N° D'ORDRE : 12/2011-M/IN

République Algérienne démocratique et Populaire

Ministère de l'Enseignement Supérieur et

de la Recherche Scientifique

Université des Sciences et de la Technologie

« HOUARI BOUMEDIENE »

Faculté d' Electrique et d'Informatique

MEMOIRE

Présentée pour l'obtention du diplôme de Magister

En Informatique

Spécialité : Intelligence artificielle

Par Mr Dahmri Oualid

SUJET

UNE APPROCHE MÉMÉTIQUE PARALLÈLE POUR L'EXTRACTION DES CONNAISSANCES

Soutenu publiquement le 10/07/2011, devant le jury composé de :

Mr H. Azzoune	Professeur (USTHB)	Président
Mr. Ahmed Riadh BABA-ALI	Maitre de conférences (USTHB)	Directeur de mémoire
Melle D. Boughaci	Maitre de conférences (USTHB)	Examinateur
Mr A. Boukra	Maitre de conférences (USTHB)	Examinateur

Sommaire

Chapitre III : Les algorithmes évolutionnaires

Chapitre IV : Parallélisation des algorithmes évolutionnaires

Chapitre V : Conception

Chapitre VI : Expérimentations

Remerciements

Je tiens à remercier, tout d'abord : Mr H. Azzoune, Melle D. Boughaci et Mr A. Boukra pour nous avoir fait l'honneur d'être membres du jury. Ainsi que pour avoir consacré une partie de leur temps précieux pour lire et corriger ce mémoire.

Je souhaite remercier monsieur Baba Ali, pour avoir accepter de me suivre dans l'élaboration de ce mémoire. Je lui suis particulièrement reconnaissant pour sa patience, son aide, sa disponibilité et ses nombreuses suggestions qui ont amélioré ce travail.

Je remercie également madame Baba Ali pour m'avoir fait bénéficier de ses connaissances, ainsi que pour ses conseils tout au long de ce travail.

Je me dois également de remercier mes parents qui, par leurs encouragements et leur soutien, m'ont permis d'aller avant dans mes projets.

Finalement, mes remerciements les plus sincères vont à ma femme Ahlem, pour sa patience malgré mes horaires de travail imprévisibles et mes nombreuses nuits blanches.

Je remercie tous ceux qui ont aidé de loin ou de près dans la réalisation de ce mémoire.

Merci à tout le monde.

Introduction générale

De nos jours, les bases de données, aussi bien commerciales que scientifiques, sont devenues gigantesques. Leur taille se compte le plus souvent en téraoctets (2^{40} octets) voire en petaoctets (2^{50} octets). Par ailleurs, on estime que la quantité de données collectée et numérisée dans le monde double tous les 20 mois. Il est évident qu'une telle quantité de données ne peut être traitée et analysée humainement. Ces entrepôts de données, s'ils ne sont pas correctement exploités, risquent de devenir des cimetières de données inutiles (data tombs).

C'est pour pallier à ce problème que la notion de data mining est née dans les années 1990. C'est un concept qui regroupe plusieurs méthodes mathématiques et statistiques d'analyse de données (ACP, régression, classification, etc.) qui sont souvent apparues bien antérieurement, mais qui n'ont jamais été combinées et utilisées de manière à exploiter des bases de données et en extraire des informations cachées. Le data mining a été englobé plus tard dans un processus plus général appelé KDD1 (pour Knowledge Discovery in Databases) dont il ne constitue qu'une des nombreuses étapes.

La classification est l'une des tâches du data mining, Il en existe plusieurs types : classification supervisée, classification floue, classification non supervisée, etc. Nous nous intéresserons dans ce travail à la classification supervisée. Cette dernière elle permet d'affecter un objet à des classes dont le nombre est connu, ayant une signification et une interprétation bien précise.

Le problème de classification est un problème d'optimisation combinatoire et la résolution de ce problème de manière optimale s'avère dans la plupart des cas impossible à cause de son caractère fortement combinatoire. Les méthodes exactes requièrent un effort calculatoire qui croît exponentiellement avec la taille du problème. Alors, des méthodes approchées ont été proposées pour résoudre ce problème en temps raisonnable. Parmi ces méthodes, apparaissent celles dites "Métaheuristiques". Cependant le choix d'une méthode dépend du problème et de l'espace de recherche, dans notre cas la résolution de ce problème de classification fait appel aux algorithmes évolutionnaires : les algorithmes génétiques ou les algorithmes mémétiques, ces derniers étant des algorithmes génétiques hybridés des méthodes de recherches locales.

L'efficacité des métaheuristiques a été démontrée dans plusieurs travaux de recherche fondamentale et appliquée. Le succès de ces méthodes s'explique par un ensemble de

facteurs, notamment par leur potentiel d'adaptation aux différentes contraintes associées à des problèmes spécifiques, par leur facilité d'implantation dans des programmes d'application divers et par la qualité des solutions.

Mais les métaheuristiques demandent un temps de calcul et une quantité de mémoire considérables qui sont étroitement liées à la taille du problème et à l'obtention d'une certaine qualité de solution. De ce fait, ces algorithmes deviennent intéressants à paralléliser. Dans ce contexte, les objectifs de réduction du temps de calcul et de traitement de gros problèmes sont toujours pertinents. À cela s'ajoute l'intérêt d'utiliser le parallélisme pour améliorer la performance des métaheuristiques en termes de qualité de solution obtenue et ce, sans augmenter la charge de calcul.

Dans le cadre de notre travail, nous nous attelons à construire un classifieur en utilisons d'abord l'approche génétique pure, puis l'approche mémétique, ce dernier étant un algorithme génétique hybridé avec une recherche tabou et comme cette hybridation augmente le temps d'exécution, donc il faut améliorer ce temps sans diminuer la qualité des résultats, ce qui fait l'objectif de la parallélisassions du classifieur basé sur l'approche mémétique.

Le mémoire est structuré en six chapitres qui permettent un cadrage progressif du sujet :

- Le premier chapitre présente d'abord le data mining en tant que domaine, puis la classification (et particulièrement la classification supervisée), qu'est une tâche qui nous intéresse. En fin il définit le problème de classification et les méthodes de classification.
- Le deuxième chapitre présente les méthodes d'optimisation combinatoires les plus populaires et chaque méthode et ses propres caractéristiques.
- Dans le troisième chapitre, Nous donnons les principes généraux des algorithmes évolutionnaires et nous détaillons les algorithmes génétiques et les algorithmes mémétiques qui peuvent être une bonne solution pour résoudre des problèmes d'optimisation combinatoire.
- Dans le quatrième chapitre nous présentons la parallélisation des métheuristiques et nous détaillons les trois principales stratégies de parallélisations des algorithmes génétiques : Parallélisation de forme maître-esclave sur une population unique, parallélisation de grain fin sur une population unique (Modèle de diffusion) et la parallélisation de gros grain sur des populations multiples (Modèle de migration).

- Le cinquième chapitre présente en détail notre conception commençant par l'adaptation de l'algorithme génétique, la recherche Tabou et l'algorithme mémétique à la classification pour passer après à la présentation de l'architecture parallèle utilisée.
- Le sixième chapitre, fait appel à une démarche expérimentale afin de montrer l'efficacité des approches utilisées dans la résolution du problème étudié. Avec l'outil développé, plusieurs expérimentations sont effectuées sur des Benchmarks retenus comme échantillon de test. Les résultats obtenus sont discutés et analysés afin de faire ressortir certaines conclusions importantes.
- Finalement, un bilan et des perspectives de recherche issues de ce travail sont présentés en conclusion.

1. Data Mining et KDD

1.1. Introduction

Aujourd'hui, le data mining prend de plus en plus d'importance dans l'entreprise. La cause en est, d'un côté, la quantité de données collectées et de l'autre les exigences du marché. On peut citer par exemple la société Wal-Mart (la plus grande chaîne de supermarchés aux Etats-Unis) qui effectue et enregistre en moyenne 21 millions de transactions par jour dans ses bases de données [CIO 07] et qui doit faire face à la concurrence en cernant au mieux le profil et le comportement de ses clients. Ces deux facteurs obligent les dirigeants à recourir à des outils puissants et le plus automatisés possible afin d'extraire les informations cachées qui seront utiles à la prise de décisions stratégiques et commerciales. Le data mining est donc né du regroupement de méthodes mathématiques et statistiques qui ont pour but de satisfaire ces exigences. Mais, ne pouvant être appliqué que sur un format de données bien précis, le data mining doit être précédé par de nombreuses étapes de sélection et de nettoyage des données. Il est ainsi englobé dans un processus plus large, appelé KDD (*Knowledge Discovery in Databases*), qui démarre des données brutes et aboutit à l'intégration des connaissances acquises dans le processus opérationnel de l'entreprise.

1.2. Définitions

1.2.1. Quelques définitions préliminaires [BOU 07]

Donnée : Valeur d'une variable pour un objet (ex : l'âge d'un patient.)

Information : Résultat d'Analyse sur les données (ex : âge moyen des patients.)

Connaissance : information utile pouvant aider à la prise de décision (ex : telle catégorie d'âge est exposée plus souvent à telle affection.

1.2.2. Définition du data mining

Plusieurs définitions ont été proposées, parmi celles qui font référence :

« Le data mining est l'exploration et l'analyse de grandes quantités de données afin de découvrir des règles et des formes significatives, en utilisant des moyens automatiques ou semi-automatiques. » [BER 97].

« Le data mining est un processus d'analyse fine et intelligente des données détaillées, interactif et itératif, permettant aux managers d'activités utilisant ce processus de prendre des

décisions et de mettre en place des actions sur mesure dans l'intérêt de l'activité dont ils ont la charge et de l'entreprise pour laquelle ils travaillent.» [NAÏ 01].

« Le data mining est un processus non trivial d'identification de modèles inconnus, valides, utiles et compréhensibles dans les bases de données.» [NAÏ 01], [BOU 07] et [CIO 07].

On peut en ressortir quelques mots clés en les expliquant et en soulignant leur importance :

Non trivial : en effet, le processus n'est jamais tout à fait automatisé, les experts interviennent forcément à un moment ou à un autre, ne serait-ce que pour choisir les outils et les méthodes et spécifier les buts recherchés.

Inconnus : les modèles et les relations extraits sont inconnus. Sinon le processus ne servirait à rien.

Ex : Prenons l'exemple d'un processus de data mining appliqué à une base de données médicale et ayant pour but de découvrir la relation entre certaines caractéristiques des patients et certaines affections cardiaques qui les touchent. Si ce processus permet de découvrir une relation déjà connue par les cardiologues et bien établie (par exemple que le taux de cholestérol a une incidence directe sur le risque d'infarctus), il est inutile. Les connaissances acquises doivent donc être nouvelles et non établies.

Valides : les modèles découverts doivent êtres valides, c'est-à-dire vérifiés et confirmés par des outils statistiques et des experts dans le domaine.

Utiles : les modèles découverts doivent avoir une utilité, c'est-à-dire qu'ils doivent permettre une application dans le domaine opérationnel.

Ex : considérons encore notre base de données médicale avec le même processus et le même cardiologue et supposons qu'on est dans le cadre d'un programme de prévention. Si une relation est établie entre une caractéristique et une affection, mais que cette caractéristique n'est vérifiable qu'une fois le patient décédé, cette connaissance est inutile car inapplicable dans un but préventif.

Compréhensibles : il faut que la connaissance acquise soit simplement exprimable, compréhensible et interprétable. Par exemple sous la forme d'une règle (SI fonction de caractéristiques ALORS telle affection).

On peut également revenir sur deux notions dans les deux premières définitions. Car bien que la troisième définition soit complète concernant les propriétés des méthodes et des résultats souhaités, les deux premières donnent des propriétés intéressantes concernant les données à

analyser. A. Berry et G. Linoff [BER 97] qui parlent de « grandes quantités de données », cette notion est importante, car elle souligne la nécessité de recourir à des méthodes à complexité réduite pour pouvoir être appliquées. La seconde définition nous parle de « données détaillées » pour permettre une analyse « fine et intelligente » et atteindre plus facilement les objectifs fixés dans la troisième définition.

1.3. Le KDD (Knowledge Discovery in Databases)

Le KDD est né de la nécessité d'englober et de compléter le data mining pour que le processus soit le plus automatisé possible (sans pour autant l'être totalement.)

1.3.1. Définition du KDD

« Le KDD est un processus non trivial d'identification de modèles inconnus, valides, utiles et compréhensibles dans les bases de données ou d'autres sources de données. Il consiste en plusieurs étapes (dont l'une est le data mining). Chacune de ces étapes a pour but d'effectuer une tâche participant à la découverte de connaissances et en utilisant des méthodes spécifiques. Le KDD inclut toutes les étapes d'extraction de connaissances, à partir de l'acquisition et de l'accès aux données jusqu'à la visualisation et l'interprétation des résultats. » [CIO 07].

1.3.2. Les différents modèles de KDD

Il existe cinq principaux modèles de KDD qu'on peut répartir en trois catégories :

A. Les modèles académiques

Les premiers modèles de KDD ont émergé de la recherche au milieu des années 1990. Il en existe deux principaux : le modèle de Fayyard et al. en neuf étapes développé en 1996 [FAY 96] et le modèle d'Anand et Buchner en huit étapes développé en 1998 [ANA 98]. Nous allons présenter les étapes du premier qui est construit selon un modèle séquentiel avec possibilité de retour en arrière.

Etape 1 – Développer et comprendre le domaine d'application **:**

Cette étape consiste principalement à comprendre le domaine d'application et à spécifier les buts et les objectifs recherchés.

Etape 2 – Sélectionner les données cibles **:**

Cette étape consiste à sélectionner les données et les variables qui vont servir au processus.

Etape 3 – Nettoyage et prétraitement des données **:**

Cette étape consiste à nettoyer les données, c'est-à-dire supprimer les données bruitées (c'est-à-dire des données indésirables qui faussent les résultats) et à remplir les champs qui sont incomplets, soit par duplication (dupliquer la donnée en mettant à chaque occurrence une des valeurs possibles du champ vide), soit par suppression du champ concerné.

Etape 4 – Réduction et projection de données **:**

Cette étape consiste à chercher une représentation invariante des données par des projections et des transformations (par exemple en pondérant certaines variables ou en réduisant la dimension des données).

Etape 5 – Choisir les tâches du Data Mining à effectuer **:**

Ici l'expert choisit la tâche appropriée aux buts établis pendant l'étape 1 (Ex : classification, régression, estimation, etc.)

Etape 6 – Choix de l'algorithme et du modèle **:**

Cette étape consiste à choisir la méthode à appliquer pour accomplir la tâche fixée dans l'étape précédente.

Etape 7 – fouilles de données (Data Mining) :

Cette étape correspond au Data Mining. C'est-à-dire à l'application des algorithmes choisis pour accomplir la tâche que l'on s'est fixé et découvrir des modèles inconnus.

Etape 8 – Interprétation du modèle :

Cette étape consiste à visualiser et interpréter le modèle découvert.

Etape 9 – Intégration des connaissances acquises :

La dernière étape consiste donc à intégrer les connaissances dans le processus opérationnel ou décisionnel de l'entreprise.

B. Les modèles industriels

Succédant aux modèles académiques, plusieurs modèles industriels (c'est-à-dire développés au sein même de l'environnement industriel) ont vu le jour. Les deux plus connus étant le modèle en 5 étapes de Cabena (au sein d'IBM) [CAB 98] et le modèle CRISP-DM en 6 étapes développé par un consortium d'entreprises (DaimlerChrysler, OHRA, NCR, etc.).

Le modèle CRISP-DM (pour CRoss-Industry Standard Process for Data Mining) est le plus utilisé en Europe actuellement. Il contient 6 étapes dont certaines permettent un retour en arrière. Les étapes se divisent elles-mêmes en sous étapes. Ce modèle se caractérise par une grande simplicité et une facilité d'application dues à son développement dans un milieu industriel.

C. Les modèles hybrides

Après l'avènement des modèles purement académiques ou industriels, des méthodes hybrides (c'est-à-dire développées dans le milieu universitaire et industriel en parallèle) ont vu le jour. Le plus connu est le modèle de Cios et al. en 6 étapes [CIO 07]. Ce modèle se base principalement sur le modèle CRISP-DM avec quelques modifications orientées vers la recherche académique.

1.4. Les tâches du Data Mining

Il existe quatre principales tâches en Data mining [NAÏ 01] :

1. La classification non supervisée (ou segmentation1.)
2. L'association.
3. La prédiction.
4. La classification supervisée.

On peut diviser les tâches du data mining en deux catégories : les tâches descriptives et les tâches prédictives .

1.4.1. Les tâches descriptives

Les tâches descriptives du data mining sont la classification non supervisée et l'association.

A. Classification non supervisée (segmentation)

La classification non supervisée consiste à rechercher des groupes homogènes dans un ensemble de données. Les données appartenant à la même classe doivent être les plus proches entre elles et les données appartenant à des classes différentes doivent être les plus différentes possible. Le nombre de groupes et leur signification ne sont pas connus [NAÏ 01].

Exemple : Diviser un groupe de clients selon plusieurs caractéristiques (nombre d'achat, âge, etc.) en groupes homogènes.

B. L'association

L'association consiste à rechercher les règles de dépendances entre certaines caractéristiques des données, le plus souvent sous forme de règles d'association.

Exemple : L'exemple le plus célèbre est la découverte de la très forte corrélation en l'achat de bières et de couches-culottes dans les supermarchés Wal-Mart. Particulièrement dans le sens « acheter des couches acheter de la bière » [NAÏ 01].

1.4.2. Les tâches prédictives

Les tâches prédictives se distinguent par le fait qu'elles n'ont pas pour but de découvrir des informations cachées, mais plutôt de prédire la valeur de certaines caractéristiques.

A. Classification supervisée

La classification supervisée consiste à prévoir le placement futur d'une donnée dans des classes préexistantes et ayant une signification précise.

Exemple : Classer des tumeurs dans les groupes « bénignes » ou « malignes » à partir d'images ou de certaines caractéristiques (taille, texture, régularité, etc.)

B. La prédiction

Alors que la classification supervisée consiste à prévoir une classe à laquelle on va affecter un objet, c'est-à-dire estimer une variable qualitative, la prédiction (ou estimation) consiste à estimer la valeur d'une variable quantitative.

Exemple : estimer le volume des ventes dans les six prochains mois.

1.5. Conclusion

Le data mining est une technologie puissante permettant d'extraire des connaissances pertinentes et utiles de grandes quantités de données. Les tâches du data mining sont accomplies grâce à des méthodes mathématiques, statistiques et logiques qui font encore aujourd'hui l'objet de recherches avancées. La classification supervisée est une des tâches du data mining, elle permet d'affecter un objet à des classes dont le nombre est connu, ayant une signification et une interprétation bien précise. L'objectif de cette partie a été de présenter le data mining en tant que domaine et d'y situer la classification (et particulièrement la classification supervisée), tâche qui nous intéresse. La partie suivante lui sera consacrée en détail.

2. Le problème de classification [JAI 88]

D'un point de vue général, les méthodes de classification ont pour but de regrouper les éléments d'un ensemble $X = \{X1,..., X\,n\,,..., X\,N\}$ en un nombre K optimal de classes selon leurs ressemblances. En effet, les objets sont souvent répertoriés par rapport à des catégories ou des classes auxquelles ils sont censés appartenir. Cette appartenance est, la plupart du temps, vague et/ou graduelle.

De manière générale, les problèmes de classification s'attachent à déterminer des procédures permettant d'associer une classe à un objet (individu). Ces problèmes se déclinent essentiellement en deux variantes: la classification dite " *supervisée* " et la classification dite " *non supervisée* ".

2.1. Les différents types de classification

Il existe plusieurs types de classifications : supervisée, non supervisée, hiérarchique, partitionnement, floue, exclusive, etc. **[JAI 88]** a proposé une taxonomie qui permet d'organiser ces types de manière claire et hiérarchique (Voir Figure 1)

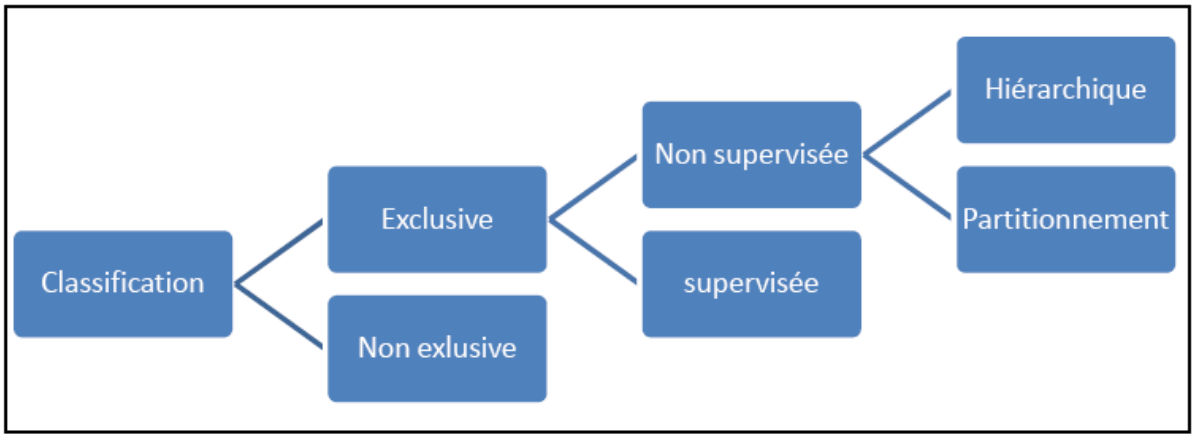


Figure 1 : Taxonomie de la classification.

Classification non exclusive (ou classification floue) : La classification non exclusive est non pas l'affectation d'un objet à une classe unique, mais plutôt le calcul de la probabilité d'appartenance de l'objet à chaque classe. La somme des probabilités d'appartenance d'un objet aux différentes classes doit être égale à 1.

Classification exclusive : La classification exclusive consiste à affecter chaque objet à une classe unique. Il en existe deux types :

Classification supervisée : La classification supervisée consiste à affecter un objet à des classes dont le nombre est connu, ayant une signification et une interprétation bien précise et disposant d'une population de base. La classification supervisée est, en général, issue d'un apprentissage avec des données initiales qu'un expert va classer et qui serviront de modèles aux futurs objets classés.

Classification non supervisée : La classification non supervisée consiste à affecter un objet à des classes dont le nombre est inconnu et qui n'ont aucune signification particulière à priori. Il s'agit donc d'un regroupement de données par similitude dans des classes homogènes. Il en existe deux principaux types :

Classification hiérarchique : Cette classification est une hiérarchie de classe qui va de la simple donnée jusqu'à la classe contenant l'ensemble des données. Elle est exprimée par un arbre construit sur les données appelées dendrogramme.

Classification Par partitionnement : La classification par partitionnement cherche à trouver une partition de l'espace de départ tel que les données appartenant à un même groupe soient plus similaires entre elles qu'avec les données issues d'un autre groupe.

2.2. Les méthodes de classifications supervisées [Bac 04]

Les méthodes de classification ont pour but d'identifier les classes auxquelles appartiennent des objets à partir de certains traits descriptifs. Les différentes méthodes de classification supervisée sont les algorithmes génétiques, les réseaux de neurones, les arbres de décision etc.

2.2.1. Arbres de décision

1. Définition

C'est une structure de données utilisée comme modèle pour la classification, c'est l'une des formes les plus simples d'apprentissage qui connait le plus de succès. Aussi c'est une méthode récursive basée sur le principe de diviser-pour-régner pour créer des sous-groupes (plus) purs (un sous-groupe est pur lorsque tous les éléments du sous-groupe appartiennent à la même classe). On peut dire alors que l'arbre de décision est une représentation graphique d'une procédure de décision, qui permet de modéliser simplement, graphiquement, rapidement un phénomène mesuré plus ou moins complexe, et enfin de construire le plus petit arbre de décision possible.

2. Représentation

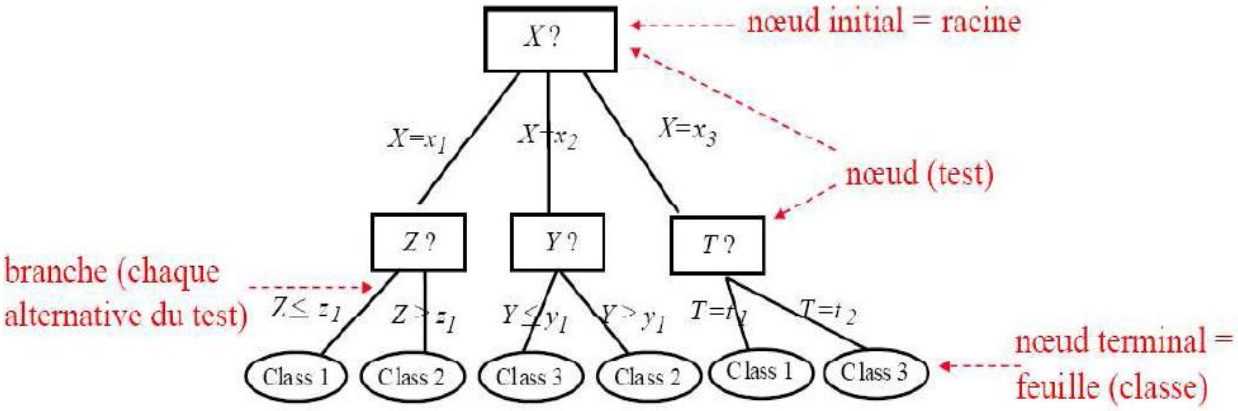
Chaque nœud = Test sur un attribut.

Une branche correspond à une valeur possible d'un attribut.

Les feuilles désignent la classe de l'objet à classer, c'est-à-dire à une decision.

Problèmes: Choix de l'attribut, terminaison.

Exemple



Le ***nœud racine*** représente toute la population, les ***autres nœuds*** représentent le sous-ensemble de la population, chaque nœud est divisé en plusieurs nœuds fils, selon la valeur d'un attribut. Chaque ***feuille*** représente un nœud indivisible.

3. Algorithmes utilisées

Les deux algorithmes les plus connus et les plus utilisés sont CART (Classification And Regression Trees) et C4.5 la version la plus récente après l'algorithme ID3. Ces algorithmes sont très utilisés car performants et car ils génèrent des procédures de classification exprimables sous forme de règles.

A. Algorithme de CART

Présentation de CART :

L'algorithme CART (Classification and Regression Trees) développée par Breiman et J.H Friedman R.A Olshen et C.J Stone en (1984). Sert à construire un arbre de décision en classifiant un ensemble d'enregistrements. Cet arbre fournit un modèle pour classer de nouveaux échantillons. L'arbre généré par CART est binaire (un nœud ne peut avoir que 2 fils) et le critère de segmentation est l'indice de Gini.

Principes de CART

● Considérons un ensemble de variables catégorielles X_1,... X_p. Le partitionnement récursif divise l'espace des p variables en n rectangles qui ne se chevauchent pas.

● La division est accomplie récursivement. Par exemple soit la variable Xi et une valeur S_i de cette variable. On trouve que le partitionnement où $X_i < S_i$ et $X_i > S_i$ sépare bien les données en deux ensembles disjoints. Ensuite une des deux parties est à son tour divisée par une valeur de X_i ou par la valeur d'une autre variable. On aboutit alors à 3 rectangles et ainsi de suite.

● L'idée est de créer n rectangles de telle sorte que l'ensemble de données contenu dans un rectangle soit homogène.

On rappelle qu'un arbre binaire est constitué d'un ensemble de nœuds ; de chaque nœud partent 0, 1 ou 2 branches, vers le bas ; on appelle feuille un nœud dont ne part aucune branche. Considérons par exemple l'arbre ci-après, qui a un seul nœud, et deux branches menant à deux feuilles.

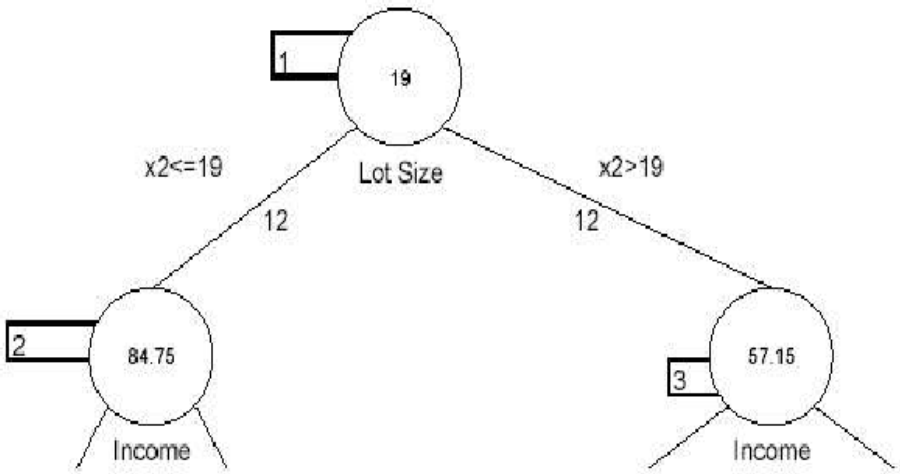


Figure 2 : Exemple d'une arbre binaire.

On lit un tel arbre de la façon suivante : si X < =19 alors on emprunte la branche de gauche, qui mène à une feuille ; à cette feuille est attribuée la valeur 84.75, on obtient donc Y=84.75 . Si au contraire X>19 alors on emprunte la branche de droite ; on lit : Y =57.15. L'arbre établit donc un lien fonctionnel entre Y et X, que l'on pourrait d'ailleurs expliciter au moyen de

fonctions indicatrices. On peut proposer des arbres plus complexes, le principe de lecture est le même.

B. Algorithme d'ID3

Présentation d'ID3

Cette algorithme a été crée par J. Ross Quinlan(1993). Il est utilisée pour construire un arbre de décision, étant donnée un ensemble d'attributs non cibles C1, C2, ..., Cn, l'attribut cible C, et un ensemble S d'enregistrements d'apprentissage.

Principe d'ID3

ID3 construit un arbre de décision de façon récursive en choisissant l'attribut qui maxime le gain d'information selon l'entropie de Shannon. Cet algorithme fonctionne exclusivement avec des attributs catégoriques et un nœud est créé pour chaque valeur des attributs sélectionnés.

Entropie de Shannon est calculée comme suit:

$$E(S) = -\sum_{j=1}^{|S|} p(j)\log_2 p(j)$$

Où p(j) est la probabilité d'avoir un élément de caractéristique j dans l'ensemble S.

$$Gain(S,A) = E(S) - \sum^{v}(\frac{|S_v|}{|S|} * E(S_v))$$

Avec :

S est un ensemble d'entraînement.

A est l'attribut cible.

Sv le sous-ensemble des éléments dont la valeur de l'attribut A est v.

|***Sv***| = nombre d'éléments de Sv .

|***S***| = nombre d'éléments de S.

3. Algorithme de C4.5

Principe de C4.5 :

Cet algorithme a été proposé en 1986, par Ross Quinlan, pour pallier les limites de l'algorithme ID3 vu précédemment. Nous n'allons pas tout redévelopper pour décrire C4.5

car il repose complètement sur l'algorithme ID3 que nous avons déjà décrit. Nous nous focaliserons donc d'avantage ici sur les limites de l'algorithme ID3 et les améliorations apportées par C4.5.

Les phases d'algorithme C4.5

Nous supposons prédéfini un ensemble de tests n-aires. Pour définir l'algorithme, nous allons définir les trois opérateurs utilisés par l'algorithme C4.5 pour calculer un bon arbre de décision (phase d'expansion), puis nous verrons la phase d'élagage. On suppose disposer d'un ensemble d'apprentissage A.

Phase d'expansion: Construction de l'arbre de décision en divisant récursivement l'ensemble d'apprentissage A. On utilise la fonction entropie.

Décider si un nœud est terminal :

Un nœud p est terminal si $\mathbf{i_{(p)} <= i_0}$ ou $\mathbf{N_{(p)} <= n_0}$, où $\mathbf{i_0}$ et $\mathbf{n_0}$ sont des paramètres à fixer.

Sélectionner un test à associer à un nœud : Soit p une position et soit T un test n-aire. Si ce test devient l'étiquette du nœud à la position p, alors on appelle $\mathbf{P_i}$ la proportion d'éléments de l'ensemble des exemples associés à p qui vont sur le nœud en position pi (ième fils du nœud en position p). Le gain associé au choix du test T en position p est alors :

$$gain(p,T) = i(p) - \sum_{i=1}^{n} (Pi * i(pi))$$

où i est la fonction entropie. En position p (non maximale), on choisit le test qui maximise la quantité gain(p,T).

Affecter une classe à une feuille : On attribue la classe majoritaire.

Soit t l'arbre obtenu en sortie de la phase d'expansion.

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 (problème du biais inductif), il semble que la méthode donne des résultats corrects.

2.2.2. Réseaux de neurones

Les réseaux de neurones sont composés d'éléments simples (ou neurones) fonctionnant en parallèle. Ces éléments ont été fortement inspirés par le système nerveux biologique. Comme dans la nature, le fonctionnement du réseau (de neurone) est fortement influencé par la connections des éléments entre eux. Les réseaux de neurones formels sont des modèles théoriques de traitement de l'information inspirés des observations relatives au fonctionnement des neurones biologiques et du cortex cérébral. Par analogie aux neurones biologiques, les neurones artificiels ont pour but de reproduire des raisonnements « intelligents». Ces neurones peuvent adopter certaines qualités habituellement propres à leurs équivalents biologiques, entre autres, la généralisation, l'évolutivité, et une certaine forme de déduction.

Un modèle formel :

Les réseaux de neurones formels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur calcule une sortie unique sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau [Tou 92].

La figure 3 montre la structure d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire. Il reçoit un nombre variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associée un poids w abréviation de weignt (poids en anglais) représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals. A chaque connexion est associée un poids.

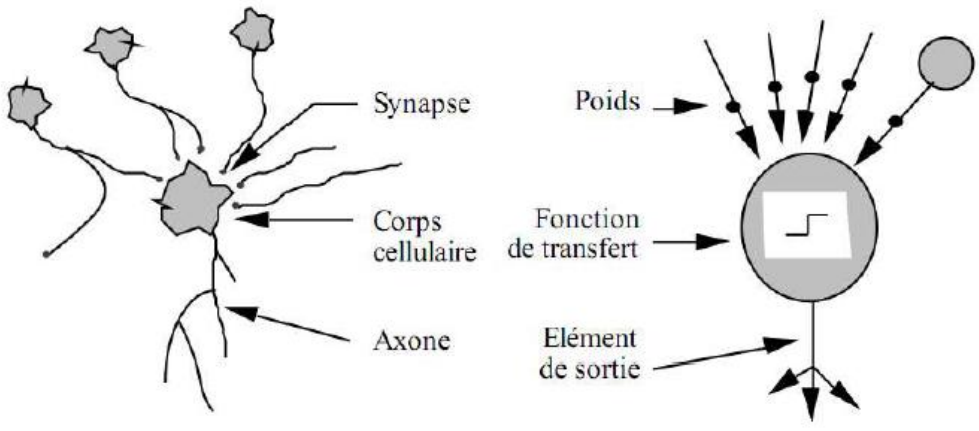


Figure 3 Mise en correspondance neurone biologique / neurone formel.

Comportement d'un neurone artificiel:

Un neurone artificiel est un automate à deux états « actif» et «inactif» : un neurone devient «actif» si la somme de l'ensemble des signaux entrants pondérés dépasse un certain seuil, il devient «inactif» sinon.

On distingue deux phases. La première est habituellement le calcul de la somme pondérée des entrées (a) selon l'expression suivante :

$$a = \sum(w_i . e_i)$$

A partir de cette valeur, une fonction de transfert calcule la valeur de l'état du neurone. C'est cette valeur qui sera transmise aux neurones avals. Il existe de nombreuses formes possibles pour la fonction de transfert. Les plus courantes sont présentées sur la figure 4.

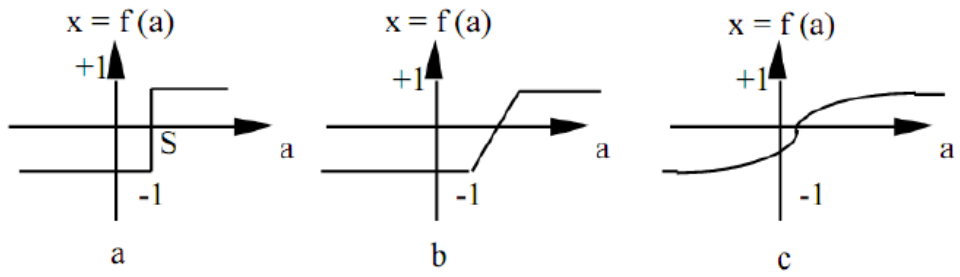


Figure 4 : Fonction à seuil (S, la valeur du seuil), b) Linéaire par

morceaux, c) Sigmoïde

Réseaux de neurones et l'apprentissage supervisé

Les réseaux de neurones se divisent en deux principales classes, les réseaux à apprentissage supervisés (supervised learning) et les réseaux à apprentissage non supervisés (unsupervised learning).

Pour les réseaux à apprentissage supervisés (Perceptron, Adaline, etc.), on présente au réseau des entrées, et au même temps les sorties que l'on désirerait pour cette entrée. Par exemple on lui présente en entrée une lettre " a " manuscrite et en sortie un code correspondant à la lettre " a ". Le réseau doit alors se reconfigurer, c'est-à-dire calculer ses poids afin que la sortie qu'il donne corresponde bien à la sortie désirée.

2.2.3. Les algorithmes génétiques

Le principe de l'algorithme génétique se résume par l'algorithme suivant :

Début

Génération d'une population initiale de k individus

Réputer

1. Sélectionner pour la reproduction
2. Croisement des individus sélectionnés
3. Mutation des individus sélectionnés
4. Sélection pour le remplacement

Jusqu'à (condition d'arrêt)

Retourner la meilleure solution

Fin

Les algorithmes génétiques sont détaillés dans le chapitre 3

2.3. Les systèmes de classifieurs

Il existe deux approches pour les systèmes de classifieurs qui sont l'approche Pittsburgh et l'approche michigan.

2.3.1. Approche Pittsburgh

L'approche Pittsburgh propose un système qui utilise le cycle traditionnel d'un AG, où chaque individu est une solution complète au problème de classification, Une solution complète au problème de classification signifie une disjonction d'un ensemble de règles de classification. Chaque règle a une longueur fixe, mais le nombre de règles de l'ensemble est variable. Les individus concurrencent eux-mêmes pour classifier correctement le maximum d'instances et la population converge vers de bons ensembles de règle.

Il y a deux systèmes de représentation pour cette famille :

- GABIL (DeJong & Spears, 1991).
- GIL (Janikow, 1991).

2.3.2. Approche Michigan

L'approche du Michigan est caractérisée par la création d'un modèle cognitif appelé le système de classeur où les membres de la population sont les différentes règles, et la population entière est la solution au problème de classification. Ceci signifie qu'un mécanisme qui récompense de bonnes règles et pénalise de mauvaises règles est nécessaire. Dans cette approche l'AG n'est pas l'élément central, mais seulement une partie du système employé de temps en temps pour découvrir de nouvelles règles.

2.4. Evaluation d'une règle de classification [AGR 94] [BAB 09]

L'extraction de règles de classification tend à découvrir un petit ensemble de règles qui constituent le classifieur le plus exact possible. Pour cette raison une règle est évaluée par sa qualité de prédiction sur une ensemble de test. On mesure le taux de succès, par le pourcentage d'exemples de l'ensemble de test qui sont bien classé par le modèle. Pour une règle $A \rightarrow C$ ce taux peut s'écrire en terme de probabilité $P(A \cap C) + P(\text{ non } A \cap \text{ non } C)$. Pour mesurer la qualité de classement, la plupart des algorithmes utilisent cette mésure qui reste globale.

La mesure de taux d'erreur, bien que significative, ne Prend pas en compte les performances réelles du classifieur, car elle ne prend pas en compte le déséquilibre qui peut exister dans la distribution des classes dans l'ensemble d'exemples. Aussi les performances des modèles prédictifs font souvent état de mesure sur les ***vrais positif*** et les ***vais négatifs***. Il est courant d'utilisé les notations suivantes pour une classe C et un classifieur [FRE 00]:

TP (ou true positives) : Le nombre d'exemples qui satisfont A et C.

FP (ou false positives) : Le nombre d'exemples qui satisfont A et non C.

TN (ou true négatives) : Le nombre d'exemples qui ne satisfont ni A et ni C.

FN (ou false positives) : Le nombre d'exemples qui ne satisfont pas A mais satisfont C.

Il existe de nombreux critères pour mesurer la qualité d'une règle de classification, dont nous citons quelques uns :

2.4.1. Couverture

La couverture d'une règle R, noté ***Couv(R)*** représente le nombre d'instances dans la base de test B qui vérifie la partie A de la règle R (peut importe la classe).

2.4.2. Spécifité

La spécifité d'une règle R noté ***Sp(R)***, représente le taux de la précision de la règle et cela peut être calculé comme suit :

Sp(R) = TN/(TN+TP) Avec :

FP (ou false positives) : Le nombre d'exemples qui satisfont A et non C.

TN (ou true négatives) : Le nombre d'exemples qui ne satisfont ni A et ni C.

2.4.3. La précision

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

2.5. Complexité du problème de classification

Avant de commencer, nous sommes d'abord amenés à situer la complexité de classification. Le nombre de façons de classer *n* objets dans *k* groupes est donné par Liu [Liu 68] :

$$S(n,K) = \frac{1}{K!}\sum_{i=1}^{K}(-1)^{K-i}\binom{K}{i}i^n \quad \text{où} \quad \binom{K}{i} = \frac{K!}{i!(K-i)!}$$

Par exemple, si nous prenons n=25 et k=5, il y a 2 436 684 974 110 751 façons de répartir les 25 objets dans 5 groupes. Si on ne connait pas le nombre optimal de classes, on doit donc énumérer le nombre de partitions possibles pour K = 1 jusqu'à K = *n* (*n* étant le nombre d'objets à classer.) Donc le nombre de partitions possibles sur un ensemble de *n* objets satisfaisant les contraintes est donné par le nombre de Bell :

$$B(n) = \sum_{k=1}^{n} S(n,k)$$

On peut voir sur le tableau et le graphe suivants la croissance des 10 premiers nombres de Bell.

n	B(n)
1	1
2	2
3	5
4	15
5	52
6	203
7	887
8	4140
9	21147
10	115975

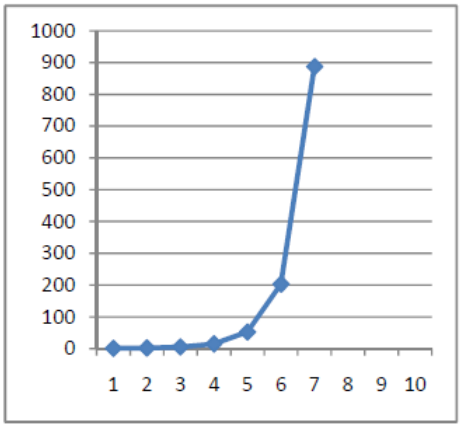


Figure 6 : Tableau et graphe de croissance des nombres de Bell.

Il est évident que la croissance du nombre de partitions possibles est exponentielle.

L'énumération de tous les cas est donc impossible à partir de quelques dizaines d'objets, l'utilisation d'heuristiques et de métaheuristiques est donc justifiée.

2.6. Conclusion

Après avoir défini la classification et les méthodes de classification, Nous avons pu voir que la classification se fait de manières approchée car il n'existe pas de méthodes exactes qui nous donnent des règles pertinentes, utiles et lisibles. Donc, l'utilisation de critères pour l'évaluation des règles est nécessaire. Mais la prise en considération de plusieurs critères en même temps est un problème combinatoire auquel nous devons trouver une solution satisfaisante en un temps raisonnable. Donc par conséquence nous devons étudier les méthodes d'optimisation combinatoires qui sera l'objet du chapitre suivant.

1. Introduction [SIA 05]

Résoudre un problème d'optimisation combinatoire, c'est trouver un optimum d'une fonction, parmi un nombre fini de choix, souvent très grand. Les applications concrètes sont nombreuses, que ce soit dans le domaine de la production industrielle, des transports ou de l'économie – partout où se fait sentir le besoin de minimiser des fonctions numériques, dans des systèmes où interviennent simultanément un grand nombre de paramètres.

2. L'optimisation combinatoire

L'optimisation combinatoire recouvre toutes les méthodes qui permettent de déterminer un optimum d'une fonction, avec ou sans contraintes.

En théorie, un problème d'optimisation combinatoire se définit par l'ensemble de ses instances, souvent très nombreuses. Dans la pratique, le problème se ramène à résoudre numériquement l'une de ces instances, par un procédé algorithmique. A chaque instance du problème est associé un ensemble discret de solutions S, un sous-ensemble X de S représentant les solutions admissibles (réalisables), ainsi qu'une fonction de coût f (appelée aussi *fonction objective*). f assigne à chaque solution $s \in X$ le nombre $f(s)$.

Résoudre un problème d'optimisation combinatoire consiste alors à trouver une solution $s \in X$ optimisant la valeur de la fonction de coût f. Formellement, on cherche donc $s^* \in X$ tel que $f(s^*) \leq f(s)$ pour tout $s \in X$ *(dans un problème de minimisation)*. Une telle solution s^* s'appelle une *solution optimale*, ou un *optimum global*.

On peut tout aussi bien résoudre des problèmes de maximisation, les principes de résolution restant naturellement les mêmes.

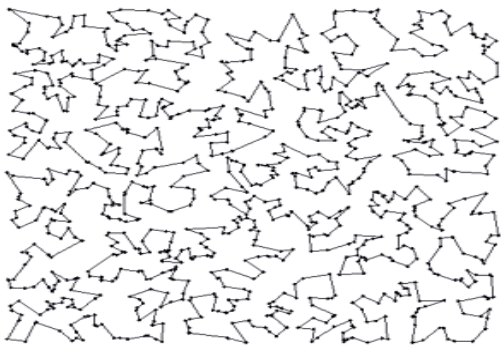


Figure 7 : représentation du chemin le plus court passant par une distribution aléatoire de 1000 points.

Dans le schéma ci-dessus, le problème était de trouver le chemin minimal passant une seule fois par un certain nombre de points Ce graphique correspond à une certaine *solution* du problème. Avec une distribution différente de points, nous aurions eu une autre instance du problème.

3. L'optimisation difficile

L'optimisation combinatoire consiste à trouver la meilleure solution entre un nombre fini de choix. Autrement dit, à minimiser (où à maximiser) une fonction, avec ou sans contraintes, sur un ensemble fini de possibilités. Quand le nombre de combinaisons possibles est exponentiel par rapport à la taille du problème, le temps de calcul devient rapidement critique.

Ce temps de calcul devient si problématique que pour certains problèmes, on ne connaît pas d'algorithme exact *polynomiaux*, c'est-à-dire dont le temps de calcul soit proportionnel à N^n, où N désigne le nombre de paramètres inconnus du problème, et où n est une constante entière. Lorsqu'il n'existe pas une telle constante n telle qu'un polynôme de degré n puisse borner le temps de calcul d'un algorithme, on parle alors d'optimisation difficile, ou de problèmes *NP-difficiles* (c'est tout l'objet de la théorie de la NP-complétude).

4. Les méthodes de résolution des problèmes d'optimisation combinatoire [BLU 03] [HAO 99]

4.1. Méthode exacte et méthode approchée

Une recherche exhaustive par énumération explicite de toutes les solutions est impensable pour résoudre un problème d'optimisation difficile en raison du temps de calcul induit. Dans le cas du problème du voyageur de commerce, par exemple, l'espace de recherche croit en $(n\text{-}1)!$, où n est le nombre de villes à visiter, ce qui dépasse rapidement les capacités de calcul de n'importe quel ordinateur.
Avec seulement 50 villes, il faudra évaluer 49! trajets, soit $6{,}08.10^{62}$ trajets.
C'est l'explosion combinatoire.

Néanmoins, la résolution d'un tel problème d'optimisation peut se faire de manière exacte, en modélisant soigneusement le problème, puis en appliquant un algorithme ad-hoc, qui écarte d'emblée l'examen de certaines configurations, dont on sait d'ores et déjà qu'elles ne peuvent pas être optimales.

Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de *séparation et évaluation* (branch-and-bound), ou les *algorithmes avec retour arrière* (backtracking). Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Mais malgré les progrès réalisés (notamment en matière de programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution augmente exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés avec les applications de taille importante.

Si les méthodes de résolution exactes permettent d'obtenir une ou plusieurs solutions dont l'optimalité est garantie, dans certaines situations, on peut cependant se contenter de solutions de bonne qualité, sans garantie d'optimalité, mais au profit d'un temps de calcul réduit. On utilise pour cela une méthode **heuristique**, adaptée au problème considéré, avec cependant l'inconvénient de ne disposer en retour d'aucune information sur la qualité des solutions obtenues.

4.2. Heuristique et métaheuristique

Afin d'améliorer le comportement d'un algorithme dans son exploration de l'espace des solutions d'un problème donné, le recours à une méthode *heuristique* (du verbe grec heuriskein, qui signifie « trouver ») permet de guider le processus dans sa recherche des solutions optimales. Feignebaum et Feldman (1963) définissent une **heuristique** comme une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre sorte de système qui limite drastiquement la recherche des solutions dans l'espace des configurations possibles. Newell, Shaw et Simon (1957) précisent qu'un processus heuristique peut résoudre un problème donné, mais n'offre pas la garantie de le faire. Dans la pratique, certaines heuristiques sont connues et ciblées sur un problème particulier.

La **métaheuristique**, elle, se place à un niveau plus général encore, et intervient dans toutes les situations qui ne connaît pas d'heuristique efficace pour résoudre un problème donné, ou lorsqu'on estime qu'on ne dispose pas du temps nécessaire pour en déterminer une.

En 1996, I.H. Osman et G. Laporte définissaient la métaheuristique comme « un processus itératif qui subordonne et qui guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales ».

En 2006, le réseau Metaheuristics (metaheuristics.org) définit les métaheuristiques comme « un ensemble de concepts utilisés pour définir des méthodes heuristiques, pouvant être appliqués à une grande variété de problèmes. On peut voir la métaheuristique comme une « boîte à outils » algorithmique, utilisable pour résoudre différents problèmes d'optimisation, et ne nécessitant que peu de modifications pour qu'elle puisse s'adapter à un problème particulier ». Elle a donc pour objectif de pouvoir être programmée et testée rapidement sur un problème. Comme l'heuristique, la métaheuristique n'offre généralement pas de garantie d'optimalité, bien qu'on ait pu démontrer la convergence de certaines d'entre elles. Non déterministe, elle incorpore souvent un principe stochastique pour surmonter l'explosion combinatoire. Elle fait parfois usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.

4.3. Classification des métaheuristiques

Les métaheuristiques n'étant pas, à priori, spécifiques à la résolution de tel ou tel type de problème, leur classification reste assez arbitraire. On peut cependant distinguer :

4.3.1. Les approches à base de trajectoire

Ces algorithmes partent d'une solution initiale (obtenue de façon exacte, ou par tirage aléatoire) et s'en éloignent progressivement, pour réaliser une trajectoire, un parcours progressif dans l'espace des solutions. Dans cette catégorie, se rangent :

- la méthode de descente
- le recuit simulé
- la méthode Tabou
- la recherche par voisinage variable

Le terme de *recherche locale* est de plus en plus utilisé pour qualifier ces méthodes.

4.3.2. Les approches à base de population (ou évolutionnaires)

Elles consistent à travailler avec un ensemble de solutions simultanément, que l'on fait évoluer graduellement. L'utilisation de plusieurs solutions simultanément permet naturellement d'améliorer l'exploration de l'espace des configurations. Dans cette seconde catégorie, on recense :

- les algorithmes génétiques
- les algorithmes par colonies de fourmi
- l'optimisation par essaim particulaire
- les algorithmes à estimation de distribution
- le path relinking (ou chemin de liaison)

Une autre manière, plus intuitive, de classifier les métaheuristiques consiste à séparer celles qui sont inspirées d'un **phénomène naturel**, de celles qui ne le sont pas. Les algorithmes génétiques ou les algorithmes par colonies de fourmi entrent clairement dans la première catégorie, tandis que la méthode de descente, ou la recherche Tabou, vont dans la seconde.

On peut également raisonner par rapport à l'usage que font les métaheuristiques de la **fonction objectif**. Certaines la laissent « telle quelle » d'un bout à l'autre du processus de calcul, tandis que d'autres la modifient en fonction des informations collectées au cours de l'exploration – l'idée étant toujours de « s'échapper » d'un minimum local, pour avoir davantage de chance de trouver l'optimal. La *recherche locale guidée* est un exemple de métaheuristique qui modifie la fonction objectif.

Enfin, il faut distinguer les métaheuristiques qui ont la faculté de **mémoriser** des informations à mesure que leur recherche avance, de celles qui fonctionnent sans mémoire, en aveugle, et qui peuvent revenir sur des solutions qu'elles ont déjà examinées. On distingue la mémoire à court terme (celles des derniers mouvements effectués), et la mémoire à long terme (qui concerne des paramètres synthétiques plus généraux). Le meilleur représentant des métaheuristiques avec mémoire reste la recherche Tabou. Dans les « sans mémoire », on trouve le recuit simulé par exemple.

4.4. Les principales métaheuristiques

4.4.1. La méthode de descente

Le principe de la méthode de descente (dite aussi *basic local search*) consiste à partir d'une solution *s* et à choisir une solution *s'* dans un voisinage de *s*, telle que *s'* améliore la recherche (généralement telle que $f(s') < f(s)$).

On peut décider soit d'examiner *toutes* les solutions du voisinage et prendre la meilleure de toutes (ou prendre la première trouvée), soit d'examiner un sous-ensemble du voisinage.

La méthode de recherche locale la plus élémentaire est la *méthode de descente*. On peut la schématiser comme suit :

Procédure **descente_simple** (solution initiale *s*)

Répéter :

Choisir *s'* dans N(s)

Si $f(s') < f(s)$ alors $s \leftarrow s'$

Jusqu'à ce que $f(s') \geq f(s)$, $\forall s' \in S$

Fin

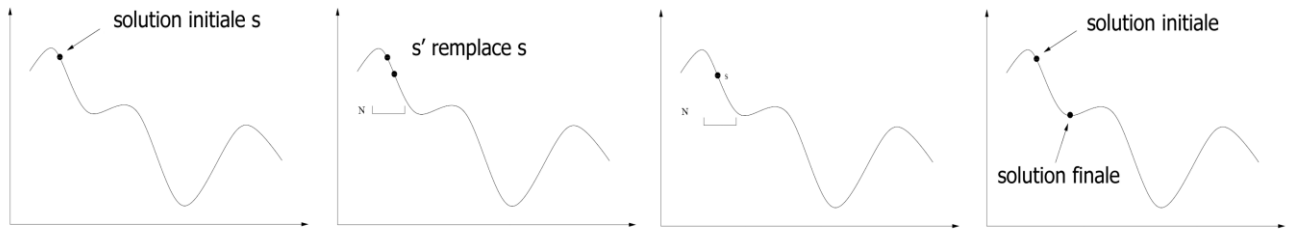


Schéma 3 : évolution d'une solution dans la méthode de descente

On peut varier cette méthode en choisissant à chaque fois la solution *s'* dans N(*s*) qui améliore le plus la valeur de *f*.

4.4.2. La méthode du recuit simulé

Le recuit simulé (simulated annealing) est souvent présenté comme la plus ancienne des métaheuristiques, en tout cas, la première à mettre spécifiquement en œuvre une stratégie d'évitement des minima locaux (Kirkpatrick, Gelatt, Vecchi, 1983).

Elle s'inspire d'une procédure utilisée depuis longtemps par les métallurgistes qui, pour obtenir un alliage sans défaut, chauffent d'abord à blanc leur morceau de métal, avant de laisser l'alliage se refroidir très lentement (technique du recuit). Pour simuler cette évolution d'un système physique vers son équilibre thermodynamique à une température *T*, la méthode du recuit simulé exploite l'algorithme de *Métropolis*.

Dans l'algorithme de *Métropolis*, on part d'une configuration donnée, et on fait subir au système une modification élémentaire. Si cette perturbation a pour effet de diminuer la fonction objectif (ou *énergie*) du système, elle est acceptée. Sinon, elle est acceptée avec la probabilité exp(ΔE/T). En appliquant itérativement cette règle, on engendre une séquence de configurations qui tendent vers l'équilibre thermodynamique.

On peut systématiser l'algorithme avec le pseudo-code suivant :

```
Procédure recuit_simulé (solution initiale s)
Début
    Poser T ← T0
    Répéter :
        Choisir aléatoirement s' ∈ N(s)
        Générer un nombre réel aléatoire r dans [0,1]
        Si r < e^((f(s)−f(s'))/T) alors Poser s ← s'
        Mettre à jour T
    Jusqu'à ce que le critère de terminaison soit satisfait
Fin
```

Et voici l'interprétation de son fonctionnement :

· Si *f(s')<f(s)* alors $e^{\frac{f(s)-f(s')}{T}}$ >1, donc r est toujours inférieur à cette valeur, et on accepte la solution *s'* (une meilleure solution est donc toujours acceptée, ce qui paraît logique).

· Si *f(s')>f(s)* et *T* est très grand, alors $e^{\frac{f(s)-f(s')}{T}} \cong 1$, et on il y a de fortes chances d'accepter *s'* (bien que la solution *s'* soit plus « mauvaise » que *s* !)

· Si *f(s')>f(s)* et *T* est très petit, alors $e^{\frac{f(s)-f(s')}{T}} \cong 0$, et on va donc probablement refuser *s'*

Dans un premier temps, *T* étant généralement choisi très grand, beaucoup de solutions, même celles dégradant la valeur de *f*, sont acceptées, et l'algorithme équivaut à une visite aléatoire de l'espace des configurations. Mais à mesure que la température baisse, la plupart des mouvements augmentant l'énergie sont refusés, et l'algorithme se ramène à une amélioration itérative classique. A température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction objectif. Il laisse ainsi une chance au système de s'extraire d'un minimum local.

Cet algorithme est parfois amélioré en ajoutant une variable qui mémorise la meilleure valeur rencontrée jusqu'à présent (sans cela, l'algorithme pourrait converger vers une certaine solution *s*, alors qu'on avait visité auparavant une solution *s'* de valeur inférieure à *f(s)* !)

L'algorithme du recuit simulé a montré son efficacité sur les problèmes combinatoires classiques, surtout sur les échantillons de grande taille. Par exemple, l'expérience a montré qu'il ne devenait vraiment efficace sur le problème du voyageur de commerce qu'au-delà d'environ 800 villes. Il a été testé avec succès sur le problème du partitionnement de graphe. Des analyses ont montré qu'il était efficace avec certaines catégories de problème où l'ensemble des solutions possèdent certaines propriétés particulières. Ceci expliquerait le succès du recuit simulé dans le domaine du placement des circuits électroniques, où il est très employé.

4.4.3. La méthode Tabou

La méthode Tabou est une technique de recherche dont les principes ont été proposés pour la première fois par Fred Glover dans les années 80, et elle est devenue très classique en optimisation combinatoire. Elle se distingue des méthodes de recherche locale simples par le recours à une mémoire des solutions visitées, de façon à rendre la recherche un peu moins « aveugle ». Il devient donc possible de s'extraire d'un minimum local, mais, pour éviter d'y retomber périodiquement, certaines solutions sont bannies, elles sont rendues « taboues ».

A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine $s' \in N(s)$ à chaque itération, Tabou examine un échantillonnage de solutions de *N(s)* et retient

la meilleure *s'* même si *f(s')>f(s)*. La recherche Tabou ne s'arrête donc pas au premier optimum trouvé.

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.

L'algorithme général peut se représenter avec le pseudo-code suivant :

Si on nomme N_T (s) toutes les solutions qui ne sont pas taboues ainsi que celles qui le sont mais dont le statut tabou est levé en raison des critères d'aspiration :

$N_T(s) = \{ s' \in N(s)$ tel que $s' \notin T$ ou $f(s') < f(s^*) \}$

```
Procédure methode_Tabou (solution initiale s)
Début
    poser T ← ∅ et s* ← s;
    Répeter
        Choisir s'qui minimise f(s') dans N_T(s)
        Si f(s') < f(s*) alors poser s* ← s'
        Poser s ← s' et mettre à jour T
    Jusqu'à ce que le critère de terminaison soit satisfait
Fin
```

Comme critère d'arrêt on peut par exemple fixer un nombre maximum d'itérations sans amélioration de *s**, ou bien fixer un temps limite après lequel la recherche doit s'arrêter.

4.4.4. La méthode GRASP

La méthode GRASP (pour Greedy Randomized Adaptative Search Procedure) est une métaheuristique simple, développée à la fin des années 90 par Feo et Resende.

Elle est adaptée aux problèmes dont les solutions se construisent pas à pas, comme le problème d'ordonnancement de tâches, dont la solution consistera en une certaine suite d'opérations à ordonner sur des machines.

Son algorithme contient deux phases :

- Une phase de construction d'une solution
- Une étape d'amélioration de la solution

La phase d'amélioration sera souvent faite grâce à une autre métaheuristique, une recherche locale simple par exemple. Ce qui est tout à fait caractéristique de la méthode GRASP, c'est sa phase de construction de la solution. Pour ce faire, l'algorithme maintient à jour une liste de fragments de solutions possibles (RCL, restricted candidate list). Par exemple, dans le problème de l'ordonnancement de tâches, la RCL sera composée d'une sélection d'opérations élémentaires restant encore à ordonner.

La solution se construit donc pas à pas en allant « piocher » les opérations à programmer dans cette RCL, qui est mise à jour à mesure que s'élabore la solution.

Deux points sont à remarquer :

la RCL est mise à jour avec des éléments sélectionnés en fonction d'une heuristique spécifique, adaptée au problème considéré.

le choix d'un élément dans la RCL pour construire la solution est aléatoire.

L'algorithme général peut se représenter avec le pseudo-code suivant :

```
Procédure methode_GRASP ()
Début
    s*←∅
    Répeter
        s' ← ConstruireSolution()
        Améliorer(s')
        Si f(s') < f(s*) alors s* ← s'
    Jusqu'à ce que le critère de terminaison soit satisfait
Fin
```

```
Procédure ConstruireSolution()
Début
    S ← ∅
    Tant que s n'est pas complète
        RCL ← GenererCandidat(s)
        x← ChoisirAuHasard(RCL)
        s ←  s U {x};
    Fin
    Retourner s
Fin
```

4.4.5. Les algorithmes génétiques

Avec les algorithmes évolutionnaires, nous passons à une autre catégorie de métaheuristiques, celles des méthodes dites évolutionnaires, qui manipulent un ensemble de plusieurs solutions simultanément.

La métaheuristique la plus connue dans cette branche est celle reposant sur un algorithme génétique, inspiré du concept de sélection naturelle élaboré par Darwin. Nous parlerons donc d'individus, pour parler de solutions (complètes, ou partielles). L'ensemble des individus formera une population, que nous ferons évoluer pendant une certaine succession d'itérations appelées générations, jusqu'à ce qu'un critère d'arrêt soit vérifié. Pour passer d'une génération à une autre, nous soumettrons la population à des opérateurs de sélection. Cette approche sera détaillée à la section 5.

4.4.6. Les algorithmes de colonies de fourmi

Comme les algorithmes génétiques, les algorithmes de colonies de fourmi font évoluer une population d' « agents », selon un modèle stochastique.

Cet algorithme est encore inspiré de la nature et de son organisation, et a été mis au point par Doringo au début des années 90. Son principe repose sur le comportement particulier des fourmis qui, lorsqu'elles quittent leur fourmilière pour explorer leur environnement à la recherche de nourriture, finissent par élaborer des chemins qui s'avèrent fréquemment être les

plus courts pour aller de la fourmilière à une source de nourriture intéressante. Chaque fourmi laisse en effet derrière elle une traînée de phéromone à l'attention de ses congénères ; les fourmis choisissant avec une plus grande probabilité les chemins contenant les plus fortes concentrations de phéromones, il se forme ainsi ces « autoroutes » à fourmis, qui sillonnent le paysage. Ce mode de communication particulier, qui fait intervenir des modifications dans l'environnement, est appelé stigmergie.

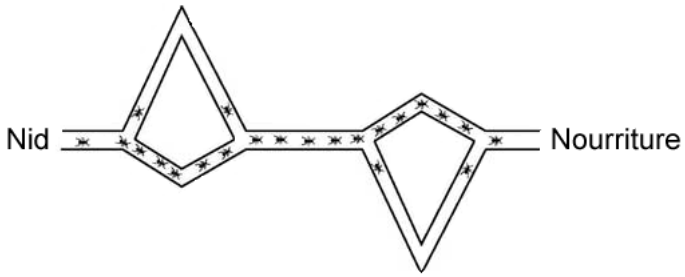


Figure 8 : recueil de ressources par des fourmis

Principe de l'algorithme

Le premier algorithme conçu selon ce modèle était destiné à résoudre le problème du voyageur de commerce.Le principe consiste à « lancer » des fourmis, et à les laisser élaborer pas à pas la solution, en allant d'une ville à l'autre. C'est donc un algorithme qui repose sur la *construction* progressive de solutions, un peu comme dans la méthode GRASP, qui inclue également une phase de construction. Afin de ne pas revenir sur ses pas, une fourmi tient à jour une liste Tabou, qui contient la liste des villes déjà visitées.

Soit A un ensemble de k fourmis :

```
Répeter :
    Pour i=1 à k faire
        ConstruireTrajet(i)
    Fin Pour
    MettreàJourPheromones()
Jusqu'à ce que le critère de terminaison soit satisfait.
```

Dans la procédure *ConstruireTrajet*(i), chaque fourmi se construit une route en choisissant les villes selon une règle de transition aléatoire.

4.5. L'hybridation des Métaheuristiques

Pour élargir le spectre d'application d'une méthode, ou augmenter ces performances, le recours au principe d'hybridation est envisagé depuis longtemps, ce qui permet de créer de nouvelles méthodes hybrides plus sophistiquées et plus performantes.

Une méthode hybride est une méthode de recherche constituée d'au moins deux méthodes de recherche distinctes. Il est possible d'hybrider toutes les méthodes, y compris heuristiques et méthodes exactes [Duv 00], mais il faut être prudent dans le choix des méthodes à utiliser pour obtenir une bonne coopération. On peut choisir par exemple, l'hybridation d'une méthode évolutionnaire qui renforce la diversification, et une méthode de recherche locale qui assure l'intensification de la recherche.

L'hybridation peut se faire par plusieurs stratégies [Duv, 00] :

Hybridation séquentielle : consiste à exécuter séquentiellement différentes méthodes de recherche, de telle manière que les résultats d'une méthode servent de solutions initiales à la suivante.

Hybridation parallèle synchrone : est obtenue par incorporation d'une méthode de recherche particulière dans un opérateur d'une autre. C'est à dire, une méthode est englobée dans une autre. C'est une technique plus complexe que la précédente.

Hybridation parallèle asynchrone : consiste à faire évoluer en parallèle plusieurs méthodes de recherche qui peuvent être identiques. Cette coévolution nécessite un mécanisme coordinateur qui permet d'assurer une bonne coopération des méthodes.

5. Conclusion

Nos avons vu dans ce chapitre les méthodes d'optimisation combinatoires les plus populaires. Cependant il faut choisir la bonne méthode pour résoudre un problème donnée. Certaines métaheuristiques présentent l'avantage d'être simples à mettre en œuvre, comme nous l'avons vu avec le recuit simulé ; d'autres sont plutôt bien adaptées à la résolution de certaines classes de problème, très contraints, comme le système de colonies de fourmis. La qualité des solutions trouvées par les métaheuristiques dépend de leur paramétrage (il faut éviter que les algorithmes ne convergent trop rapidement vers un optimum local), et de l'équilibre à trouver entre un balayage de tout l'espace des solutions et une exploration locale poussée.

1. Introduction

L'évolution biologique a engendré des systèmes vivants extrêmement complexes. Elle est le fruit d'une altération progressive et continue des êtres vivants au cours des générations et s'opère en deux étapes : la sélection et la reproduction.

La sélection naturelle est le mécanisme central qui opère au niveau des populations, en sélectionnant les individus les mieux adaptés à leur environnement.

La reproduction implique une mémoire : l'hérédité, sous la forme de gènes. Ce matériel héréditaire subit, au niveau moléculaire, des modifications constantes par mutations et recombinaisons, aboutissant ainsi à une grande diversité.

Ces principes, présentés pour la première fois par Darwin, ont inspiré bien plus tard les chercheurs en informatique. Ils ont donné naissance à une classe d'algorithmes regroupés sous le nom générique d'*Algorithmes Evolutionnaires* (ou *Evolutionary Algorithms* (EA)). Les sections suivantes en présentent les fondements.

2. Historique

Il existe une catégorie de problèmes pour lesquels il est difficile, voire impossible, de trouver une solution en un temps limité. Il est alors utile de trouver une technique permettant la localisation rapide de solutions sous-optimales, sachant que l'espace de recherche a une taille et une complexité suffisamment importantes pour éliminer toute garantie d'optimalité [Jon 88]. Pour cela, un système capable de s'automodifier au cours du temps, tout en améliorant sa performance dans l'accomplissement des tâches auxquelles il est confronté, semble ouvrir la voie à une recherche intéressante.

Les EA sont basés sur des principes simples. En effet, peu de connaissances sur la manière de résoudre ces problèmes sont nécessaires, même si certaines peuvent être exploitées afin de rendre plus efficace l'évolution (en effet, il n'est pas réaliste d'espérer obtenir une méthode d'optimisation raisonnablement efficace si on a aucune connaissance sur le domaine à traiter). C'est pourquoi, dans de nombreux domaines, les chercheurs ont été amenés à s'y intéresser.

Un certain nombre de travaux ont porté sur l'optimisation de fonctions mathématiques complexes telles celles proposées dans [Jon 75] Les EA ont été appliqués à différents problèmes issus de la recherche opérationnelle : la coloration de graphes, le problème du voyageur de commerce, la gestion d'emploi du temps, etc.

On trouve aussi des applications en robotique pour la recherche de trajectoire optimale ou l'évitement d'obstacles, en vision pour la détection ou la reconnaissance d'images, en recalage d'images médicales, en reconnaissance de formes mais aussi en mécanique pour l'optimisation de formes, en chimie, économie et gestion de production, gestion de trafic aérien, etc.

3. Principes généraux

Les EA sont une classe d'algorithmes d'optimisation par recherche probabiliste basés sur le modèle de l'évolution naturelle. Ils modélisent une *population* d'*individus* par des points dans un espace. Un individu est codé dans un *génotype* composé *de gènes* correspondant aux valeurs des paramètres du problème à traiter. Le génotype de l'individu correspond à une solution potentielle au problème posé, le but des EA est d'en trouver une solution.

La plupart des problèmes peuvent être résolus par une méthode de type recherche locale. C'est le cas, par exemple, de l'apprentissage d'un réseau de neurones par rétro-propagation du gradient. Partant d'une configuration de poids initiale, la méthode va chercher la meilleure solution dans le voisinage de cette configuration. Cette solution est optimale localement mais peut ne pas correspondre à un optimal global car il est possible qu'il existe une meilleure solution qui n'est pas dans le voisinage de la configuration initiale. La Figure 9 illustre ce type de problème.

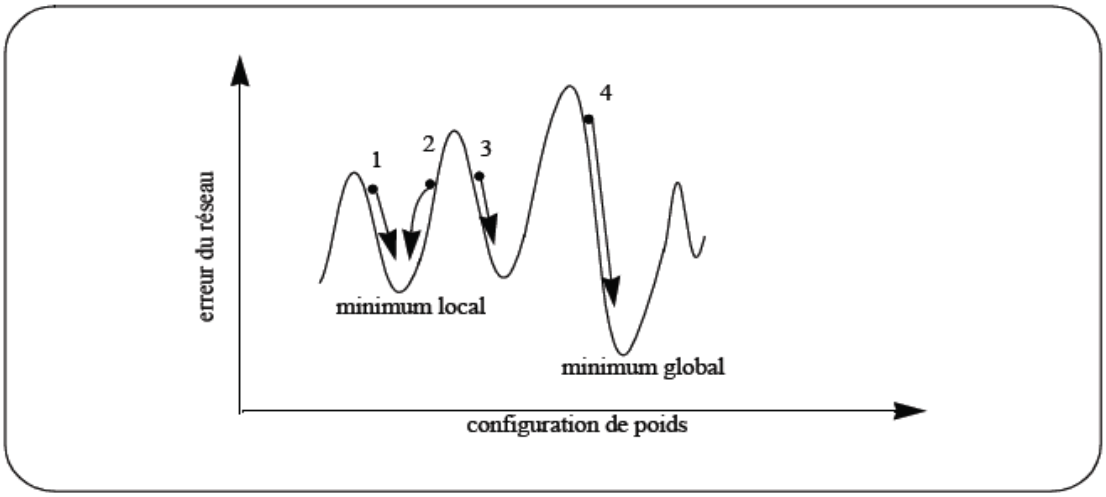


Figure 9 : Exemple de convergence locale ou globale d'un réseau de neurones.

Les EA ont montré leur capacité à reduire la convergence des solutions vers des optima locaux, aussi bien lorsqu'ils sont combinés avec des méthodes de recherche locale comme la rétro-propagation du gradient [Bel 90] que lorsqu'ils sont seuls [Gol 89]. Quel que soit le type de problème à résoudre, les EA opèrent selon les principes suivants : la population est initialisée de façon dépendante du problème à résoudre (l'*environnement)*, puis évolue de génération en génération à l'aide d'opérateurs de *sélection*, de *recombinaison* et de *mutation*. L'environnement a pour charge d'évaluer les individus en leur attribuant une performance (ou *fitness*). Cette valeur favorisera la sélection des meilleurs individus, en vue, après reproduction (opérée par la mutation et/ou recombinaison), d'améliorer les performances globales de la population.

Plusieurs types d'évolution ont été développés, donnant naissance à quatre grandes tendances : les Algorithmes Génétiques (ou *Genetic Algorithms* (GA)), les Stratégies d'Evolution (ou *Evolution Strategies* (ES)), la Programmation Evolutive (ou *Evolutionary Programing* (EP)) et la Programmation Génétique (ou *Genetic Programing* (GP)).

De ces quatre méthodes classiques ont dérivé différentes techniques mélangeant les méthodes d'évolution des unes et des autres. Impossible à classer dans l'une des quatre familles citées ci-dessus, elles sont néanmoins considérées comme des EA.

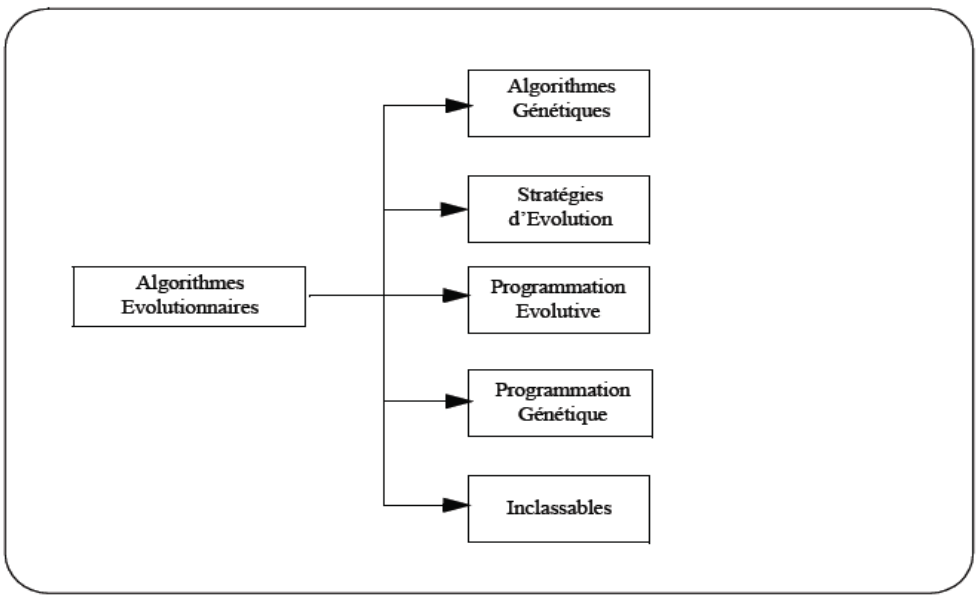


Figure 10 : Différentes branches des algorithmes évolutionnaires.

4. Algorithmes génétiques

4.1. Principes généraux

Développés dans les années 70 avec le travail de Holland [Hol 75] puis approfondis par Goldberg [Gol 89], les GA sont certainement la branche des EA la plus connue et la plus utilisée. La particularité de ces algorithmes est le fait qu'ils font évoluer des populations d'individus codés par une chaîne binaire. Ils utilisent les opérateurs de mutation binaire et de recombinaison de différents types.

4.2. Représentation

La représentation des individus est basée sur le codage binaire de l'information. La difficulté de cette représentation est de choisir le codage approprié. [Mic 96] suggère de trouver une représentation adaptée à l'espace de recherche associé au problème à optimiser et de choisir des opérateurs génétiques adaptés à cette représentation. Dans une telle représentation, deux génotypes donnant des solutions proches doivent différer de peu dans leur représentation. Ce n'est pas vraiment le cas dans la représentation binaire classique. On peut parler des falaises de Hamming traduisant le passage du nombre 7 (0111) au nombre 8 (1000) en 4 mutations. Différentes possibilités ont été proposées, comme le codage de Gray proposant (0100) pour 7 et (1100) pour 8, l'adaptation de la probabilité de mutation des gènes en fonction de leur position dans le génotype [Fog 89], ou la représentation de Dedekind [Sur 96].

4.3. Principe de l'algorithme génétique

Le principe de l'algorithme génétique se résume par l'algorithme suivant :

Début

Génération d'une population initiale de k individus

Réputer

5. Sélectionner pour la reproduction
6. Croisement des individus sélectionnés
7. Mutation des individus sélectionnés
8. Sélection pour le remplacement

Jusqu'à (condition d'arrêt)

Retourner la meilleure solution

Fin

4.4. Opérateurs de sélection

Il y a donc deux étapes de sélection : l'une pour déterminer quels seront les n éléments, parmi les μ individus de la population totale qui vont se reproduire entre eux, et former λ enfants. Et une seconde étape pour déterminer quels sont ceux qui vont survivre. Ces nombres μ, n et λ sont des paramètres de l'algorithme, fixés par l'opérateur, qui peuvent conduire à différents scénarios, par exemple, l'opérateur peut décider qu'à chaque génération, les enfants remplacent tous les parents. Ou au contraire, il peut décider de faire se conserver la plupart des parents d'une génération à l'autre, et ne faire intervenir qu'un petit nombre d'enfants à chaque fois.

Sélection déterministe :

C'est la méthode de sélection la plus simple à mettre en œuvre, puisqu'elle consiste à choisir les n meilleurs individus parmi la population, n étant un paramètre fixé (égal à la moitié de la population par exemple).

Elle présente l'inconvénient de faire se conserver les éléments ayant une mauvaise performance, si la performance de la population est globalement mauvaise.

Sélection par tournois :

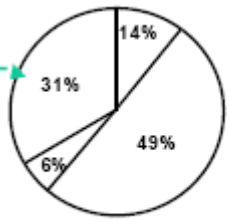
L'idée de cette méthode est de sélectionner aléatoirement un nombre k d'individus dans la population, et de choisir parmi eux celui qui a la meilleure performance. On organise autant de tournois qu'il y doit y avoir d'individus à sélectionner. On peut décider de faire des tirages avec, ou sans remise. Le tirage sans remise permet d'éviter de favoriser excessivement un individu. Dans la mesure où seul l'ordre des éléments importe dans la sélection par tournois, il est parfois possible simplifier l'évaluation de la fonction objectif, et donc de gagner en temps de calcul.

Sélection proportionnelle :

Dans la méthode proportionnelle, le nombre espéré de sélections d'un individu est proportionnel à sa performance. Pour expliciter cela, imaginons une roulette de casino, qui comporterait autant de cases que d'individus, et dont la taille des cases serait proportionnelle à la performance des individus. Chaque sélection est alors déterminée par le lancer d'une bille sur la roulette.

Ainsi, avec une population de quatre individus, dont les valeurs de performance seraient de 169, 576, 64 et 361, on obtiendrait la roulette suivante :

f(x)	f/Σ f
169	0.14
576	0.49
64	0.06
361	0.31
1170	1.00



Avec cette technique, il faut s'arranger pour que *fi* prenne des valeurs positives, en bornant inférieurement la fonction à 0, ou à une très petite valeur, pour que tout individu, même mauvais, ait quelque chance d'être sélectionné.

La sélection proportionnelle risque cependant de favoriser excessivement un très bon individu au détriment des autres. Pour éviter cela, on peut se contenter d'un seul tirage aléatoire, qui fixe le premier élément sélectionné, les autres étant déterminés à partir de la position du premier, selon une succession de points équidistants.

fi	5	4	1	10	4
	*	*		* *	*

Par exemple : ici, un premier jet a élu l'individu de performance 5 : on a placé une étoile sur le ruban, au niveau de la case 5. Puis, un certain offset a déterminé les quatre autres enfants (on voit sur le schéma que la distance entre chaque étoile est fixe). C'est ainsi que, dans notre exemple, que l'individu de performance 1 n'est pas sélectionné.

Cette méthode (*Stochastic Universal Sampling*) a l'avantage de pénaliser les mauvaises solutions, sans mettre trop en avant les meilleures.

4.5. Opérateurs de variation

On distingue les opérateurs de croisement, qui génèrent de nouveaux individus à partir de plusieurs (un couple, le plus souvent), et les opérateurs de mutation, qui transforment un seul individu.

Croisement :

Il n'y a pas de règle absolue pour construire des opérateurs de croisement. Il est possible d'apparier au hasard, mais en général, on préférera définir une distance dans l'espace de recherche, et combiner les individus proches selon cette distance. Par exemple, dans un algorithme génétique appliqué à un problème de régulation de transport, on associera ensemble des individus générant peu de conflits, ou le moins de retards.

Pour pouvoir croiser, il faut déjà représenter les solutions. C'est dans les années 60 que John Holland, le père des algorithmes génétiques, a mis au point un système de représentation binaire, inspiré du modèle phénotype-génotype du monde vivant. Ce modèle est devenu classique, quoique controversé.

Dans ce système, les solutions du problème sont comparables à des chromosomes, et sont constituées d'une série de gènes, chaque gène étant associé à une séquence de bits, à une chaîne binaire codée en fonction d'une règle de conversion particulière. Croiser des individus consiste alors à « croiser » les séquences binaires de deux parents, pour former les « génomes » des enfants.

Il existe plusieurs méthodes :

Dans le croisement par point, le génome des deux enfants est constitué d'une partie de chacun des deux parents, par coupure et échange. Dans le croisement uniforme, on génère un masque binaire, de même longueur que le gène, qui détermine l'opération appliquée sur le gène des enfants (copie intacte, ou croisement des bits, selon la valeur des bits du masque).

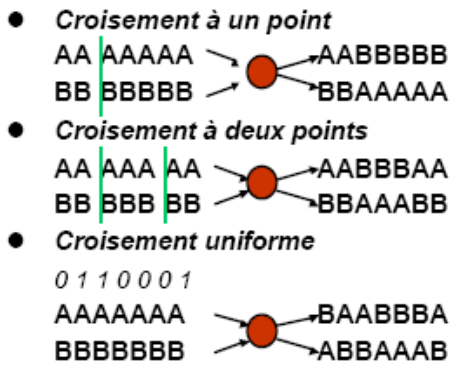


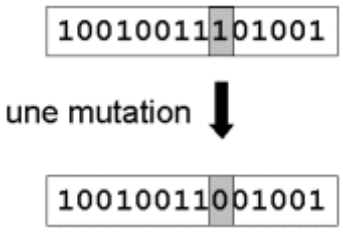
Figure 11 : les différents types de croisements

Dans certains problèmes, on représente les individus sous forme d'arbre binaire, et le croisement se fera par coupure et inversion des branches : ceci est tout à fait adapté à la résolution de problème de calculs formels par exemple.

Mutation :

Dans la mutation binaire, on modifie de temps en temps, avec une faible probabilité, les gènes des enfants : quelques bits, dans le gène d'un individu, sont inversés (un 0 devient un 1, un 1 devient un 0). Ce nombre de bits modifiés peut être fixe, ou choisi aléatoirement.

La mutation permet bien sûr d'introduire de la variété dans la population, et explorer davantage l'espace de recherche, mais si la mutation s'applique trop souvent, l'algorithme génétique revient à faire une marche aléatoire dans l'espace des solutions.



5. Stratégies d'Evolution

Les stratégies d'évolution (SE) constituent une technique d'AE développée par I. Rechenberg et H.-P. Schwefel à Berlin dans les années 1960 [Tom 02][Rec 73]. Les SE représentent les individus comme un ensemble de caractéristiques de la solution potentielle. En général, cet ensemble prend la forme d'un vecteur de nombres réels de dimension fixe. Les SE s'appliquent à une population de parents (de dimension μ) à partir de laquelle des individus sont sélectionnés aléatoirement afin de générer une population d'enfants (de dimension $\lambda \geq \mu$). Ceux-ci sont ensuite modifiés par des mutations qui consistent à ajouter une valeur générée aléatoirement selon une fonction de densité de probabilité paramétrée. Les paramètres de la fonction de densité de probabilité, nommés les paramètres de la stratégie, évoluent eux aussi dans le temps selon les mêmes principes que les paramètres caractérisant les individus. Pour former la nouvelle population, μ individus sont choisis (approche (μ, λ)) parmi les μ meilleurs individus des λ enfants, ou (approche $(\mu + \lambda)$) parmi les μ meilleurs individus des μ parents et λ enfants.

Les SE modernes peuvent aussi intégrer une approche multi-échelle où des stratégies d'évolution sont imbriquées.

6. Programmation Evolutionnaire

La programmation évolutionnaire (PE) a été développée par L.J. Fogel dans les années 1960 et reprise par D.B. Fogel et d'autres chercheurs dans les années 1990 [Tom 02] [Law 66].

La PE a été initialement conçue dans le but de faire évoluer des machines à états finis et a été par la suite étendue aux problèmes d'optimisation de paramètres. Cette approche met l'emphase sur la relation entre les parents et leurs enfants plutôt que de simuler des opérateurs génétiques d'inspiration naturelle. Contrairement aux trois autres AE classiques, la PE n'utilise pas une représentation spécifique mais plutôt un modèle évolutionnaire de haut niveau, jumelé à une représentation et à un opérateur de mutation directement appropriés au problème à résoudre.

Pour effectuer de la PE, une population de solutions potentielles au problème est d'abord générée aléatoirement. Chaque individu ι de la population produit λ enfants résultant de mutations. Une opération de sélection naturelle est alors appliquée afin de former une nouvelle population d'individus. Le processus de mutation - sélection est répété itérativement jusqu' à ce qu'une solution acceptable soit trouvée.

7. Programmation Génétique

La programmation génétique (PG) [Wol 98] [Joh 92] est un paradigme permettant la programmation automatique d'ordinateurs par des heuristiques basées sur les mêmes principes d'évolution que les AG : des opérations de variation génétique, par des croisements et des mutations, et des opérations de sélection naturelle. La différence entre la PG et les AG réside essentiellement dans la représentation des individus. En effet, la PG consiste à faire évoluer des individus dont la structure est similaire à des programmes informatiques.La PG a été exprimée formellement par Koza au début des années 1990 [Joh 92].

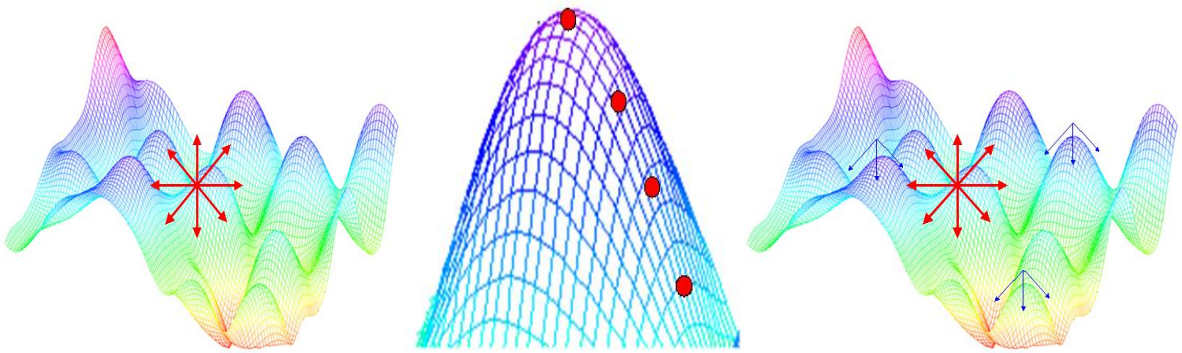
La PG utilisée par Koza représente les individus sous forme d'arbres, c'est-à-dire des graphes orientés et sans cycle, dans lesquels chaque noeud est associé à une opération élémentaire relative au domaine du problème. Plusieurs autres représentations comme des programmes linéaires [Wol 98] et des graphes cycliques [Ast 96], ont été utilisées depuis. La

PG est particulièrement adaptée à l'évolution de structures complexes de dimensions variables.

8. Algorithmes mémétiques

Les algorithmes mémétiques ont été introduits pour la première fois par Moscato [Mos 89]. Fondamentalement, ils hybrident les algorithmes génétiques avec des méthodes de recherche locale. Pour cette raison, certains chercheurs les ont vus comme des algorithmes génétiques hybrides. La combinaison qui semble la plus fructueuse est celle qui utilise une méthode de recherche locale dans un algorithme génétique. On remarque que les algorithmes génétiques peuvent être une bonne solution pour résoudre des problèmes d'optimisation combinatoire. Cependant, un inconvénient d'un algorithme génétique est que les opérateurs standard de croisement et de mutation ne permettent pas d'intensifier suffisamment la recherche comme le rappellent [Hoo 04]. L'opérateur de mutation apporte une légère modification à l'individu. Son rôle est de favoriser la diversification des individus alors que la sélection se charge de conserver les meilleurs. C'est pourquoi les algorithmes génétiques sont souvent hybridés avec des méthodes de recherche locale.

Ces deux méthodes sont complémentaires car l'une permet de détecter de bonnes régions dans l'espace de recherche alors que l'autre se concentre de manière intensive à explorer ces zones de l'espace de recherche. Ainsi, on peut explorer rapidement les zones intéressantes de l'espace de recherche pour les exploiter en détail comme le montre la figure 12.



- Algorithme génétique (b) Recherche locale (c) Algorithme mémétique

Figure 12 : Comportement schématique d'un algorithme génétique, d'une recherche locale et d'un algorithme mémétique.

Il ya plusieurs raisons pour hybridés un algorithme génétique avec une recherche locale [KRA 04], parmi lesquels on peut citer les raisons suivants :

8.1. Les raisons de l'hybridation :

1. les algorithmes évolutionnaires ou les méthodes de recherche locale peuvent étre 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. Par ailleurs, on peut disposer de méthode de recherche locale très puissante 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.
2. dans certains cas, il existe des méthodes exactes ou approximatives pour les sous problemes du probléme traité, capables de prendre en charge certaines spécificités du probléme. Lorceque celles-ci sont disponibles, elles peuvent etre incorporées dans l'algorithme évolutionnaire afin de produire de meilleures solutions.
3. les problémes associent des contraintes à des solutions et les heuristiques de rechrche 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 premiére préoccupation du concepteur alors une définition plus riche des métaheuristiques hybrides d'adaptation est possible : les stratégies de recherche locale pourraient etre générées en méme 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 paramétres de probabilité à des membres d'individus de la population dans chaque génération.

Certaines, décisions de conception [KRA 04] doivent être prises lors de la conception d'un algorithme mémétique, on peut citer :

- 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 :

8.2. Les stratégies de remplacement des individus

Lors de l'intégration de la recherche locale dans l'algorithme évolutionnaire, nous somme confrontés à ce qu'il faut avec la solution améliorée produite par la recherche locale. Par exemple, 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 :

Approche de LAMARCKIAN :

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.

Approche de BALDWINIAN

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

P reste la même et F(i) = f(i')

La question qui se pose est de savoir laquelle des deux approche est meilleure, Il est à priori difficile de décider quelle est la meilleure méthode et probablement aucune n'est bonne dans tous les cas.

8.3. Préservation de la diversité

Nous avons mentionné ci-dessus les difficultés lors de l'utilisation d'une methode très agressive de recherche locale, au premier rang de ces difficultés est la nécessité de préserver la diversité. Cette diversité pourrait être perdue si par exemple, dans le cas de recherche locale partielle, l'espace de recherche possède une forme (ou topologie) a très larges bassins desquels le croisement et la mutation ne peuvent pas facilement éviter d'être piégés dans des optimums 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.
- L'introduction d'opérateurs de croisement [FRI 96] 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.

- Des recherches locales multiples où chacune introduit un espace de recherche différent avec des optimums locaux différents afin d'éviter les pièges locaux [KRA 04].
- La mise en œuvre des ensembles et systèmes flous pour contrôler explicitement la diversité au sein d'une règle de décision dans la phase de recherche [KRA 02].
- Finalement, il est possible aussi de modifier l'opérateur de sélection ou les critères d'acceptations de recherche locale pour utiliser la méthode adaptative de Boltzmann [KRA 00]. 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. Dans cette méthode et durant la recherche locale au moins un voisinage peut être accepté avec une probabilité qui augment exponentiellement avec un facteur k de normalisation. Ce mécanisme permet à

l'algorithme mémétique d'osciller entre les périodes d'exploitation et les périodes d'exploration.

8.4. Le choix de voisinage dans la recherche locale

Ce choix 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 solution. Si l'espace de solution 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, dans ce cas l'analyse exhaustive peut conduire à une longue recherche qui peut être exponentielle.

8.5. Utilisation des profils de performance

Les profils de performance peuvent être utilisés pour suivre le progrès de la recherche vis-à-vis des différents composants 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 précédentes, vues comme un moyen pour guider l'optimisation, par exemple la recherche tabou n'autorise pas de revisiter des solutions contenues dans la liste tabou.

9. Conclusion

Nous avons vu dans ce chapitre les principes généraux des algorithmes évolutionnaires et nous avons détaillé les algorithmes génétiques qui peuvent être une bonne solution pour résoudre des problèmes d'optimisation combinatoire, Toutefois les opérateurs standard de croisement et de mutation ne permettent pas d'intensifier suffisamment la recherche, c'est pourquoi les algorithmes génétiques sont souvent hybridés avec des méthodes de recherche locale (pour avoir les algorithmes mémétiques). D'un autre coté cette hybridation augmente le temps d'exécution, donc il faux améliorer ce temps sans diminuer la qualité des résultats, ce qui fait l'objectif du chapitre suivant (Parallélisations des métaheuristiques).

1. Introduction

De façon générale, le parallélisme est utilisé pour la résolution de problèmes complexes nécessitant des algorithmes coûteux en temps d'exécution. Dans le contexte d'algorithmes exacts produisant une solution déterminée et optimale à un problème donné, les principaux buts recherchés sont de réduire le temps de résolution, c'est-à-dire d'obtenir cette solution plus rapidement, et de s'attaquer à des instances de plus grande taille.

Il existe cependant certains problèmes qui ne peuvent être résolus en temps polynomial par des algorithmes exacts de nature séquentielle ou parallèle. La plupart des problèmes d'optimisation combinatoire font partie de classe de problèmes dits NP-Difficiles [Gar 79]. Une stratégie de résolution de plus en plus appliquée ce type de problème est l'utilisation de métaheuristiques. Ces algorithmes d'approximation ne garantissent pas l'obtention de solutions optimales mais trouvent généralement de bonnes solutions dans un temps de calcul raisonnable.

Plusieurs travaux réalisés au cours des dernières années ont démontré l'utilité et l'efficacité des métaheuristiques pour la résolution de problèmes d'optimisation combinatoire. Il demeure toutefois que ces algorithmes demandent un temps de calcul et une quantité de mémoire considérables qui sont étroitement reliés à la taille du problème et à l'obtention d'une certaine qualité de solution. De ce fait, ces algorithmes deviennent intéressants à paralléliser. Dans ce contexte, les objectifs de réduction du temps de calcul et de traitement de gros problèmes sont toujours pertinents, mais à cela s'ajoute l'intérêt d'utiliser le parallélisme pour améliorer la performance des métaheuristiques en termes de qualité de solution obtenue et ce, sans augmenter la charge de calcul.

2. Le parallélisme et les métaheuristiques

Même si les métaheuristiques offrent des stratégies efficaces pour la recherche de solutions à des problèmes d'optimisation combinatoire, les temps de calcul associés à l'exploration de l'espace de recherche peuvent être très importants [Cun 01]. Une façon d'accélérer cette exploration est l'utilisation du parallélisme. Une autre contribution du calcul parallèle aux métaheuristiques est de plus en plus constatée : en leur appliquant des paramètres appropriés, les métaheuristiques parallèles peuvent être beaucoup plus robustes que leurs équivalents séquentiels en termes de qualité de solutions trouvées [Cra 98]. Certains mécanismes parallèles permettent donc de trouver de meilleures solutions sans nécessiter un nombre total d'opérations plus grand.

Le parallélisme peut donc représenter un apport considérable aux métaheuristiques et un nombre grandissant de travaux sont effectués afin d'exploiter ce potentiel. Actuellement, la plupart de ces travaux sont limités à une métaheuristique ou à une stratégie de parallélisation particulière et rares sont ceux qui apportent une vue globale et fondamentale aux métaheuristiques parallèles.

3. Les stratégies de parallèlisation des algorithmes génétiques

Les algorithmes génétiques sont ceux qui ont fait l'objet du plus grand nombre de travaux de parallélisation, notamment en raison de leur nature parallèle fondamentale. [Cra 98]. Cantú-Paz (1998) [Can 98] ont présenté une revue des principales publications ayant trait aux algorithmes génétiques parallèles. Ils distinguent trois principales catégories d'algorithmes génétiques parallèles :

- Parallélisation de forme maître-esclave sur une population unique
- Parallélisation de grain fin sur une population unique (Modèle de diffusion)
- Parallélisation de gros grain sur des populations multiples (Modèle de migration)

Dans le premier modèle, il existe une seule population résidant sur un seul processeur appelé le maître. Ce dernier effectue, de génération en génération, les différents opérateurs génétiques de l'algorithme sur la population et distribue ensuite l'évaluation des individus aux processeurs esclaves.

Dans le deuxième modèle, qui est adapté aux ordinateurs massivement parallèles, les individus de la population sont répartis sur les processeurs, préférablement à raison d'un individu par processeur. Les opérateurs de sélection et de reproduction des individus sont

limités à leurs voisinages respectifs. Cependant, comme les voisinages se chevauchent (un individu peut faire partie du voisinage de plusieurs autres individus), un certain degré d'interaction entre tous les individus est possible.

La troisième catégorie, plus sophistiquée et plus populaire, consiste en plusieurs populations qui sont réparties sur les processeurs. Celles-ci peuvent évoluer indépendamment les unes des autres ou échanger occasionnellement des individus. Cet échange facultatif, appelé le phénomène de migration, est contrôlé par différents paramètres et permet généralement une meilleure performance de ce type d'algorithme. Cette classe d'algorithmes génétiques parallèles est cependant plus difficile à comprendre parce que les effets de la migration sont complexes et difficiles à formaliser. Cette catégorie est aussi appelée entre autres « *algorithmes génétiques distribués* » ou « *algorithmes génétiques parallèles en îles* ».

3.1. Parallélisation de forme maître-esclave

La parallélisation de forme maître-esclave est classée comme un parallélisme d'exploration avec un chemin unique. Donc, le chemin suivi par ce parallélisme est le même qu'une implémentation séquentielle. Donc, cette stratégie manipule une seule population et distribue les individus parmi les processeurs pour les traiter. Le traitement qui est fait en parallèle par les processeurs est l'évaluation de la fonction de coût des individus ou l'application des opérateurs de croisement et mutation. Le maître est le seul qui a la population globale dans sa propre mémoire. Le processeur maître fait la sélection des individus puis les distribuer parmi les esclaves, pour faire le croisement et la mutation. Dans cette méthode on distingue deux types selon le type de communication entre le maître et les esclaves.

L'algorithme est dit synchrone, si le maître attend les valeurs de tous les individus de la population. Ce type donne les mêmes résultats comme pour l'implémentation séquentielle, seulement le temps de calcul est réduit. Les inconvénients de cette implémentation sont: le temps perdu si quelques esclaves terminent leurs taches plus vite que les autres.

L'algorithme est dit asynchrone, si le maître n'attend pas tous les esclaves, mais il envoie un nouvel individu à l'esclave dès que ce dernier est libre, Ce type de shema ne retourne pas forcement les mêmes résultats.

3.2. Parallélisation de grain fin sur une population unique (Modèle de diffusion)

Le modèle de diffusion est développé pour prendre l'avantage de la structure des machines parallèles. La population est divisée en sous populations typiquement constituées d'un seul individu. Chacun est assigné à un processeur élémentaire (cellule). La sélection et l'accouplement (mutation et croisement) ne sont possibles qu'avec les individus qui sont résidents sur les processeurs voisins. Chaque individu sélectionne le meilleur parmi les voisins ou bien selon un schéma de sélection local pour choisir les parents, puis on fait le croisement pour produire un nouvel individu fils. Ce dernier peut remplacer (ou pas) l'individu parent dans le processeur élémentaire. Cela dépend du schéma de remplacement. Ce modèle dépend des trois critères suivants :

- L'architecture de connexion entre les processeurs élémentaires.
- Le schéma de sélection.
- Le schéma de remplacement.

3.3. Parallélisation de gros grain sur des populations multiples (Modèle de migration)

Le modèle de migration est connu aussi par le nom *modèle des îles* ou recherche par coopération des populations. Ce modèle est inspiré de l'observation de la nature comme plusieurs autres méthodes de recherche (les algorithmes génétiques inspirés du système génétique et la métaheuristique la colonie de fourmis par l'observation des fourmis).

Le monde des humains ou tout autre monde d'un type d'espèce vivante peut être vu comme une collection de plusieurs populations qui vivent et évoluent dans des régions séparées (ou bien des îles). De temps en temps, quelques individus d'une population migrent à une autre population. Ces individus migrants prennent des nouveaux attributs à la population de destination. L'évolution entre ces individus et la population permet de générer des individus avec les caractéristiques des deux populations. Ici, la migration permet de partager les caractéristiques d'une population avec les autres tout en faisent en apparaitre de nouvelles. La projection de cette observation sur une méthode de recherche qui se base sur l'évolution de la population tels que les algorithmes génétiques, permet de découvrir une nouvelle stratégie de parallèlisation avec le même principe.

Cette nouvelle stratégie, le modèle des îles, divise la population globale en sous populations, chacune d'elles est assignée à un esclave (dit île ou site par des autres) pour la traiter avec un algorithme génétique séquentiel. La sélection et le croisement ne sont faits qu'avec les individus de la même sous population. Durant la recherche, les esclaves échangent les individus entre eux par migration. L'implémentation de cette opération reste compliquée, puisque il y a plusieurs critères à déterminer. Leur variation d'une implémentation à une autre peut donner des résultats différents (la qualité de la solution et le temps écoulé). Ces critères sont :

- La sélection des individus pour la migration: Les caractéristiques des individus à migrer (les meilleures, aléatoire ou autre méthode de sélection).
- Le taux de migration (*migration rate*): Le nombre des individus dans une migration.
- L'intervalle de migration (*migration interval*): Le nombre (fréquence) de migration à faire.
- Le schéma de communication: Elle définit les esclaves de destination pour chaque esclave.
- La méthode de remplacement: Elle sélectionne les individus à détruire (remplacer) après la réception des individus de migration d'un autre esclave. Généralement, les individus les plus pires qui sont remplacés.

Les trois stratégies précédentes sont considérées comme les plus connues et les plus utilisées pour paralléliser les algorithmes génétiques. Il existe d'autres stratégies parallèles pour les algorithmes génétiques qui sont présentées dans d'autres classifications. Par exemple dans [Can 98], Cantù-Paz a ajouté la parallèlisation hiérarchique comme le résultat de combinaison des trois méthodes précédentes.

3.4. Parallèlisation hiérarchique

Quelques études ont essayé de combiner deux des méthodes précédentes de parallèlisation des algorithmes génétiques. Cette combinaison a donné la parallèlisation hiérarchique qui est une nouvelle méthode pour paralléliser les algorithmes génétiques. La parallèlisation hiérarchique utilise deux méthodes de parallèlisation en hiérarchie à deux niveaux. Souvent, le niveau externe représente le modèle des îles et le niveau interne représente une autre méthode des trois méthodes de parallèlisation. Dans cette hybridation,

l'algorithme génétique est parallélisé globalement par le modèle des îles dont chaque sous population est parallélisée par une autre méthode.

4. Parallélisation hybrides des métaheuristiques

Chaque métaheuristique possède ses propres caractéristiques et sa propre manière d'effectuer la recherche de solutions. Par conséquent, il peut être intéressant de faire coopérer plusieurs métaheuristiques différentes afin de créer de nouveaux comportements de recherche. À ce sujet, Bachelet *et al*. (1998) [Bac 98] ont déterminé trois principales formes d'algorithmes hybrides :

- Hybride séquentiel où deux algorithmes s'exécutent l'un après l'autre, les résultats fournis par le premier étant les solutions initiales du deuxième.
- Hybride parallèle synchrone où un algorithme de recherche est utilisé à la place d'un opérateur. Un exemple de ce type est de remplacer l'opérateur de mutation d'un algorithme génétique par une recherche taboue.
- Hybride parallèle asynchrone où plusieurs algorithmes de recherche travaillent concurremment et s'échangent des informations. Ces auteurs, par l'utilisation avec succès d'un algorithme hybride parallèle asynchrone combinant un algorithme génétique et un algorithme de recherche avec tabou, montrent une autre facette de la puissance du parallélisme qui est la possibilité de collaboration de plusieurs méthodes de recherche en parallèle.

5. Conclusion

Dans ce chapitre nous avons vu que le parallélisme est utilisé pour la résolution de problèmes complexes nécessitant des algorithmes coûteux en temps d'exécution et nous avons détaillé les stratégies de parallélisations des algorithmes génétiques.

1. Introduction

Après avoir introduit les notions essentielles concernant : la théorie du Data Mining, le problème de la classification, les Métaheristiques, ainsi qu'une présentation des algorithmes évolutionnaires, nous allons dans ce chapitre présenter en détail notre conception commençant par l'adaptation de l'algorithme génétique, la recherche Tabou et l'algorithme mémétique à la classification pour passer ensuite à la présentation de l'architecture parallèle utilisée.

2. Adaptation de l'Algorithme Génétique à la classification

Dans cette partie nous allons montrer comment adapter un algorithme génétique au problème de classification en détaillant les points suivants :

- Codage des règles
- Création de la population initiale
- Evaluation des individus
- Application du cycle de l'Algorithme Génétique
- Construction du classifieur

2.1. Codage des règles [BEN 07]

Il existe deux approches différentes pour extraire des règles en utilisant un algorithme génétique : l'approche de Pittsburgh et l'approche de Michigan que nous avons adopté. La première consiste à coder plusieurs règles au sein d'un individu tandis que dans la seconde, une règle ne code qu'un seul individu. L'individu représente la conjonction de termes attribut-valeur qui correspond à l'antécédent d'une règle. Le codage est un codage de position qui consiste en une suite de gènes rangés dans le même ordre que les attributs du jeu de données, le dernier gène de l'individu contient la valeur de la classe prédite. Chaque condition est codée par un gène et consiste en un triplet de la forme (A_i op V_{ij}) où A_i est le $i^{ème}$ attribut de la table sur laquelle l'algorithme est appliqué. Le terme op est un des opérateurs '=', '≤' ou '≥' et V_{ij} est une valeur du domaine de l'attribut A_i. Chaque gène comporte un champ booléen B_i qui indique si le $i^{ème}$ gène est actif ou pas, c'est-à-dire si la $i^{ème}$ condition est présente dans la régle où non. Ainsi bien que les individus aient la même longueur, les règles associées à ceux-ci sont de longueur variable. La structure d'un individu est indiqué sur la figure 13, où m est le nombre total d'attributs de la table.

Figure 13 : La structure d'un individu.

2.2. Création de la population initiale

Avant de lancer l'Algorithme Génétique, une population initiale doit être générée. La génération de la population initiale dans notre application se fait en deux étapes afin d'apporter une certaine diversité à la population initiale :

1ère Etape : Nous générons aléatoirement le premier individu de la population.

2ère Etape : Nous générons le reste de la population à condition que chaque individu ajouté soit différent des autres en calculant à chaque fois sa similarité (On dit que deux individus sont similaire à 100% si et seulement si sont identiques, sinon s'il y a une déférence entre les individus la valeur de la similarité est compris entre 0% et 100%) par rapport à l'ensemble des individus déjà crées et si elle est inferieure au seuil prédéfini par l'utilisateur (appelé *Seuil_ Similarité*) Alors on rajoute l'individu à la population sinon on ne le rajoute pas.

La fonction suivante illustre comment générer la population initial

```
Population Create_population()
Début
    Population = {vide} ; Individu indiv ;
    Seuil_ Similarité = Le seuil de similarité définie par l'utilisateur ;
    Taille = la taille de la population initial définie par l'utilisateur ;
    Pour i=0 à  (Taille-1)
    Faire
        Répéter
            indiv  = Create_Individu() ; // Créer un nouveau individu
        Jusqu'à (Similarité(indiv,Population) <= Seuil_ Similarité)
        Population = Population + indiv ; // ajouter indiv à la population
    Fait
    Retourner (Population) ;
Fin
```

La fonction Create_Individu() :

Le code suivant montre comment créer et initialiser des individus

```
Individu Create_Individu ()
Début
    Individu indiv = Nouveau individu vide ;
    Entier i, L = Taille de l'individu;
    Pour  i = 0 à (L-1)
    Faire
        // Attribut actif (1) ou non actif(0)
        Etat de l'attribut i  = fonction aléatoire (0, 2) ;
        // Opérateur de l'attribut i
        Si  (Type de l'attribut i est nominal)
        Alors
            // Opérateur ==
            Opérateur de l'attribut i = 0 ;
        Sinon
            //Choisir aléatoirement un des Opérateur : >=, <=, ==
            Opérateur de l'attribut i = fonction aléatoire (0, 3) ;
        Finsi
        // La valeur de l'attribut i
        Valeur de l'attribut i = choisir aléatoirement une valeur permi
        les valeurs possibles de l'attribut i ;
    Fait
    // La classe de l'individu indiv
    Classe de indiv = choisir aléatoirement une classe permi les classes possibles ;
Fin
```

La fonction Similarité :

Le code suivant montre comment mesurer la similarité d'un individu.

```
Individu Similarité (indiv,Population)
Début
    Taille = Taille de la population ; // la taille de la population créé jusqu'au maintenant
    Entier i,j, L = Taille de l'individu;
    Réal Similarité, Similarité_Max = 0;
    Pour  i = 0 à (Taille-1)  // parcourir toutes la population
    Faire
        Similarité = 0;
        Pour  j = 0 à (L-1)  // parcourir tous les attributs
        Faire
            // indiv[j] : C'est l'attribut j de l'individu indiv
            // Population[i][j] : C'est l'attribut j de l'individu Population[i]
            Si ((indiv[j] est active) et (Population[i][j] est active))
            Alors
                Si (Opérateur de indiv[j] = Opérateur de Population[i][j])
                Alors
                        Si (La valeur de indiv[j] = La valeur de Population[i][j])
                        Alors   Similarité = Similarité +3/3 ;
                        Sinon  Similarité = Similarité + 2/3 ;
                Sinon
                        Si (La valeur de indiv[j] = La valeur de Population[i][j])
                        Alors   Similarité = Similarité +2/3 ;
                        Sinon  Similarité = Similarité + 1/3 ;
            Sinon
                Si ((indiv[j] est non active) et (Population[i][j] est non active))
                Alors   Similarité = Similarité +3/3 ;
        Fait
         Si (Similarité > Similarité_Max)
         Alors  Similarité_Max = Similarité ;
    Fait
    Retourner  (Similarité_Max) ;
Fin
```

2.3. Evaluation des individus

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 qu'on cherche à résoudre. Notons qu'il n'y a pas de fonction précise qui permet de calculer avec exactitude la performance de chaque individu. Pour cela nous avons opté pour les critères suivant :

- 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 son nombre d'attributs.

La fonction d'adaptation est définie par [BEN 07] :

Fitness = + λ1 * Couverture / Nbr instances total
(Maximisé la couverture de la règle)
+ λ2 * TP / Couverture
(Maximisé le taux de précision de la règle)
- λ 3 * Nbr attributs de la Régle / Nbr attributs total
(Minimisé la taille de la règle)

λ_i = Est une valeurs entière comprise entre 0 et 1 avec $\sum_{i=1}^{3} \lambda i = 1$

Les Fonctions correspondants au calcul de la fitness sont :

La fonction Couverture [ben 07] [bab 09]

Est une fonction qui calcule la couverture d'une règle R (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 antécédente de l'instance *inst* satisfait la partie prémisse A de la règle, peut importe la classe C.

```
Boolean Fonction Instance_Est_Couverte( Instance inst , Règle R)
Début
    Entier Nb = nombre d'attributs de la partie condition de la règle R;
    Pour i = 1 à (Nb-1)
    Faire
        Si ( le i^ème antécédent de l'instance inst ne satisfait pas le i^éme attribut de la règle R)
        Alors  retourne Faux
    Fait
    Retourne Vrai
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 ( Instance_Est_Couverte ( la i^ème instance inst_i , la règle R) = Vrai)
            Alors  Couverture++ ;
        Fait
        Retourne Couverture ;
Fin
```

La fonction TP [BEN 07] [BAB 09]

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.

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

Début

Entier Nb = nombre d'attributs de la partie condition de la règle ***R***;

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 retourne ***Faux***

Fait

Si (la classe de l'instance ***inst*** est égale à la classe de la règle **R**)

Alors Retourne ***Vrai*** ;

Sinon Retourne ***Faux*** ;

Fin

Entier Fonction **TP** (Règle ***R***)

Début

Entier N = nombre d'instances de la base de données ;

Entier TP = 0 ;

Pour i = 0 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 la

// instance est égale à la classe de la règle.

Si (**Instance_Est_TP** (la i^{ème} instance ***inst*** $_i$, la règle ***R***) = Vrai)

Alors TP ++ ;

Fait

Retourne TP ;

Fin

2.4. Application du cycle de l'Algorithme Génétique

Nous avons utilisé l'Algorithme génétique suivant :

```
Début
    Sélectionner deux individus :
        Individus 1 = Tournament Selection() ;
        Individus 2 = Tournament Selection() ;
    Enfant = Croisement(Individu1, Individu2) ;
    Enfant_Muté = Mutation(Enfant) ;
    Remplacement(Enfant_Muté) ;
Fin
```

Les fonctions correspondantes à chaque opération du cycle sont :

La Sélection

La sélection est appliquée afin de favoriser au cours du temps les individus les mieux adaptés, à les faire se produire. Dans notre application, nous avons utilisé la sélection par tournoi. Elle se fait en comparant des paires d'individus tirées aléatoirement de la population, pour sélectionner le meilleur.

Le code suivant illustre cette opération :

```
Individu Tournament Selection()
Début
    Entier p1, p2, taille ; // taille  représente la taille de la population
    Répéter
        Tirer aléatoirement deux nombre p1et p2 ;
    Jusqu'à (p1≠p2) et (p1<=taille) et (p2<=taille)
    Si (Fitness (population [p1])> Fitness (population [p2]))
    Alors  Retourner (population [p1]); //l'individu correspond à l'indice p1
    Sinon  Retourner (population [p2]); //l'individu correspond à l'indice p2
Fin
```

Le croisement

Le croisement est un opérateur très important des Algorithmes Génétiques. Dans notre application, nous avons utilisé le croisement en un point. Ce point est choisi aléatoirement :

```
Individu Croisement (Indiv1, Indiv2)
Début
  Entier k ; //k est le point de coupure
  Entier L ; //la taille d'un individu
  Réel r, pc = La probabilité de croisement ;
  Individu Enfant1, Enfant2 ;
      r = fonction aléatoire (0, 1) ;
      si (r  > pc) Retourner un des deux parents (indiv1 ou indiv2).
      k = fonction aléatoire (0, L) ;
  // Générer l'enfant 1
      Pour  i=0 à k-1   Faire    Enfant1 [i] := Indiv1 [i] ;   Fait
      Pour  i=k à L-1   Faire   Enfant1 [i] :=Indiv2 [i] ;    Fait
  //Générer l'enfant 2
      Pour  i=0 à k-1   Faire    Enfant2 [i] :=Indiv2 [i] ;   Fait
      Pour  i=k à L-1  Faire    Enfant2 [i] :=Indiv1 [i] ;   Fait
   Retourner le meilleur des 2 enfants admissible;
Fin.
```

La mutation

La mutation permet de transformer au hasard le codage d'un individu afin d'apporter une certaine diversité dans la population et empêcher que celle-ci ne converge trop vite vers un seul type d'individu parfait. Dans notre application nous avons utilisé la mutation par valeur d'un gène où plusieurs selon la probabilité de mutation.

Individu **Mutation** (***Indiv***)
Début
Entier i, L = taille de l'individu;
Double r, pm = Probabilité de mutation ;
r = fonction aléatoire (0, 1) ;
Pour i = 0 à (L-1)
Faire
Si(r < pm)
Alors
Si(r < 0.5)
Alors
- ***Muter la case opérateur*** dans le cas où l'attribut est Numérique mais avec une autre valeur différente de La valeur actuelle ;
- ***Muter la case valeur*** de l'attribut par une autre valeur parmi les valeurs possibles de l'attribut ***i*** mais elle doit être différente de la valeur actuelle ;
- Si (l'attribut ***i*** n'est pas active)
Alors Activer l'attrubut i

Sinon Muter la case Active/Désactive ;
Fsi
Fait
Fin

Le remplacement

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.

2.5. Construction du classifieurs [BEN07]

Notre classifieur contient les meilleures règles extraites par l'algorithme génétique, la fonction suivante illustre le fonctionnement de cette opération.

```
Classifieur Construction_du_classifieur (La base d'apprentissage)
Début
    Classifieur = {vide} ;
    Nb_Inst = le nombre d'instances de la base d'apprentissage;
    Booléen ; couvre ;
    Tant que(Le critère d'arrêt n'est pas atteint)
    Faire
        Appliquer l'AG() ;
        Régle R = le meilleur individu obtenu par l'AG ;
        Couvre = faux ;
       // Chercher toutes les instances couvertes par cette règle
        Pour i=1 à Nb_Inst
        Faire
            Si(Instance_Est_Couverte (instance inst_i, régle R))
            Alors  Supprimer cette instance de la base d'apprentissage;
                   Couvre = vrai ;
        Fait ;
        Si (Couvre = vrai)  Alors  Ajouter la règle au classifieur ;
    Fait ;
   Retourner (Classifieur) ;
Fin
```

Le critère d'arrêt du classifieur :

Le critère d'arrêt de notre classifieurs est la conjonction des deux conditions suivantes :

La base d'apprentissage est vide Où bien au bout un certain nombre d'itération car l'algorithme risque de se dérouler de manière infinie dans le cas où il existe des instances dans la base d'apprentissage qui ne peuvent être couvertes par aucune règle.

La fonction Instance_Est_Couverte :

Est une fonction qui permet de tester si une instance est couverte par la régle extraite par l'AG. Nous dirons que l'instance ***inst*** est couverte par la règle ***R*** si et seulement si la partie antécédente de l'instance ***inst*** satisfait la partie prémisse de la règle ***R***, peut importe la classe.

Boolean Fonction **Instance_Est_Couverte** (Instance *inst* , Règle *R*)
Début
    Entier Nb = nombre d'attributs de la partie condition de la règle ***R***;
    Pour i = 1 à (Nb-1)
    Faire
        Si (le $i^{ème}$ antécédent de l'instance ***inst*** ne satisfait pas le $i^{éme}$ attribut de la règle ***R***) Alors retourne ***Faux***
    Fait
    Retourne ***Vrai***
Fin

3. Adaptation de la recherche Tabou à la classification

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
- Implantation de la liste des tabous
- Application de la recherche Tabou
- Construction du classifieurs

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 avec l'Algorithme Génétique.

Le recours à l'Algorithme Génétique pour initialiser la Recherche Tabou 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()
Début
        Appliquer l'AG ;
        Individu indiv = le meilleur individu obtenu par l'AG ;
        Retourner (indiv) ;
Fin
```

3.2. Génération de voisinage

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 **Create_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 La 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

3.3. Implantation de la liste des tabous

Dans la Recherche Tabou on examine un ensemble de voisinage N(s) et on retient la meilleure s' même si f(s')<f(s). La recherche Tabou ne s'arrête donc pas à la première solution trouvé. 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 Tablou par une liste des individus 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 dégagé dans le cas où il n'y plus des places vide.

La fonction suivante montre comment gérer la liste Tabou

```
Individu Ajouter_Tabou (Liste_Tabou, Indiv)
Début
    Si (La liste Tabou n'est pas plein)
    Alors
        Liste_Tabou = Liste_Tabou + Indiv ;
    Sinon
        Liste_Tabou = Liste_Tabou – mauvais_indiv(Liste_Tabou) ;
        Liste_Tabou = Liste_Tabou + Indiv ;
    Retourner   (Liste_Tabou);
Fin
```

3.4. Application de la recherche Tabou

Le processus de la Recherche Tabou consiste, pour chaque itération à choisir la meilleure solution qui n'est pas taboue dans le voisinage de la solution courante, même si cette solution n'entraîne pas une amélioration. La fonction suivante illustre le fonctionnement de la Recherche Tabou.

```
Individu Recherche_Tabou()
Début
    Individu meilleur_ voisin;
    Individu indiv = Générer_Solution_Initiale() ;
    Liste_Tabou = vide ;
    Tant que(Le critère d'arrêt n'est pas atteint)
    Faire
        Ensemble_de_voisins = Create_Voisinage(indiv) ;
        meilleur_voisin =  Meilleur_Voisin(Ensemble_de_voisins);
        Liste_Tabou = Ajouter_Tabou(Liste_Tabou, Indiv);
        indiv = meilleur_voisin ;
    Fait
    Retourner   Meilleur_Individu(Liste_Tabou);
Fin
```

3.5. Construction du classifieurs

Notre classifieur contient les meilleures règles extraites par la Recherche Tabou, la fonction suivante illustre le fonctionnement de cette opération.

```
Classifieur  Construction_du_classifieur (La base d'apprentissage)
Début
    Classifieur = {vide} ;
    Nb_Inst = le nombre d'instance de la base d'apprentissage;
    Booléen ; couvre ;
    Tant que(Le critère d'arrêt n'est pas atteint)
    Faire
        Appliquer la Recherche Tabou() ;
        Régle R = le meilleur individu obtenu par la Recherche Tabou ;
        Couvre = faux ;
        // Chercher toutes les instances couvertes par cette règle
        Pour i=1 à Nb_Inst
        Faire
            Si(Instance_Est_Couverte (instance inst_i, régle R))
            Alors   Supprimer cette instance de la base d'apprentissage;
                    Couvre = vrai ;
        Fait ;
        Si (Couvre = vrai)   Alors   Ajouter la règle au classifieur;
    Fait ;
    Retourner (Classifieur) ;
Fin
```

Le critère d'arrêt du classifieur :

Le critère d'arrêt de notre classifieur est la conjonction des deux conditions suivantes :

La base d'apprentissage est vide, où bien au bout un certain nombre d'itérations, car l'algorithme risque de se dérouler de manière infinie dans le cas où il existe des instances dans la base de d'apprentissage qui ne peuvent être couverte par aucune règle.

4. Adaptation de l'algorithme mémétique à la classification

4.1. Les stratégies d'application de la recherche Tabou

Nous allons intégrer la Recherche Tabou dans l'algorithme génétique en appliquant les stratégies suivantes :

1. Appliquer la Recherche Tabou à toute la population initiale.
2. Appliquer la Recherche Tabou après mutation.
3. Appliquer la Recherche Tabou après croisement.
4. Appliquer la Recherche Tabou à toute la population initiale et après mutation.
5. Appliquer la Recherche Tabou à toute la population initiale et après croisement.
6. Appliquer la Recherche Tabou à toute la population initiale et après mutation et croisement.

Le code suivant montre un exemple d'intégration de la Recherche taboue dans l'algorithme génétique ($6^{ème}$ Stratégie).

```
Individu Algorithme_Mémétique()
Début
    Population[] = Création de la population initiale ;
    Entier  taille = La taille de la population ;

    Pour i=1 à (taille)
    Faire
        // Appliquer la Recherche Tabou à toute la population initiale
        Population[i] = Recherche_Tabou(Population[i]) ;
    Fait

    Tant que(Le critère d'arrêt n'est pas atteint)
    Faire
        // Sélectionner deux individus
        Individus 1 = Tournament_Selection() ;
        Individus 2 = Tournament_Selection() ;
        // Croisement des deux individus sélectionnés
        Enfant= Croisement(Individu 1, individu 2) ;
        // Appliquer la Recherche Tabou après croisement.
        Enfant = Recherche_Tabou(Enfant);
        // Mutation de l'enfant
        Enfant_Muté=Mutation(Enfant) ;
        //Appliquer la Recherche Tabou après croisement.
        Enfant_Muté = Recherche_Tabou(Enfant_Muté);
        Remplacement(Enfant_Muté) ;
    Fait

    Meilleur_indiv = Le meilleur individu de la population ;
    Retourner (Meilleur_indiv) ;
Fin
```

5. L'approche parallèle proposée

Dans cette partie nous présentons notre modèle parallèle basé sur les algorithmes mémétiques, est un modèle parallèle synchrone de la forme maître-esclave sur une population unique résidant sur un seul processeur appelé le maître. Ce dernier effectue, les différents opérateurs génétiques de l'algorithme sur la population et distribue ensuite l'évaluation des individus aux processeurs esclaves.

5.1. Le modèle parallèle synchrone maître-esclave

Dans ce modèle on à une machine maitre, et les autres sont des machines esclaves. Dans chacune des machines esclaves l'algorithme de la Recherche Tabou s'exécute comme expliqué dans la partie précédente. La différence est que les machines esclaves, ne génèrent pas leurs individus de départ aléatoirement où par AG, ils les reçoivent de la machine maitre.

Dans la machine maitre l'algorithme Mémétique s'exécute et le maitre est le seule machine qui a la population globale dans sa propre mémoire. Le processeur maître fait la sélection des individus puis le croisement et la mutation ensuite il les distribue sur les esclaves, pour faire la Recherche tabou sur les individus et il attend les résultats de tous les esclaves pour faire l'opération de remplacement.

La base d'apprentissage est la seule chose commune entre la machine maitre et les 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 par tous, la machine maitre envoie le meilleur individu sélectionné après chaque génération de l'algorithme Mémétique à toutes les machines esclaves pour qu'ils mettent à jours leurs bases d'apprentissages.

5.2. Communication maitre/esclave

Les différents types de communications peuvent être résumés comme suit :

5.2.1. Le maitre vers l'esclave

Les différentes informations communiquées par le maitre vers un esclave sont :

1. L'individu résultant des opérations de sélection, croisement et mutation
2. Le meilleur individu de chaque génération

5.2.2. Un esclave vers le maitre

Une fois qu'il termine sa tache, l'esclave envoie à son maitre la meilleure solution qu'il a trouvée.

5.3. Synchronisation

Dans ce modèle la synchronisation est au niveau du lancement des différents processus esclaves. Au début le maitre les lance tous, puis à chaque fois que le maitre a besoin de faire la Recherche tabou sur un ensemble d'individu il les distribue sur les processus esclaves et il attend tous les résultats puis il les remplace dans la population.

5.4. Algorithme du maitre

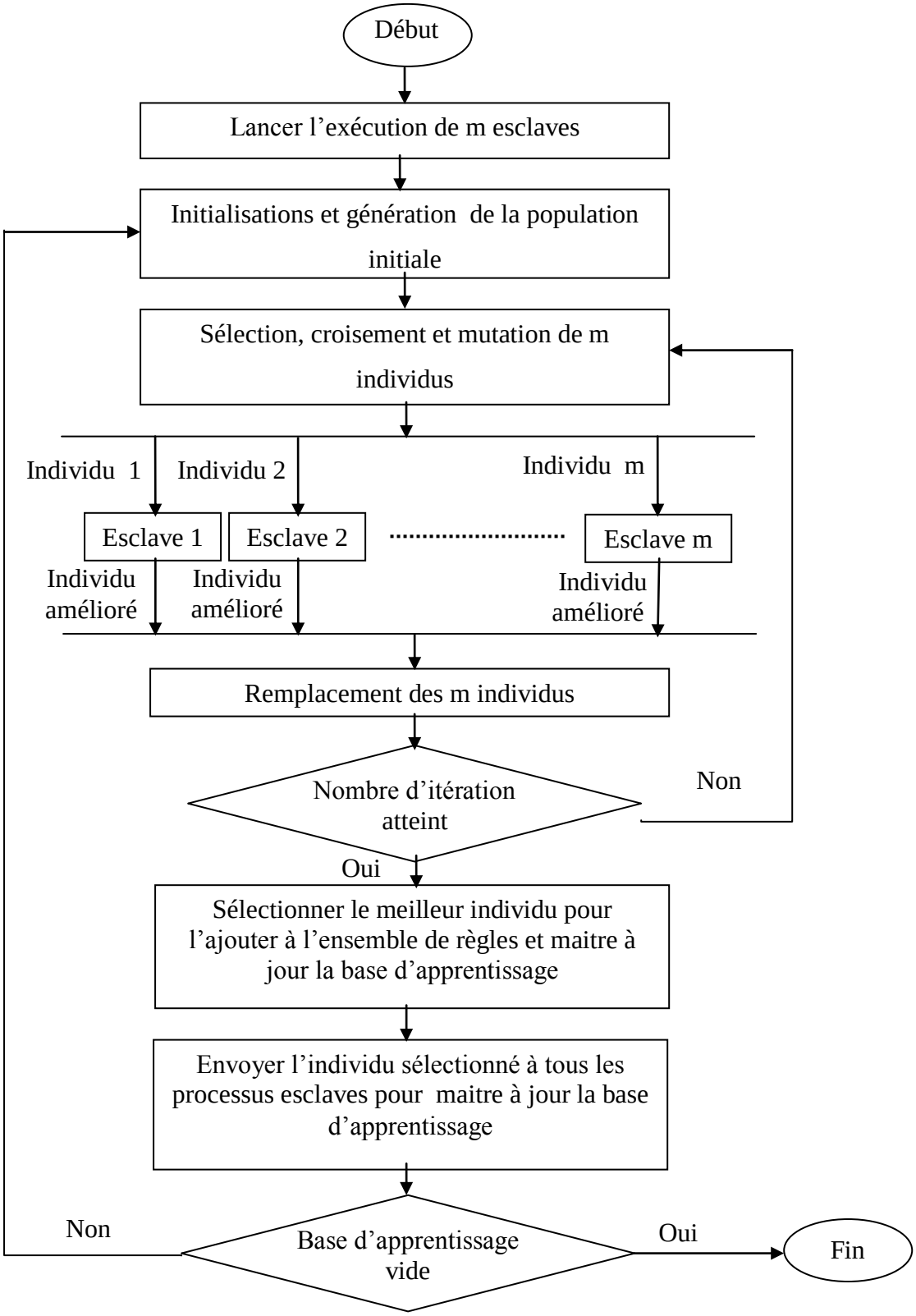


Figure 14 : Organigramme de l'algorithme du maitre

5.5. Algorithme des esclaves

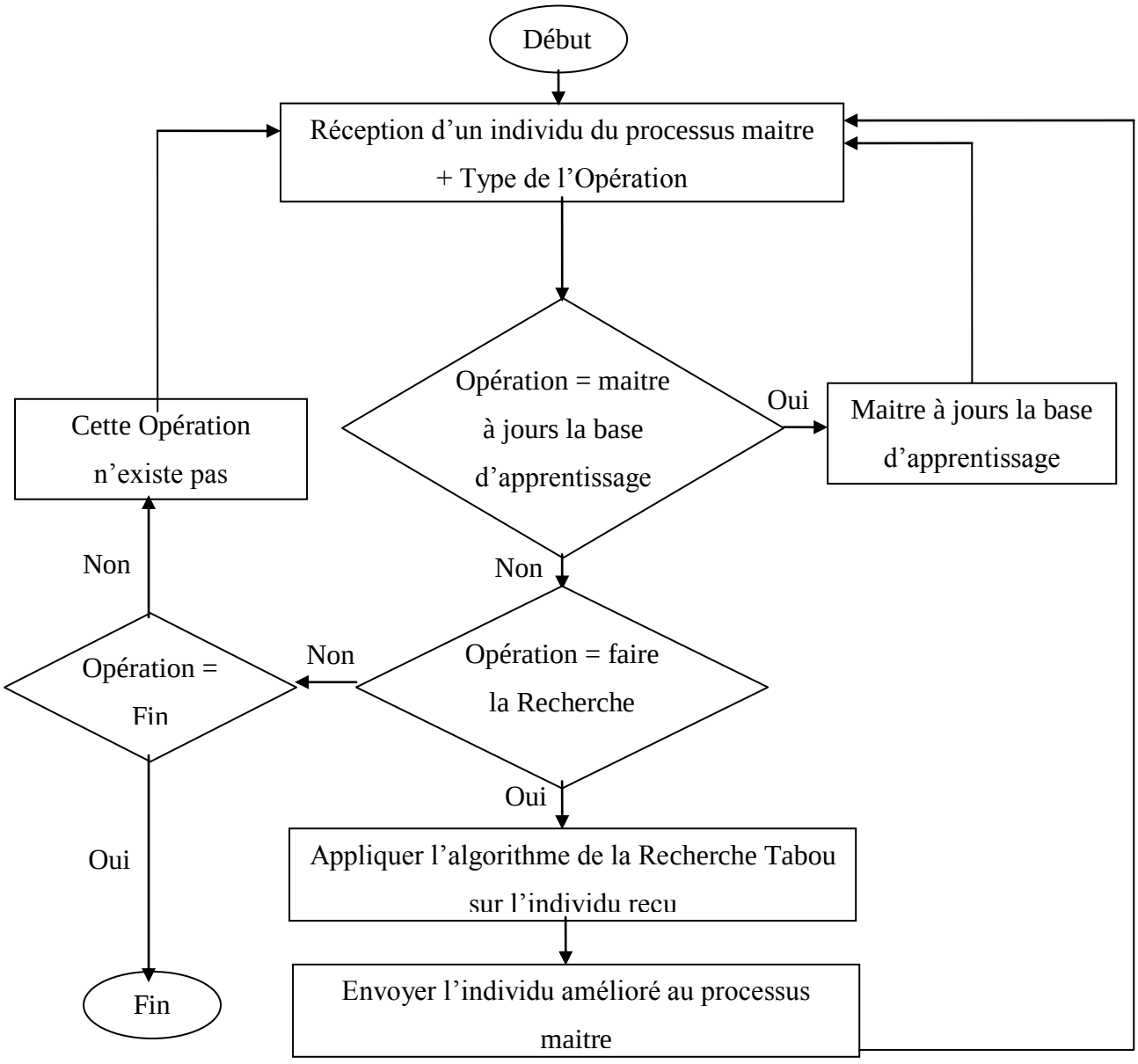


Figure 15 : Organigramme d'algorithme des esclaves

6. Conclusion

Dans ce chapitre nous avons présenté en détail notre conception. Nous avons commencé par présenter la conception d'un classifieur basé sur les algorithmes génétiques et un classifieur basé sur la recherche Tabou et un autre basé les algorithmes mémétiques. Ensuite nous sommes passés à la présentation de notre architecture parallèle utilisée. Le chapitre suivant contiendra les résultats des tests effectués sur cette application.

1. Introduction

Afin de tester l'efficacité de l'application conçue, ainsi que le rôle et l'importance de certains paramètres, nous avons effectué une série de tests que nous résumons dans ce chapitre.

Nous présentons d'abord le langage de programmation utilisé ainsi que la bibliothèque externe Weka, les bases de données de l'UCI et une étude paramétrique décrivant l'influence des paramètres sur la qualité des solutions, En suite nous présentons les résultats obtenus par :

1. L'application du classifieur construit à l'aide d'un algorithme génétique sur quelques benchmarks de l'UCI.
2. L'application du classifieur construit à l'aide de la Recherche Tabou sur quelques benchmarks de l'UCI.
3. L'application du classifieur mémétique sur quelques benchmarks de l'UCI.
4. L'application du classifieur mémétique parallèle sur quelques benchmarks de l'UCI.

Les testes de ce chapitre sont faits sur un PC Pentium 4 de fréquence 3GH et les tests du classifieur mémétique parallèle sont faits sur un réseau en étoile de 10 PC Pentium 4 de fréquence 3GH.

2. Environnement de développement et de test

2.1. Le langage de programation JAVA

On a utilisé le langage de programmation Java 1.6 [RIN 01], est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton employés de Sun Microsystems, présenté officiellement le 23 mai 1995 au SunWorld.

Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modifications. Le langage reprend en grande partie la syntaxe du langage C++.

2.2. La bibliothéque externe WEKA

On a utilisé le logiciel Weka (Waikato Environment for Knowledge Analysis) [RIN 02] qui est une suite de logiciels d'Apprentissage automatique populaire. Weka est écrit en Java, et il est développé à l'université de Waikato, Nouvelle-Zélande.

Il se compose principalement :

- De classes Java permettant de charger et de manipuler les données.
- De classes pour les principaux algorithmes de classification supervisée ou non supervisée.
- D'outils de sélection d'attributs, de statistiques sur ces attributs.
- 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, vérifier son efficacité.
- Invoquer un algorithme sur la ligne de commande.
- Utiliser les classes définies dans ses propres programmes pour créer d'autres méthodes, implémenter d'autres algorithmes, comparer ou combiner plusieurs méthodes.

C'est cette troisième possibilité qui sera utilisée dans notre cas.

2.3. Les Benchmarks de l'UCI

Est une très grande bibliothèque de base de données (Benchmarks) sélectionnées par l'Université de Californie Irine (UCI) [RIN 03] et mise à la disposition de la communauté des chercheurs de Data Mining. 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 d'autre algorithmes.

Nous allons utiliser les bases de données suivantes :

Base de données	Nombre d'instances	Nombres d'attributs	Nombre de class
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

Tableau 1 : Les bases de données utilisées.

2.4. Performance des méthodes de classification

Pour évaluer les performances d'une méthode de classification, En général, on divise l'ensemble de données disponible 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. L'ensemble de test sert pour tester les performances de la méthode en calculant le taux (la précision) de classification correct de l'ensemble des cas. Ce taux est déterminé en divisant le nombre de cas bien classés sur le nombre total des cas testés.

Parfois on est 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.

2.4.1. Technique de validation croisée (cross-validation)

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 saient 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 paramétrique de l'influence des paramètres sur la précision du classifieur

3.1. Influence des paramètres de l'AG sur la précision du classifieur

Le processus de l'Algorithme Génétique est guidé par un certain nombre de paramètres fixés à l'avance. La valeur de ces paramètres influence la réussite ou non de l'algorithme. Ces paramètres sont les suivants [Hau 04] :

- 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, la population peut converger trop rapidement vers un mauvais individu.
- 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 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 classifieur et nous choisissons des tailles différentes de la base d'apprentissage. Puis nous exécutons le classifieur basé sur l'algorithme génétique 10 fois successives pour chaque taille.

Les paramètres PC et PM de l'algorithme génétique sont PC=0.1 et PM=0.8 et les paramètres λ_1,λ_2 et λ_3 de la fonction objectif du classifieur sont λ_1=0.2, λ_2=0.8 et λ_3=0. Les résultats trouvés sont dans le tableau suivant :

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

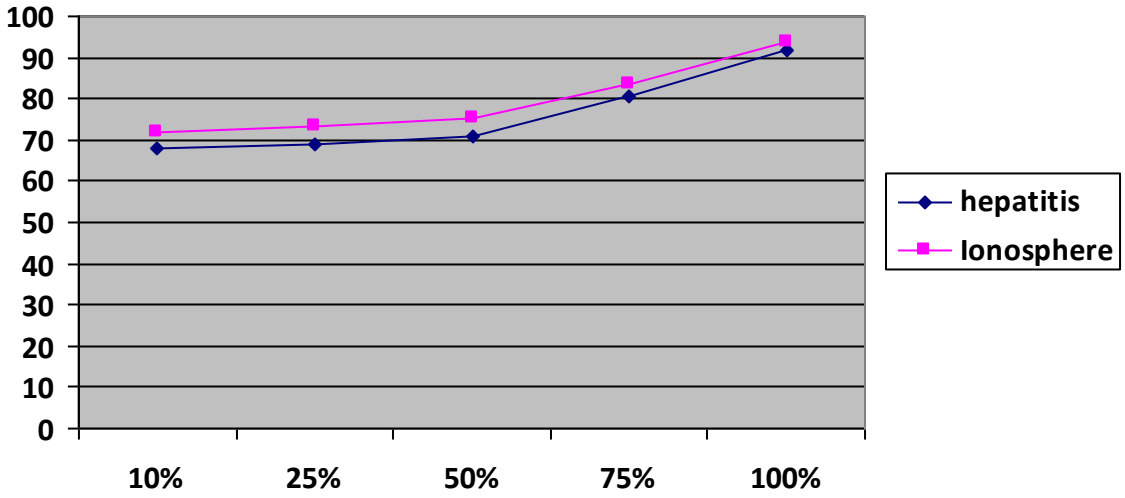


Figure 16 : Influence de la taille d'apprentissage sur la précision du classifieur

À travers les résultats du tableau 2 représentés par les deux graphiques de la figure 16, nous pouvons déduire que l'impact de la taille de la base d'apprentissage est important. En effet, la précision moyenne du classifieur augment 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

Nous allons utiliser les bases de donnés *Hepatitis* et *Ionosphere* pour illustrer l'impact de ce paramètre sur la précisions du classifieur. Nous allons modifier les paramètres λ_1, λ_2 et λ_3, Puis pour chaque cas nous exécutons le classifieur 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 résultats trouvés sont dans le tableau suivant :

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.

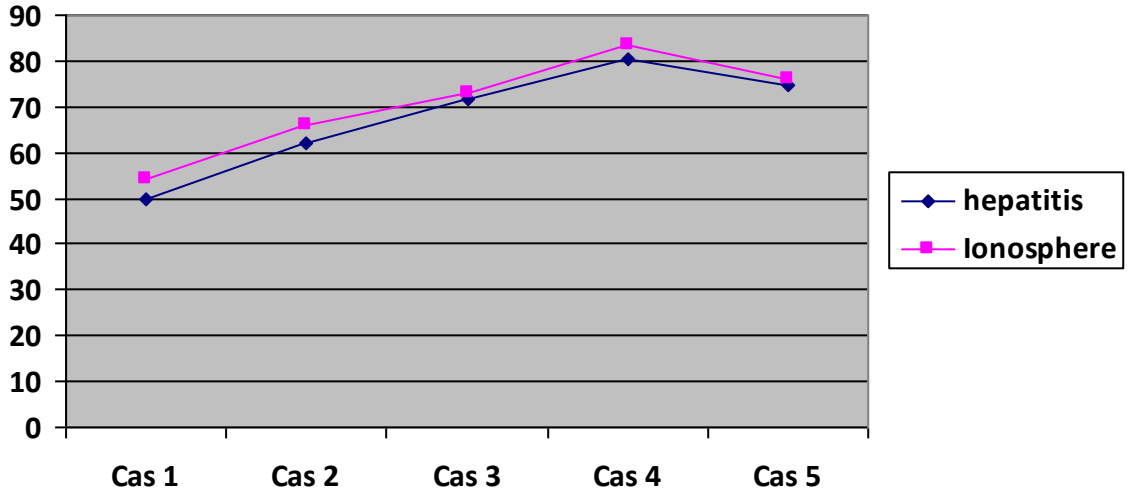


Figure 17 : Influence des paramètres λ_1, λ_2 et λ_3 de la fonction objectif sur la précision du classifieur.

3.1.3. Influence du nombre de générations

Le processus de recherche par les Algorithmes Génétiques pour tous les problèmes testé suit une allure générale illustrée par la courbe de la figure 18. Cette courbe montre l'existence de deux temps de recherche. Un premier temps, se caractérise par une croissance rapide de la précision du classifieur après quelques générations successives, ce qui signifie que la recherche est efficace. Durant un deuxième temps, la recherche commence à se stabiliser, et l'amélioration de la solution se fait très lentement jusqu'à stabilisation de la recherche. L'évolution durant les dernières générations devient inutile.

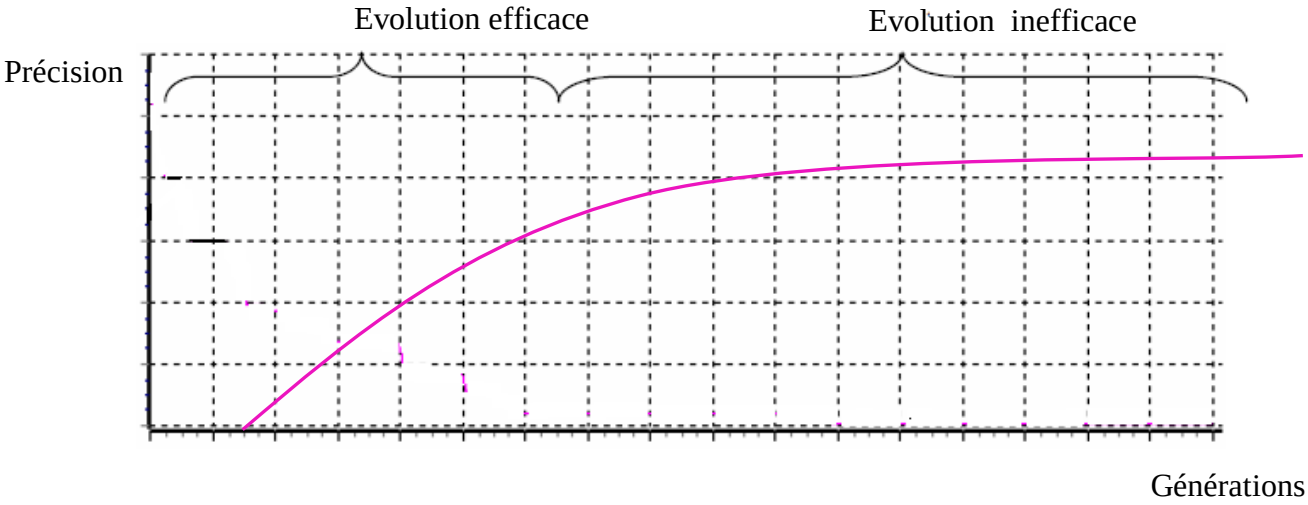


Figure 18 : Amélioration de la solution durant l'évolution des générations de l'Algorithme Génétique.

3.1.4. Influence de la taille de la population initiale

Nous allons utiliser les bases de donnés *Hepatitis* et *Ionosphere* pour illustrer l'influence de ce paramètre sur la précisions du classifieur. Nous allons modifier la taille de la population initiale, Puis nous exécutons le classifieur 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 classifieur sont λ_1=0.2, λ_2=0.8 et λ_3=0. Les résultats trouvés sont dans le tableau suivant :

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

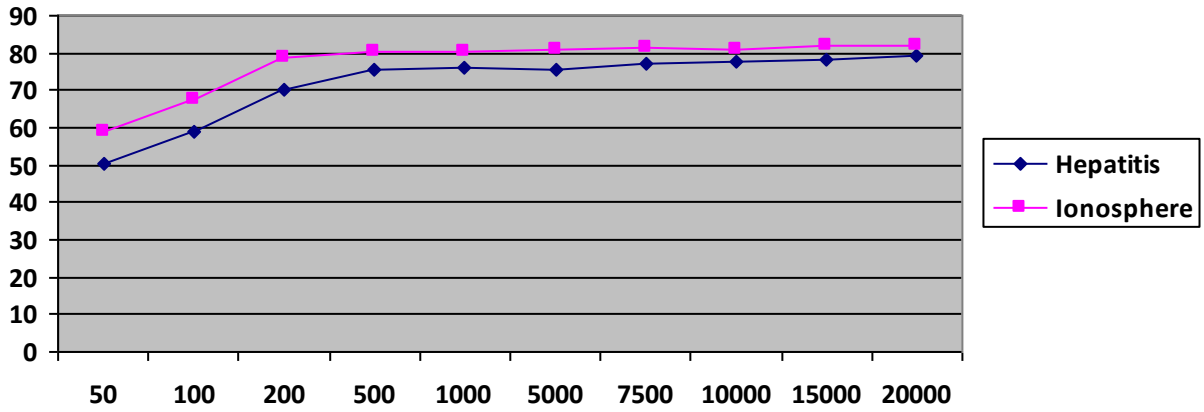


Figure 18 : Effet de la taille de la population sur la précision du classifieur

Sur la courbe de la figure 18, nous remarquons l'existence de deux phases :

- Une première phase où l'ajout de plus d'individus à la population initiale implique l'augmentation de la précision du classifieurs.
- Après une certaine limite, commence la deuxième phase où l'augmentation de la taille de la population n'influence plus sur la précision du classifieurs.

3.1.5. Influence de la diversification de la population initiale

Nous allons utiliser la base de donnés *Hepatitis* pour illustrer l'influence de la diversification de la population initiale sur la précisions du classifieur. On va utiliser deux populations initiales, une diversifiée et une autre aléatoire, En suite nous exécutons le classifieur génétique 10 fois successivement pour chaque population en augmentant le nombre d'itérations à chaque fois.

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 $\lambda 1$, $\lambda 2$ et $\lambda 3$ de la fonction objectif du classifieur sont $\lambda 1=0.2$, $\lambda 2=0.8$ et $\lambda 3=0$. Les résultats trouvés sont dans le tableau suivant :

Algorithme Génétique		
Itérations	**Hepatitis**	
	Population diversifiée	**Population aléatoire**
50	65,38	54,36
100	69,59	57,26
150	72,95	60,15
200	77,56	64,75
250	77,50	63,89
300	78,18	65,67
350	78,50	70,45
400	76,89	73,37
450	79,75	75,60
500	80,21	78,11

Tableau 5 : Résultats obtenus pour des itérations différents

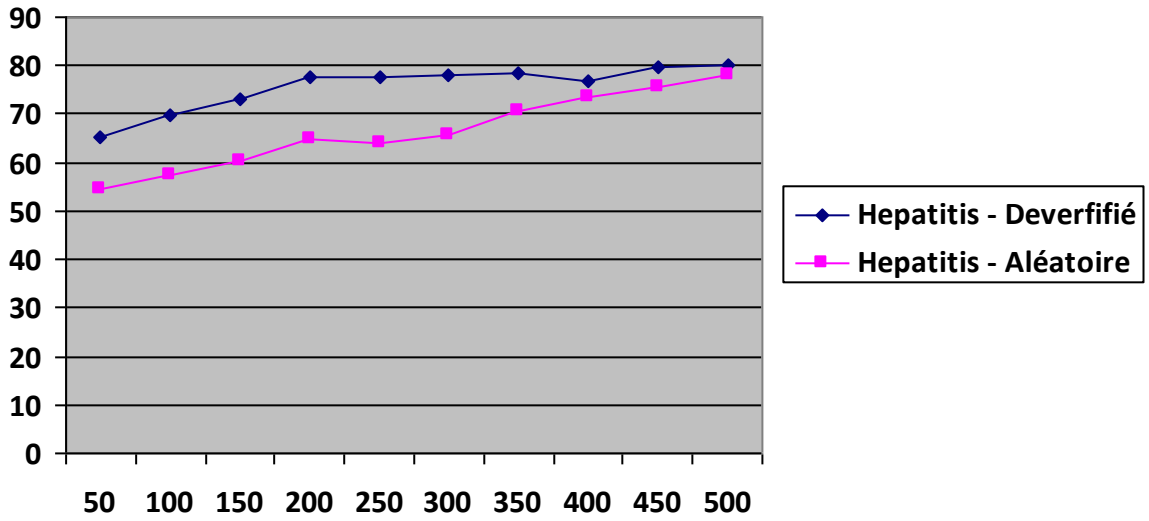


Figure 19 : Effet de la diversification de la population sur la précision du classifieur

D'après les résultats obtenus, nous remarquons que la diversification de la population initiale aide à la convergence rapide vers les meilleurs résultats.

D'après les résultats obtenus par les séries d'expériences réalisées sur l'influence des paramètres de l'AG sur la précision du classifieur, nous allons utiliser les paramètres de l'AG cités ci-dessous (tableau 6) pour la suite des expériences du chapitre.

Paramètres	Valeurs
Taille de la base d'apprentissage	75%
Taille de la base de teste	25%
Taille de la population initiale	200
Nombre d'itération	500
Probabilité de croisement (PC)	0.8
Probabilité de mutation (PM)	0.025
Le paramètre y1 de la fonction objectif	0.1
Le paramètre y2 de la fonction objectif	0.8
Le paramètre y2 de la fonction objectif	0.1

Tableau 6 : Les paramètres utilisés par l'AG

3.2. Influence des paramètres de la Recherche Tabou sur la précision du classifieur

L'efficacité de la méthode Tabou, est déterminée par ses composants principaux : la qualité de la solution initiale, la mémoire Tabou, le nombre d'itération et la fonction de voisinage. Afin d'illustrer l'influence de ces différents paramètres sur le processus de recherche, nous avons effectué une série d'expérimentations.

3.2.1. Influence de la qualité de la solution initiale

Nous allons utiliser la base de donnés *Hepatitis* pour illustrer l'influence de la qualité de la solution initiale sur la précisions du classifieur. On va utiliser deux types d'initialisations différentes, une initialisation avec un AG et un autre aléatoire, En suite nous exécutons le classifieur basé sur la recherche Tabou 10 fois successives pour chaque type d'initialisation, en augmentant le nombre d'itération à chaque fois.

Tous les tests sont effectués dans les mêmes conditions et avec les mêmes paramètres de la méthode.

Recherche Tabou		
Itérations	**Hepatitis**	
	Initialisation avec AG	**Initialisation aléatoire**
50	71,56	57,39
100	63,37	60,18
150	65,78	72,85
200	70,74	55,11
250	62,65	50,69
300	67,89	66,72
350	70,30	72,72
400	66,78	60,34
450	64,99	67,49
500	73,39	63,76

Tableau 7 : Résultats obtenus pour deux types d'initialisations

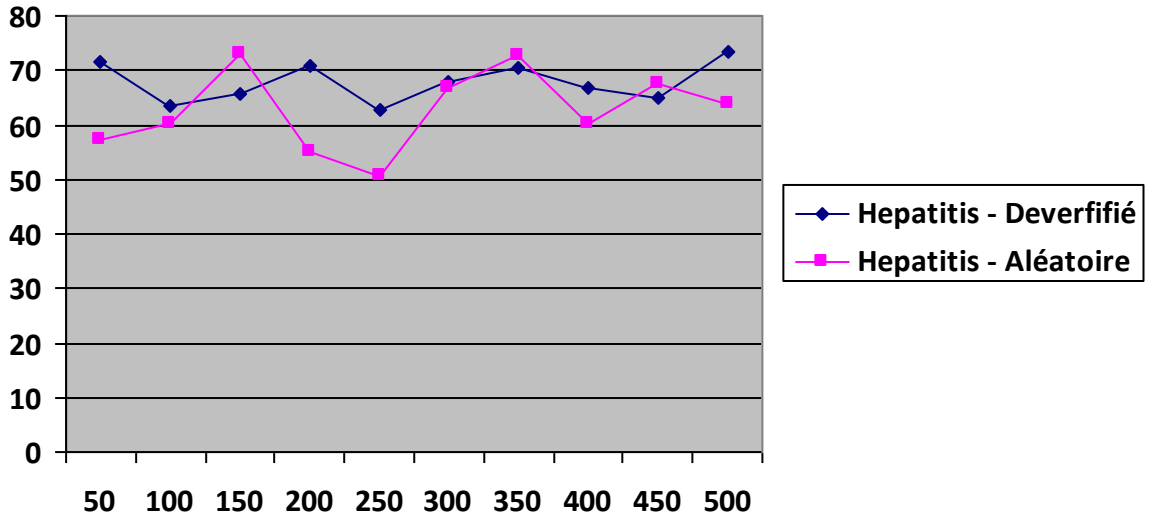


Figure 20 : Effet de la qualité de la solution initiale sur la précision du classifieur

D'après les résultats obtenus de cette série d'expériences, nous remarquons que la qualité de la solution initiale peut influencer la recherche. Ainsi, une mauvaise solution de départ peut empêcher le processus de recherche à explorer les bons endroits de l'espace d'états. Effectivement, dans nos expériences, l'initialisation avec AG nous a fournit la meilleure solution dans la majorité des cas (7/10), contre (3/10) pour l'initialisation aléatoire.

3.2.2. Influence du nombre d'itération

Nous allons utiliser les bases de donnés *Hepatitis* et *Ionosphere* pour illustrer l'influence de ce paramètre sur la précision du classifieur. Nous allons exécutons le classifieur Tabou 10 fois successivement avec un nombre croisant d'itérations. On va utiliser la Recherche Tabou avec une initialisation aléatoire.

Recherche tabou		
Itérations	**Hepatitis**	**Ionosphere**
250	53,79	60,37
500	57,30	58,80
1000	55,68	61,15
2500	58,76	63,96
5000	61,53	65,49
10000	63,18	67,66
20000	60,67	65,78
30000	64,79	67,99
40000	65,89	69,09
50000	68,21	73,17

Tableau 8 Influence du nombre d'itérations sur la précision du classifieur

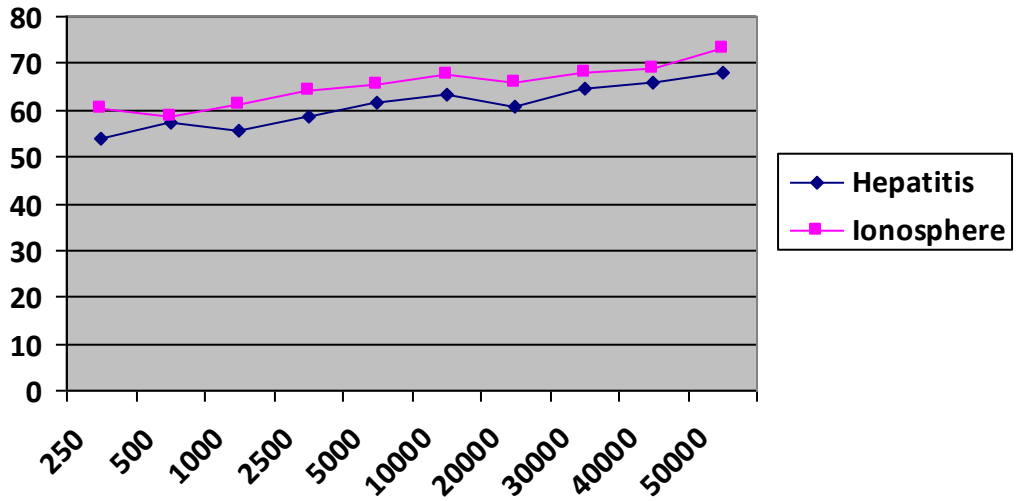


Figure 21 : Effet du nombre d'itérations sur la précision du classifieur

L'observation du graphique de la figure 21, et les données du tableau 8 met en évidence l'importance de l'ajout de plus d'itérations à la recherche. Contrairement à l'Algorithme Génétique qui arrive à une "stagnation" durant les dernières générations, du fait de la convergence de la population, la Recherche Tabou "profite" toujours de toute itération supplémentaire.

3.2.3. Influence de la mémoire Tabou

Afin de découvrir l'effet de la taille de la mémoire Tabou sur la précision du classifieur, nous avons effectué une série d'essais en faisant varier à chaque fois la taille de la mémoire Tabou, Nous allons utiliser la base de donnés *Hepatitis* avec un petit nombre d'itérations (500).

Taille de la mémoire Tabou Tabou	Hepatitis
1	55,49
2	57,74
3	60,67
4	64,70
5	71,53
6	69,27
7	64,95
8	62,45
9	61,79
10	59,86

Tableau 9 Influence de la taille de la mémoire Tabou sur la précision

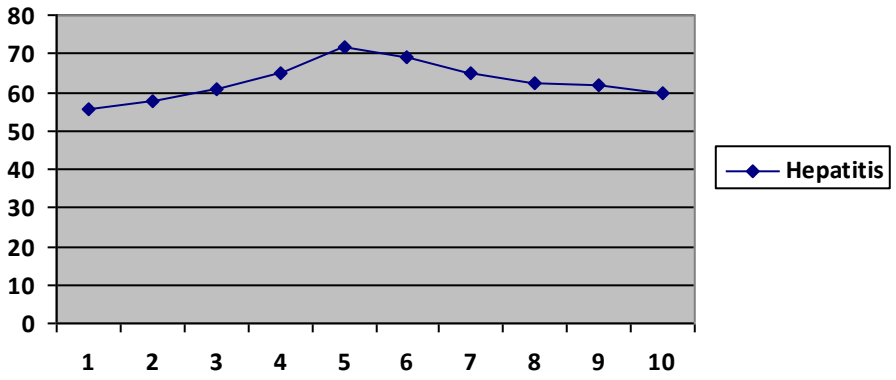


Figure 22 : Représentation graphique des résultats du tableau 9.

Les résultats obtenus permettent de constater que l'emploi d'une mémoire trop courte ou trop longue donne de mauvaises solutions. Une taille trop faible risque de favoriser le cyclage, alors qu'une taille trop grande risque de restreindre le voisinage.

D'après les résultats obtenus par les séries d'expériences réalisées sur l'influence des paramètres de la Recherche Tabou sur la précision du classifieurs, nous allons utiliser les paramètres de la Recherche Tabou cités ci-dessous (tableau 10) pour la suite des expériences du chapitre.

Paramètres	**Valeurs**
Solution Initial créer par un	AG
Taille de la Mémoire Tabou	5
Nombre d'itération	1000

Tableau 10 : Les paramètres utilisés par la recherche Taboue

4. Résultats obtenus par l'approche génétique

A fin d'évaluer les performances de l'approche génétique Nous allons exécuter le classifieur 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.

	Benchmarks					
	Hepatitis	Heart-statlog	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Précision du classifieur (%)	81,20	77,7	86,45	83,33	93,16	62,44
	87,30	70,10	77,76	85,66	88,44	65,35
	81,75	47,50	79,66	77,34	97,38	74,27
	86,66	85,55	80,59	68,45	96,80	66,66
	86,55	77,45	73,50	83,67	98,70	72,03
	73,70	74,11	80,58	77,45	56,89	78,00
	86,80	70,00	68,56	89,11	82,13	63,00
	87,20	81,34	84,77	82,84	83,11	64,56
	46,50	70,15	74,69	86,41	82,00	72,33
	80,22	47,89	74,00	72,44	97,90	63,83
Moyenne	**79,79 %**	**70,18 %**	**78,06 %**	**80,67 %**	**87,65 %**	**68,25 %**

Tableau 11 : Les précisions obtenues par l'approche génétique

	Benchmarks					
	Hepatitis	Heart-statlog	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Nombre de regèles	35	27	83	71	28	3
	41	44	95	65	23	26
	39	23	79	55	21	3
	32	23	113	60	25	10
	28	50	117	73	27	4
	38	49	116	59	21	4
	34	43	117	59	16	5
	39	44	124	60	20	6
	36	48	92	66	14	5
	26	50	91	60	25	5
Moyenne	**34,8**	**40,1**	**102,7**	**62,8**	**22**	**7,1**

Tableau 12 : Nombre de règles de l'approche génétique

5. Résultats obtenus de l'application de la Recherche Tabou

Dans cette partie nous allons exécuter le classifieur basé sur la recherche tabou 10 fois sur les benchmarks de l'UCI données précédemment et nous donnons à chaque fois les précisions obtenues ainsi que le nombre de règles.

	Benchmarks					
	Hepatitis	Heart-statlog	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Précision du classifieur (%)	75,00	77,66	76,36	71,11	93,78	60,50
	87,11	62,29	46,78	75,33	96,30	55,80
	53,30	74,85	34,44	77,45	95,24	61,00
	53,75	70,48	36,12	87,68	19,55	61,00
	73,22	62,13	70,34	67,19	82,45	55,55
	80,00	59,25	84,54	77,95	87,40	59,00
	46,00	70,08	41,11	73,47	9,00	70,77
	66,66	66,33	64,92	65,11	20,43	61,87
	73,33	70,17	73,11	78,70	77,39	58,76
	87,50	70,88	69,55	71,63	11,11	66,79
Moyenne	**69,59 %**	**68,41 %**	**59,73 %**	**74,56 %**	**59,27 %**	**61,10 %**

Tableau 13 : Les précisions obtenues par l'application de l'approche tabou

	Benchmarks					
	Hepatitis	Heart-statlog	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Nombre de regèles	17	13	64	40	34	4
	11	37	37	41	22	3
	14	13	56	50	34	4
	18	16	44	54	23	7
	10	21	70	32	27	2
	19	17	50	39	27	6
	10	25	47	46	54	6
	18	24	68	47	49	3
	17	31	56	44	23	2
	12	28	57	37	33	5
Moyenne	14,6	22,5	54,9	43	32,6	4,2

Tableau 14 : Nombre de règles de l'approche tabou

6. Résultats obtenus par l'approche mémétique

L'efficacité de l'approche mémétique, est déterminée par la stratégie d'hybridation de la Recherche Tabou dans l'Algorithme Génétique. Afin de trouver la meilleure stratégie d'hybridation, nous avons effectué une série d'expérimentations sur toutes les stratégies suivantes :

Stratégie 1 : Appliquer la Recherche Tabou à toute la population initiale.

Stratégie 2 : Appliquer la Recherche Tabou après croisement.

Stratégie 3 : Appliquer la Recherche Tabou après mutation.

Stratégie 4 : Appliquer la Recherche Tabou à toute la population initiale et après croisement.

Stratégie 5 : Appliquer la Recherche Tabou à toute la population initiale et après mutation.

Stratégie 6 : Appliquer la Recherche Tabou à toute la population initiale et après mutation et croisement.

Nous allons exécuter le classifieur mémétique sur une base de données 10 fois pour chaque stratégie, En suite nous calculons la moyenne des 10 résultats trouvés. Les résultats obtenus sont dans le tableau suivant :

Benchmarks	**Stratégies**					
	Stratégie 1	**Stratégie 2**	**Stratégie 3**	**Stratégie 4**	**Stratégie 5**	**Stratégie 6**
Hepatitis	78,68	85,50	84,60	83,77	85,94	83,54
Hert-statlog	73,06	83,66	90,66	85,56	89,67	91,89
Segment-challenge	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
Diabestes	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 15 : Résultats Obtenus des différentes stratégies

D'après les résultats obtenus par les séries d'expériences réalisées, nous remarquons que la troisième stratégie d'hybridation (appliquer la Recherche Tabou après mutation) donne des bons résultats. Donc on va utiliser cette stratégie pour la suite des tests.

Nous allons exécuter le classifieur mémétique basé sur la 3ème stratégie d'hybridation, 10 fois pour chaque benchmarks (base de données) et nous donnons à chaque fois les précisions obtenues ainsi que le nombre de règles. Les résultats obtenus sont dans les tableaux suivants :

	Benchmarks					
	Hepatitis	Heart-statlog	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Précision du classifieur (%)	87,57	89,98	93,15	97,77	98,80	79,88
	87,73	93,66	94,99	96,90	99,50	76,00
	87,11	91,33	93,80	95,37	98,11	83,78
	86,66	92,79	93,60	93,10	99,23	83,00
	81,11	89,91	95,89	90,96	98,74	81,03
	81,23	88,00	93,45	92,39	99,35	80,49
	87,50	95,38	94,77	97,40	99,33	81,93
	80,00	88,29	93,00	96,84	99,49	77,44
	73,35	89,67	93,88	96,50	98,84	80,50
	93,76	87,58	94,00	94,20	99,46	83,11
Moyenne	**84,60 %**	**90,66 %**	**94,05 %**	**95,14 %**	**99,09 %**	**80,72 %**

Tableau 16 : Les précisions obtenues par l'approche mémétique

	Benchmarks					
	Hepatitis	Heart-statlog	Segment-challenge	Ionosphere	Kdd-train	Diabetes
Nombre de regèles	8	15	45	16	5	3
	10	22	44	8	12	3
	6	23	40	3	7	3
	12	21	42	11	6	4
	11	21	40	3	6	3
	9	21	41	9	7	3
	9	19	47	10	8	3
	4	21	47	15	3	3
	11	22	46	10	6	5
	11	19	41	4	10	4
Moyenne	**9,1**	**20,4**	**43,3**	**8,9**	**7**	**3,4**

Tableau 17 : Le nombre de règles de l'approche mémétique

À travers les tests effectués, On a pu constater que la précision du classifieur basé sur l'approche génétique est meilleure que celle basé sur l'approche Tabou, par contre le nombre de règles dans l'approche tabou est meilleur que celle de l'approche génétique.

Nous avons constaté aussi que l'hybridation de la Recherche Tabou avec l'Algorithme Génétique donne de bons résultats en termes de précision et nombre de règles par apport à l'approche génétique et l'approche Tabou.

Enfin, comme le principal inconvénient des algorithmes mémétique est leur temps d'exécution, le traitement parallèle de la recherche Tabou des différents individus réduit considérablement le temps de calcul.

7. Résultats obtenus par l'approche mémétique parallèle

Pour évaluer les performances de notre algorithme parallèle conçu, nous allons effectuer une série de tests sur un réseau en étoile de 10 PC Pentium 4 de fréquence 3GH. Nous allons prendre la base de données *Hepatitis* pour voir l'accélération du classifieur basé sur l'algorithme mémétique parallèle en augmentant à chaque fois le nombre des machines esclave.

Les paramètres PC et PM de l'algorithme mémétique sont PC=0.025 et PM=0.8 et les paramètres λ_1,λ_2 et λ_3 de la fonction objectif du classifieur sont λ_1=0.1, λ_2=0.8 et λ_3=0,1 et les paramètres taille de la mémoire Tabou et le nombre d'itération de la recherche Tabou sont 5 et 300.

Les résultats trouvés sont dans le tableau suivant :

Nombre des processus esclaves utilisés	Temps d'exécution
1	21m 25s
2	11m 37s
3	7m 30s
4	5m 8s
5	4m 8s
6	3m 55s
7	3m 32s
8	3m 19s
9	3m 12s
10	3m 10s

Tableau 18 : Résultats obtenus pour un nombre d'esclaves différents

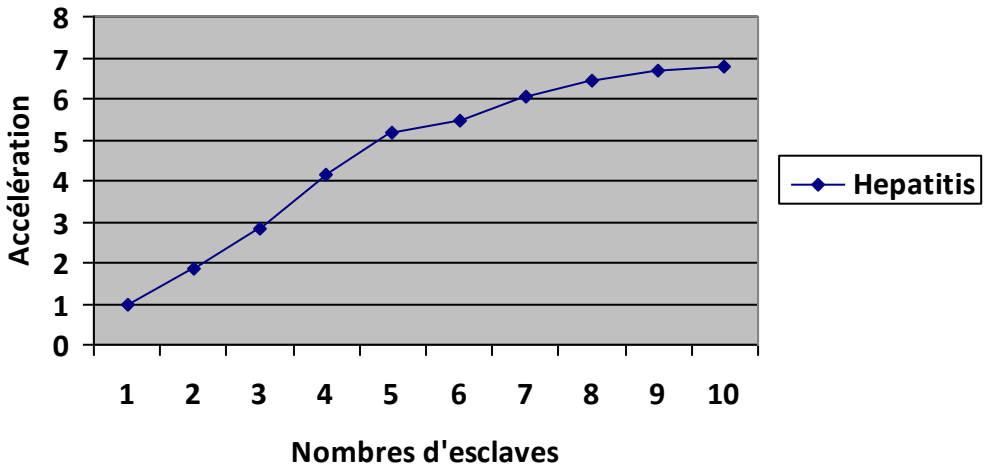


Figure 23 : Représentation graphique des résultats du tableau 18

Du point de vue de l'accélération, on remarque d'après le graphique de la Figure 23 que l'algorithme mémétique parallèle donne des bons résultats, à chaque fois qu'on augmente le nombre des esclaves l'accélération augmente aussi. Globalement, on constate que l'accélération commence à diminuer lorsque le nombre des esclaves dépasse 10 ce que peut être justifié par l'augmentation du coût de communication entre les esclaves et le maitre, le fait que le nombre des esclave augment la taille des messages échangés entre les esclaves et le maitre augmente aussi, par conséquent, le coûts de communications freine l'accélération.

Pour pousser encore plus les limites de notre algorithme mémétique parallèle développé, on a augmenté la taille de la population et le nombre d'itération de l'algorithme mémétique. Il a été constaté que le comportement de l'algorithme mémétique parallèle n'est pas le même, la qualité de la précision du classifieur est meilleure mais l'accélération diminue. On peut ainsi déduire une relation directe entre la taille de la population et le nombre d'itération et la fréquence d'échange.

La parallélisation des algorithmes mémtiques est une voie intéressante pour obtenir des résultats de qualité dans des temps raisonnables. Cependant, il faut savoir définir les paramètres appropriés pour en tirer le meilleur profit. De plus, il a été démontré que l'augmentation du nombre des esclaves est bénéfique jusqu'à un certain point au-delà. duquel aucune amélioration de la qualité n'est possible.

8. Conclusions

Nous avons présenté dans ce chapitre les résultats obtenus par l'application de l'approche génétique, l'approche taboue et l'approche mémétique sur le problème de classification et nous avons obtenu des classifieurs de qualité avec une haute précision et avec un petit nombre de régles.

À travers les tests effectués, On a pu constater que les résultats du classifieur basé sur l'approche génétique sont meilleurs que ceux basés sur l'approche Tabou, et l'hybridation de ces deux approches (pour avoir l'approche mémétique) a donné de bons résultats en termes de précision et nombre de règles par apport à l'approche génétique et l'approche Tabou.

Les résultats de la parallèlisation de l'algorithme mémétique ont montré l'efficacité de notre architecture parallèle, nous avons pu diminuer le temps d'exécution de l'algorithme mémétique.

Conclusion générale et perspectives

L'information numérique étant de plus en plus présente dans le monde d'aujourd'hui, il est crucial de l'exploiter de manière optimale. Le data mining a pour but de fournir des méthodes formelles et automatiques permettant d'y arriver au mieux. L'une de ces tâches est la classification supervisée toute fois cette classification est un problème combinatoire classé NP-Complet.

Dans ce travail, nous nous sommes d'abord proposé de situer cette tâche au sein du data mining afin de voir son utilité et ses applications pratiques, ensuite nous nous sommes intéressés à la classification et les méthodes de classification de manière plus détaillée et nous avons abordé la grande complexité du problème et déduit que l'utilisation des métaheuristiques était inévitable.

Puis nous avons détaillé les méthodes d'optimisations combinatoires les plus populaires et leurs caractéristiques. Nous avons abordé plus en détail les approches évolutives, parmi lesquelles les algorithmes génétique et les algorithmes mémétiques. Ces derniers étant des algorithmes génétiques hybridés avec des méthodes de recherches locales et nous avons étudié toutes les questions auxquelles on est confronté lors de la conception d'une approche mémétique pour la résolution d'un problème complexe. D'autre coté cette hybridation augmente le temps d'exécution, en conséquence il fallait améliorer ce temps d'execution sans diminuer la qualité des résultats, ce qui fait l'objectif du quatrième chapitre où nous avons détaillé les trois principales stratégies de parallélisations des algorithmes génétiques.

Nous avons donc utilisé deux approches pour la construction d'un classifieur. La première approche est l'approche génétique et la deuxième est l'approche mémétique dans laquelle nous avons intégré la recherche taboue en appliquant plusieurs stratégies d'intégration dans le but de voir les performances de ces stratèges et leurs effets sur la qualité de classifieur obtenu. Puis nous avons parallélisé le classifieur basé sur l'approche mémétique suivant le modèle maître-esclave Synchrone.

Afin de montrer l'efficacité des approches utilisées dans la résolution du problème de classification, plusieurs expérimentations sont effectuées sur des Benchmarks de l'UCI, À travers les tests effectués, On a pu constater que la précision du classifieur basé sur l'approche génétique est meilleure que celle basé sur l'approche Tabou, par contre le nombre de règles dans l'approche tabou est meilleur que celle de l'approche génétique. Nous avons constaté

aussi que l'hybridation de la Recherche Tabou avec l'Algorithme Génétique donne de bons résultats en termes de précision et de nombre de règles par apport à l'approche génétique et l'approche Tabou, Ceci est principalement du aux performances de la recherche tabou qui améliore l'exploration de l'espace de recherche.

Nous avons montré aussi que la parallélisation des algorithmes mémétiques est une voie intéressante pour obtenir des résultats de qualité dans des temps raisonnables. Cependant, il faut savoir définir les paramètres appropriés pour en tirer le meilleur profit. De plus, il a été démontré que l'augmentation du nombre des esclaves est bénéfique jusqu'à un certain point au-delà duquel aucune amélioration de la qualité n'est possible.

À fin de compléter notre travail nous proposons les perspectives suivantes :

1. Diminuer les messages échangés entre les esclaves et le maitre, pour améliorer l'accélération de l'algorithme.
2. Essayer de trouver des nouvelles architectures réseaux pour ces algorithmes évolutionnaires.
3. Essayer d'appliquer la parallèlisation hiérarchique qui est une nouvelle méthode de parallèlisation des algorithmes évolutionnaires. La parallèlisation hiérarchique utilise deux méthodes de parallèlisation en hiérarchie à deux niveaux. Souvent, le niveau externe représente le modèle des îles et le niveau interne représente une autre méthode.

Bibliographie

[AGR 94] : R.Agrwal et A. Srikant. Fast algoritms for mining assosiations rules. Proc Pp 787-499, VLDB 1994.

[ANA 98] : S. Anand et A. Buchner. « Decision support using Data Mining. » Financial Times Pitman Publishers, Londres. 1998.

[Ast 96] : Astro Teller et Manuela Veloso. PADO : « A new learning architecture for ob-ject recognition ». Dans Symbolic Visual Learning, pages 81–116. Oxford University Press, 1996.

[BAB 09] : BABA-ALI Sadjia, MAHANI Aouatef « Une approche mémétique pour l'extraction des connaissances » these de magister , USTHB, 2009.

[Bac 04] : J. Bacardit, « Pittsburgh Genetic-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time »

[Bac 03] : Bacardit J., Garrell J. M., « Evolving Multiple Discretizations with Adaptive Intervals for a Pittsburgh Rule-Based Learning Classifier System », in E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S.Wilson,M. Har- man, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, J. Miller (eds), Genetic and Evolutionary Computation – GECCO-2003, Springer-Verlag, Berlin, p. 1818-1831, 2003.

[Bac 98] : Bachelet, V., Hafidi, Z., Preux, P. et Talbi, E.-G. (1998). "Vers la coopération des métaheuristiques." Calculateurs parallèles, réseaux et systèmes répartis 10(2).

[Bea 94] : Beasley D. et J. Heitkoetter, (1994), « The Hitch-Hiker's Guide to Evolutionary Computation : a list of Frequently Asked Questions (FAQ) », USENET : comp.ai.genetic.

[Bele 90] : Belew B., McInerney J. et Schraudolph N., « Evolving Networks : Using the Genetic Algorithms with Connectionist Learning » . CSE Technical Report CS90-174, Computer Science, UCSD, 1990.

[BEN 07] S. Benkhider & al, « A new generationless parallel evolutionary algorithm for combinatorial optimization" , proceeding of the IEEE CEC2007.

[BER 97] : M. J. A. Berry et G. Linoff. « Data mining : techniques appliquées au marketing, à la vente et au service client. » InterEditions. 1997.

[BLU 03] : Christian Blum, Andrea Roli. « Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison » . ACM Computing Survey, Vol. 35 No 3, Sept. 2003.

[BOU 07] : M. Boudjemia et A. Bekri. « Application d'un algorithme évolutif au problème de la classification non supervisée ». Mémoire d'ingénieur. INI. 2007.

[CAB 98] : P. Cabena, P. Hadjinian, R. Stadler, J. Verhees et A. Zanasi. « Discovering Data Mining : From concepts to implementation. » Editions Prentice Hall. 1998.

[Can 98] : Cantú-Paz, E. (1998). "A survey of parallel genetic algorithms." Calculateurs parallèles, réseaux et systèmes répartis 10(2): 141-171.

[CIO 07] : K. J. Cios, W. Pedryecz, R. W. Swinniarsky et L. A. Kurgan. « Data Mining : A Knowledge Discovery Approach .» Editions Springer Science. 2007.

[Cra 98] : Crainic, T. G. et Toulouse, M. (1998). Parallel Metaheuristics. Fleet Management and Logistics. T. G. C. a. G. Laporte. Norwell, MA., Kluwer Academic: 205-251.

[Cun 01] : Cung, V.-D., Martins, S. L., Ribeiro, C. C. et Roucairol, C. (2001). Strategies for the parallel implementation of metaheuristics. Essays and Surveys in Metaheuristics. C. C. R. a. P. Hansen, Kluwer: 263-308.

[Duv 00] : Duvivier D. Etude de l'hybridation des méta-heuristiques, Application à un problème d'ordonnancement de type jobshop, Thèse de Doctorat, Université du Littoral Côte d'Opale, LIL, Calais 2000.

[FAY 96] : U. Fayyard, G. Piatesky-Shapiro, P. Smyth et R. Uthurusamy. « Advances in Knowledge Discovery and Data Mining. » AAAI Press, Cambridge. 1996.

[Fog 89] : Fogarty T.C., Varying the probability of mutation in the genetic algorithm. In Schaffer, J. D. Ed. Proceedings of the Third International Conference on Genetic Algorithms (Morgan Kaufmann): 104-109, 1989.

[FRE 00] : A.A. Freitas Undertanding the crucial differences between classification ans descovery of assosiation rules, a position paper.ACM SIGKDD Explorations (ACM 2000), pp 65-69, 2000.

[FRI 96] : FRIESLEBEN, B. and MERZ, P. (1996), "*A genetic local search algorithm for solving the symmetric and assymetric travelling salesman problem"*. In Proceedings of the 1996 {IEEE} Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ.

[Gar 79] : Garey, M. S. et Johnson, D. S. (1979). Computer and Intractability : A Guide to the Theory of NP-Completeness. New York, W.H. Freeman and Co.

[Gér 03] : Gérard P., Sigaud O., « Designing Efficient Exploration with MACS: Modules and Function Approximation », Proceedings of the Genetic and Evolutionary Computation Conference 2003 (GECCO'03), Springer-Verlag, Chicago, IL, p. 1882-1893, july, 2003.

[Gol 88] : Golberg D. E., Holland J. H., « Guest Editorial : Genetic Algorithms and Machine Learning »,Machine Learning, vol. 3, p. 95-99, 1988.

[Gol 89] : Goldberg D.E., « Genetic Algorithms in Search, Optimization and Machine Learning ». Addison Wesley, Massachusetts, 1989.

[Gol 94] : Goldberg, D. (1994), Algorithmes génétiques, Paris, Addison Wesley.

[HAO 99] : Jin-Kao Hao, Philippe Galinier, Michel Habib. « Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes ». Revue d'intelligence artificielle, 1999.

[Hau 04] : Randy L. Haupt & Sue Ellen Haupt, practical genetic algorithms, 2nd ed. John Wiley & Sons, Inc., New Jersey 2004.

[Hol 75] : Holland J.H., « Adaptation in Natural and Artificial Systems ». Ann Arbor : The University of Michigan Press, 1975.

[Hol 86] : Holland J. H., « Escaping Brittleness : The possibilities of general-purpose learning algorithms applied to parallel rule-based systems », Machine Learning, An Artificial Intelligence

Ap-proach (volume II), Morgan Kaufmann, 1986.

[Hoo 04] : H.H. Hoos and T. Stützle. Stochastic local search: Foundations and applications. Morgan Kaufmann, 2004.

[JAI 88] : A. K. Jain et R. C. Dubes. « Algorithms for clustering data.» Editions Prentice Hall Advanced Reference Series. 1988.

[Joh 92] : John R. Koza. « Genetic Programming : On the Programming of Computers by Means of Natural Selection ». MIT Press, Cambridge, MA, USA, 1992.

[Jon 75] : Jong, K., « An analysis of the behavior of a class of genetic adaptive Systems ». Ph.D. thesis, University of Michigan, 1975.

[Jon 88] : Jong K.A., « Learning with Genetic algorithm : An Overview.

Machine Learning, vol. 3, pp. 121-138, 1988.

[Koh 95] : Kohavi R. (1995) "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: Cs Mellosh (ed.) Proceedings of the 14th International Joint Conf of Artificial Intelligence, Morgan Kaufmann publishers, Inc: 1137-1143.

[Kov 02] : Kovacs T., « Learning Classifier Systems Resources », Journal of Soft Computing, vol. 6, n° 3-4,p. 240-243, 2002.

[KRA 00] : KRASNOGOR, N. and SMITH, J. (2000), "A memetic algorithm with self-adaptive local search: {TSP} as a case study". In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp 987-994, Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I. and Beyer, H.G. (eds.). Morgan Kaufmann, San Francisco.

[KRA 02] : KRASNOGOR, N., BLACKBOURNE, B., BURKE, E., HIRST, J. (2002), "Multimeme algorithms for protein structure prediction". In *Proceedings of the Parallel Problem Solving From Nature*, pp 769-778, Lecture Notes in Computer Science, Springer.

[KRA 04] : KRASNOGOR, N. (2004), "Self-generating metaheuristics in Bioinformatics: The protein structure comparison case". In *Genetic Programming and Evolvable Machines*, vol 5, pp

181--201. Kluwer academic Publishers

[LAB 03] : N. Labroche. « Modélisation du système de reconnaissance chimique des fourmis pour le problème de la classification non supervisée : application à la mesure d'audience sur Internet. » Thèse de doctorat. Université François Rabelais, Tours. 2003.

[Lan 05] : Landau S., Sigaud O., Schoenauer M., « ATNoSFERES revisited », in H.-G. Beyer, U.-M.O'Reilly, D. Arnold, W. Banzhaf, C. Blum, E. Bonabeau, E. Cantú Paz, D. Dasgupta, K. Deb, J. Foste r, E. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G. Raidl,T. Soule, A. Tyrrell, J.-P.Watson, E. Zitzler (eds), Proceedings of the Genetic and Evolutio-nary Computation Conference, GECCO-2005, ACM Press, Washington DC, p. 1867-1874,june 25-29, 2005.

[Law 66] : Lawrence J. Fogel, A. J. Owens, et M. J.Walsh. « Artificial Intelligence through Simulated Evolution ». John Wiley & Sons, New York, 1966.

[Liu 68] : G. L. Liu. Introduction to combinatorial Mathematics. McGraw Hill, 1968.

[Llo 02] : Llorà X., Garrell J. M., « Co-evolving Different Knowledge Representations with fine-grained Parallel Learning Classifier Systems », inW. B. Langdon, E. Cantu-Paz, K.Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A.Potter, A. C. Schultz, J. F. Miller, E. Burke, N. Jonoska (eds), Proceeding of the Genetic and Evolutionary Computation Conference (GECCO2002), Morgan Kaufmann, 2002.

[MAC 67] : J. B. MacQueen. « Some methods for classification analysis of multivariate observations. » Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. Pages 281-297. University of California Press. 1967.

[Mic 96] : Michalewicz Z., Genetic Algorithms + Data Structures = Evolution Programs. Ed Springer-Verlag, 1996.

[Mos 89] : P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report 05.20 02.50 87.10, California Institute of technologie, Pasadena, CA 91125, U.S.A., 1989.

[NAÏ 01] : P. Naïm et M. Bazsalicza. « Data mining pour le web. » Editions Eyrolles. 2001.

[Rec 73] : Ingo Rechenberg. « Evolutionsstrategie ». Friedrich Frommann Verlag (GuntherHolzboog KG), Stuttgart, 1973.

[RIN 01] : http://www.iniplus.net/home/index.php

[RIN 02] : http://www.cs.waikato.ac.nz/ml/weka

[RIN 03] : BLAKE, C.L. and C.J. MERZ, UCI repository of machine learning databases, 1998. URL http://www. ics. uci. edu/~ mlearn/MLRepository. html.

[Roc 95] : Roch, J. L., Villard, G. et Roucairol, C. (1995). Parallélisme et Applications Irrégulières. Algorithmes Irréguliers et Ordonnancement, Hermès: 73-88.

[Sch 89] : Schaeffer J. et D. Schuurmans (1989), « Representational difficulties with classifier systems », International Conference of Genetic Algorithms, p. 328-333.

[SIA 05] : Johann Dréo, Alain Pétrowski, Patrick Siarry, Eric Taillard. « Métaheuristiques pour l'optimisation difficile ». Eyrolles, 2005.

[Smi 80] : Smith S. F., « A Learning System Based on Genetic Algorithms», PhD thesis, Department of Com-puter Science, University of Pittsburg, Pittsburg, MA, 1980.

[Sur 96] : Surry P.D. et Radcliffe N.J., Real Representation. Foundation of Genetic Algorithms IV, Ed. Morgan Kaufmann, San Mateo, 1996.

[Sut 90] : Sutton R. S., « Planning by Incremental Dynamic Programming », Proceedings of the Eighth In-ternational Conference on Machine Learning, Morgan Kaufmann, San Mateo, CA, p. 353-357, 1990.

[Tom 02] : Thomas Back, David B. Fogel, et Zbigniew Michalewicz, éditeurs. « Evolu-tionary Computation 1 : Basic Algorithms and Operators ». Institute of Physics Publishing, Bristol, UK, 2000.

[Tou 92] : C. Touzet, Les réseaux de neurones artificiels, 1992.

[Ver 95] : Verhoeven, M. G. A. et Aarts, E. H. L. (1995). "Parallel Local Search." Journal of Heuristics 1: 43-65.

[Wei 91] : 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.

[Wil 85] : Wilson S. W., « Knowledge Growth in an Artificial Animat », in J. J. Grefenstette (ed.), Pro-ceedings of the 1st international Conference on Genetic Algorithms and their applications (ICGA85), L. E. Associates, p. 16-23, july, 1985.

[Wil 95] : Wilson S.W., « Classifier Fitness Based on Accuracy », Evolutionary Computation, vol. 3, n˚ 2, p. 149-175, 1995.

[Wol 98] : Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, et Frank D. Francone. « Genetic Programming – An Introduction ; On the Automatic Evolution of Com-puter Programs and its Applications ». Morgan Kaufmann, dpunkt.verlag, 1998.