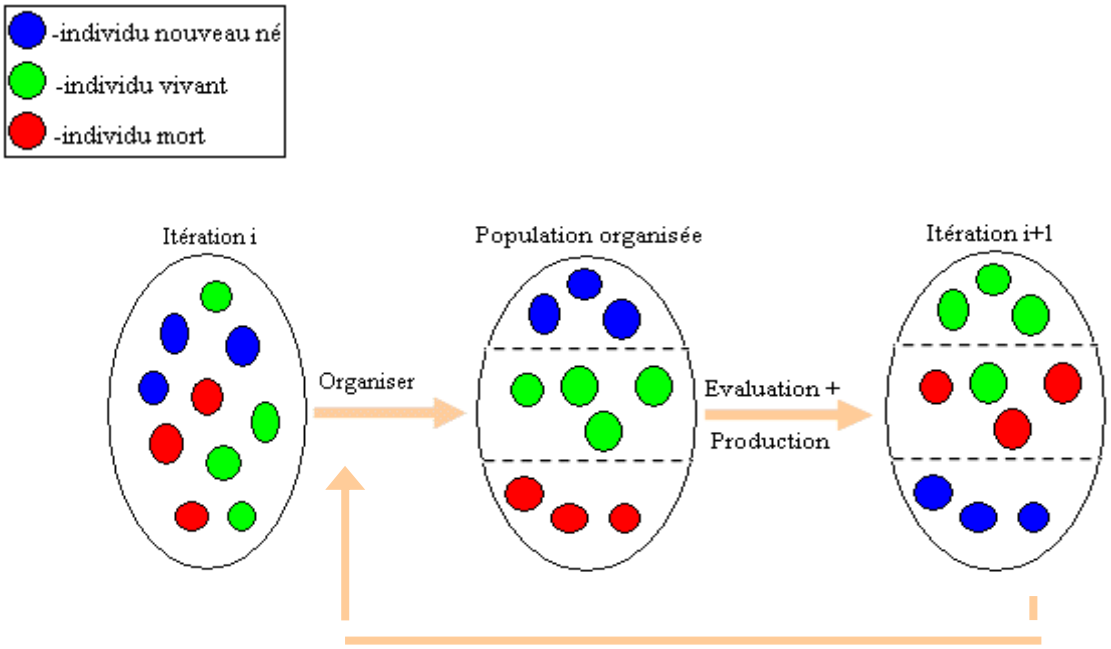
- des individus meurent : ceux dont la date de mort vient d'être atteinte
- des individus naissent et remplacent les disparus.
- des individus restent en vie pour un certain nombre d'itérations encore, tant qu'ils n'ont pas atteint leurs dates de fin de vie.

Pour cela, notre population est hétérogène.

2.3 Opérations de l'algorithme

Les opérations successives utilisées dans le nouvel algorithme génétique pour faire évoluer la population sont illustrées sur la figure 6.1

Légende :



. **Figure 6.1: Etapes de l'algorithme**

2.3.1 Organisation de la population

On commence par organiser les individus de la population comme illustré par la figure 6.1. Nous avons séparé les individus "Nouveaux Nés" des individus ayant atteint le statut de fin de vie et ceux qui ont le statut de "Vivants" dans le but de pouvoir les retrouver facilement dans la population, pour permettre l'évaluation des "Nouveaux Nés" ainsi que l'évolution des "Vivants" indépendamment les uns des autres.

On évalue chaque individu nouveau né, c'est-à-dire on procède au calcul de la valeur de leur fonction fitness et on remplace dans la population des individus "Morts" par ceux "Vivants". Ceci est effectué afin de pouvoir maintenir la taille de la population tout le temps fixe.

2.3.2 Production des individus

Deux parents (**individus vivants**) sont ensuite sélectionnés (P1 et P2) en fonction de leurs adaptations selon une méthode de sélection. On applique aléatoirement l'opérateur de croisement : nous avons choisi de générer un seul enfant C1 au lieu de deux comme dans le cas général. On modifie ensuite un gène de C1 en appliquant l'opérateur de mutation avec la probabilité Pm, ce qui produit un nouvel individu C'1(**nouveau né**) qui remplace un individu mort. On réitère les opérations de sélection, de croisement et de mutation afin de remplacer

tous les individus en statut de "morts" de la population, ceci termine le processus d'évolution d'une itération.

2.4 Fonctionnement de l'algorithme

Globalement, le fonctionnement de l'algorithme peut être résumé par le pseudo-code ci-dessous :

```
Début
    - Initialiser le nombre d'itérations total (nb_itérations) de l'algorithme.
    - Initialiser la durée de vie maximale.
    - Initialiser_la_population.
    - Pour chaque itération de 1 à nb_itérations
      Faire
          - Organisation( population )
          - Evaluation (nouveaux nés)
          - Production
      Fait
Fin

Fonction Initialiser_la_population ( )

 Début
    - Créer n individus aléatoirement. // n = taille de la population.
    - Pour chaque individu
      Faire
        durée de vie  = aléatoire (1, durée de vie max)
        date de mort = itération _courante de l'algorithme + durée de vie
        nn  = vrai // booléen qui signifie que l'individu est un nouveau né.
                   //Ce flag prendra la valeur "faux" dés que l'individu a été évalué
        vivant = vrai // booléen qui signifie que l'individu est vivant.
        Remplir chromosome ( )//permet d'initialiser le chromosome par tirage
                              //aléatoire de chacun de ses gènes dans la plage des
                               //valeurs qu'il peut prendre
      Fait
    - Retourner (pop) //objet population(transmis dans les procédures de l'algorithme).
Fin

Procédure  Evaluation (Nouveaux nés, population)
   Début
      Evaluer chaque individu "Nouveau né". // Calculer la valeur de sa fitness .
    Fin
```

Procédure Organisation (population)
Début
- Insérer les individus "Nouveaux nés" au sommet de la population puis les vivants, ensuite les morts.
- Pour chaque "Nouveau né"
Faire
nn = faux // car il n'a pas encore de valeur de fitness.
vivant = vrai
Fait
- Pour chaque Individu avec vivant=vrai
Faire
Si (date de mort = itération courante de l'algorithme)
alors
vivant = faux // il a atteint sa date de mort.
fsi
Fait
Fin

En fait, nous avons eu recours à des flags booléens "nn" et "vivant" dont le but est de pouvoir retrouver les différents individus de la population. Un individu arrivé en fin de vie possède les flags nn=faux et vivant=faux. Un "Nouveau né" possède les flags nn=vrai et vivant=vrai et enfin un individu "vivant" possède les flags nn=faux et vivant=vrai.

Par ailleurs, nous avons mis au point ces procédures dans le but de garder la taille de la population fixe, sachant que chaque individu arrivé en fin de vie est remplacé par un "Nouveau né", pour assurer ainsi la convergence de l'algorithme.

Procédure **Production** (population)
Début

- Pour chaque individu mort
Faire
-Sélectionner deux individus "Vivants"
-Leur appliquer l'opérateur de croisement avec la probabilité de croisement Pc.
-Appliquer l'opérateur de mutation sur l'individu généré avec la probabilité de mutation Pm.
-enfant.durée de vie = aléatoire (1, durée de vie max)
-enfant.date de mort = itération _courante de l'algorithme + enfant.durée de vie
- enfant.nn = vrai
- enfant.vivant = vrai
-Remplacer l'individu mort par l'enfant produit.
Fait
Fin

2.5 Paramètres de l'algorithme

La rapidité et la qualité du nouvel algorithme sont influencées par de nombreux paramètres, à savoir ceux de l'AG classique d'abord, ensuite le plus important :

- **La durée de vie maximale**, qui fixe le nombre d'itérations maximum qu'un individu peut vivre (exister dans la population). Sa valeur à une influence très importante sur la qualité de convergence de l'algorithme. En effet, si la durée de vie maximale possède une valeur assez grande par rapport au nombre d'itérations maximum de l'algorithme, un individu vit longtemps dans la population : il y a peu d'individus qui meurent, on ne peut donc pas produire de "Nouveaux Nés" sachant qu'un "Nouveau né" remplace un mort, pour maintenir la taille de la population fixe. Par ailleurs, si la durée de vie maximale est assez faible, les individus n'ont pas le temps d'évoluer suffisamment dans la population. Ils sont vite retirés de la population même s'ils possèdent d'excellentes performances.

Ce paramètre est donc à choisir adéquatement et expérimentalement.
Pour éclaircir ce point, nous avons appliqué le nouvel AG à un problème simple qui consiste à maximiser le nombre de "1" dans un chromosome binaire. Les résultats obtenus sont illustrés par les figures suivantes 6.3

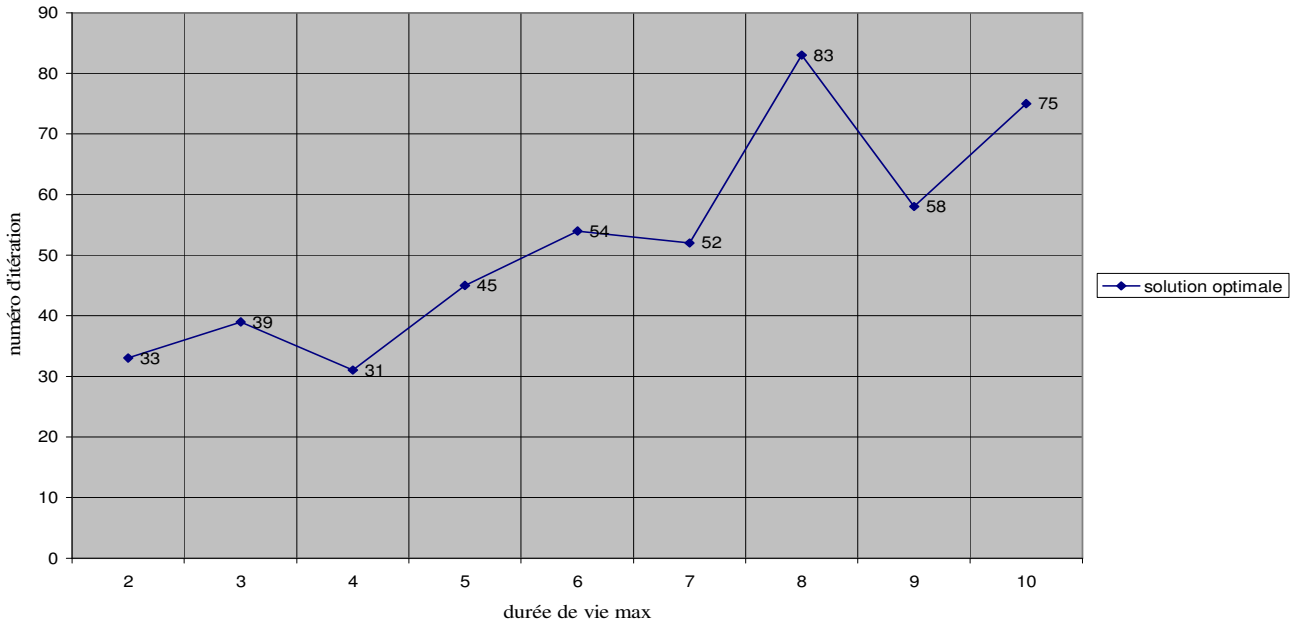


Figure 6.3 Influence de la durée de vie max sur la convergence de l'algorithme

Pour une population de 200 individus créés aléatoirement et un nombre d'itérations de l'ordre de la centaine, nous remarquons que plus la durée de vie maximale est importante et plus le nombre d'itérations pour aboutir au chromosome final (contenant que des "1") est important. Pour ce problème, la durée de vie maximale est égale à 4 tel que obtenu dans la figure 6.3, pour que le nombre d'itérations pour obtenir le meilleur chromosome soit minimum (de l'ordre de la trentaine dans ce cas).

Dans le graphe de la figure 6.4, il apparaît clairement que pour une population de 200 individus, pour 2500 itérations, plus on a augmenté la durée de vie maximale et plus le nombre d'itérations pour obtenir le meilleur chromosome est important. Sur la figure suivante la durée de vie maximale égale à 80 a engendré le nombre d'itérations minimum pour converger vers la meilleure solution.

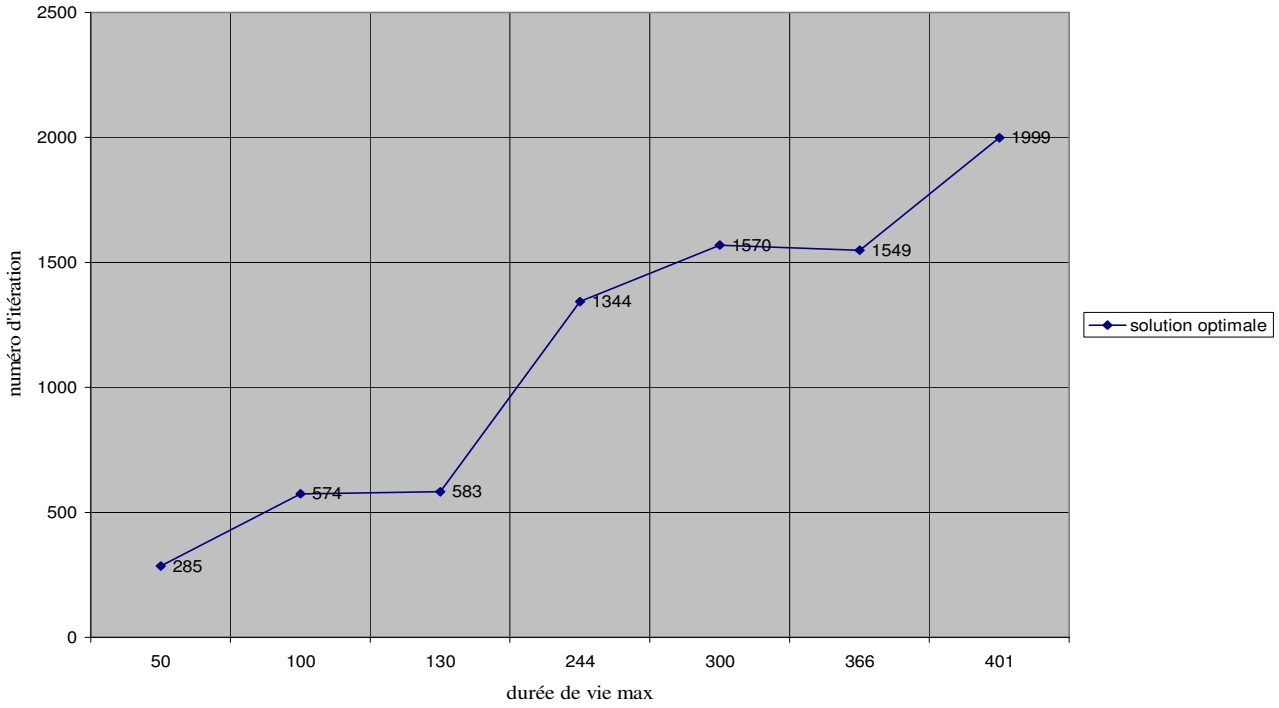


Figure 6.4 : Durée de vie maximale pour un grand nombre d'itérations de l'AG

D'où, nous avons conclu que pour un nombre d'itérations égal à 100 la durée de vie maximale est égale à 4 et pour un nombre d'itérations égal égal à 2000 la durée de vie maximale est égale à 80. Nous avons donc un rapport de 4% : la durée de vie maximale qui donne la solution optimale est de l'ordre de 4% du nombre d'itérations total de l'algorithme.

Après cela, nous avons étudié la diversité de la population en terme d'hétérogénéité de la population. Les histogrammes suivants montrent le nombre d'individus "Nouveaux Nés", "Vivants" et "Morts" dans la population en fonction de la durée de vie maximale.

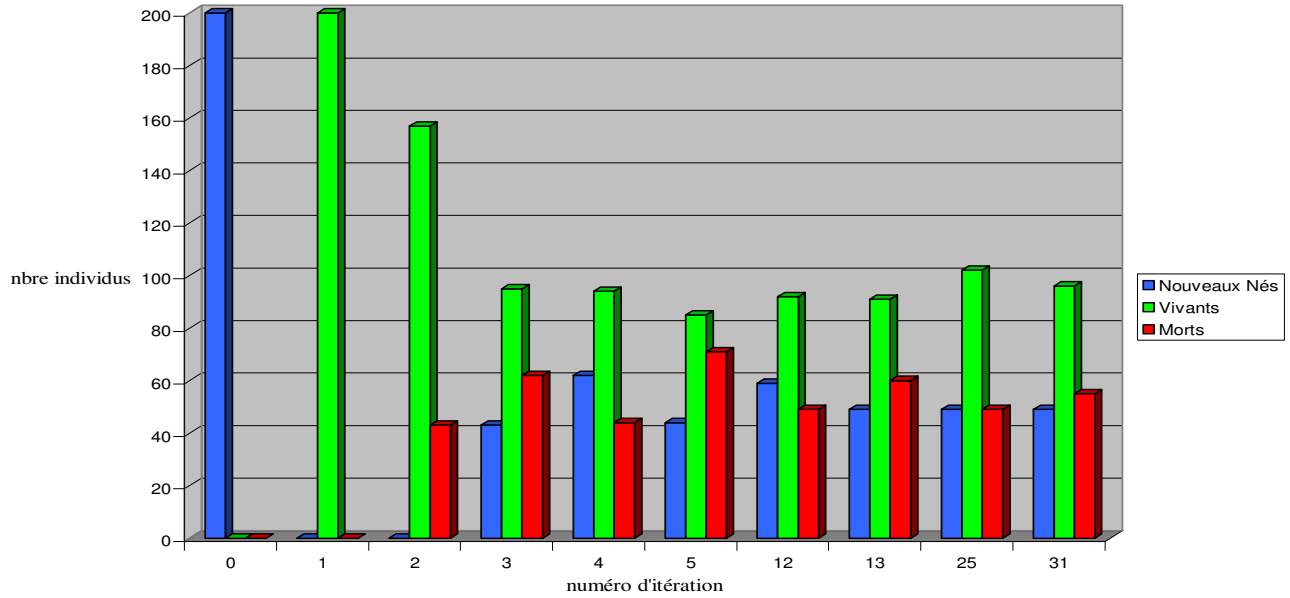


Figure 6.5 : Hétérogeinité de la population pour une petite durée de vie maximale

Au départ, tous les individus sont considérés comme "Nouveaux Nés", à leur création, puis ils deviennent tous "Vivants" juste après le calcul de leur fonction fitness, ensuite la population commence à être hétérogène après les productions. On remarque que pour une population de 200 individus et un nombre d'itérations = 100, la durée de vie maximale = 4 a donné une bonne répartition de la population, à savoir de l'ordre de 50% de "Vivants", 25% de "Nouveaux Nés" et 25% de "Morts".

Dans l'histogramme suivant, nous avons opté pour un nombre d'itérations égal à 2000 et une durée de vie maximale égale à 400.

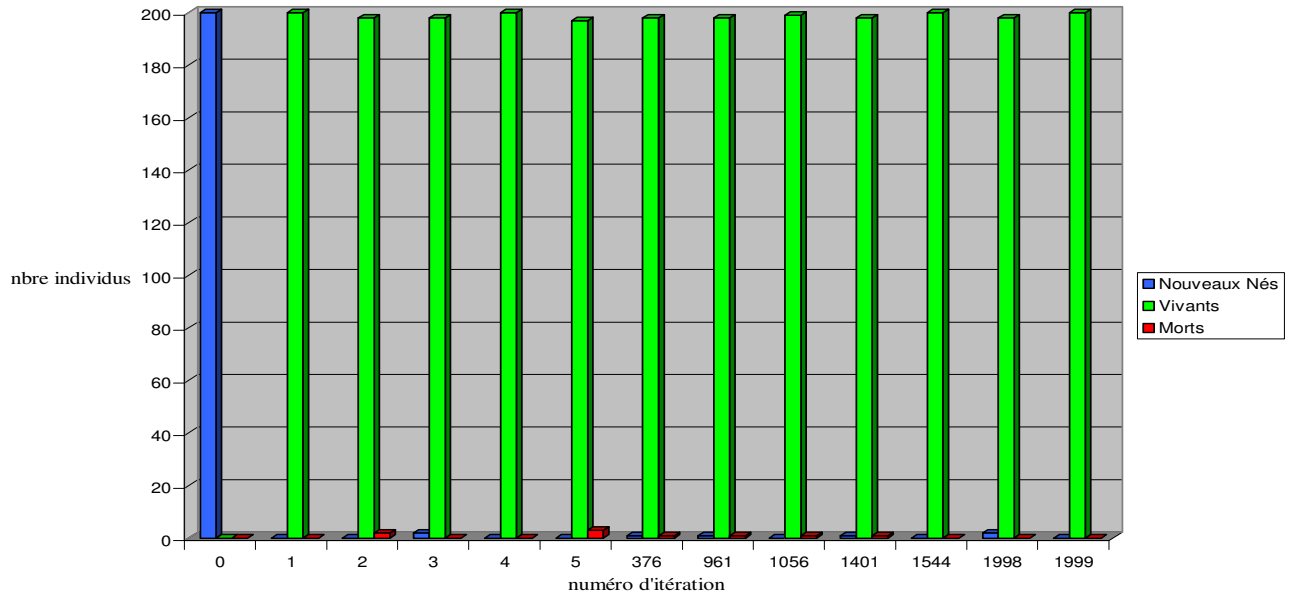


Figure 6.6 : Hétérogeinité de la population pour une large durée de vie maximale

Il est clair ici que le nombre d'individus "Vivants" est très grand par rapport aux autres types d'individus : la production stagne car il n'y a pas d'individus "en fin de cycle" pour pouvoir les remplacer par des nouveaux.

En conclusion, le paramètre Durée_de_Vie_Maximale est important à trouver, nous avons pu montrer qu'il est de l'ordre de 4% du nombre total d'itérations de l'algorithme.
Par ailleurs et afin de montrer la rapidité de ce nouvel algorithme, on compare sa complexité de calcul à un algorithme génétique classique, et cela en terme de nombre d'individus à évaluer, du moment où l'étape d'évaluation consomme plus de 80% du temps de calcul total de l'algorithme.

En effet, dans un algorithme génétique classique, les individus d'une génération ***g*** sont tous évalués, donc si on considère que la population est de taille ***n***, alors la génération ***g*** est caractérisée par ***n*** évaluations, ce qui implique ***n*k*** évaluations pour ***k*** générations.

Par contre, dans le nouvel algorithme, nous ne procédons qu'à l'évaluation des "Nouveaux Nés" et non pas TOUS les individus de la génération. Si le nombre de "Nouveaux Nés" est de l'ordre de 25% de la population totale donc le quart seulement des individus seront évalués, c'est-à-dire : ***n/4*** évaluations pour une population de taille ***n à chaque itération***, ce qui implique ***n*k/4*** évaluations pour ***k*** itérations.

3. Proposition d'un schéma parallèle du nouvel AG sans génération

3.1 introduction

Tout le formalisme et les nouveaux concepts que nous avons introduits sur l'AG classique ont été effectués dans le but de proposer un nouveau schéma parallèle de l'AG, basé sur le schéma Maitre/Esclave. En effet, étant donné que la fonction fitness est la plus consommatrice en temps de calcul de l'AG, nous avons choisi le modèle Maitre/Esclave qui consiste à utiliser des processeurs esclaves pour l'évaluation en parallèle des individus de la population.

Dans la littérature nous avons pu trouver plusieurs travaux concernant la parallélisation de l'AG classique. Parmis ces travaux nous citons ceux de Muhleinben [MUH 1989], Cantu Pazz [CAN 1995] et [CHI 1996]. Leurs travaux au début et milieu des années 90, ont consisté à proposer des modèles parallèles de l'AG classique et que nous avons décrits dans le chapitre 3.

3.2 Modèle Maitre/Esclave de l'AG classique

Ce modèle est aussi appelé Global. Il consiste à maintenir toute la population dans un processus dit "Maitre". Ce dernier est chargé de l'évolution de la population (selection, croisement et mutation) puis de distribuer tous les individus par paquets sur chaque processeur dit "Esclave" afin de calculer la valeur de la fitness des individus. Les processeurs "Esclaves" renvoient ces valeurs au processus Maitre une fois calculées. Voir le diagramme de la figure 6.4
Dans ce schéma parallèle de l'AG classique Maitre/Esclave, toute la population est répartie sur les processus Esclaves en vue de calculer la valeur de la fonction d'évaluation (ou fitness). Tant que toutes ces valeurs n'ont pas été retournées au processus Maitre, celui ci ne peut commencer ***aucune*** *nouvelle opération* et donc ne peut entamer aucune nouvelle génération, puisqu'il faut que tous les individus de la population soient évalués

pour pouvoir commencer la sélection des individus de la nouvelle génération. Le processus Maitre reste en attente de toutes les valeurs de la fonction d'évaluation. Ce schéma est donc complètement *synchrone*.

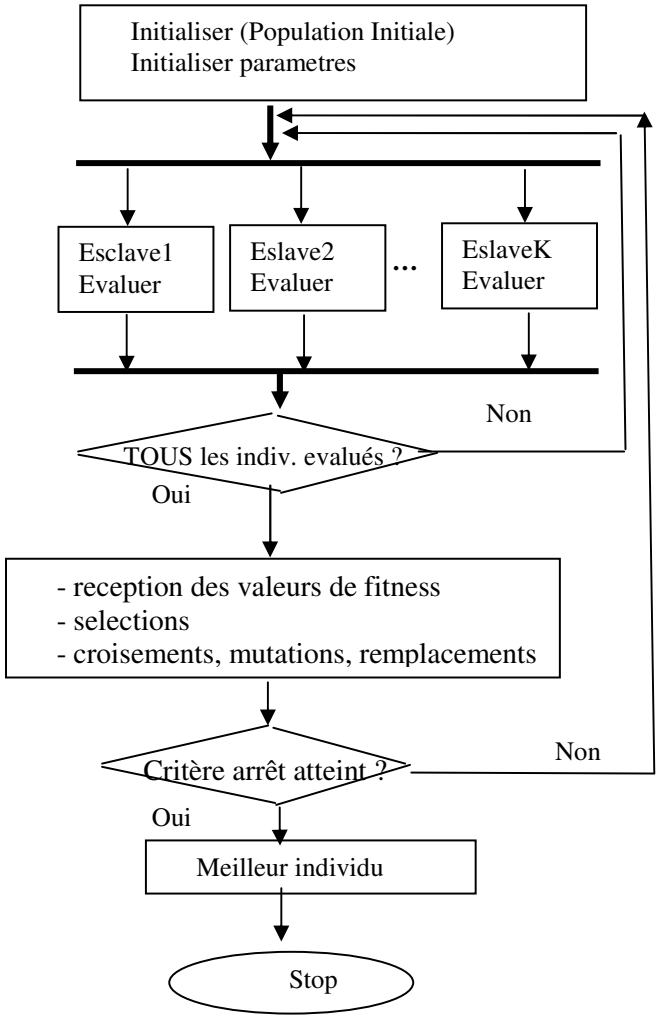


Figure 6.4 : Schéma Parallèle Maitre/Esclave de l'AG classique

Dans notre travail, nous avons apporté des améliorations à l'AG classique de manière à pouvoir lancer la production et l'évaluation des individus en parallèle, ce qui entraine la suppression de la notion de génération et donc de la notion de synchronisme de l'AG parallèle classique.

3.3 Proposition d'un nouveau modèle Maitre/Esclave semi asynchrone

Dans une première étape, nous avons proposé un modèle Maitre/Esclave que nous avons appelé semi_Asynchrone, où la production (sélection, croisement, mutation) peut avoir lieu en parallèle avec les évaluations. Voir la figure 6.5

Dans ce schéma, le processus Maitre procède aux différentes initialisations, à savoir les paramètres de l'AG et la population initiale.

Notre méthode consiste à maintenir la population totale sur une seule machine (Maître). Cette dernière a recours à des machines esclaves pour effectuer les évaluations des différents individus nouveaux nés, en parallèle.

On utilise ici des machines esclaves ou chacune d'entre elles est dotée d'un processus de calcul de fitness dont la seule fonction est de recevoir un individu et de renvoyer la valeur de son adaptation.

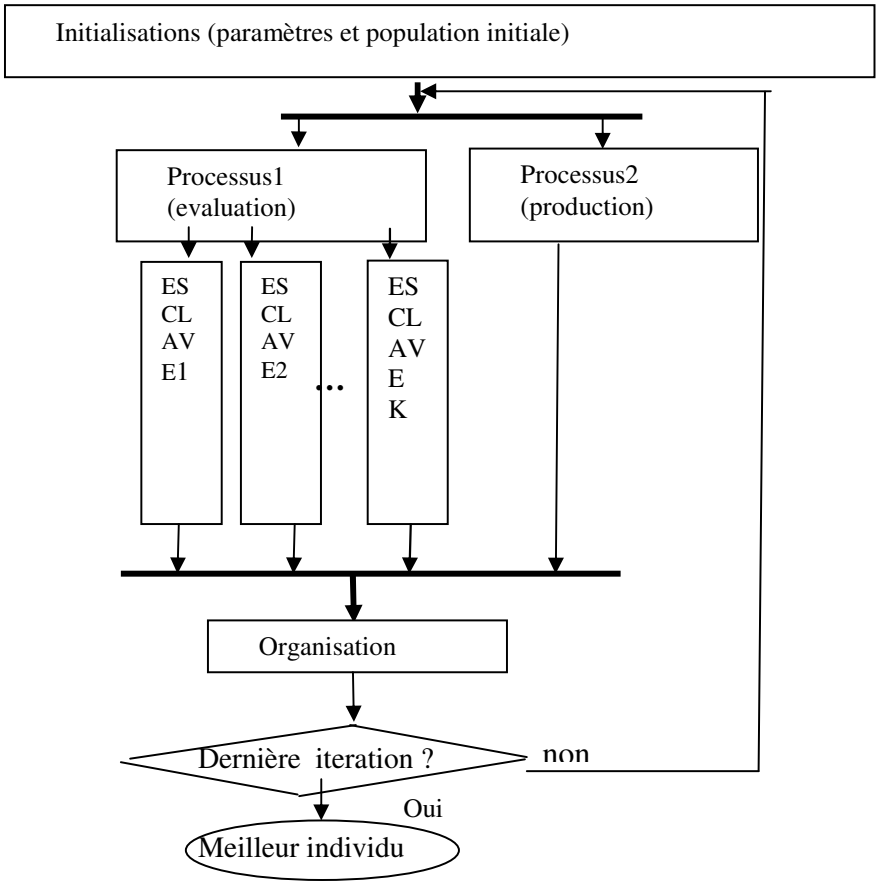


Fig 6.7 : Schéma Parallèle Semi-asynchrone de l'AG amélioré

Le principe se trouve résumé sur la figure Fig. 6.7: le programme Maître se charge d'exécuter les différentes étapes de la production (sélection, croisements,…etc). En d'autres termes, il fait évoluer la population, et en même temps répartit les calculs sur l'ensemble des machines esclaves. Plus précisément, le programme Maître lance deux processus en parallèle :

- ***processus d'évaluation*** : s'occupe de la distribution des individus "Nouveaux nés" sur l'ensemble des machines esclaves d'une façon équitable, et cela en créant un certain nombre de threads (processus légers) qui est égal au nombre d'esclaves connectés au Maître.

Chaque thread communique avec l'esclave qui lui correspond. Dans notre cas la correspondance est définie par l'ordre de connexion, c'est-à-dire que le premier thread créé communique avec le premier esclave connecté au maître, et ainsi de suite.

Donc la distribution des individus est établie localement sur l'ensemble des threads ou chacun d'eux envoie ses propres individus à l'esclave correspondant. Les adaptations reçues sont affectées aux individus correspondants de la population par le biais des threads.

- ***processus de production*** : s'occupe de la production de nouveaux individus à partir de ceux qui ont un statut de "vivants" afin de remplacer les individus "morts" de la population comme déjà spécifié dans la première partie de notre travail.

Au début, les individus de la population initiale sont tous considérés comme "Nouveaux Nés". Lorsqu'un certain nombre d'individus ont été évalués et qu'un autre nombre d'individus ont atteint la date de mort, le processus Maitre lance la "Production" de nouveaux individus en parallèle avec les évaluations. Dés que tous les Esclaves ont renvoyé les valeurs des fonctions fitness au processus Maitre, les esclaves s'arrêtent et la production s'arrête, le temps que le processus Maitre réorganise la population, comme spécifié dans la figure 6.7. A la fin de cette opération, il y a relancement des processus Esclave et Production en parallèle. L'algorithme s'arrête après un certain nombre d'itérations.
Dans ce nouveau schéma Maitre/Esclave parallèle, il y a autonomie de production et d'évaluations mais seulement à un certain degré car il y a synchronisation afin d'effectuer la réorganisation de la population : les "Nouveaux nés" en premier, puis les "Vivants" et enfin tous les individus en fin de vie. Les processus Evaluation et Production s'arrêtent un laps de temps pendant cette réorganisation puis ils reprennenr. Pour cela, ce schéma est dit semi-asychrone.

Le fonctionnement de l'algorithme est le même que celui en séquentiel auquel vient s'ajouter la procédure d'initialisation des contacts, c'est-à-dire la connexion des esclaves au maître (elle est lancée en parallèle avec la procédure d'initialisation de la population), et la procédure de déconnexion des esclaves (appelée à la fin de l'algorithme) ainsi que le changement au niveau de la procédure Evaluation des "Nouveaux nés".

Coté Maître

Procédure initialisation des contacts ( )
Début
- Récupérer n : le nombre d'esclaves. //paramètre défini par l'utilisateur
- créer tableau : infoEsclave[n] = nul //contient les informations de chaque //machine esclave connectée
- **Socket** soc ;
- **ServerSocket** servsoc ;
- i = 0
- servsoc = new ServerSocket (port) //le port est un entier défini par l'utilisateur.

- Tant que (i<n)
  Faire
  -soc = servsoc.accept ( ) //accepter connexion sur port,
  //accept : retourne un objet de type Socket.
  -infoEsclave[i].socket = soc // la sauvegarder dans le tableau
  -infoEsclave[i].DataIn = soc.DataIn //récupérer son flux d'entrée
  // et le sauvegarder
  -infoEsclave[i].DataOut = soc.DataOut //récupérer le flux de sortie
  // et le sauvegarder
  -i = i+1
  Fait

Fin

Procédure init_Evaluation _des_nouveaux_nés (population)

Début
- créer threads[n] ou n est le nombre d'esclaves. //paramètre défini par l'utilisateur
- distribuer les individus nouveaux nés de la population de sorte que :
  - Pour chaque thread i
    Faire
    threads[i].nbreind = nbre // le nombre d'individus que le thread doit envoyer à l'esclave qui lui correspond
    threads[i].début = deb // l'indice de son premier individu dans la population.
    Fait
  - Si tous les esclaves sont connectés
    Alors : envoyer les individus aux esclaves
    Fsi ;
- recevoir les adaptations et les affecter aux individus correspondants.

Fin.

Procédure Déconnexion_des_esclaves ( )

Début
- Pour i = 0; i < infoEsclave.taille; i++
  Faire
  - infoEsclave[i].socket.close // déconnexion de la machine esclave numéro i.

  Fait

Fin

Coté Esclave

Concernant les esclaves, ils sont connectés et déconnectés selon les procédures suivantes :

Procédure Connexion_au_maître ( )
Début
-**Socket** soc ;
-soc = new Socket (adresse IP maître, port maître)//établissement d'une connexion
//l'adresse IP et le port
//sont définis par l'utilisateur.
Fin

Procédure Déconnexion ( )
Début
-soc.close // déconnexion de la machine.
Fin

Protocole de communication entre le maître et un esclave est illustré dans le tableau suivant:

Maître	***Esclave***
-	Attente d'ordre
Envoyer message (1) qui signifie évaluation →	Recevoir message (1) (attente individu)
Envoyer individu →	Recevoir individu
Recevoir adaptation et l'affecter à l'individu correspondant ←	Evaluer et envoyer son adaptation
Envoyer message (-1) qui signifie déconnexion →	Recevoir message (-1) qui signifie déconnexion

3.4 Proposition d'un modèle Maitre/Esclave complètement asynchrone

Pour gagner encore plus en temps d'exécution et pour rendre l'algorithme parallèle plus efficace, nous avons réduit le schéma précédent à un schéma où les deux processus Evaluation et Production sont complètement indépendants.
Pour cela, le schéma précédent a été modifié de telle sorte qu'il ne contienne plus aucun point de synchronisation. Ceci est illustré par le diagramme de la figure 6.8

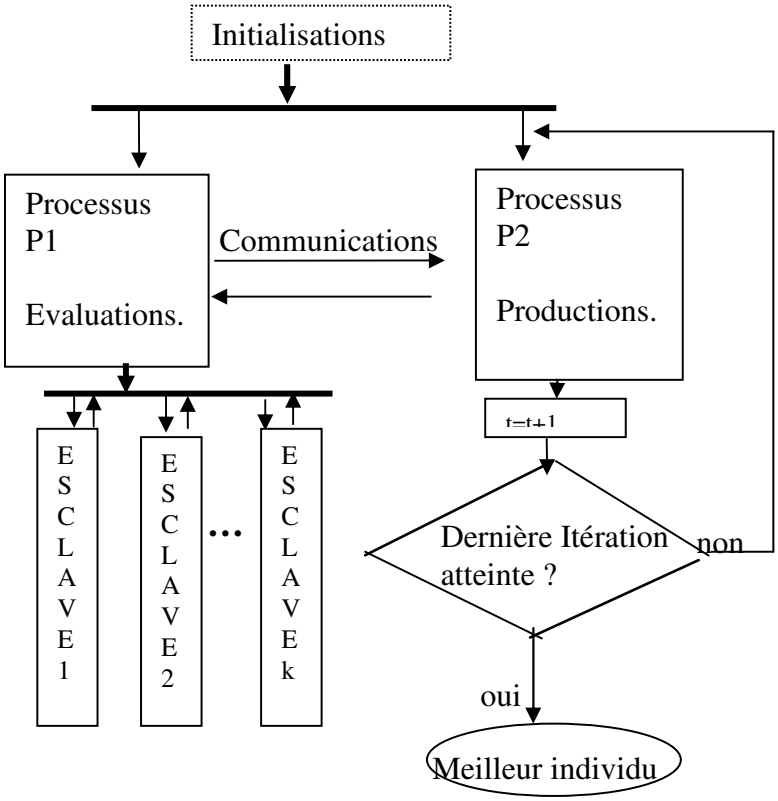


Figure 6.8 : Nouveau Schéma parallèle **asynchrone** Maitre/Esclave

En effet, dans ce schéma deux processus concurrents s'exécutent en parallèle. Ils sont lancés par le processeur Maitre qui contient un processus d'initialisations au départ. Ce dernier procède à l'initialisation de la première population et celles des paramètres de l'AG, avec notamment la durée de vie maximale des individus.
Une fois finie l'étape des initialisations, le Maitre lance les deux processus parallèles : l'un pour effectuer la répartition des individus "Nouveaux Nés" sur les différents Esclaves, l'autre pour effectuer les productions (sélection, mutation, croisement, remplacement). Dés que ces deux processus parallèles sont lancés, ils continuent à travailler indépendamment l'un de l'autre. Ils communiquent via le tableau de la population.

Toute la population est placée dans un seul tableau nommé POP qui regroupe toutes les informations des individus. Les individus "Vivants" renferment le chromosome, la valeur de sa fonction fitness ainsi que sa durée de vie. Les individus "Nouveaux nés" contiennent le chromosome et la durée de vie ainsi que le champ fitness qui ne contient pas encore une valeur alors que pour les individus en fin de vie on aura le chromosome et la fitness calculés mais un flag est positionné indiquant leur statut de fin de vie atteinte.
Cependant, un inconvénient majeur a été recensé en regroupant ainsi toute la population dans un seul tableau : différencier entre les différents types d'individu (vivant, inactif ou nouveau né) est facile mais l'accès à un parmi toute la population devient une tâche lourde.
Pour cela, nous avons introduit des structures supplémentaires qui consistent en des listes chaînées qui contiennent essentiellement les indices des individus dans le tableau POP. Nous avons donc introduit une liste pour les individus "Vivants", une autre pour les individus "Nouveaux nés" et une troisième pour les individus en fin de vie.

Une quatrième liste est ajoutée pour une utilisation ultérieure :c'est une liste additive qui conserve les individus ayant atteint le statut inactif mais qui possèdent la meilleure fitness à chaque itération, nous verrons l'utilité de cette liste ultérieurement. Voir la figure 6.9 qui illustre cette structure de données.
Chaque liste est constituée de nœuds, chaque nœud regroupe les informations représentant un individu avec:

-Liste des "nouveaux nés" : sert à conserver l'indice de l'individu "nouveau né" ;

-Liste des "vivants" : sert à conserver l'indice, la fitness et la durée de vie, l'indice suffit pour récupérer l'individu à partir du tableau POP.

-Liste des individus en fin de vie et donc inactifs : les indices de tous les individus dont la durée de vie est écoulée seront conservés dans cette liste pour être remplacés par les nouveaux nés dans la population.

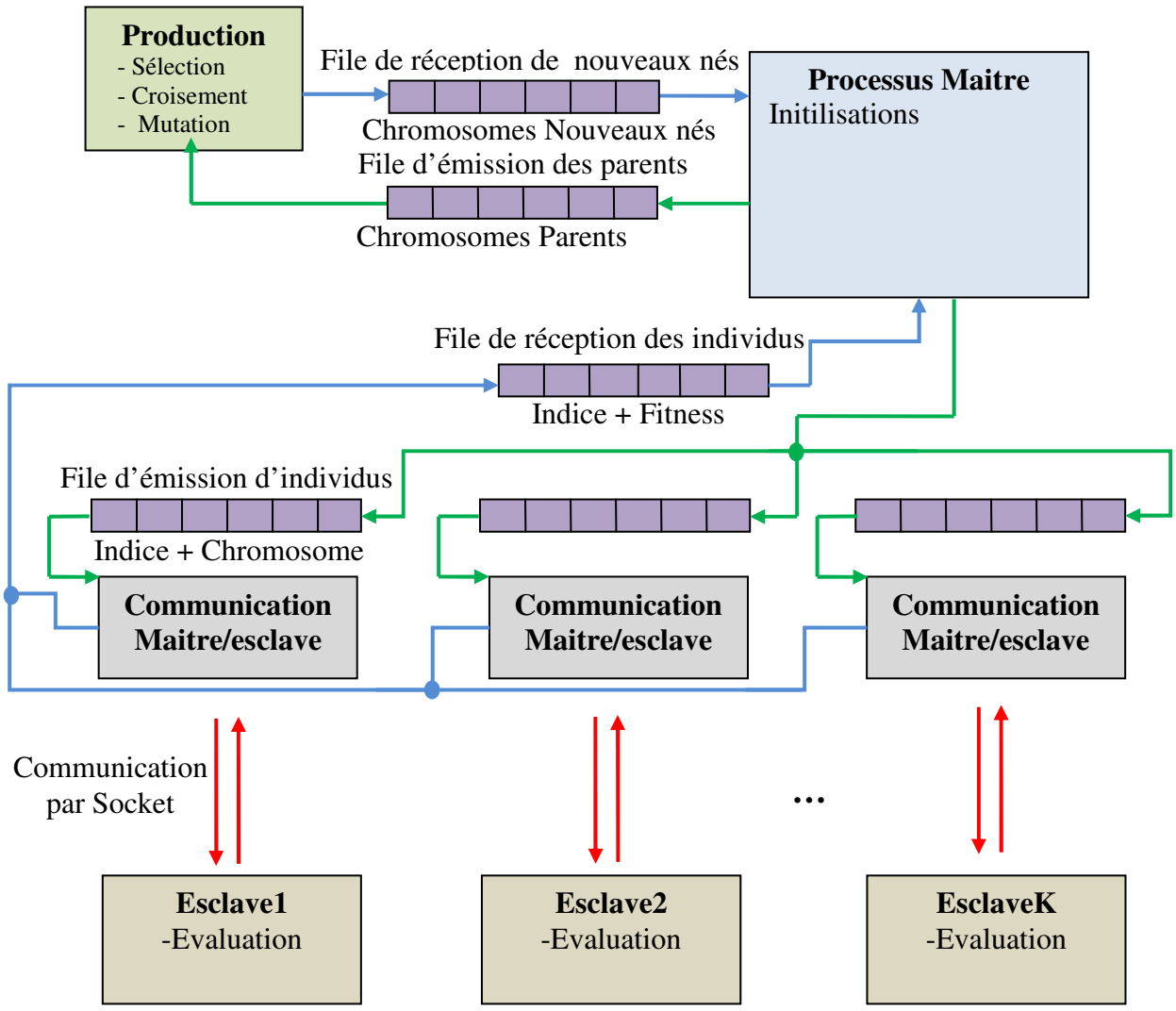


Figure 6.9 : Représentation du mécanisme de parallélisation

Individu 1	Individu 2	Individu 3	Individu 4	Individu 5	Individu 6
Vivant	Inactif	Vivant	Nouveau né	Inactif	Nouveau né

Tableau POP

Liste des nouveaux nés :

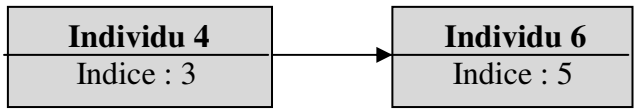


Figure 6.10 : Exemple de conservation d'indice dans les listes

Les avantages des listes sont :
-l'envoi direct de la liste des "Nouveaux nés" pour évaluation au lieu de chercher dans toute la population les individus concernés, ce qui permet un accès direct à ces individus.
-produire de nouveaux individus seulement à partir de la liste des "Vivants", la recherche est évitée encore une fois.
-remplacement direct des individus inactifs par les nouveaux nés dans la population ; c'est la liste des individus inactifs contenant les indices permettant un accès direct.
- la recherche du meilleur individu de la population s'effectue directement au sein de la liste des "Vivants".

3.4.1 Fonctionnement de l'algorithme

Suivant le modèle de parallélisation Maître/Esclave, un nombre limité de tâches sont accomplies par le processus Maître tout en laissant les tâches d'évaluation aux processus Esclaves.
Dans notre schéma, on aura deux grandes tâches qui vont se réaliser en parallèle : l'évaluation et la production.
En fait, les principaux processus recensés, sont :
-l'initialisation.
-l'évaluation.
-la production.

1-L'initialisation : le processus d'initialisation consiste en la génération aléatoire ou par une méthode quelconque de la population initiale, les individus générés sont considérés comme nouveaux nés; ainsi que la génération aléatoire d'une durée de vie pour chacun, appartenant à l'intervalle [1, Durée de vie Max]. Ces individus sont placés dans le tableau POP et sont aussi insérés dans la liste des "Nouveaux nés".
2-La production : le processus de production sert à produire des "Nouveaux nés", à partir de la liste des "Vivants". Il consiste à effectuer le croisement et appliquer l'opérateur de mutation sur l'individu obtenu.
3-L'évaluation : c'est une tâche effectuée par les processeurs esclaves, à chaque fois qu'une machine esclave est libre un individu "nouveau né" lui sera envoyé (chromosome) ; la machine calcule sa fitness et la renvoie au Maître qui lui donnera le statut de "Vivant", il sera

rajouté à la liste des "Vivants" et retiré de celle des "Nouveaux nés" , le tableau POP est mis à jour par le Maitre.

Fonctionnement détaillé de l'algorithme

Coté maitre **:**

Processus de production :

Début
Tant que (critère d'arret non atteint)
Faire
-Réceptionner les 2 individus à croiser de la file Emit-Product.
-Croisement.
-Mutation.
-Déposer l'individu produit dans la file Recept-Product.
Fait
Fin.

Processus Communication :

Début
-Initialisation de server socket
-Tant que (!fin)
Faire
- Se mettre à l'écoute d'une nouvelle machine demandant une communication
Si (demande== vrai)
-Lancer le Thread associé à cette machine esclave.
Fsi
Fait
Fin

Thread associé à un esclave

Début
-Initialisation de la socket
-Tant que (!FIN)
Faire
-Se mettre dans la file d'attente
-Se bloquer
-Réceptionner des individus à évaluer de la file Emit-Eval
-Envoyer sur la socket
-Réceptionner la socket
-Déposer les individus évalués dans la file Recept-Eval
Fait
Fin

Algorithme principal

```
Début
    -initialisation des paramètres le l'AG   // Durée de vie Maximale des individus
                                             //  Taille de la population
                                             //   Taille du chromosome
                                             //Nombre_d'itération_Max_de_l'algorithme
                                            //Probabilités de croisement et de mutation
    -initialisation des files d'attentes      //taille maximale des files.

    -générer la population initiale dans le tableau POP.

    -copier tous les individus dans la liste des "Nouveau nés ".

    -Lancer le processus de production.

    -Lancer le processus d'évaluation.  // Evaluation parallèle  avec la Production
    -Pour  i =0 jusqu'à Nombre_d'Itérations_Maximum
          Faire
          Nb-demande=Taille de la file d'attente ;
           Tant que (Liste NN != Vide())
                Faire
                    Tant que (Nb-demande !=0)
                        Faire
                            -Nb-chrom=taille de la Liste NN /Nb-demande ;
                            -Concerné=tête de la file d'attente
                            Tant que (Nb-chrom!=0)
                               Faire
                                    -Prélever l'indice de  l'individu de la liste.
                                    -récupérer le chromosome du tableau POP.
                                   -déposer le chromosome dans la file
                                        Emit-Eval[Concerné].
                                   - Nb-chrom--;
                               Fait
                            -Nb-demande-- ;
                        Fait
                Fait
          Si  (liste des vivants = = vide ())
               Tant que ((liste additive != vide ( ))
                        Faire
                            -récupérer un individu en fin de vie  de la liste additive.
                            -affectation d'une durée de vie
                            -le rajouter dans la liste des vivants.
                            -le rajouter dans la population.
                        Fait
          FSi.
```

Tant que ((liste des mort != vide ()) ET (Nb vivants >=2))
Faire
-Déposer individus vivants dans la file
// les individus seront
//prélevés par le processus de production.
Fait
Tant que (la file de réception des nouveaux né != vide())
Faire
-Récupérer l'individu nouveau né de la file.
-Remplacer un individu en fin de vie par cet individu dans la population.
-L'insérer dans la liste NN.
Fait.

Tant que (la file de réception des individus évalués ! = vide( ))
Faire
-récupérer l'individu évalué de la file.
-Affecter une durée de vie.
-le rajouter a la liste des individus vivants.
Fait.
Si (Liste des morts est vide) OU (Nb_Vivants<2) alors
-Rechercher l'individu vivant qui possède la plus petite durée de vie.
-Soustraire cette valeur de toutes les durées de vie des individus vivants.
-Mise à jour des listes des vivants et des morts.
-rechercher l'individu mort qui a la meilleure fitness et le rajouter dans la liste additive.
Fsi
Fait
-retourner le meilleur individu de la liste des vivants qui représente la meilleure solution au problème traité dés que critère de fin est atteint.
Fin.

Coté esclave

Début

Initialisation de la socket ;
Demande de connexion ;
Tant que (! FIN)
Faire
-Réception des individus à évaluer
-Evaluer les individus.
-Emettre les valeurs de la fitness des individus évalués.
Fait.
Fermeture de la socket ;
Fin.

Remarque :
D'après la description des différents processus, il n'existe pas de point de synchronisation entre les processus Evaluation et Production. Cependant, il est possible que les processus production et évaluation soient inter-bloqués.

3.4.2 Les cas d'inter-blocage entre les deux processus

On dit que deux processus sont inter-bloqués lorsqu'arrivés à une étape la suite de l'exécution de chacun d'eux dépend d'un évènement déclenché par l'autre et vice versa.
Dans notre algorithme nous avons mis au point un schéma Producteur/Consommateur spécial : nos 2 processus sont à la fois tantôt consommateur et tantôt producteur. En effet, pour que le processus "Productions" puisse produire de "Nouveaux nés" il faudra qu'il soit en mesure de consommer les individus "Vivants" et pour que le processus "Evaluations" puisse produire des "Vivants" il faudra qu'il soit en mesure de consommer des individus "NouveauxNés". En fait, ces deux processus travaillent indépendamment l'un de l'autre mais chacun d'eux consomme ce que produit l'autre.

Pour cela, deux cas d'interblocage risquent de se poser :

1- Supposons qu'à une certaine itération de l'algorithme la durée de vie de tous les individus de la population a atteint la valeur zéro. A ce moment là, tous les individus sont dans la liste des individus inactifs et il n'en reste aucun dans la liste des "Vivants". Par conséquent, le processus Production ne trouvera plus d'individus "Vivants" à sélectionner dans le but d'en créer de nouveaux : il sera donc mis en attente jusqu'à ce que la liste des "Vivants" contienne de nouveau des éléments. Sachant que cette liste est remplie par le processus d'évaluation à partir des "Nouveaux nés", le processus "évaluation" sera lui aussi bloqué en attente d'individus "Nouveaux nés". Arrivé à cette situation, l'algorithme se bloque.
Pour pallier cette situation, nous avons pris des mesures consistant à introduire des structures supplémentaires avec des modifications de l'algorithme :

Une liste chaînée a été ajoutée dans le but de conserver à chaque itération, l'individu inactif ayant la meilleure fitness qui a été remplacé dans le tableau POP. Rappelons que ceci est fait avec le souci de garder tout le temps une taille de population fixe. Ainsi, quand la liste des vivants ne contient aucun élément, le processus Production pourra prélever des individus inactifs de cette liste additive et les remettre dans la liste des "Vivants", avec une nouvelle durée de vie. Le cycle des individus devient schématisé comme illustré par la figure 6.11

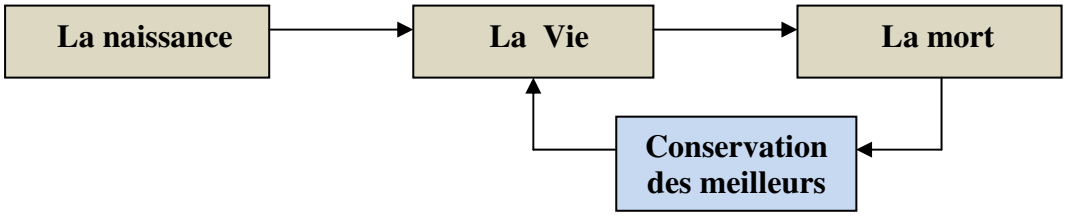
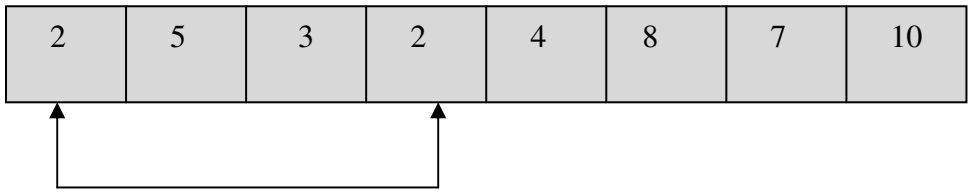


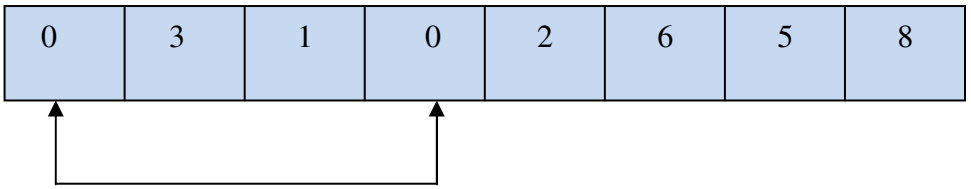
Figure 6.11 : Cycle de vie d'un individu

2- Un autre cas possible d'inter-blocage peut survenir quand la liste des individus inactifs est vide, c'est-à-dire : il se peut qu'à une itération donnée de l'algorithme aucun individu n'a atteint le cycle de disparition de la population. Etant donné que le processus "Production" recherche d'abord s'il existe des éléments arrivés en fin de vie pour produire un "Nouveau né", ce processus sera bloqué par conséquent pas de "Nouveau nés" par conséquent pas d'évaluations.
Pour éviter ce blocage, au lieu de décrémenter les durées de vie de chaque individu de 1, nous recherchons dans la liste des "Vivants" l'individu ayant la plus petite durée de vie. Cette dernière sera soustraite de la durée de vie de chacun des individus de la population et le nombre d'itérations total de l'algorithme sera avancé de cette durée de vie minimale.
Exemple : supposant qu'arrivés à une étape de l'algorithme on a des durées de vie comme suit :



La durée de vie minimale

La durée de vie de chaque individu devient alors comme suit :



Individus devenus Inactifs

L'individu dont la durée de vie a atteint la valeur zéro est maintenant un individu inactif donc on a assuré au moins un individu inactif à chaque itération dans le but de pouvoir le remplacer par un nouveau né.

3.4.3 Application des opérateurs génétiques dans l'algorithme

-La sélection :

Etant donné que c'est l'opérateur clé de l'évolution de l'algorithme en terme de qualité de solutions, la sélection est l'étape qui permet de déterminer quels individus nous allons croiser. La méthode pour laquelle nous avons optée est le TOURNOI BINAIRE :
La sélection se fera à partir de la liste des vivants.
On tire 2 individus aléatoirement de la liste, le tournoi leur sera ensuite appliqué c'est à dire que l'individu qui possède la meilleur fitness gagne, on aura ainsi tiré le premier individu à croiser, le deuxième sera tiré de la même manière. Voir figure 6.12

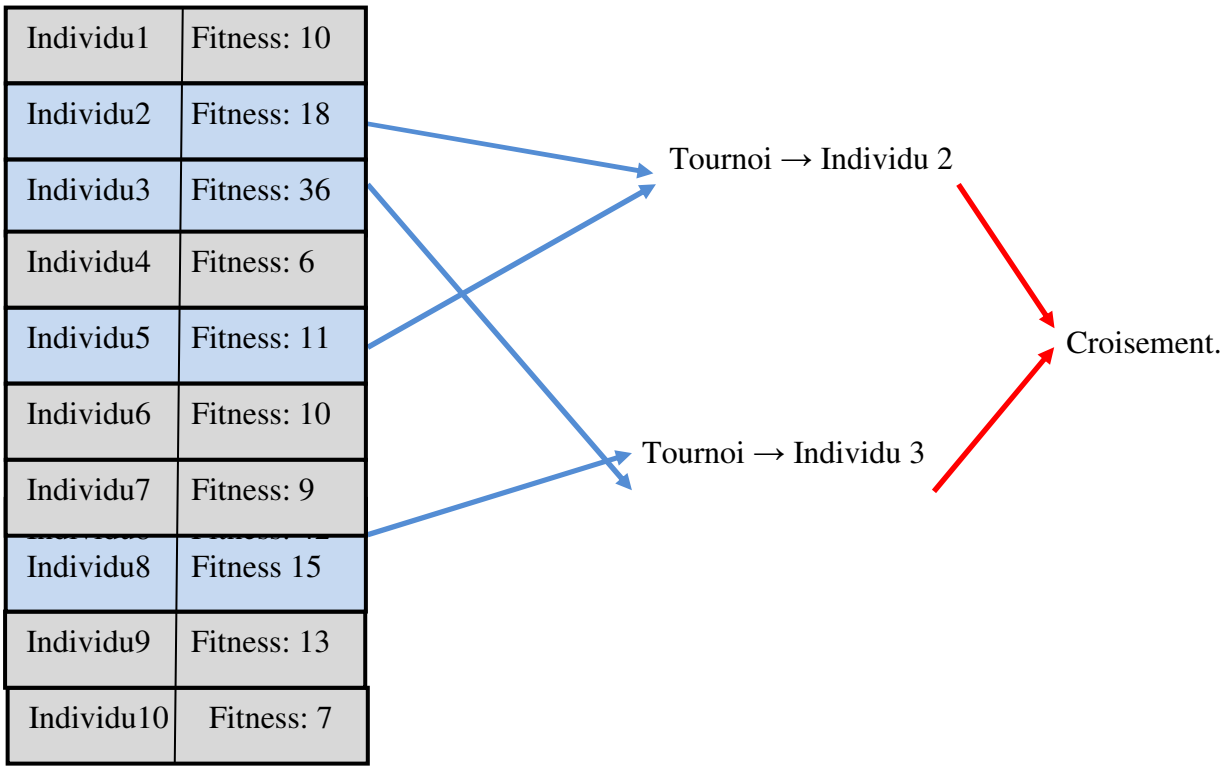


Figure 6.12 Exemple de sélection par tournoi de l'algorithme

-Le croisement :

Une fois que les 2 individus sont sélectionnés, il va falloir les croiser pour produire un "Nouveau né ", la probabilité de croisement est d'abord appliquée pour déterminer s'il y aura croisement ou pas ; le croisement se fera en un seul point qui sera déterminé aléatoirement.

-La mutation :

L'opérateur de mutation est appliqué sur l'individu obtenu après croisement ; si la probabilité de mutation est satisfaite, on mute un gêne du chromosome dont la position est tirée aléatoirement.

-Le remplacement :

Chaque nouvel individu va remplacer un individu Inactif dans la population en sachant qu'à chaque itération le nombre de productions est égal au nombre d'individus Inactifs.

3.5 Application du nouvel algorithme au problème de la classification

Comme nous le constatons, la conception de l'algorithme présentée ne fait pas de classification, nous allons maintenant aborder la conception du système d'extraction de règles à l'aide du nouvel algorithme. Tout le formalisme présenté dans le chapitre précédent pour extraire des règles de classification sera gardé, sauf que l'algorithme génétique parallèle aura à gérer la base de données apprentissage.

La base d'apprentissage est la seule information commune entre le processeur Maitre et les processeurs esclaves. De ce fait on trouve la même base d'apprentissage dans tous les esclaves. Pour avoir toujours la même image de la base d'apprentissage pour tous, le

processeur Maitre envoie le meilleur individu sélectionné à tous les processeurs esclaves pour qu'ils mettent à jour leurs bases d'apprentissage.

Le principe de fonctionnement de l'algorithme reste le même: une population initiale est générée, elle évolue durant un certain nombre d'itérations, prédéfini par l'utilisateur puis la règle correspondant au meilleur chromosome est extraite. Les instances de la base de données couvertes par cette règle sont ignorées pour le reste de l'exécution.

Comme l'opération d'évaluation des individus (règles) entraîne le calcul de leurs couvertures, alors les machines esclaves doivent être informées de la mise à jour de la base de données (suppression des instances couvertes par la meilleure règle d'une évolution).

Pour cela, il suffit de définir un nouveau message qui indique la mise à jour de la base de données. Ce dernier doit être diffusé vers tous les processeurs esclaves, suivi par l'individu (meilleure règle), afin que chaque machine esclave puisse ignorer les instances de la base de données couvertes par cette règle pour le reste de l'exécution (voir tableau ci-après).

Message indiquant la mise à jour de la BD.

Maître	***Esclave***
Envoyer message 0 (maj. BD) →	Recevoir message 0 (attente individu (règle))
Envoyer individu →	Recevoir individu (règle) et suppression des instances couvertes par cette règle

4. Conclusion

Dans ce chapitre, nous avons défini tout le formalisme conçu pour mettre en œuvre un nouvel AG, de manière à pouvoir proposer un nouveau schéma Maitre/Esclave complètement asynchrone. Ce schéma sert à distribuer la population de l'AG sur plusieurs processeurs Esclaves pour calculer les fonctions d'évaluation ***sans stopper*** la production. Dans ce schéma, la notion de « génération » a donc été supprimée. Il est à noter que notre nouveau schéma parallèle assure une distribution équilibrée des tâches sur les différents processeurs Esclaves, sachant qu'une variable X qui représente le nombre d'Esclaves libres était mise à jour régulièrement par le processeur Maitre : à chaque fois qu'un processeur Esclave reçoit des individus à évaluer, cette variable est décrémentée d'une unité et elle est incrémentée d'une unité dés qu'un processeur Esclave renvoie des résultats au Maitre. Cette variable X (nombre d'Esclaves libres) doit le plus souvent possible rester nulle pour qu'un équilibre des tâches soit assuré entre les processeurs Esclaves. Dans notre cas, la conception de l'algorithme parallèle a été assurée dans ce souci. Il est à noter que ces travaux sont décrits dans [BEN2007] et [BEN2013] .

1. Introduction

L'algorithme génétique (AG) [HOL1962][GOL1989] est une meta heuristique robuste qui a montré ses preuves dans la résolution de problèmes d'optimisation complexes et à grande taille. En effet, c'est une méthode efficace d'exploration de grands espaces de recherche, convergeant toujours et quelque soit la complexité du problème traité, vers une solution acceptable et en un temps raisonnable. Cependant, un inconvénient majeur de cet algorithme est parfois rencontré : il s'agit de sa convergence prématurée vers un optimum local. Pour éviter cette convergence qui risque d'être prématurée dans certains cas, Moscato [MOS1989] s'est inspiré de la notion de "mème" proposé dans les travaux de R. Dawkins [DAW1982] et a introduit la recherche locale dans un AG pour mettre au point un nouvel algorithme évolutionnaire, appelé Algorithme Mémétique (AM). Cet algorithme représente actuellement une classe d'algorithmes stochastiques de recherche dans lequel des approches évolutionnaires sont combinées avec des approches locales qui dépendent fortement du problème traité. L'aspect évolutionnaire apporte une grande exploration de l'espace de recherche alors que la recherche locale incorporée dans l'exploration apporte quant à elle une sorte de "zoom" sur les régions prometteuses de bonnes solutions [MOS 2000]. Les AMs combinent donc l'approche évolutionnaire avec une intensification de recherche puissante de la recherche locale. Pour cela, ils représentent une perspective pragmatique pour trouver une (ou des) solution(s) plus intéressante(s) que celle apportée par un AG pur.

Dans le contexte de la classification par des approches mémétiques, nous avons recensé dans la littérature les travaux de Tan [TAN 2008] [TAN 2010] qui a mis au point une approche mémétique pour extraire un classificateur basé toujours sur l'approche de Pittsburgh. Dans ces travaux, les auteurs se sont basés sur un chromosome de longeur variable et ils ont proposé une structure de données assez complexe ainsi que des opérateurs specifiques pour la manipulation génétique des chromosomes de différentes tailles. Comme recherche locale, ils ont mis au point une approche basée sur les systèmes immunitaires AIS [HOL 1992] et une autre basée sur une approche évolutionnaire trés réduite.

Dans cette partie de nos travaux, nous allons décrire un AM pour résoudre le problème de l'extraction de règles de classification. Notre souci majeur à présent, est d'augmenter la précision et la compréhensibilité de chaque règle extraite. Evidemment, tout le formalisme décrit pour extraire un classificateur par l'approche génétique pure, décrit dans le chapitre précédent, est gardé. Nous allons décrire à présent deux approches locales ainsi qu'une approche basée sur la recherche tabou que nous avons mise au point. Par la suite, nous décrivons toutes les stratégies d'application de ces approches locales dans le cycle génétique, afin de pouvoir étudier la cynergie des opérateurs génétiques avec la recherche locale appliquée [KRA2002].

2. Description des Approches locales mises au point

Tout le formalisme développé précédemment pour l'approche génétique pure pour l'extraction de règles de classification est gardé. A ce niveau, nous allons décrire les méthodes de recherche locale que nous avons mises au point afin de les intégrer dans le cycle génétique.

Ceci est fait dans le but d'améliorer la recherche de l'AG et donc d'améliorer les classificateurs obtenus en terme de précision.

Dans la littérature, nous avons trouvé plusieurs formes d'hybridation : meta_exact qui consiste à introduire une méthode exacte de recherche locale et meta_meta hybridation qui consiste à hybrider une metaheuristique comme recherche locale. Nous avons mis au point ces deux formes d'hybridation. La première approche de recherche locale consiste en une méthode simple dite "one-move" que nous avons appelée S_LS alors que la deuxième est elle même basée sur un mini algorithme génétique, appelée LGA (Local GA). Nous avons choisi de mettre au point LGA en nous inspirant un peu des travaux de J. Smith [SMI 03], qui a combiné un algorithme de programmation génétique avec un AG. Ainsi, notre approche mémétique renfermant un mini AG comme recherche locale reste purement évolutionnaire. Nous avons fait ce choix dans le but de pouvoir paralléliser cette approche mémétique dans des travaux futurs, sachant que le parallélisme intrinsèque des approches évolutionnaires nous apportera un gain considérable en temps d'exécution de la méthode. Par ailleurs, nous avons également mis au point une recherche locale basée sur la recherche tabou. Toutes ces approches sont décrites dans ce qui suit.

2.1 Description d'une méthode de recherche locale simple S_LS

Cette méthode, que nous avons mise au point, consiste à créer tous les voisins d'un individu (***I)*** qui diffèrent par un seul attribut et choisir le meilleur individu entre I et ses voisins .

Dans cette approche, il s'agit de récupérer à une itération t de l'AG, un individu I de la population Pop(t). Nous créeons son voisinage constitué par autant d'individus que de nombre d'attributs de la base d'apprentissage considérée: Si cette base contient K attributs décrivant une instance, nous formons un voisinage de l'individu I formé de K nouveaux individus. Ces K nouveaux individus sont créés chacun par mutation d'un seul gène de l'individu I. Nous obtenons alors K individus autour de I, tous différents entre eux par un seul gène. L'opérateur de mutation utilisé reste le même que celui défini par l'approche génétique pure.

Cette nouvelle population S_LSPop(I) est alors évaluée en utilisant la même fonction fitness précédemment définie. Le meilleur individu I' de S_LSPop(I) est récupéré.

En fait, la méthode locale opère ***un seul mouvement*** sur un individu d'une itéation t et se termine par retourner à l'AG, le meilleur individu entre I de départ et I' trouvé par S_LS.

S_LS(I,I') // Simple Local Searcher
Début

- Récupérer un individu I dans la population Pop(t);
- Opérer un seul mouvement sur chaque attribut de I;
- Initialiser le voisinage de I formant une petite population S_LSpop(I) ;
- Evaluer la population S_LSpop(I);
- Prendre le meilleur individu I' de S_LSPop(I);
- retourner le meilleur individu entre *I et I'* trouvé par S_LS;

Fin.

2.2 Description d'une méthode de recherche locale basée sur un AG "LGA"

Dans cette approche, il s'agit de récupérer un individu I de la population Pop(t) de l'AG. Nous créeons son voisinage constitué par autant d'individus que de nombre d'attributs de la base d'apprentissage considérée. Si cette base contient K attributs décrivant une instance, nous formons un voisinage de l'individu I formé de K nouveaux individus, de la même manière que dans S_LS.

Cette mini population ainsi obtenue et appelée LGAPop(I) est alors évaluée en utilisant la même fonction fitness précédemment définie.

La méthode locale, à présent, consiste à déclencher un nouvel AG faisant évoluer LGAPop(I) à travers quelques itérations. Il a fallu adapter ses propres paramètres: la taille de sa population est la taille de LGAPop(I) qui vaut le nombre d'attributs de la base d'apprentissage. Pour garder l'aspect local de la méthode, nous avons choisi une probabilité de croisement assez faible et un taux de mutation assez fort par rapport à l'habituel. Nous utilisons également un nombre d'itérations faible, de l'ordre de la vingtaine.

L'AG local se termine par retourner à l'AG principal le meilleur individu entre I de départ et I' trouvé par LGA.

Le pseudo_code de LGA est donné par:

LGA // Local Genetic Algorithm

Début

- k=0;
- Initialiser les parametres de LGA // Pc, Pm, NbIterations;
- Récupérer un individu I de la population Pop(t);
- Initialiser la population LGAPop(t);
- Evaluer la population LGAPop(t);

Répéter

-Selection;
-Croisement; //avec une faible probabilité
-Mutation; //avec une relativement forte probabilité
-k=k+1;
-Garder le meilleur individu localement

Jusqu'à épuisement de NbItérations choisi faible;

retourner le meilleur individu entre *I et le meilleur* individu *I'* trouvé par LGA .

Fin.

2.3 Description de la recherche Tabou comme recherche locale pour l'extraction de connaissances

Dans cette partie nous allons montrer comment adapter la recherche Tabou au problème de la classification en détaillant les points suivants :

- Génération de la solution initiale
- Génération de voisinage
- Implémentation de la liste des tabous
- Application de la recherche Tabou

- Construction du classificateur

2.3.1 Génération de la solution initiale

Il a été démontré que l'efficacité des méthodes basées sur le principe de la recherche locale dépend étroitement de la qualité de la solution initiale choisie. Dans notre application, nous avons initialisé la solution initiale à l'aide d'un Algorithme Génétique.

Le recours à l'Algorithme Génétique pour initialiser la Recherche Taboue est considéré comme une phase d'initialisation de la méthode de recherche locale et non plus une hybridation proprement dite de Métaheuristiques. Effectivement, cette initialisation ne fait pas partie du processus de recherche et peut se faire éventuellement par d'autres méthodes.
La fonction suivante montre comment générer la solution initiale

```
Individu Générer_Solution_Initiale(Données_Apprentissage, indiv)
Début
    | Appliquer l'AG(Données_Apprentissage) ;
    | Individu indiv = le meilleur individu obtenu par l'AG ;
    | Retourner (indiv) ;
Fin
```

2.3.2 Génération des voisinages : Le voisinage d'un individu courant est constitué de toutes les solutions obtenues en effectuant une seule mutation sur l'individu. La fonction suivante montre comment générer le voisinage d'un individu.

```
Ensemble de voisins Créer_Voisinage (Indiv)
Début
    Ensemble_de_voisins = {vide} ;
    Entier i, L = taille de l'individu;
    Individu Voisin ;
    Pour i = 0 à (L-1)
    Faire
        Voisin = une copie de Indiv ;
        Muter la case opérateur de l'attribut Voisin[i] dans le cas où l'attribut
        est Numérique mais avec une autre valeur différente de sa valeur actuelle ;
        Ensemble_de_voisins = Ensemble_de_voisins + Voisin ;
        Voisin = une copie de Indiv ;
        Muter la case valeur de l'attribut Voisin[i] par une autre valeur
        parmi les valeurs possibles de l'attribut i mais elle doit être différente
        de la valeur actuelle ;
        Ensemble_de_voisins = Ensemble_de_voisins + Voisin ;
        Voisin = une copie de Indiv ;
        Muter la case Active/Désactive de l'attribut Voisin[i];
        Ensemble_de_voisins = Ensemble_de_voisins + Voisin ;
    Fait
    Retourner(Ensemble_de_voisins) ; Fin
```

2.3.3 Implémentation de la liste des tabous

Dans la Recherche Tabou un ensemble de voisinage N(s) est examiné et on retient la meilleure s' même si f(s')<f(s). La recherche Tabou ne s'arrête donc pas à la première solution trouvée. Le danger serait alors de revenir à s immédiatement, puisque s est meilleure que s'. Pour éviter de tourner ainsi en rond, on crée une liste T qui mémorise les dernières solutions visitées et qui interdit tout déplacement vers une solution de cette liste. Cette liste T est appelée liste Tabou.

Dans notre application nous avons implémenté la liste Tabou par une liste de règles et la mise à jour de la liste Tabou se fait à chaque itération de la recherche. A chaque itération un nouvel individu est introduit et le plus mauvais de la liste est retiré dans le cas où il n'y plus de places vides.
La fonction suivante montre comment gérer la liste Tabou :

```
Individu Ajouter_Tabou (Liste_Tabou, Indiv)
 Début
      Si (La liste Tabou n'est pas pleine)
        Alors
   Liste_Tabou = Liste_Tabou + Indiv ;
     Sinon
   Liste_Tabou = Liste_Tabou – mauvais_indiv(Liste_Tabou) ;
   Liste_Tabou = Liste_Tabou + Indiv ;
   Retourner   (Liste_Tabou);
Fin
```

2.3.4 Application de la recherche Tabou TS

Le processus de la Recherche Tabou TS consiste, pour chaque itération à choisir la meilleure solution qui n'est pas tabou dans le voisinage de la solution courante, même si cette solution n'entraîne pas une amélioration. Les individus du voisinage sont évalués en ayant recours à la fonction d'évluation définie dans la chapitre 5. La fonction suivante illustre le fonctionnement de la Recherche Tabou.

```
Individu Recherche_Tabou(individu)
 Début
 Individu meilleur_ voisin;
 Individu indiv = Générer_Solution_Initiale(Données_Apprentissage, indiv) ;
          Liste_Tabou = vide ;

        Tant que(Le critère d'arrêt n'est pas atteint)
        Faire
            Ensemble_de_voisins = Créer_Voisinage(indiv) ;
            Evaluer les individus du voisinage de I ;
            List_Tabou = Ajouter_Tabou(Liste_Tabou, Indiv);
            indiv = meilleur_voisin selon la fonction d'évaluation ;
       Fait ;
      Retourner   Meilleur_Individu de la Liste_Tabou;
Fin
```

3. Mise en oeuvre des Algorithmes Mémétiques (AMs)

Une fois que les méthodes de recherche locale ont été définies, l'algorithme mémétique est conçu. Le pseudo_code suivant donne son illustration.

```
Début
   -t=0;
   -Initialiser les parametres de l'AG global : TaillePop,NbreItérationsMax, Pc, Pm;
   -Initialiser la 1ere Population Pop(t);
   -Evaluer la 1ere Population Pop(t);
Répéter
   -Sélection
   -Coisement ;
   -Mutation;
   -Appliquer LGA ou S_LS ou TS à chaque individu de Pop(t+1);
   -Evaluer la nouvelle population obtenue;
   -Remplacer les anciens par les nouveaux individus en appliquant une stratégie de
    remplacement préalablement définie.
   -t=t+1;
Jusqu'à NbreIterationsMax ;
Afficher le meilleur individu;
Fin.
```

En fait l'application de la recherche locale peut avoir lieu à n'importe quel niveau de l'AG. Elle peut être appliquée à la population initiale ou bien après croisement ou bien après la mutation ou alors après chaque opérateur génétique, aux individus nouvellement crées et avant leur remplacement dans la population.
Le fait de changer de niveau d'application de la méthode de recherche locale nous permet d'étudier la synergie des opérateurs génétiques avec la recherche locale ensemble. Cela permet en fait de montrer à quel niveau de l'AG l'application de la recherche locale peut être la plus efficace et travaille le plus correctement possible avec le croisement et la mutation. Nous verrons tout cela dans la partie expérimentale en comparant les résultats obtenus par les différentes stratégies d'application de la recherche locale.

Par ailleurs, nous avons utilisé le remplacement après recherche locale selon la stratégie de Baldwin et "steady state" dans l'AG. La diversité de la population quant à elle, est protégée par le croisement et la mutation spécifiques mis au point pour le problème de l'extraction de règles de classification, sans compter la nature même des différentes recherches locales mises au point.

Pour ce qui est d'appliquer cet algorithme à l'extraction de règles de classification, le pseudo-code suivant résume l'application de l'AM à l'extraction des règles de classification. Rappelons que tout le formalisme concernant l'extraction de règles de classification définie dans l'approche purement génétique, à savoir les opérateurs génétiques, la sélection et le remplacement et même le calcul de la fonction de fitness, reste valable ici.

```
Début
        Classificateur={vide};
        Nb_Inst=le nombre d'instances de la base de données apprentissage ;
        Booléen : couvre;
        Tant que(le critère d'arrêt n'est pas atteint)
        Faire
              Appliquer le cycle de l'AM ;
              Individu = le meilleur individu obtenu par l'AM ; //la meilleure solution
              Règle= Décoder(Individu) ;
              couvre=faux ;
              Count=0 ;
              Pour i=1 à Nb_Inst  //Chercher toutes les instances couvertes par cette règle
              Faire
                    Si (Est_Couverte(règle, instance))
                       Alors
                             Supprimer cette instance de la base de données ;
                             couvre=vrai ;
                             Count - = count ;
              Fait ;
              Si (couvre=vrai)
                 Alors
                       Ajouter règle au classificateur ;
                       Nb_inst = count ;
        Fait ;
Fin.
```

L'AM est donc appliqué pour extraire chaque règle du classificateur, une à une. A chaque fois qu'une règle est extraite, nous recherchons toutes les instances de la base d'apprentissage couvertes par cette règle et nous les supprimons. L'algorithme s'arrête lorsqu'il n'existe plus d'instances dans la base d'apprentissage ou bien que le nombre d'itérations maximum défini à l'initialisation de l'algorithme est atteint.

Pour évaluer les performances du classificateur obtenu, la base de test est utilisée. Dans la base de tests, les individus possèdent des attributs qui les caractérisent mais pas la classe à laquelle ils appartiennent. Nous appliquons les règles de classification extraites à ces instances dont le champ classe n'est pas défini, puis on calcule le taux d'instances bien classées par le classificateur. Plus ce taux se rapproche de 100% et plus le classificateur obtenu est valable et précis et donc l'algorithme qui l'a extrait est performant.

4. Conclusion

Les approches mémétiques décrites dans ce chapitre avaient pour but d'augmenter la précision de chaque règle extraite et donc d'augmenter les performances du classificateur obtenu en terme de précision. Nous avons opté pour une recherche locale basée sur un AG

dans le but de garder l'approche complètement évolutionnaire, sachant que les méthodes évolutionnaires présentent un parallélisme intrinsèque et que la fouille de données manipule une masse très volumineuse de données, ce qui induit des temps de calculs élevés. Le parallélisme de données donne d'excellentes performances dans leurs traitements tout en maintenant des temps de calculs relativement bas.

1. Introduction

Dans ce chapitre, il est décrit les plateformes de base qui ont servi au développement de nos méthodes ainsi que les bibliothèques externes utilisées et les bases de données de tests utilisées. Afin de montrer l'efficacité de nos algorithmes et stratégies développées, une étude comparative sera effectuée avec d'autres algorithmes de classification bien connus dans la littérature.

2. Environnements d'implémentation

2.1 Le langage

Nous avons utilisé le langage de programmation Java qui présente la particularité principale : les logiciels développés dans ce langage sont modulables d'une part et très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Microsoft Windows, Mac OS ou Linux avec très peu ou pas de modifications d'autre part. Le langage reprend en grande partie la syntaxe du langage C++.

2.2 La bibliothéque externe WEKA

Le logiciel Weka (Waikato Environment for Knowledge Analysis) [WIT 2005] est une suite de logiciels d'Apprentissage automatique populaire.

Weka a été entièrement écrit en Java. Il a été développé à l'université de Waikato, en Nouvelle Zelande.

Il se compose principalement :

- De classes Java permettant de charger et de manipuler les données, telles que les bases de données benchmark de data mining.
- De classes pour les principaux algorithmes de classification supervisée ou non supervisée, tels que ID3, OneR, Part, … que nous avons utilisés pour comparer leurs résultats avec ceux obtenus par nos approches sur les mêmes bases de données.
- D'outils de sélection de données, de statistiques sur ces données.
- De classes permettant de visualiser les résultats.

On peut l'utiliser à trois niveaux :

- Via l'interface graphique, pour charger un fichier de données, lui appliquer un algorithme et vérifier son efficacité.
- Exécuter un algorithme sur la ligne de commande de manière intéractive.
- implémenter d'autres algorithmes, comparer ou combiner plusieurs méthodes.

2.3 Les Benchmarks de l'UCI

L'UCI (Université de Californie Irvine) [UCI 1998] a mis à la disposition des chercheurs en data mining des bibliothèques de données benchmarks. En fait, c'est une très grande bibliothèque de base de données (Benchmarks) sélectionnées par cette université. Les Benchmarks de l'UCI sont largement utilisés et considérés comme une référence, d'où

l'importance de les utiliser pour évaluer des algorithmes et comparer leurs performances par rapport à d'autres algorithmes.

Nous avons utilisé les bases de données suivantes :

Base de données	Nombre d'instances	Nombres d'attributs	Nombre de classes
Weather	14	5	2
Iris	150	5	3
Zoo	101	18	7
Mushroom	8124	23	2
Hepatitis	155	20	2
Heart-statlog	270	14	2
Segment-challenge	1500	20	7
Ionosphere	351	35	2
Kdd-train	11419	42	2
Diabetes	768	9	2
Chess End-Game	8124	22	2

Tableau 1 : Les bases de données BENCHMARKS utilisées.

Remarque : Kdd_train est la seule base de données qui ne fait pas partie du répertoire de l'UCI. C'est une base extraite d'une base de données appelée **Darpa** et qui décrit des événements concernant la sécurité dans les réseaux où il s'agit de détecter des attaques sur les réseaux, selon différents facteurs : le protocole utilisé, le serveur (http, nntp…etc.), les adresses IP de destination…etc.

2.4 Performance des méthodes de classification

Pour évaluer les performances d'une méthode de classification, généralement l'ensemble de données disponible est divisé en deux sous-ensembles : l'un servira pour l'apprentissage et l'autre pour le test. L'ensemble d'apprentissage est utilisé pour déterminer le modèle de classification alors que l'ensemble de tests contient des individus dont la classe a été supprimée avant d'être présenté au classificateur extrait par l'algorithme. Ce classificateur doit être en mesure de retrouver la classe de chaque individu. Cela sert à tester les performances de la méthode en calculant le taux (la précision) de classification correct de l'ensemble des cas. Ce taux est déterminé par le rapport du nombre de cas bien classés sur le nombre total des cas testés.

Parfois il arrive que l'on soit confronté à des problèmes où l'ensemble de données est restreint et on veut exploiter ces données disponibles pour construire le classificateur d'une part et tester les performances de la méthode d'autre part. Pour cela on fait appel aux techniques de rééchantillonage (resampling techniques) [Koh 95] [Wei 91] ; parmi lesquelles la technique de validation croisée (cross-validation) est la plus utilisée. Le principe de cette technique consiste à diviser aléatoirement l'ensemble des données en *m* partitions mutuellement exclusives (*m*-fold cross-validation). Ensuite, la méthode est construite à partir de l'ensemble des partitions moins une qui servira de test. Après on réitère le processus en introduisant la partition testée dans l'ensemble d'apprentissage et en prenant une autre partition d'apprentissage pour tester la méthode et ainsi de suite jusqu'à ce que toutes les

données soient utilisées tantôt pour l'apprentissage et tantôt pour le test. La moyenne des taux de classification correcte sur toutes les partitions de test correspond au taux de prédiction.

3. Etude Expérimentale de l'approche génétique pure

3.1 Etude de l'influence des paramètres de l'AG sur la précision du classificateur obtenu

Le processus de l'Algorithme Génétique est guidé par un certain nombre de paramètres que nous devons fixer à l'avance. La valeur de ces paramètres influence la réussite ou non de l'algorithme. Ces paramètres sont les suivants [GOL1989][HAU 2004] :

- La taille de la population N : si N est trop grand, le temps de recherche par l'algorithme devient important. Si N est trop petit, l'algorithme peut converger trop rapidement vers un optimum local.
- La probabilité de croisement pc : elle dépend de la forme de la fonction de fitness. Plus elle est élevée, plus la population subit des changements importants. Les valeurs généralement admises sont comprises entre 0,5 et 0,9.
- La probabilité de mutation pm : ce taux est généralement faible puisqu'un taux élevé risque de conduire à une solution sous-optimale, et à la perte de la population originale.
- La taille de la base d'apprentissage
- Les paramètres λ_1, λ_2, λ_3 de la fonction objectif.
- Le nombre de générations qui peut également être défini à priori comme critère d'arrêt.

Afin d'illustrer l'influence de ces différents paramètres sur le processus de recherche, nous avons effectué une série d'expérimentations.

3.1.1 Influence de la taille de la base d'apprentissage

Nous allons prendre les bases de données *Hepatitis* et *Ionosphere* pour voir l'influence de ce paramètre sur la précision du classificateur et nous choisissons des tailles différentes de la base d'apprentissage. Puis nous exécutons le classificateur basé sur l'algorithme génétique 10 fois successives pour chaque taille.

Les paramètres P_C et P_M de l'algorithme génétique sont $P_M = 0.1$ et $P_C = 0.8$ et les paramètres λ_1, λ_2 et λ_3 de la fonction objectif du classificateur sont λ_1=0.2, λ_2=0.8 et λ_3=0. Les résultats obtenus sont résumés dans le tableau 2.

Algorithme Génétique									
10%		**25%**		**50%**		**75%**		**100%**	
Hepatitis	***Ionosphere***	***Hepatitis***	***Ionosphere***	***Hepatitis***	***Ionosphere***	***Hepatitis***	***Ionosphere***	***Hepatitis***	***Ionosphere***
70,28	70,59	70,00	73,67	70,35	77,56	87,17	83,48	93,41	95,46
71,42	75,68	66,75	77,30	71,84	76,78	82,05	84,35	90,35	93,85
69,85	70,46	69,57	69,68	69,64	70,55	84,61	82,11	87,41	90,30
65,42	69,67	65,29	73,56	70,24	79,56	81,05	79,99	89,99	93,66
69,29	70,34	68,46	74,33	71,15	75,39	79,48	83,39	90,70	91,36
72,85	68,19	69,31	68,99	67,14	70,11	84,61	85,76	94,45	92,45
62,85	73,55	73,84	73,55	77,69	77,34	76,31	86,09	91,06	93,56
59,42	77,99	65,89	75,80	72,82	75,94	74,35	80,77	93,39	96,99
69,28	76,44	67,86	68,29	71,94	78,48	82,50	87,05	94,35	95,40
67,85	65,66	71,88	76,94	65,84	69,11	73,68	81,11	92,15	96,19
67,85 %	**71,86 %**	**68,89 %**	**73,21 %**	**70,87 %**	**75,08 %**	**80,58 %**	**83,41 %**	**91,73 %**	**93,92 %**

Tableau 2 : Résultats obtenus avec des tailles différentes de la base d'apprentissage

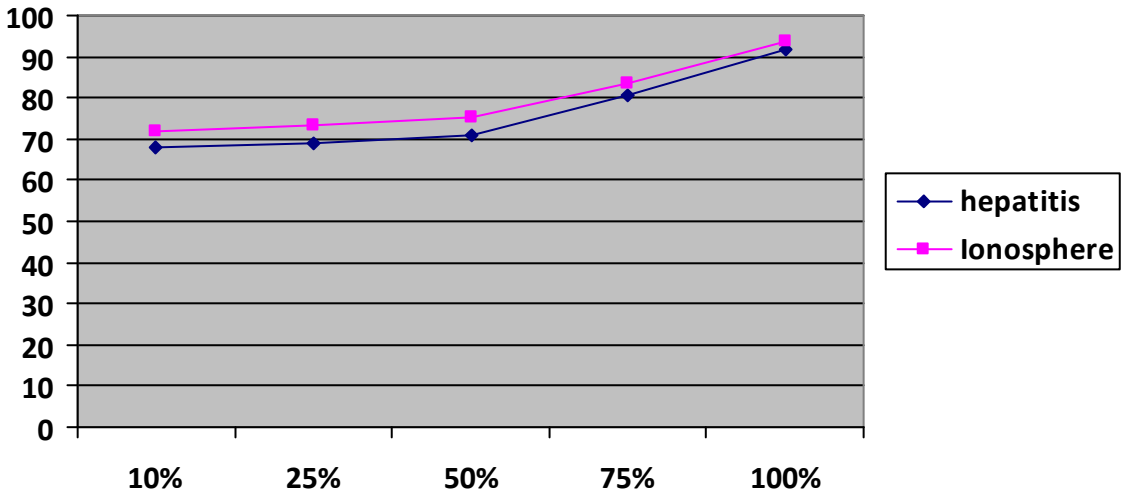


Figure 7.1 : Influence de la taille d'apprentissage sur la précision du classificateur

À travers les résultats du tableau 2 représentés par les deux graphiques de la figure 1, nous pouvons déduire que l'impact de la taille de la base d'apprentissage est important. En effet, la précision moyenne du classificateur augmente avec l'augmentation de la taille de la base d'apprentissage.

3.1.2. Influence des paramètres λ_1, λ_2 et λ_3 de la fonction objectif

Les mêmes bases de données *Hepatitis* et *Ionosphere* sont utilisées pour illustrer l'impact de ces paramètres sur la précisions du classificateur. Chacun des paramètres λ_1, λ_2 et λ_3, est modifié puis pour chaque cas nous exécutons le classificateur génétique 10 fois successives.

Les paramètres P_C et P_M de l'algorithme génétique sont $P_M = 0.1$ et $P_C = 0.8$ et la taille de la base d'apprentissage est 75%. Les résultats obtenus sont résumés dans le tableau suivant :

λ_1 = facteur de pénalisation de la Couverture

λ_2 = facteur de pénalisation de la Précision

λ_3 = facteur de pénalisation de la Compréhensibilité en terme de taille de la règle

Algorithme Génétique									
Cas 1 : λ1=0.80 λ2=0.10 λ3=0.10		**Cas 2 : λ 1=0.50 λ 2=0.25 λ 3=0.25**		**Cas 3 : λ 1=0.50 λ 2=0.50 λ 3=0.00**		**Cas 4 : λ 1=0.10 λ 2=0.80 λ 3=0.01**		**Cas 5 : λ 1=0.10 λ 2=0.10 λ 3=0.80**	
Hepatitis	**Ionosphere**	**Hepatitis**	**Ionosphere**	**Hepatitis**	**Ionosphere**	**Hepatitis**	**Ionosphere**	**Hepatitis**	**Ionosphere**
45,16	56,37	71,29	68,87	78,70	73,25	87,17	83,48	81,29	75,40
37,09	59,74	72,25	73,94	72,58	69,45	82,05	84,35	72,58	80,39
43,07	62,11	35,48	60,45	71,93	77,39	84,61	82,11	84,51	76,67
47,09	55,35	74,51	70,67	73,29	65,87	81,05	79,99	77,41	72,67
54,51	49,49	60,64	65,56	79,35	79,44	79,48	83,39	75,16	81,67
65,48	45,56	67,74	68,02	60,87	69,79	84,61	85,76	80,64	78,45
54,83	56,38	71,61	60,11	75,48	71,35	76,31	86,09	71,93	74,46
58,06	57,63	62,58	53,34	59,35	73,59	74,35	80,77	55,43	71,37
70,64	50,17	48,38	69,97	68,70	75,47	82,50	87,05	73,87	70,36
21,93	48,99	58,06	68,45	75,43	73,56	73,68	81,11	75,43	77,45
49,79 %	**54,18 %**	**62,25 %**	**65,94 %**	**71,57 %**	**72,92 %**	**80,58 %**	**83,41 %**	**74,83 %**	**75,89 %**

Tableau 3 : Résultats obtenus avec des valeurs différents des paramètres λ_1, λ_2 et λ_3 de la fonction objectif.

Le tableau 3 donne pour chaque cas la précision moyenne de 10 essais successifs du claasifieur. D'après ces résultats nous remarquons que le quatrième cas (λ_1=0.1, λ_2=0,8 et λ_3=0.1) donne le meilleur résultat. Nous remarquons que la précision moyenne du classificateur augmente si le paramètre $\boldsymbol{\lambda_2}$ associé au taux de précision augmente.

3.1.3 Influence de la taille de la population

Nous allons utiliser les bases de donnés *Hepatitis* et *Ionosphere* pour illustrer l'influence de ce paramètre sur la précision du classificateur. Nous allons modifier la taille de la population initiale, Puis nous exécutons le classificateur génétique 10 fois successives.

Les paramètres PC et PM de l'algorithme génétique sont PC=0.1 et PM=0.8 et la taille de la base d'apprentissage est 75%. les paramètres λ_1,λ_2 et λ_3 de la fonction objectif du classificateur sont λ_1=0.2, λ_2=0.8 et λ_3=0. Les résultats trouvés sont présentés dans le tableau4.

Algorithme Génétique		
Taille population	**Hepatitis**	**Ionosphere**
50	50,39	59,09
100	58,73	67,47
200	69,95	78,93
500	75,36	80,56
1000	76,23	80,45
5000	75,55	81,03
7500	76,89	81,28
10000	77,45	81,05
15000	78,18	81,86
20000	79,32	82,13

Tableau 4 : Résultats obtenus avec des tailles différents de la population

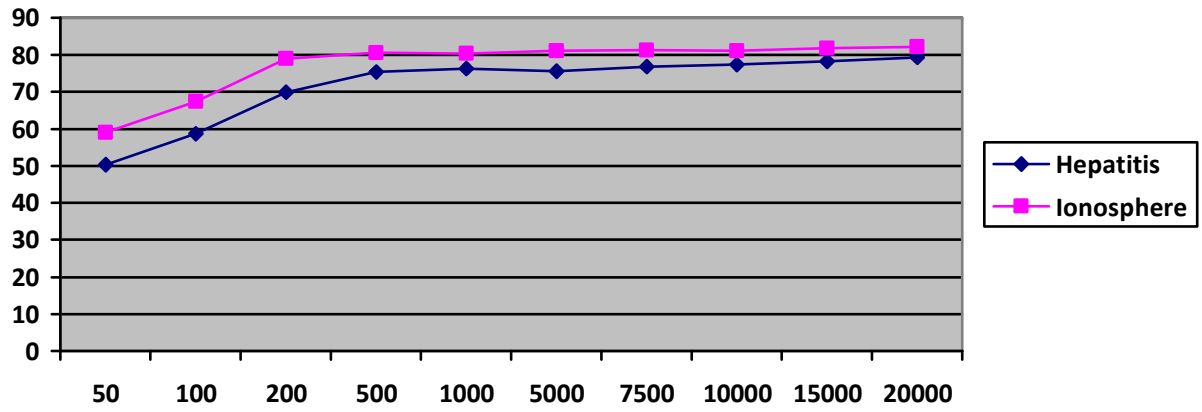


Figure 7.2 : Influence de la taille de la population sur la précision du classificateur génétique obtenu

3.1.4 Influence du nombre d'itérations sur la précision du classificateur obtenu

Algorithme Génétique		
Nombre d'Itérations	**Ionosphere**	**Hepatitis**
100	85%	83,22%
200	93%	84,51%
500	96,35%	93,54%
800	97,39	95,50%
1000	98,34	98,06%

Tableau 5 : Nombre d'itérations total et Précision obtenue

Sur le tableau 5, nous constatons que plus le nombre d'itérations est grand et plus la précision du classsifieur obtenue est meilleure.

3.2 Résultats Obtenus par l'approche génétique pure

Après avoir effectué ces différents tests des paramètres de l'AG et leurs influences sur la qualité du classificateur obtenu, nous avons opté pour réaliser les tests de toutes nos approches développées sur les bases de données de l'UCI, à l'aide des valeurs des paramètres de l'AG résumées dans le ***tableau 6.***

:

Paramètres	Valeurs
Taille de la base d'apprentissage	85%
Taille de la population	200 individus
Nombre d'itérations	1000 itérations pour la version séquentielle
Probabilité d'application des opérateurs génétiques	• Probabilité de croisement (pc)=0.8 • Probabilité de mutation (pm)=0.1
Les paramètres « λ_i » de la fonction objectif	• $\lambda 1$=0.2 • λ_2=0.8 • λ_3=0.0

Tableau 6: les paramètres utilisés par l'algorithme génétique.

Afin d'évaluer les performances de l'approche génétique nous avons exécuté le classificateur génétique 10 fois successives sur les benchmarks de l'UCI données précédemment et nous donnons à chaque fois les précisions obtenues ainsi que le nombre de règles.

	Hepatitis	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Précision du classificateur (%)	85,16	67.65	92,15	97.36	90.75
	96,77	66,26	88,31	83.67	98.17
	94,19	65,46	80,06	99.66	93.48
	81,29	70,46	70,90	84.67	91.01
	95,48	66,06	72,64	94.81	98.95
	96,48	67,55	92,50	95.41	92.57
	87,09	69,45	80,05	96.14	95.31
	90,96	70,05	93,73	99.73	88.15
	96,77	69 ,85	89,50	83.67	81.38
	94,19	68,24	92,30	86.37	91.14
Moyenne	**91,83**	**68,10**	**85,21**	***92.14***	***92.09***

Tableau 7 : Résultats des tests obtenus sur différents benchmarks par l'AG pur en terme de précision

	Hepatitis		Segment-challenge		Ionosphere		Kdd-train	Diabetes	
Nombre de règles et taille d'une règle	23	3	25	4	12	4	28	3	7
	18	4	15	3	13	5	23	26	4
	8	4	15	2	2	7	21	3	7
	18	3	14	4	16	3	25	10	5
	15	3	15	4	10	5	27	4	6
	11	5	22	4	11	5	21	4	5
	25	2	25	3	8	4	16	5	3
	17	4	22	4	12	4	20	6	4
	17	4	24	3	7	4	14	5	5
Moyenne	**15**	**3**	**17**	**3**	**9**	**4**	**22**	**7**	**5**

Tableau 8 : Résultats des tests obtenus sur différents benchmarks par l'AG pur en terme de compréhensibilité

Notons que pour chaque base de données nous avons relevé le nombre de règles du classificateur obtenu et la taille moyenne d'une règle.
En observant les deux tableaux précédents, nous pouvons constater que les classificateurs obtenus pour différents benchmarks de l'UCI sont satisfaisants : la précision est assez élevée et le nombre et taille des règles sont intéressants et induisent un classificateur compréhensible.

4. Etude Expérimentale de l'approche mémétique pour l'extraction de règles de classification

Afin de pouvoir étudier l'efficacité de l'approche mémétique, nous avons implémenté les deux opérateurs de recherche locale : SLS correspondant à "Simple Local Search" et LGA correspondant à "Local GA" Search où la recherche locale elle-même est basée sur un mini GA comme décrit dans le chapitre VII.

4.1 Tests des approches mémétiques avec S_LS et LGA comme méthodes de recherche locale

Dans le but d'étudier la synergie des opérateurs génétiques avec l'opérateur de recherche locale, six stratégies d'application de la recherche locale ont été implémentées, à savoir les points d'application de la recherche locale suivants :
SLS(M): correspond à l'approche mémétique avec la recherche locale simple appliquée juste aprés la mutation.
SLS(C): correspond à l'approche mémétique avec la recherche locale simple appliquée juste aprés le croisement.
SLS(C+M) : correspond à l'approche mémétique avec la recherche locale simple appliquée juste aprés le croisement et juste aprés la mutation.
LGA(M): correspond à l'approche mémétique avec la recherche locale basée sur l'AG appliquée juste aprés la mutation.
LGA(C): correspond à l'approche mémétique avec la recherche locale basée sur l'AG appliquée juste aprés le croisement.

LGA(C+M): correspond à l'approche mémétique avec la recherche locale basée sur l'AG appliquée juste aprés le croisement et juste aprés la mutation.

SGA : Standard GA et qui correspond à l'approche génétique pure (sans application de la recherche locale).
Les paramètres utilisés concernant la recherche locale sont données dans le tableau 9
Nous avons exécuté 10 fois chaque stratégie et nous avons reporté dans le tableau 10 la précision maximale obtenue et dans le tableau 11 la compréhensibilité maximale en terme de nombres de règles et de nombre de prémisses de chaque règle..

	Pour S_LS	*Pour LGA*
Taille Population	*Nb_Attributs de la base d'apprentissage*	*Nb_Attributs de la base d'apprentissage*
Nbre d'itérations	*1*	20
P. croisement		0.25
P. mutation		0.5

Tableau9 : Les paramètres de la Recherche Locale

DataSet	*SGA*	*S_LS(M)*	*S_LS(C)*	*SLS(C+M)*	*LGA (M)*	*LGA(C)*	*LGA(C+M)*
Weather	92,8	100	100	100	100	100	100
Iris	94,7	100	100	100	99,33	100	100
Zoo	80,2	99	99,02	100	96,03	99,02	100
Kdd_train	98,6	99,99	99,92	99,98	100	100	100
Breast_C.	97,5	98,41	100	100	100	99,3	99,82
Seg_Chal.	59,8	99,20	96,20	98,8	99,06	97,8	97

Tableau10 : Maximum de précison (%) obtenue

Data Set	*SGA*		*SLS (M)*		*SLS (C)*		*SLS (C+M)*		*LGA (M)*		*LGA (C)*		*LGA (C+M)*	
Weather	4	2	4	1	5	2	4	1	4	1	4	1	6	2
Iris	8	1	6	2	4	2	5	1	5	1	6	2	4	1
Zoo	11	3	15	3	9	3	16	3	16	3	9	3	14	3
Kdd_train	11	5	6	6	3	4	6	6	6	6	10	6	6	6
Breast_Cancer	18	4	12	4	9	5	11	3	11	3	3	4	13	3
Seg_Challenge	14	3	26	4	30	3	40	5	40	5	26	5	35	4

Tableau11 : Maximum de Compréhensibilité

Sur ces tableaux nous constatons un excellent taux de précision obtenu par les approches mémétiques : il avoisine les 100% dans presque tous les cas d'application de la recherche locale. Dans le cas de la base de données Kdd_train qui comporte plus de 11400 instances chacune comportant 42 attributs, tous les classificateurs obtenus par application de LGA présentent une précision de 100% avec seulement 6 règles de 6 attributs (voir tableau 11), d'où compréhensibilité et précision assurées dans un excellent compromis.

Autre exemple, utilisant l'ensemble de données *Breast_Cancer* qui est un ensemble de tuples représentant des patients atteints et non atteints de maladies incurables : Pour la stratégie LGA(C+M), le classificateur obtenu est constitué de 13 règles contenant au plus 2 prémisses pour chacune d'elles, alors qu'avec la stratégie LGA(C) nous avons obtenu 3 règles contenant chacune 4 prémisses (voir Tableau 11).
C'est donc clair que tous les objectifs ont été largement atteints.

4.2 Tests de l'approche mémétique avec Tabu Search comme recherche locale

Nous présentons à ce niveau les résultats de l'approche mémétique qui a consisté à hybrider la recherche taboue comme recherche locale à l'AG. Pour cela, nous avons utilisé les paramètres résumés par le tableau 12 suivant. Ici, nous avons utilisé un taux d'apprentissage à 75% de la base d'apprentissage.

Parametres	Paramètres de l'AG	Parametrer de la recherche **Taboue**
Taille de la base d'apprentissage	75%	75%
Taille de l'ensemble de tests	25%	25%
Taille de la population Initiale	200	1
Nombre d'itérations	500	500
Taux de Croisement (C_P)	0.8	Taille de la liste Tabou = 5
Taux de Mutation (P_M)	0.25	

Tableau12 : Paramètres de l'approche mémétique avec la recherche taboue

Benchm.	Nombre de Règles(GA)	Précision(GA)	Nombre de Règles(TS)	Précision(TS)
Hepatitis	34	79,79 %	14	69,59 %
Heart-statlog	40	70,18 %	22	68,41 %
Segment Chal.	102	78,06 %	54	59,73 %
Ionosphere	63	80,67 %	43	74,56 %
Kdd-train	22	87,65 %	32	59,27 %
Diabetes	7	68,25 %	4	61,10 %

Tableau 13 : Compréhensibilité et Précision obtenues par l'AG pur et par la Recherche Tabou pure

Nous remarquons sur ce tableau que la recherche taboue pure a donné des classificateurs moins précis que ceux obtenus par AG pur alors que le nombre de règles par AG est plus important que celui obtenu par TS. Il y a donc un compromis à trouver entre la précision et la compréhensibilité obtenus par chaque approche.

Le tableau 14 résume les résultats obtenus par l'approche hybride.

Dans le but de montrer l'efficacité de la méthode nous avons implémenté 6 stratégies d'application de la recherché locale TS dans l'AG.

Strategy 1 : appliquer la Tabu Search à toute la population initiale.

Strategy 2 : appliquer la Tabu Search après mutation

Strategy 3 : appliquer la Tabu Search après le croisement.

Strategy 4 : appliquer la Tabu Search à toute la population initiale et après la mutation.

Strategy 5 : appliquer la Tabu Search à toute la population initiale et après le croisement.

Strategy 6 : appliquer la Tabu à toute la population initiale et après la mutation et après le croisement.

Benchmark	Strategies					
	1	2	3	4	5	6
Hepatitis	78,68	85,50	84,60	83,77	**85,94**	83,54
Heart-statl.	73,06	83,66	90,66	85,56	89,67	**91,89**
Segment-Ch	79,67	**94,47**	94,05	93,67	94,38	93,87
Ionosphere	82,89	**93,67**	95,14	92,74	93,79	92,67
Kdd-train	85,46	98,56	99,09	**99,45**	99,18	98,93
Diabetes	70,83	**82,73**	80,72	80,39	80,59	80,35
Moyenne	**78,43**	**89,77**	**90,71**	**89,26**	**90,59**	**90,21**

Tableau 14 : Résultats obtenus par l'AM avec TS comme recherche locale.

Benchmark	Nombre de règles	Précision(%)
Hepatitis	9	84,60
Heart-Statlog	20	90,66
Segment-Ch	43	94,05
Ionosphere	9	95,14
Kdd-train	7	99,08
Diabetes	3	80,72

Tableau15 : Compréhensibilité exprimée en terme de nombre de règles

Ces tableaux montrent bien la cynergie des opérateurs génétiques avec la recherche locale. Nous remarquons qu'avec seulement 75% comme taux d'apprentissage, nous avons obtenus de hautes précisions et une compréhensibilité intéressante pour les classificateurs obtenus pour chaque benchmark par la méthode mémétique avec hybridation de la Tabu Search comme recherche locale.

5. Comparaison des résultats obtenus par nos approches et ceux obtenus par PART, Decision Table et OneR

Ces algorithmes : PART [FRA 1998] basé sur C4.5 [QUI 1993], Decision Table [KOH 1995] et OneR [BUD 1992] sont des algorithmes d'extraction de règles de classification bien connus, les trois algorithmes sont sommairement décrits dans le chapitre 2.
Le tableau 16 résume tous les résultats obtenus en terme de précision du classificateur par nos différentes stratégies et par les 3 algorithmes mentionnés ci-dessus.

DataSet	*Weather*	*IRIS*	*Zoo*	*Kdd_Train*	*Breast_ Cancer*	*Segment_ Challenge*
SGA	92,8	94,7	80,2	98,6	97,5	59,8
SLS(M)	100	100	99	99,99	98,41	99,20
SLS(C)	100	100	99	99,92	100	96,20
SLS(C+M)	100	100	100	99,98	100	98,8
LGA(M)	100	99,33	96,03	100	100	99,06
LGA(C)	100	100	99,02	100	99,3	97,8
LGA(C+M)	100	100	100	100	97,82	97
OneR	**64,3**	**92,66**	**42,57**	**98,01**	**90,33**	**63,53**
Dec_Table	**57,30**	**94**	**87,12**	**99,76**	**92,09**	**91,87**
PART	**42,8**	**93,3**	**92,07**	**99,01**	**93,67**	**94,46**

Tableau 16 : Précisions obtenues par l'AG, l'AM et ses différentes stratégies et les algorithmes OneR, Decision Table et PART.

Ce que nous observons sur le tableau 16, c'est que toutes nos approches ont obtenu de meilleurs classificateurs en terme de précision par rapport aux algorithmes connus dans la littérature. Même PART qui est basé sur l'algorithme C4.5 de classification n'a pas donné d'aussi bons taux de classification, à part par rapport à l'AG classique (SGA dans ce tableau). En effet, l'algorithme mémétique a obtenu de très bons scores pour toutes les stratégies d'application de la recherche locale. Ceci prouve la bonne synergie entre les opérateurs génétiques et la recherche locale d'une part et la grande exploration/exploitation de l'espace de recherche : exploration faite par les opérateurs génétiques et exploitation par la recherche locale.
Par ailleurs, nous avons constaté en effectuant nos tests que les temps d'execution de SGA (approche génétique pure) sont plus longs que ceux de l'approche mémétique pour toutes les stratégies. Ceci a été une surprise car nous nous attendions au contraire vue que la recherche locale prend un certain temps en plus pour s'exécuter. En fait, ceci s'explique par le fait que l'AG pur explore "aveuglément" l'espace de recherche alors que l'AM est en quelque sorte guidé dans ses recherches de la meilleure solution par la recherche locale. Cette dernière joue

le rôle d'un "zoom" sur une solution trouvée par l'AG pour atteindre une meilleure solution dans le voisinage immédiat dans l'espace de recherche.

6. Tests et Comparaisons avec les Approches: DOEA [TAN 2006] et PSO/ACO2 [FRE 2008]

Pour montrer l'efficacité de notre approche mémétique, nous avons également mené des comparaisons avec des approches basées sur des metaheuristiques, notamment les travaux de A. A. Freitas et K.C Tan qui ont mis au point des hybridations de metaheuristiques pour l'extraction de règles de classification. Nous avons décrit leurs travaux dans le chapitre 5.
Pour cela nous avons utilisé les mêmes benchmark de l'UCI. Le tableau 17 résume les résultats obtenus en terme de précision (%) des classificateurs.

DataSet	Weather	ZOO	Credit_g	Breast Cancer	Hepatitis	Diabetes	Heart Statlog
DOEA	98	99		84	97		
PSO/ACO2	99,93		73,7	79,46		77,65	87,27
SGA	94,7	80,2	99,7	97,54	93,5	89,6	99,6
SLS(M)	100	99	99,6	98,41	100	98,82	100
SLS(C)	100	99,02	99,8	100	100	98,8	100
SLS(C+M)	100	100	100	100	100	99,86	100
LGA(M)	99,33	96,03	96	100	100	98,45	100
LGA(C)	100	99,02	99,7	99,30	100	99,86	100
LGA(C+M)	100	100	98,2	99,82	100	99,6	99,62

Tableau 17 Résultats (%précision) obtenus par l'approche mémétique et les approches basées sur des metaheuristiques.

Cet ensemble de tests montre bien que dans la plupart des stratégies et pour la plupart des benchmarks utilisés les résultats en terme de précision sont bien supérieurs à ceux obtenus par DOEA et PSO/ACO2, excepté pour l'AG Standard (SGA) qui a donné moins de performances que DOEA, bien que pour la base de données Breast_Cancer il reste meilleur que DOEA ET PSO/ACO2. Nos stratégies de l'AM : SLS(M) SLS(C+M) et LGA(C+M) ont donné 100% de précision pour la plupart des ensembles de données étudiés. Ceci est dû en grande partie à l'intensification de la recherche induite par la recherche locale.
.

7. Tests Expérimentaux de l'AG sans génération et parallèle

Le développement de l'AG parallèle sans génération a été effectué dans le langage JAVA qui contient les classes THREAD avec toutes les fonctions nécessaires pour la connexion des machines esclaves à la machine Maitre.
Pour cela, nous avons besoin de positionner certains paramètres.

Coté serveur (maître)

Ajouter les paramètres suivants :

evalthreads = entier positif à préciser //le nbre de machines esclaves à accepter.
master.host = l'adresse IP à laquelle les esclaves doivent se connecter.
master.port = 15000 // le numéro de port.

Coté esclave

Sur chaque machine esclave on doit disposer d'un fichier nommé "esclave", qui à comme paramètre :

Parent.0 = chemin du fichier maître.

Evidemment, une copie du fichier maître doit exister dans chaque machine, et cela pour ne pas réécrire les mêmes paramètres dans chaque fichier « esclave ».

Les résultats de l'évaluation du classificateur, les règles extraites ainsi que le temps de calcul seront affichés dans un fichier « out » à préciser de la machine maître.

7.1 Test et évaluation du nouvel AG sans générations

Nous avons implémenté une version séquentielle de l'AG modifié, ie : sans génération et avec durée de vie pour chaque individu. Pour montrer l'efficacité de cette approche, nous allons donner les temps d'exécution de la version séquentielle qui sont déjà nettement améliorés par rapport à l'AG classsique. En effet, dans le nouvel AG (avec durée de vie) seuls les individus nouveaux nés sont à évaluer, alors que dans l'AG classique lorsque le tableau de population est rempli d'individus de la nouvelle génération après la phase de sélection, on procède à l'évaluation de *tous* les individus la population, y compris ceux qui ont été sélectionnés sans avoir subi de modifications. Sachant que le calcul de la fonction d'évaluation consomme plus de 80% du temps total de l'AG, forcément l'AG classique consomme plus de temps d'exécution que le nouvel AG.

Les datasets pour lesquels le Nouvel AG (NGA) a été testé en version séquentielle sont Kdd_Train, Mushroom et Chess-end-Game. Les tests du tableau 18 ont été effectués sur une machine IBM (Pentium 4 à 1,7GHz)

Dataset	**NGA(NewGA)**	SGA
Kdd_Train	**3mn58s**	14mn12s
Mushroom	**3mn11s**	9mn24s
Chess End-Game	**2mn14s**	7mn5s

Tableau 18 : Temps d'exécution du nouvel AG comparé à l'AG classique

Ce tableau montre bien que le temps d'exécution du nouvel AG (NGA) est nettement amélioré par rapport à celui de l'AG standard (SGA) et ceci en séquentiel.

7.2 Tests et résultats de l'AG parallèle semi-asynchrone

Pour tester l'approche parallèle semi asynchrone, nous avons eu recours à 10 machines ayant les mêmes performances, connectées en réseau LAN. Les temps obtenus pour l'AG parallèle semi-asynchrone sont résumés dans le tableau 19, pour le dataset **Insurances** de l'UCI.

Nombre de Machines Esclaves Connectées à la machine Maitre	Temps d'exécution obtenu (en minutes et secondes)
1	00 :04:25
2	00 :02 :35
3	00 :01 :42
4	00 :01 :29
5	00 :01 :16
6	00 :01 :15
7	00 :01 :11
8	00 :01 :08
9	00 :01 :02
10	00 :01 :02

Tableau 19 : Temps obtenus par l'approche parallèle semi-asynchrone pour 10 Machines Esclaves connectées en réseau LAN

Nous remarquons qu'à chaque fois qu'une nouvelle machine est rajoutée dans le réseau, le temps est divisé par 2 au début il commence à se stabiliser pour devenir tout à fait stable à la 10ème machine raccordée au réseau. Ceci est dû aux temps de communication via le réseau que nous avons utilisé et qui n'avait pas des performances interressantes. D'autre part, cela est dû également en partie à l'algorithme parallèle qui n'est pas complètement asynchrone, où il reste un point mineur de synchronisation.

Ces résultats ont donné la courbe de speed-up suivante. Voir figure 8.2

Figure 8.2 Speed-up obtenu avec la version parallèle semi-asynchrone pour la base de données INSURANCES.

Rappelons que le speed-up est la courbe représentant l'accélération du temps d'exécution obtenu par un algorithme parallèle, avec :

Speed-up=Ts / Tp

Où Ts est le temps obtenu en séquentiel et Tp le temps obtenu en parallèle, ce qui permet de mesurer la performance d'un algorithme parallèle.
Sur cette courbe nous voyons bien que l'accélération obtenue est de l'ordre de 2,5 avec 10 machines esclaves.

7.3 Test de l'Algorithme complètement Asynchrone

Pour ce test, nous avons utilisé le même Benchmark et le même nombre d'esclaves.
La courbe de Speed Up obtenu avec 10 machines esclaves sur la même base de données.

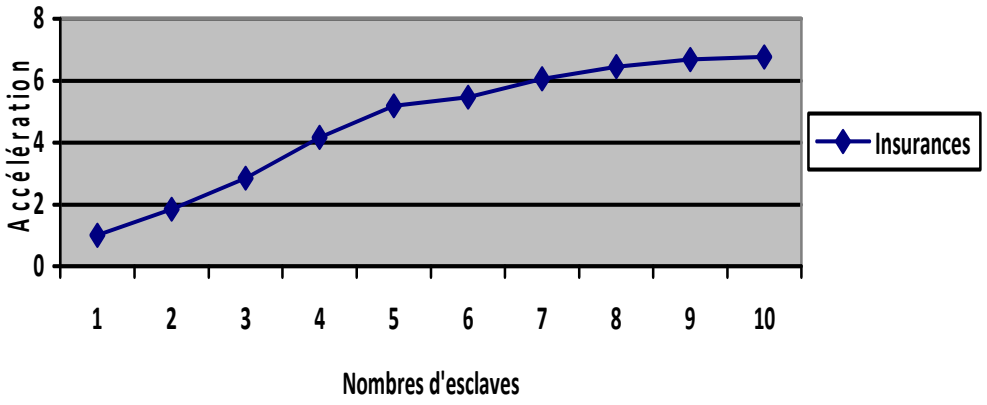


Figure 8.3 : Courbe de Speed up obtenue avec l'AGP asynchrone

La courbe dans ce cas est nettement meilleure que celle obtenue par l'approche semi_asynchrone. En effet, avec 10 machines esclaves, la courbe a accéléré et avoisine une accélération de 7. Cette approche complètement asynchrone, où les deux processus "Production" et "Evaluation" travaillent indépendamment l'un de l'autre s'avère donc très efficace et prometteuse dans les applications aux problèmes à très grande taille.

8. Conclusion

Ce chapitre a eu pour but de montrer les tests et résultats expérimentaux obtenus pour chaque approche développée. Les tests ont été menés sur des fichiers "Benchmarks" de l'Université de Californie Irvine (UCI) spécialement mis à la disposition de la communauté des chercheurs en data mining dans le monde.

Tel que constaté sur les différents tableaux, nos résultats sont concluants et montrent largement l'efficacité de toutes nos approches développées.

Les versions parallèles ont induit des courbes de "Speed Up" intéressantes montrant les gains obtenus par le parallélisme de données.

Nos différentes contributions dans la résolution du problème d'extraction de règles de classification en data mining ont montré une grande efficacité. Nous pensons qu'elles trouveront une application directe dans le domaine des problèmes à très grande taille.

Conclusion Générale et Perspectives

Les travaux de recherche que nous avons effectués ont consisté à montrer l'apport des metaheuristiques dans le problème de la classification connu pour son aspect fortement combinatoire.

Dans cette thèse, nous avons abordé et étudié le problème de recherche de règles de classification en extraction de connaissances dans les grands volumes de données, en ayant recours à des approches évolutionnaires.

Ainsi, plusieurs contributions ont été apportées :

- D'abord, nous avons mené une large étude bibliographique de la fouille de données en général puis du problème de la classification supervisée et non supervisée en particulier.
- Ensuite nous avons modélisé le problème de la recherche de règles de classification comme un problème d'optimisation multi-objectif que nous avons ramené à un problème mono_objectif par une méthode d'agrégation.
- Nous avons ensuite développé plusieurs versions d'un algorithme de résolution évolutionnaire : génétique pur, parallèle complètement asynchrone et mémétique en hybridant à l'AG différentes méthodes de recherche locale. L'étude de la synergie entre les opérateurs génétiques et la recherche locale a été effectuée dans le but de montrer l'efficacité des approches mémétiques. Ces algorithmes ont été testés sur des bases de données d'apprentissage "Benchmark" de l'UCI.

En raison du nombre important de données traitées, les techniques de datamining et d'apprentissage se sont avérées un outil trés utile pour la construction de modèles servant à classer des individus dans des classes prédéfinies. En effet, les modèles de classification extraits ont montré un grand degré de compréhensibilité dans la mesure où les classificateurs obtenus présentent une petite taille en terme de nombre de règles ainsi qu'en nombre de prémisses de ces dernières. D'autre part la précision des classificateurs en terme de nombre d'individus bien classés par rapport au nombre d'individus à classer avoisine les 100% dans la plupart des cas. Ainsi les objectifs fixés ont été largement atteints.

- Nous avons également tenté de diminuer les temps d'exécution de nos approches, étant donné le nombre de données à traiter dans ce domaine. Pour cela, nous avons modélisé un nouvel algorithme génétique parallèle basé sur le modèle Maitre/Esclave ainsi que sur la notion de cycle de vie d'individus. Ceci nous a permis de concevoir deux processus concurrents complètement indépendants : l'un pour la production de nouveaux individus dans la population de l'AG et l'autre pour l'évaluation des individus crées et non pas toute la population comme dans un modèle AG Maitre/Esclave classique. Notre modèle parallèle est complètement asynchrone et donc l'accélération obtenue s'est avérée intéressante.

Les résultats obtenus par nos approches ont été comparés à des résultats obtenus par des algorithmes de classification classiques tels que PART basé sur C4.5, OneR et Decision Table sur les mêmes bases de données apprentissage "Benchmark" de l'UCI. Nous avons comparé également par rapport à des approches basées sur d'autres metaheuristiques développées par A.A. Freitas et K.C. Tan pour le même problème. Nos résultats ont montré de meilleures précision et compréhensibilité par rapport à ces travaux, surtout l'approche mémétique basée sur la recherche locale simple et un AG réduit pour améliorer les performances d'un individu dans son voisinage.

Nous espérons donc avoir atteint les objectifs fixés dans cette thèse. Néanmoins, tout travail de recherche reste sujet à développement. Nous nous proposons d'apporter dans nos futurs travaux, une résolution multi_objectif basée sur la méthode de Pareto pour l'extraction de règles de classification pour pouvoir trouver les bons compromis entre la compréhensibilité du modèle obtenu et sa précision.
Vues les masses volumineuses des données traitées en datamining, une approche parallèle multi_objectif devrait apporter des résultats appréciables efficacement à ce problème. L'application de ces approches au domaine médical pourrait s'avérer fort interressante.

Les Références Bibliographiques

[AFS2007] A. Afshar, O.B. Haddad, M.A. Marino, B.J Adams, Honey Bee Mating Opt. (HBMO) algorithm for optimal reservoir operation, Journal of the Franklin Institute, Vol. 344, Issue 5, August 2007, Pages 452-462, Modeling, Simulation and Applied Optimization PartII. Science Direct.

[AGR1994] R.Agrawal et A. Srikant. Fast algorithms for mining association rules. Proc, pp 787-499, VLDB 1994

[BAB2009] A.R. Baba-Ali, A new two level evolutionary approach for classification rules extraction, in proceeding of the IEEE Congress on Evolutionary Computation CEC2009, Norway.

[BAC2003] J. Bacardit, J.M. Garell, 'Multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system'. In proceeding of the Genetic and Evolutionary Computation Conference GECCO 2003, pp 1818-1831 LNCS 2724,Springer.

[BAC2004] J. Bacardit, 'Pittsburgh Genetic-based Machine Learning in the data mining area,Representations, Generalisation and Run-Time', PhD Thesis Dissertation, Computer Science Department, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Barcelona 2004.

[BAC 2006] Bacardit, J. and Krasnogor, N. *Smart crossover operator with multiple parents for a pittsburgh learning classifier system.* In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 1441–1448, New York, NY, USA. ACM Press, 2006.

[BEN2006] S. Benkhider, A.R. Baba-ali, H.Benhabiles, Y.Boudi, H.Drias 'Une nouvelle approche évolutionnaire parallèle basée "Cycle de Vie" pour l'optimisation combinatoire', Proceeding of the international conference META 2006, organisée par le LIFL de l'USTL, Hammamet, Tunisie. oct.2006.

[BEN2007] S.Benkhider, A.R. Baba-Ali & H. Drias, 'A new Generationless Parallel Evolutionary Algorithm for Combinatorial Optimization', Proceeding of the IEEE International Congress on Evolutionary Computation CEC2007, Singapore, 2007.

[BEN2008] S.Benkhider and H.Drias, 'A New Memetic Approach for Classification Rule Extraction' , proceeding of the international conference META 2008, organisée par le LIFL de l'USTL, Hammamet, Tunisie. oct.2008.

[BEN2011] S. Benkhider, H. Drias, 'A New Memetic Approach for the Classification Rules Extraction Problem', International Journal of Data Mining, Modeling and Management, Special Issue on Evolutionary Computation and Related Metaheuristics, InderSciences Pub, Accepted on April 2011, published DEC. 2013.

[BEN2012] S. Benkhider, O. Dahmri, H. Drias, ' A Memetic Approach for the Knowledge Extraction', Proceeding of the LNCS International Conference on Neural Intelligent Processing ICONIP 2012, Springer Pub., Doha QATAR Nov. 2012.

[BEN2013] S. Benkhider, A.R. Baba-Ali, H. Drias, 'Evolutionary Approaches for the Extraction of Classification Rules', International Journal of Applied Metaheuristics, IGI Global, 2013.

[BER2004] M. Berry and G. Linof, Data mining techniques, for Marketing, Sales and Customer Relationship Management, John Wiley publishing. 2004.

[BRE1984] L.Breimann, J.H. Friedman, R. Olshen and C.J. Stone, Classification And Regression Trees, Wadsworth, Belmont CA. 1984.

[BUD1992] G. Budhinat and D. Derry, A Simple Enhancement to OneR Classification', available at http://goanna.cs.rmit.edu.au/~gjyatil/OtherLinks/Documents/ImprovedOneR Algorithm.

[BUR2003] E. K. Burke, G. Kendal, E. Soubeigha, 'A Tabu Search hyperheuristic for timetabling and rostering', Journal of Heuristics,Vol. 9 N°6, 2003.

[CAN1995] E. Cantu-Paz, 'A survey of parallel Genetic Algorithms', http://neo.lcc.uma.es/Articles/cantu-pazxx.pdf

[CAN2000] E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithms Kluwer Academic Publishers Norwell, MA, USA ©2000

[CHI1996] Chipperfield, A. and Flemming, P., (1996) '*Parallele GAs*', Parallel and distributed computing Handbook,A.Y.H. Zamoya(ed.), MacGrawHill, pp1118-1143. 1996.

[COL2001] M. COLARD, D. FRANCISCI, *Evolutionary* data mining: an overview of genetic based algorithms. 8th IEEE International Conference on Emerging Technologies and Factory; Octobre 2001.

[COW2000] P. Cowling, G. Kendall and E. Soubeiga, A Hyperheuristic Approach to schedululing a sales Summit, PATAT2000, Springer Lecture Notes on Computer Sciences, N° 2079, pp 176-190, Konstanz, Germany, August 2000.

[DAW1976] R. Dawkins, 'The selfishness gene', Editions : Oxford University Press, 1ere Edition: Angleterre 1976

[DAW1989] R. Dawkins, *The Extended Phenotype*. Oxford: Oxford University Press. p. xiii. ISBN 0-19-288051-9.

[DEJ 1991] DeJong, K. A. and Spears, W. M, Learning concept classification rules using genetic algorithms. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 651–656. Morgan Kaufmann, 1991.

[DOR2004] M. Dorigo, T. Stutzle, Ant Colony Optimisation, The MIT Press, Cambridge, Mass, USA, 2004.

[EIB1998] F. Eibben., I.H. Witten, Generating Accurate Rule Sets without Global Optimization, in Proceeding of the Fifteenth International Conference on Machine Learning ICML'98, pp 144-151. 1998.

[FAL2007] I. De Falco, A. Della Ciopa, E. Tarantino, 'Facing classification problems with particle swarm optimization', Applied Soft Computing, vol. 7, N°3, pp 652-658, 2007.

[FOG1966] L. Fogel, Journal of *Intelligence Through Simulated Evolution*, evolutionary computation, published by John Wiley, 1966.

[FRA2003] D. Francisci; L. Brisson, M. Collard, Extraction des règles selon des critères multiples l'art du compromis ; Rapport de recherche interne, Université de Nice,FR, Mai 2003.

[FRA1998] E. Frank and I.H. Witten, 'Generating Accurate Rule Sets without Global Optimization', *in Proceeding of the Fifteenth International Conference on Machine Learning ICML'98,* pp 144-151. 1998.

[FRE1998] A.A.Freitas, A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery, Internal report of the university 1998.

[FRE1999] A.A.Freitas et S.H. Lavington, Mining Very Large Databases with Parallel Processing , Boston: Kluwer Academic, 1998.

[FRE2000] A.A. Freitas. Understanding the crucial differences between classification and discovery of association rules, a position paper. ACM SIGKDD Explorations (ACM 2000), pp 65-69, 2000.

[FRE2008] A.A. Freitas, N. Holden, A Hybrid PSO/ACO Algorithm for discovering Classification Rules in Data Mining, Journal of Artificial Evolution and Applications, Hindawi Pub. Corporation,ArticleID 316145, 2008.

[GOL1989] D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning. Reading,* MA AddisonWesley, 1989.

[HAN2011] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, Morgan Kauffman, 2011.

[HAU2004] R. L. Haupt & Sue Ellen Haupt, Practical genetic algorithms, 2nd ed. John Wiley & Sons, Inc., New Jersey 2004.

[HPY2000] J.Han, J. Pei, and Y. Yin.Mining frequent patterns without candidate generation. In *Proc. of ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

[HOL1962] John Holland, Outline for a logical theory of adaptive systems. Journal of the Association of Computing Machinery, 3, 1962.

[HOL1975] J. Holland, Adaptation in Natural & Artificial Systems - An Introductory Analysis with Application to Biology, Control & A.I , 1ere Edition : Université Ann Harbor, Michigan, 1975.

[HOL1992] J. Holland, 'Adaptation in Natural and Artificial Systems', MIT PRESS. 1992.

[HOL1993] R. Holte, Very Simple Classification Rules Perform Well on Most Commonly used data sets, Machine Learning, Kluwer Academic Publisher, Boston, Vol. 11, pp.63-91, 1993.

[HON1996] L. Hongjun, R. Setiono, Effective Data Mining Using Neural Networks, IEEE Transactiuon on Knowledge and Data Engeneering, VOL 0, No 6, December 1996.

[HOR2007] F. Hornick, E. Marcadé and S. Venkayala, Java Data mining: strategy, standard and practice, Morgan Kaufmann publishers, 2007.

[HUA1995] S.Huan, T. Teck. X2R: A Fast Rule Genarator. Proceeding of the International Conference on Systems, Man and Cybernetics. University of Singapore; IEEE, 1995.

[IGL2006] B. de la Iglesia, G. Richards, M.S. Philpott, V.J. Rayward-Smith, 'The application and effectiveness of a multi-objective metaheuristic algorithm for partial classification', European Journal of Operational Research, pp 898-917, 2006.

[JAN1991] C. Janikow , Indutive Learning of Decision Rules in Attribute Based Examples: a knowledge intensive GA Approach, Phd thesis, University of North Carolina 1991.

[JES 2003] Jesus S.Aguilar-Ruiz, J.C. Riquelme and M.Toro. *Evolutionary Learning of Hierarchical Decision Rules*, IEEE Transaction on Systems, Man, and Cybernetics, Part B: Cybernetics, ISSN:1083-4419, vol. 33, no. 2, pp.324-331, 2003.

[KAS1980] V.G. Kass, An exploratory Technique for Investigating Large Quantities of Categorical Data'. In Journal of Applied Statitics, Vol. 29, N°2, pp. 119-127. 1980.

[KEN2001] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kauffman/Academic Press, San Fransisco, CA, USA, 2001.

[KHA2006] M. Khabzaoui, Modélisation et résolution multi-objectifs des règles d'association : Application à l'analyse de données biopuces. Thèse de Doctorat du LIFL, de l'USTL, Lille 2006.

[KOD1999] Y. Kodratoff, « Knowledge Discovery in Texts: A definition and Applications », Proceeding of the International Symposium on Methodologies for Intelligent Systems, Volume LNAI 1609, pages 16-29,1999.

[KOH1995] R. Kohavy, The power of decision tables, Machine Learning 1995: 8th European Conference on Machine Learning, Greece, Springr Verlag 1995.

[KRA2002] N. Krasnogor, 'Studies on the Theory and Design Space of Memetic Algorithms', PhD dissertation, Faculty of Computing, Engineering and Mathematical Sciences, University of the West of England, Bristol, june 2002.

[KRA2005] N. Krasnogor and J.E. Smith. "A tutorial for competent memetic algorithms: model, taxonomy and design issues ". IEEE Transactions on Evolutionary Computation, 9(5):474- 488, 2005.

[MAN1991] B. Manderick & Al, 'A Genetic Algorithm and the structure of a fitness landscape', In : 14th Conference on Genetic Algorithms, pp 143-150, Morgan Kauffman 1991.

[MAR2008] M. Marinaki, Y. Marinakis, C. Zopounidis, Honey Bees Mating Optimisation for financial classification problems, Applied Soft Computing , Elsevier, Science Direct, 2010.

[MER1998] P. Merz and B. Freisleben, 'Memetic Algorithms and Fitness landscape of the graph Bi-partitionning Problem', In: A.E. Eiben , T. Back, M? Schoenauer, and H.P. Schwefel Editors, Proc. Of the 5th Conference on Parallel Problem solving, from Nature, PPSNV, vol. 1498 of LNCS, pp 765-774, Springer 1998.

[MER1999] P. Merz and B. Freisleben, *fitness landscapes and memetic algorithm design, in: New Ideas in Optimization*, D. Corne, M. Dorigo and F.Glover, eds, McGraw Hill, London, 1999.

[MER2000] P. Merz, Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscape and Effective search strategies', PhD Thesis, Parallel Systems Research Group, University of Siegen, 2000.

[MIC1994] Z. Mickalewich, "Genetic Algorithms + Data Structures = evolution program", London : Kluwer Academic publishers, 1994.

[MIT 1997] T. Mitchell, *Machine Learning*. New York: McGraw Hill, 1997.

[MOR1996] Computer Architecture, A quantitative approach, Morgan Kaufmann Publishers,Inc, second edition 1996.

[MOS1989] P. Moscato, *On evolution search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Calteh Concurrent Computation Program, C3P Report 826, 1989.

[MOS2000] P. Moscato, C. Cotta, "A Gentle Introduction to Memetic Algorithms, in F. Glover and G. Kochenberger (eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, in preparation 2000.

[MUH1989] H. Muhleinben, « Parallel GA, Population Genetics and Combnatorial Optimization », in Proceeding of the 3rd Int. Conf. On Genetic algorithms, Schaffer Ed,pp416-421, Morgan Kauffman 1989.

[PAR2002] R.S Parpinelli, H.S Lopes and A.A Freitas "Data mining with an Ant colony optimization algorithm", IEEE trans. on Evolutionary Computation, vol (6), pp 321-332, 2002.

[PBTL1999] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proc. 7th Int. Conf. Database Theory (ICDT'99)*, pages 398–416, Jerusalem, Jan. 1999.

[QUI1993] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.

[RAB1996] A.A. Rabow, H.A. Sheraga, 'Improved Genetic Algorithm for the Protein Folding Problem by use of a cartesian combination operator', Protein Science, Vol5, pp1800-1815, 1996.

[RAD1996] N.J. Radcliffe & Al, 'Formal Memetic Algorithms', In: T. Fogarti Editors, Evolutionary Computing, AISB workshop, vol 865 of Lecture Note in Computer Sciences, pp 1-16, Springer Verlag, Berlin, 1996.

[REC1973] J. Rechenberg, Evolutions strategies - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog.

[RIS1978] J. Rissanen, Modeling by shortest data description, Automatica,Vol 14, pp 465-471, 1978.

[SMI1980] J. Smith, A learning system based on genetic algorithms. PhD dissertation, University of Pittsburgh. USA, 1980.

[SMI1998] J. Smith, 'Self Adaptation in Evolutionary Algorithms', Phd Thesis, University of the West of England, 1998.

[SMI2003] J. E. Smith, *Co-evolving Memetic Algorithms : A learning approach to robust scalable optimization*, IEEE Congress on Evolutionary Computation CEC2003, IEEE Press, pp. 498-505, 2003.

[SOU2004] T. Sousa, A. Silva, A. Neves, 'Particle Swarm based data mining algorithms for classification tasks', parallel Computing, vol. 30, N° 5-6, pp767-783, 2004.

[TAN2006] K.C. Tan , Q; Yu et J.H. Ang, 'A Dual-Objective Evolutionary Algorithm for Rules Extraction in Data Mining', in Computational Optimization and Applications, vol34, pp. 273-294, Springer: Netherland. 2006.

[TAN2008] K.C. Tan, J.H. Ang, A.A. Mamun, "A memetic Evolutionary Search Algorithm with variable length chromosome for rule extraction", Proceeding of the IEEE International Conference on Systems, Man and Cybernetics SMC 2008.

[TAN2010] K.C. Tan, J.H. Ang, A.A. Mamun, An evolutionary memetic algorithm for rule extraction, Journal of Expert Systems with Applications, Elsevier,pp 1302-1315, 2010.

[TUF2005] S. Tufféry, Data Mining et statistique décisionnelle: l'Intelligence dans les bases de données. Editions TECHNIP, Paris, 2005

[UCI1998] C.L. Blake, C.J Merz, Répertoire de bases de données benchmarks pour le data mining, disponible à www.uci.edu/~mlearn/MLRepository.html., Irvine, CA University of California, Department of Information and Computer Science, 1998.

[VBG1992] V. N. Vapnik, B. Boser and I. Guyon. A training algorithm for optimal margin classifiers. In *Proc. Fifth AnnualWorkshop on Computational Learning Theory*, pages 144–152, ACM Press: San Mateo, CA, 1992.

[VEN1993] G. Venturini, SIA: *A supervised inductive algorithm with genetic search for learning attributes based concepts*, in Proc. Eur. Conf. Machine Learning, pp. 281–296, 1993.

[WAI2003] Wai-Ho Au, K. C.C Chan and Xin Yao,"A novel evolutionary data mining algorithm with application to churn prediction", in IEEE trans. on Evolutionary Computation, pp 532-545, December 2003.

[WEI1991] Weiss SM., Kulikowski CA.: Computer Systems that learn, Classification and Prediction methods from Statistics, Neural Nets, Machine Learning and Expert Systems . Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[WIT2005] I.H. Witten et F. Eibb, Data mining: practical Machine Learning Tools and Techniques with JAVA Implementations, Morgan Kaufman Publishing. 2005.

[WON2000] M.I. Wong, K.S. Leung, Data Mining using Grammar Based Genetic Programming and Applications, Kluwer Academic Publisher, London, 2000.

[ZAK2000] M. J. Zaki. Scalable algorithms for association mining. Proc. of *IEEE Trans. Knowledge and Data Engineering*, 12:372–390, 2000.

Résumé

Le datamining ou forage de données consiste à extraire automatiquement des connaissances cachées ou des modèles à partir de données du monde réel où la connaissance découverte est idéalement précise, compréhensible et intéressante pour l'utilisateur. C'est un domaine interdisciplinaire qui comporte plusieurs tâches et techniques. Dans cette thèse, nous focalisons nos recherches sur l'extraction de règles de classification. Notre objectif dans cette thèse est de montrer l'apport des metaheuristiques pour ce problème combinatoire NP_Difficile. Pour cela, nous avons mis en œuvre plusieurs approches évolutionnaires : génétique pure, mémétiques et parallèles où un nouveau schéma parallèle de l'AG a été proposé. Toutes nos approches ont été testées sur des « benchmarks » de data mining de l'Universié de Californie Irvine (UCI) et ont montré une grande efficacité en terme de temps d'exécution et en terme de précision et compréhensibilité des modèles obtenus. Nos approches ont été concluantes, comparées à d'autres approches basées sur les metaheuristiques pour la résolution du même problème.

Summary

Data mining is to automatically extract hidden or models from real-world data where the discovered knowledge is ideally accurate, understandable and interesting to the user. It is an interdisciplinary field that includes several tasks and techniques. In this thesis, we focus our research on the extraction of classification rules . Our goal in this thesis is to show the contribution of metaheuristics for this combinatorial NP_Hard problem . For this, we have implemented several evolutionary approaches: genetic, memetic and parallel approaches where a new parallel scheme of GA was proposed. All approaches were tested on data mining "benchmarks" of the University of California Irvine (UCI) and they showed a great efficiency in terms of running time and in terms of accuracy and comprehensibility of the resulting models. Our approaches were successful, compared to other approaches based on metaheuristics for solving the same problem.