N° d'ordre: 02/2009-M/INF

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE
USTHB

FACULTE : INFORMATIQUE

Mémoire

Présentée pour l'obtention du diplôme de :

MAGISTER

Spécialité : Intelligence artificielle

Par

Melle. MAHANI AOUATEF

SUJET

UNE APPROCHE MEMETIQUE POUR L'EXTRACTION DES CONNAISSANCES

Soutenue publiquement, le 20 Décembre 2009 Devant un jury composé de :

Mme. Habiba DRIAS	Professeur à L'USTHB	Présidente
M. Ahmed Riadh BABA-ALI	Maître de Conférences.A à L'USTHB	Directeur de mémoire
M. Hamid AZZOUNE	Maître de Conférences.A à L'USTHB	Examinateur
Mme. Nacéra BENSAOU	Maître de Conférences.A à L'USTHB	Examinatrice
Mme. Thouraya TEBIBEL	Maître de Conférences.A à L'ESI	Examinatrice
Mme. Sadjia BABA-ALI	Maître Assistant.A à L'USTHB	Invitée

REMERCIEMENTS

Au terme de notre étude, nous tenons à exprimer nos plus vifs et sincères remerciements à tous ceux qui ont contribué, de prés ou de loin à sa réalisation :

- Monsieur BABA ALI, maître de conférences à l'USTHB, qui a bien voulu m'encadrer pour la réalisation de cette thèse et pour sa patience.
- Madame BABA ALI, pour son aide très précieuse et ses chers conseils et son encouragement durant le travail.
- Nos remerciements vont également au président et aux membres du jury, pour avoir accepté de juger ce travail.
- Nous n'oublierons pas également de remercier nos enseignants, particulièrement ceux du département

d'informatique pour leur contribution à notre formation

SOMMAIRE

Chapitre 3 : Présentation du problème de la classification

Chapitre 4 : Conception d'un classifieur à l'aide d'une approche mémétique : notre contribution

Liste des figures

Liste des Tableaux

Introduction Générale

Le Data Mining est la «découverte de connaissances dans les bases de données» (ou Knowledge Discovery in Data base, abrégé en KDD). Il englobe tout le processus d'extraction de connaissance à partir de données. Le mot « connaissances » est compris ici comme étant un ensemble de relations (règles, phénomènes, exceptions, tendances…). Le but de ce processus est de découvrir des tendances cachées dans de grande masse des données (la " mine " de données) et les modèles qui les traversent. Ces outils servent à déterminer des profils de comportement, à découvrir des règles, à évaluer des risques. Il est utilisé dans plusieurs applications, nous citons :

- **Grande distribution :** Dans la grande distribution (hypermarchés) : pour examiner les données des clients (sur une longue période), afin de trouver des affinités entre produits et services susceptibles d'intéresser le client, et les articles qui vont ensemble.
- **Détection des fraudes :** Les techniques de Data Mining sont également appliquées à la détection des fraudes notamment dans les domaines où plusieurs transactions s'effectuent simultanément rendant le système vulnérable aux fraudes (ex : cartes de crédit, télécommunications, systèmes informatiques…etc.).
- **Diagnostique médical :** Certains outils, par l'application de méthodes de clusterisation ou d'extraction de règles d'association, à partir des symptômes d'un patient, ou de ses données pathologiques (ex : radiographies, électrocardiogrammes), sont capables d'effectuer des diagnostiques plus fiables, et plus objectifs que ceux des médecins.
- **Web-Mining :** L'application du Data Mining sur le Web, vise à rendre la navigation plus personnalisée, plus adaptée aux besoins des utilisateurs ; parmi ces applications on peut citer :

• La déduction des relations d'associations entre les passages sur différents sites, ou catégories de sites.

• La prévision d'achats sur un site donné en fonction d'autres observations : impact sur la stratégie de communication (décisions).

• L'étude de l'efficacité de bannières (publicitaires), et autres techniques de référencement.

• La segmentation des profils des visiteurs en fonction des comportements de navigation sur Internet.

➢ **Marketing ciblé :** Le marketing est le domaine de prédilection du Data Mining, où il trouve diverses applications. Par exemple : certaines offres « limitées », qui ont un coût important ne peuvent être proposées à tous les clients, les distribuer de manière aléatoire générerait une perte énorme (clients non intéressés). L'application de méthodes de classification permet de cibler la clientèle la plus favorable, améliorant considérablement les bénéfices.

Le Data Mining est également appliqué dans beaucoup d'autres domaines : assurances, astronomie, industrie, renseignements et anti-terrorisme, profilage des fraudeurs dans les déclarations fiscales …etc.

Ça ne veut pas dire que le Data Mining est le remède miracle capable de résoudre toutes les difficultés ou besoins de l'entreprise. Cependant, une multitude de problèmes d'ordre intellectuel, économique ou commercial peuvent être regroupés, dans leur formalisation, dans l'une des tâches suivantes : ***classification, estimation, Prédiction, groupement par similitudes, segmentation, description et optimisation***.

Notre travail est focalisé sur le problème de la classification qui est la tâche la plus populaire du Data Mining. Pour un objet et ses attributs d'entrée, le résultat de la classification est : l'une des classes possibles prédéfinies, et mutuellement exclusives du problème. Cela revient à expliquer ou à prévoir une caractéristique d'un individu nouvellement présenté, à l'aide de relations découvertes entre ses autres caractéristiques (attributs d'entrée).

Ce problème de classification est un problème d'optimisation combinatoire où l'on cherche à optimiser une fonction objectif et fait appel à des méthodes de résolution.

Cependant, le choix d'une méthode dépend du problème et de l'espace de recherche. Dans notre cas, la résolution de ce problème fait appel aux algorithmes évolutionnaires : les algorithmes génétiques ou les algorithmes mémétiques, ces derniers étant des algorithmes génétiques hybridés avec les méthodes de recherche locale.

Les méthodes de recherche locale visent à améliorer de façon itérative une solution existante en explorant un voisinage de celle-ci. Les solutions voisines sont généralement

obtenues en appliquant des transformations ou mouvements plus ou moins importants à la solution courante.

Dans le cadre de notre travail, nous nous attelons à construire un classifieur en utilisant d'abord l'approche génétique pure, puis l'approche hybride dont le but est de répondre à cette question : que nous apporte l'approche mémétique par rapport à l'approche génétique ?

La structuration de notre document est la suivante :

Le premier chapitre décrit les méthodes de résolution des problèmes d'optimisation combinatoires.

Nous présentons dans le deuxième chapitre les concepts de base des algorithmes génétiques et mémétiques.

Dans le troisième chapitre, nous donnons des généralités sur le problème de classification et les méthodes de classification et nous décrivons aussi l'apprentissage, ses types et les méthodes existantes pour chaque type.

Le quatrième chapitre est divisé en deux sections : La première section présente un état de l'art sur les travaux faits dans le domaine de classification et la deuxième section contient toutes les étapes nécessaires pour la construction de notre classifieur par l'approche génétique dans la première partie et dans l'autre par l'approche mémétique.

Pour voir l'utilité de l'approche hybride par rapport à l'approche non hybride et de faire une étude comparative avec les autres algorithmes de classification, nous présentons dans le sixième chapitre des tests en appliquant le classifieur construit sur les bases de données benchmark d'UCI **[INT 01].**

1. Introduction : [INT 02]

Un problème d'optimisation combinatoire consiste à trouver la (ou les) *meilleure* solution dans un ensemble discret dit ***ensemble des solutions réalisables***. En général, cet ensemble est fini mais compte un très grand nombre d'éléments. Il est décrit de manière implicite, c'est-à-dire par une liste, relativement courte, de contraintes que doivent satisfaire les solutions réalisables.

Pour définir la notion de *meilleure solution*, une fonction, dite *fonction objectif*, est introduite. Pour chaque solution, elle renvoie une valeur (qui peut être un réel) et la meilleure solution (ou *solution optimale*) est celle qui optimise la fonction objectif.

Trouver une solution optimale dans un ensemble discret et fini est un problème facile en théorie : il suffit d'essayer toutes les solutions et de comparer leurs qualités pour voir la meilleure. Cependant, en pratique, l'énumération de toutes les solutions peut prendre trop de temps ; or, le temps de recherche de la solution optimale est un facteur très important et c'est à cause de lui que les problèmes d'optimisation combinatoire sont réputés si difficiles. La théorie de la complexité donne des outils pour mesurer ce temps de recherche. De plus, comme l'ensemble des solutions réalisables est défini de manière implicite, il est aussi parfois très difficile de trouver ne serait-ce qu'une solution réalisable.

2. Les méthodes de résolution des problèmes d'optimisation combinatoire :

La résolution d'un problème d'optimisation combinatoire signifie la détermination de la solution optimale qui nécessite un parcours ou une exploration de l'espace de recherche. L'espace de recherche est constitué de toutes les solutions qui satisfont le problème. L'exploration de l'espace de recherche permettra la sélection des solutions qui sont potentiellement de bonne qualité dans le but d'atteindre à la fin de ce processus la solution optimale.

3. Classification des heuristiques et des métaheuristiques :

Plusieurs classifications des métaheuristiques ont été proposées. On distingue globalement deux catégories : les méthodes à base de solution unique, qui travaillent sur un seul point de l'espace de recherche à un instant donné, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche.

Une classification des méthodes de résolution, basée sur différentes approches d'exploration de l'espace de recherche existent. Elle est sommairement illustrée par la ***figure 1.1.***

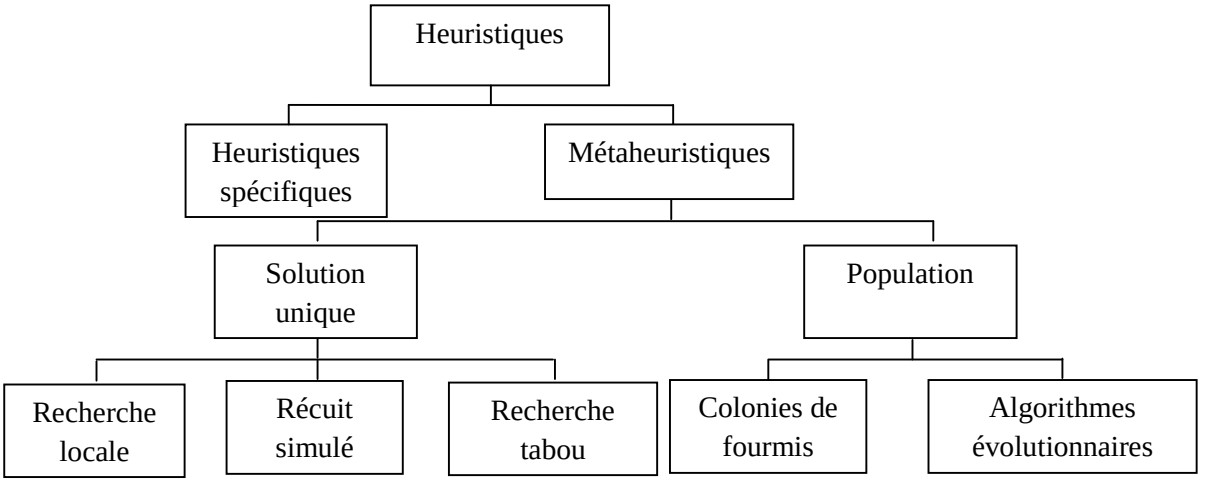


Figure 1.1. Approches de résolution de problèmes d'optimisation combinatoire

Nous limitons notre étude sur les heuristiques et les métaheuristiques.

3.1. Heuristique : [PAP 1982]

Une heuristique est un algorithme de résolution ne fournissant pas nécessairement une solution optimale pour un problème d'optimisation donné mais fournit une solution satisfaisante, proche de l'optimale en un temps satisfaisant. Une bonne heuristique possède plusieurs caractéristiques :

1. Elle est de complexité raisonnable : elle fournit souvent une solution proche de l'optimum en un temps raisonnable.
2. Elle résout un problème particulier étant donné qu'elle est conçue pour ce problème.
3. La probabilité d'obtenir une solution de mauvaise qualité est faible.

3.2. Métaheuristique : [PAP 1982]

Une métaheuristique est une famille d'algorithmes d'optimisation. Le terme métaheuristique provient de la composition de deux mots grecs. « Heuristic » dérive du verbe « heuristikein », qui signifie « trouver », et le préfixe « méta» signifie « au-dessus, dans un niveau supérieur ». Les métaheuristiques sont destinées à la résolution de problèmes difficiles d'optimisation combinatoire.

3.3. Les méthodes à base de solution unique :

Dans les problèmes d'optimisation où l'on cherche à optimiser une fonction objectif sur un espace de décision donné, une petite perturbation sur un point de cet espace induit souvent une petite variation des valeurs de la fonction objectif en ce point.

On en déduit que les bonnes solutions ont tendance à se trouver à proximité d'autres bonnes solutions, les mauvaises étant proches d'autres mauvaises solutions. D'où l'idée qu'une bonne stratégie consisterait à se déplacer à travers l'espace de recherche en effectuant de petits pas (petits changements sur le point courant) dans des directions qui améliorent la fonction objectif. Cette idée est à la base d'une grande famille d'algorithmes appelée ***méthodes de recherche locale.***

3.3.1. Les méthodes de recherche locale :

3.3.1.1. Introduction : [AAR 1997]

La recherche locale est une technique très utilisée en pratique pour aborder des problèmes d'optimisation combinatoire difficiles. L'idée, naturelle, est d'améliorer de façon itérative une solution existante en explorant un voisinage de celle-ci. Les solutions dites voisines sont généralement obtenues en appliquant des transformations (aussi appelées mouvements) plus ou moins importantes (c'est-à-dire plus ou moins locales) à la solution courante.

Après un premier essor dans les années 70 lié aux travaux de Lin et Kernighan **[LIN 1973]** sur le problème du voyageur de commerce, la recherche locale a connu un regain d'intérêt à la fin des années 90 avec de nombreuses études expérimentales et quelques tentatives d'analyse théorique. Ce regain d'intérêt, et la popularisation qui s'en est suivie dans les milieux académiques et industriels, est sans doute dû à l'avènement du concept de métaheuristique.

3.3.1.2. Définition de la recherche locale : [VAN 2007]

Les méthodes de recherche locale partent d'une configuration initiale et appliquent successivement des transformations à la solution courante tant qu'un critère d'arrêt n'est pas vérifié. Leur mise en œuvre nécessite donc le choix :

- D'une (ou plusieurs) solution initiale.
- D'une (ou plusieurs) transformation locale, on parle aussi de mouvement Voir ***figure 1.2***.

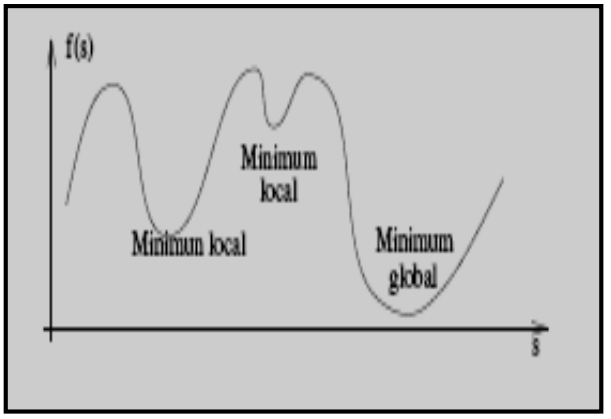


Figure 1.2. Illustration: Recherche locale

L'existence d'extrémas locaux impose l'utilisation de méthodes d'exploration efficaces pour éviter de rester bloqué aux alentours de ces minima. Plusieurs méthodes ont été proposées et ont souvent été inspirées par des phénomènes naturels :

- Le recuit simulé est basé sur les principes d'équilibre énergétique lors de la cristallisation des métaux ;
- La méthode Tabou introduit la notion d'histoire (mémoire) dans la stratégie d'exploration des solutions.

3.3.1.3. <u>Notion de voisinage</u> : [MIC 2001]

On définit le voisinage d'un point de l'espace par une transformation élémentaire permettant de passer d'une solution 1 vers une solution 2 avec une faible modification de la structure de la solution.

Graphiquement, considérons l'espace de recherche S dans la figure Fig3 et soit le point $x \in S$. Les voisinages de x notés N(x) est l'ensemble de tous les points de S qui sont définis par la mesure suivante :

Soit la fonction distance définie sur S, notée *dist* :

dist : $S*S \rightarrow R$.

On définit le voisinage N(x) comme suit :

$N(x) = \{y \in S: dist(x,y) \leq \zeta\}, (\zeta \geq 0)$.

On dit que y est dans le ζ-voisinage de x.

Exemple de voisinage dans le cas binaire :

Complémentation : remplacer un bit quelconque par son complément :

Soit le code suivant : 1010, l'ensemble de voisinage est : {0101, 1110, 1000, 1011}

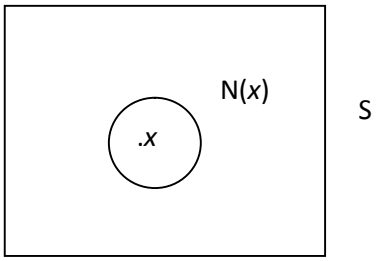


Figure 1.3. Notion de Voisinage d'un point

3.3.1.4. Les différents algorithmes :

3.3.1.4.1. La méthode de Hill-Climbing : (Grimpeur ou la descente stochastique) [MIC 2001]

On part d'une solution initiale et on lance l'exploration de son voisinage jusqu'à ce que l'on rencontre une solution meilleure, à partir de laquelle on applique le même principe. Dans ce cas, la recherche locale du voisinage d'une solution se termine après un nombre variable d'itérations, quand elle n'arrive plus à trouver une solution de meilleure qualité par rapport à la solution courante.

```
Début
    t ← 0 ;
    initialiser la meilleure solution best;
    Répéter
        locale ← FAUX ;
        générer une solution initiale vi ;
        évaluer vi ;
        Répéter
            sélectionner l'ensemble V des voisins de vi ;
            sélectionner le point vn de V qui a la meilleure valeur de la fonction
            d'évaluation eval ;
            Si eval(vn) est meilleure que eval(vi)
                Alors
                    vi ← vn ;
                Sinon
                    local ← vrai ;
```

```
                Fsi ;
            Jusqu'à local ;
            t ← t+1 ;
            Si vi est meilleure que best
              Alors
                best ← vi ;
        Jusqu'à t=MAX ;
Fin.
```

Il est clair que cette méthode fournit seulement les valeurs des optimums locaux qui sont proches de la solution initiale.

3.3.1.4.2. Le récuit simulé [ULU 1999]

Cette méthode a été proposée par S. Kirkpatrick, C. D. Gelatt et M.P.Vecchi en 1983, et indépendamment par V.Cerny en 1985.

Le recuit simulé est un algorithme local général de recherche, cherchant à réduire au minimum un objectif unique noté z(x). Il s'inspire du processus de recuit utilisé en métallurgie pour améliorer la qualité d'un solide, en cherchant un état d'énergie minimal.

En partant d'une haute température où le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à retrouver sa forme solide par une diminution progressive de la température.

Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique.

L'idée fondamentale de la méthode est d'accepter les mouvements non améliorants parce qu'ils peuvent aider à s'échapper d'un minimum local.

L'acceptation d'un mouvement non améliorant se fait selon une règle, ou plus précisément, après avoir évalué une certaine probabilité *p* qui est calculée en respectant la distribution de Boltzmann.

Sachant que $\Delta z = z(y) - z(x)$, y étant la solution potentielle obtenue, chaque mouvement qui mène à une valeur négative de Δz est un mouvement améliorant qui sera toujours accepté ($p = 1$). Lorsque Δz est positif. Il représente un retour en arrière qui est accepté ou refusé selon la formule suivante :

$$p = e^{(\Delta z/Tn)}$$

Où T_n représente la température à l'itération n.

Dans un certain sens, la température T représente le seuil pour l'acceptation d'un mouvement non améliorant :

- Une grande température implique l'acceptation de presque tous les mouvements (*p* est proche de 1) ;
- Une température très basse implique le refus de presque tous les mouvements (*p* est proche de 0).

La stratégie de recherche du recuit simulé peut être décrite comme suit :

- Au début, presque tous les mouvements sont acceptés ; ceci nous permet d'explorer l'espace de recherche.
- Ensuite la température est graduellement diminuée, ce qui signifie que la méthode devient de plus en plus sélectif quand au choix des nouvelles solutions.
- A la fin, seuls les mouvements améliorants sont acceptés.

Algorithme :

```
Début
    // Initialisation
    Tn ← grande valeur ;
    x ← solution initiale ;
    Tant que t > 0 et critère d'arrêt non atteint
        Répéter
            Générer une solution y selon la méthode utilisée ;
            Evaluer Δz = z(x) - z(y) ;
            Si Δz <0 Alors x ← y
                Sinon
                    P ← e^(Δz/Tn) ;
                    Accepter ou refuser y selon la stratégie adoptée ;
            Fsi ;
        Jusqu'à nb-iter
    Fait ;
Fin
```

3.3.1.4.3. La recherche Tabous (Tabu Search) : [KOR 1987] [VAN 2007] [GLO 1989]

- **Introduction :**

Cette méthode a été présentée pour la première fois par F. Glover. L'idée de base consiste à introduire la notion d'histoire dans la politique d'exploration des solutions.

Elle repose sur les déplacements des fonctions objectifs dans l'espace de recherche local combinés à une amélioration régulière de cet espace local grâce à la mémoire des déplacements passés. Cette méthode permet de guider les recherches en dehors des zones précédemment parcourues. D'où le nom tabou qui est donné en 1986 par Glover qui exprime l'interdiction de reprendre des solutions récemment visitées.

- **Définition de la liste taboue :** la liste taboue contient tous les mouvements interdits pour éviter d'explorer des solutions qui ont été parcourues, sa taille est fixe et si elle est pleine alors les éléments les plus anciens seront remplacés par les nouveaux éléments. La taille t de la liste taboue est à déterminer empiriquement, elle varie avec les problèmes, mais c'est une donnée primordiale. En effet une liste trop petite peut conduire à un cycle, alors qu'une liste trop grande peut interdire des transformations intéressantes.

- **Algorithme :**

Soient les paramètres suivants :

S_0 : solution initiale.

S* : meilleure solution.

f (S*) : valeur de la meilleure solution.

1. Initialisation :

Trouver une solution initiale S_0 ;

S* :=S_0 (S* est a meilleure solution rencontrée).

2. K :=0 ; liste tabou :=Φ ;

Répéter tant qu'un critère d'arrêt n'est pas atteint :

1. Choisir parmi le voisinage de S_k, $V(S_k)$, le mouvement qui minimise f et qui n'appartient pas à la liste tabou, meilleur(S_k).
2. S_{k+1} := S_k.
3. Si f (S_{k+1}) < f(S*) alors S*:=S_{k+1}, f*(S*):=f (S_{k+1}).
4. Mise à jour de la liste taboue.

- **Les stratégies d'amélioration de la recherche taboue :**

La recherche taboue peut résoudre avec succès des problèmes difficiles. Mais dans la plupart des cas, des améliorations doivent être ajoutées à la stratégie de recherche pour la rendre entièrement efficace. On peut citer comme exemple :

- **Stratégie d'intensification :** On mémorise les meilleures solutions rencontrées et l'on essaie d'en dégager quelques propriétés communes pour définir des régions intéressantes vers lesquelles on oriente la recherche (par exemple en rendant Tabou tous les mouvements qui font sortir de cette région).
- **Stratégie de diversification :** C'est le contraire. L'application de cette politique conduit à mémoriser les solutions les plus fréquemment visitées et imposer un système de pénalités, afin de favoriser les mouvements les moins souvent utilisés.

- **Conclusion :**

Pour ces méthodes (recherche Tabou et recuit simulé) nous avons trouvé dans la littérature beaucoup d'applications. Ce sont certes des méthodes de recherche locales mais qui ont montré leur grande efficacité à résoudre plus d'un problème difficile: elles sont donc classées métaheuristiques également.

3.4. Les méthodes à base de population :

Cette classe de métaheuristiques travaille explicitement avec une population de solutions. Nous pouvons distinguer dans cette classe de méthodes les algorithmes génétiques et mémétiques dits évolutionnaires, les algorithmes de colonies de fourmis, la recherche par dispersion, les algorithmes basés sur les essaims particulaires et ceux basés sur l'estimation de distributions.

3.4.1. Algorithmes de Colonies de Fourmis : [DOR 1994]

L'origine de cette approche repose sur le comportement des fourmis lors de la recherche de nourriture. Ce comportement leur permet de trouver les plus courts chemins entre les sources de nourriture et leur nid. Pendant leurs déplacements entre le nid et les sources de nourriture, les fourmis déposent une substance sur le sol. Cette substance, appelée phéromone, sert à guider les fourmis vers les chemins les plus rapides pour arriver aux sources de nourriture. Les sentiers les plus courts sont les plus concentrés en phéromone. En effet, les phéromones s'évaporent au cours du temps et les fourmis qui empruntent les sentiers les plus courts arrivent rapidement à se procurer de la nourriture et rentrent au nid en déposant de la phéromone sur leur chemin de retour.

3.4.2. Les algorithmes évolutionnaires : [BÄC 1997]

Les algorithmes évolutionnaires sont des techniques d'optimisation itératives et stochastiques, inspirées par des concepts issus de la théorie de l'évolution de Darwin. Un algorithme évolutionnaire simule un processus d'évolution sur une population d'individus, dont le but est de les faire évoluer vers les optimums globaux du problème d'optimisation considéré. Au cours du cycle de simulation, trois opérateurs interviennent : recombinaison, mutation et sélection. La recombinaison et la mutation créent de nouvelles solutions candidates, tandis que la sélection élimine les candidats les moins prometteurs.

Il existe plusieurs branches faisant partie de la famille des algorithmes évolutionnaires : les algorithmes génétiques, les algorithmes mémétiques.

3.4.2.1. Les algorithmes génétiques: ces algorithmes seront vus en détails dans ***le chapitre 2, section 2.***

3.4.2.2. Les algorithmes mémétiques : ces algorithmes seront vus en détails dans ***le chapitre 2.***

4. Conclusion :

Nous avons vu dans ce chapitre plusieurs méthodes de résolution des problèmes d'optimisation combinatoire et chaque méthode a ses propres caractéristiques. Cependant, il faut choisir la bonne méthode pour résoudre un problème donné. Par exemple, dans le cas où nous voulons résoudre un problème qui a un grand espace de recherche, les méthodes à base de population seront utilisées.

1. Introduction :

Les algorithmes mémétiques ont été proposés par Moscato 1989, ils s'inspirent de certains modèles d'adaptation dans la nature, qui combinent l'évolution adaptative de populations d'individus avec l'apprentissage des individus au cours de leur vie. Le terme "algorithme mémétique" trouve son origine dans le concept de "meme" introduit par Richard Dawkins **[MOS 1989]** qui représente une unité d'information évoluant avec les échanges d'idées entre individus. Une différence fondamentale entre les notions de gènes (manipulés par les algorithmes génétiques) et de "memes" est que ces derniers sont adaptés par la personne qui les transmet (en introduisant ses réflexions et ses déductions personnelles), alors que les gènes sont transmis tels quels.

Ces algorithmes se retrouvent dans la littérature sous plusieurs autres noms : algorithmes génétiques hybrides, recherche locale génétique, algorithmes évolutionnaires Baldwiniens, algorithmes évolutionnaires Lamarkiens et autres.

Pour cela, nous allons présenter une brève description des algorithmes génétiques dans la section qui suit :

2. Description des algorithmes génétiques : [GOL 1989] [HOL 1962]

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle : croisements, mutations, sélection, etc.

Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de recherche. Pour l'utiliser, on doit disposer des cinq éléments suivants :

1. Un principe de codage de l'élément de population.
2. Un mécanisme de génération de la population initiale.
3. Une fonction à optimiser.
4. Des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace d'état.
5. Des paramètres de dimensionnement : taille de la population, nombre total de générations ou critère d'arrêt, probabilités d'application des opérateurs de croisement et de mutation.

Le principe général du fonctionnement d'un algorithme génétique est représenté sur la ***figure 2.1.***

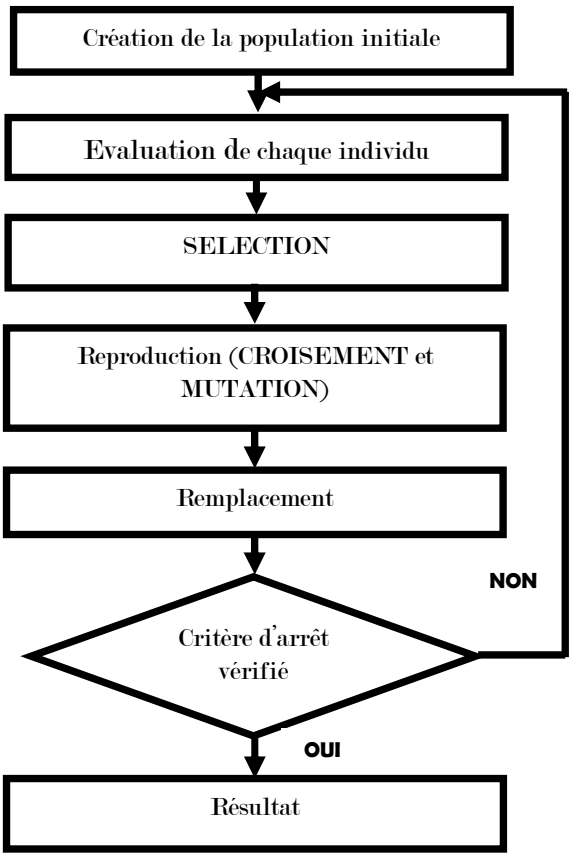


Figure 2.1. Processus de l'algorithme génétique.

2.1. Concepts de base :

- **Gène :** un gène représente un caractère ou une caractéristique d'un individu.
- **Chromosome :** est un ensemble de gènes.
- **Individu :** un individu représente un ensemble de chromosomes.
- **Population :** est un ensemble d'individus.
- **Fonction de performance ou d'évaluation « fitness » :** C'est la fonction qu'on cherche à optimiser. Elle représente l'adaptation de chaque chromosome à son environnement.

2.2. Codification : [COU 2002]

Est le premier pas dans l'implémentation des algorithmes génétiques. Le choix du codage des données dépend du problème traité et conditionne son efficacité (vitesse de convergence, précision,…), il existe 03 types de codage : binaire, entier et réel.

2.3. Sélection : [GOL 1989]

La sélection permet d'identifier statistiquement les meilleurs individus d'une population et d'éliminer partiellement les mauvais, le choix de la méthode de sélection repose sur trois objectifs principaux :

- Choisir les individus sur lesquels s'appliqueront les opérations de reproduction pour la création de la future génération.
- Favoriser les meilleurs individus.
- Permettre d'explorer les différentes parties de l'ensemble de recherche.

Parmi ses méthodes, nous citons : **la loterie biaisée (ou roulette wheel) et le tournoi.**

2.4. Les critères d'arrêt d'un algorithme génétique :

- Ils peuvent être liés à la qualité minimum obtenue : la procédure est arrêtée quand il n'y plus d'améliorations de la solution après un certain nombre d'itérations.
- La meilleure solution est atteinte.
- Après un nombre d'itération fixé.

2.5. Les opérateurs génétiques de reproduction :

- **Croisement :** (**Crossover)**

Cet opérateur mélange les gènes d'un chromosome, sans toucher à leur contenu. Il réalise la reproduction entre les individus de la population **[EIB 2007]**. Il s'applique avec une certaine probabilité P_{cro}. Il peut être effectué de plusieurs manières selon le nombre de points de coupure, d'où l'existence de plusieurs méthodes : c**roisement en un point et croisement en k points.**

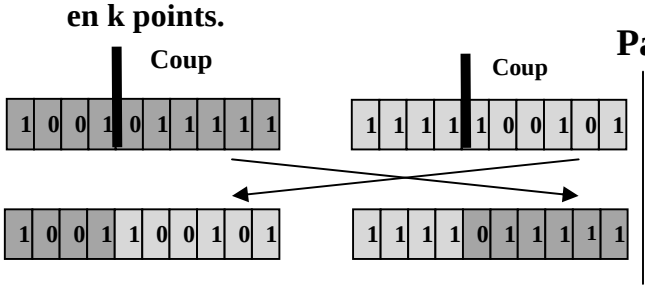
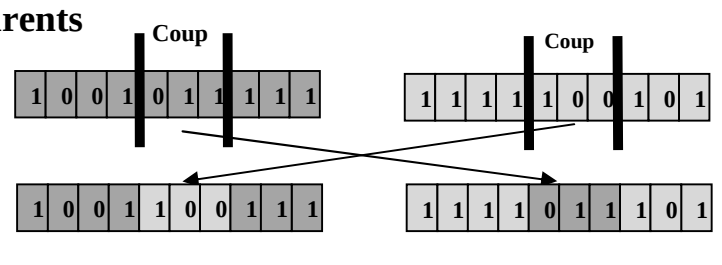


Figure 2.2. Croisement en un point.



Enfants

Figure 2.3. Croisement en k points.

- **Mutation :**

La mutation d'un chromosome consiste à remplacer un de ses gènes par une autre valeur choisie aléatoirement. Il s'applique généralement avec une faible probabilité P_{mut}. Le but est

de permettre une meilleure recherche locale et de maintenir la diversité génétique utile pour une bonne exploration de l'espace de recherche.

Figure 2.4. Mutation.

2.6. Evaluation des individus de la population .

Chaque individu de la population doit être évalué en calculant la valeur de sa fonction objective qui représente le degré d'adaptation à son environnement.

3. Concepts de base d'un algorithme mémétique : [KRA 2002] [MER 2000] [MOS 2001]

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

La ***figure 2.5*** illustre à l'aide de cercles noirs les niveaux possibles d'application d'une méthode de recherche locale dans un algorithme évolutionnaire.

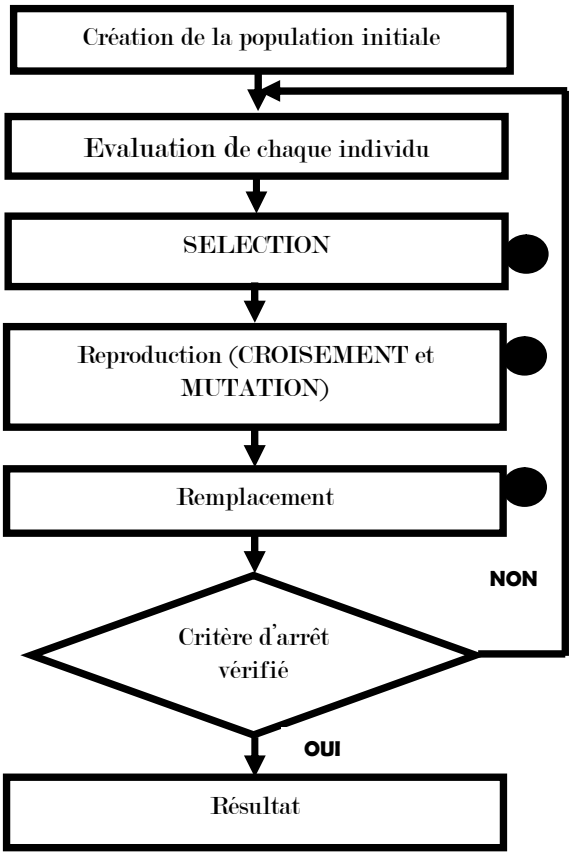


Figure 2.5. Application d'une méthode de recherche locale dans une boucle génétique.

3.1. Les raisons de l'hybridation:

1. Les problèmes complexes peuvent être décomposés en sous problèmes dans une certaine mesure.

2. Les algorithmes évolutionnaires ou les méthodes de recherche locale peuvent être utilisés comme pré/post méthode de traitement de solution. On utilise souvent une méthode de recherche locale en hybridation avec un algorithme évolutionnaire pour améliorer la meilleure solution trouvée. Par ailleurs, on peut disposer de méthodes 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.

3. L'information d'un problème spécifique peut être dissipée par la variation des opérateurs, comme les opérateurs de croisement ou de mutation, ou bien par l'algorithme de recherche

locale. L'hybridation des deux méthodes peut éviter cela efficacement en dirigeant la recherche vers des régions prometteuses de l'espace de recherche.

4. Dans certains cas, il existe des méthodes exactes ou approximatives pour les sous problèmes du problème traité, capables de prendre en charge certaines spécificités du problème. Lorsque celles-ci sont disponibles, elles peuvent être incorporées dans l'algorithme évolutionnaire afin de produire de meilleures solutions.

5. Les problèmes associent des contraintes à des solutions et les heuristiques de recherche locale sont utilisées pour améliorer les solutions trouvées par l'algorithme évolutionnaire. Si les stratégies de recherche locale dans les algorithmes mémétiques sont considérées comme première préoccupation du concepteur alors une définition plus riche des métaheuristiques hybrides d'adaptation est possible : les stratégies de recherche locale pourraient être générées en même temps avec les solutions qu'elles envisagent d'améliorer. Cependant, la forme la plus populaire de l'hybridation est d'appliquer une ou plusieurs phases de recherche locale, sur la base de certains paramètres de probabilité à des membres d'individus de la population dans chaque génération.

3.2. Les décisions de conception : [KRA 2004]

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

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

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

3.3. LAMARCKIAN / BALDWINIAN :

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

En d'autres termes, supposons:

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

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

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

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

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

1-Approche de LAMARCKIAN :

Elle consiste à remplacer i par i', donc:

P = P – {i} + {i'}

f (i)= *f* (i')

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

2-Approche de BALDWINIAN (1896)

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

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

P reste la même.

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

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

3.4. Préservation de la diversité :

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

Cette diversité pourrait être perdue si :

- Par exemple, dans le cas de recherche locale partielle, si 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 **[FRI 1996]**.
- Divers mécanismes ont été étudiés comme un moyen d'éviter la convergence prématurée dans les algorithmes mémétiques. Par exemple, si une petite population initiale est générée, seule une petite proportion d'individus devrait subir la recherche locale et non pas tous les individus de la population.

Une stratégie très commune visant à traiter la préservation de la diversité a été l'introduction d'opérateurs de croisement très spécifiques qui aident à préserver la variété génétique toujours au-dessus d'un seuil minimum **[FRI 1996]**.

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

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

1. Des recherches locales multiples : où chacune introduit un espace de recherche différent avec des optimums locaux différents afin d'éviter les pièges locaux **[KRA 2004].**

2. La mise en œuvre des ensembles et des systèmes flous pour contrôler explicitement la diversité au sein d'une règle de décision dans la phase de recherche tel que cela a été fait dans **[KRA 2002].**
3. La possibilité de modifier l'opérateur de sélection ou des critères d'acceptation de recherche locale pour utiliser la méthode adaptative de Boltzmann tel que cela été fait dans **[KRA 2000]**. Cela a été fait de manière similaire à la méthode du recuit simulé où les mouvements peuvent être acceptés avec une probabilité non nulle qui aide à s'échapper de l'optimum local. Cette méthode est prometteuse durant la recherche locale au moins un voisinage peut être accepté avec une probabilité qui augmente exponentiellement avec un facteur k de normalisation.

$$\textbf{Prob (accept) =} \begin{cases} 1 \Leftrightarrow & \Delta f > 0 \\ e^{k * \frac{\Delta f}{fmax - fmoy}} & \text{sinon} \end{cases}$$

Où :

k : facteur de normalisation, **Δf**= f (i')-f(i) (problème de maximisation).

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

3.5. Quel voisinage faut-il utiliser pour la recherche locale ?

Dans le cas de recherche locale, cela est fait en définissant la structure de voisinage d'une solution, c'est-à-dire quelles sont les solutions envisageables accessibles à partir des points de l'espace de solutions.

Si l'espace de solutions est trop petit ou trop restrictif, il est probable que l'optimum local sera déterminé directement par la recherche locale. D'autres parts, si le voisinage est complet, alors il est possible de construire un chemin de tout autre point, mais dans ce cas, l'analyse exhaustive est payée par une longue recherche qui peut être exponentielle.

3.6. Utilisation des profils de performance :

Les profils de performance peuvent être utilisés pour suivre le progrès de la recherche vis-à-vis des différentes composantes algorithmiques de l'algorithme mémétique. Nous pouvons

prendre en considération l'apprentissage acquis par l'algorithme et réutiliser cette connaissance lors de l'exploration future à travers le processus d'optimisation. Une hybridation possible utilise explicitement les connaissances sur les solutions précédemment vues comme un moyen pour guider l'optimisation, comme cela est fait dans la recherche taboue qui n'autorise pas de revisiter des solutions contenues dans la liste taboue.

3.7. Considération particulière pour les domaines continus : [RUD 1996] [HAR 2003]

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

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

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

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

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

4. Les algorithmes mémétiques adaptatifs : [COW 2000] [KEN 2002] [BUR 2003]

4.1. Définition :

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

4.2. Les catégories de choix d'un meme :

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

4.2.1. Aléatoire (Random):

Dans cette catégorie, il y a trois stratégies :

a. Stratégie aléatoire simple (Simplerandom) : dans ce cas, un meme est sélectionné aléatoirement à chaque point de décision. Elle est purement de nature stochastique, la probabilité de choisir chaque meme est maintenue constante tout au long de la recherche. Cette stratégie peut être considérée comme une référence avec laquelle d'autres stratégies de sélection peuvent être comparées.

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

c. <u>Stratégie de la descente aléatoire avec permutation (Randompermdescent)</u> : cette stratégie est similaire à la deuxième, sauf que la permutation aléatoire des memes M_1, M_2,…, M_n est fixée à l'avance, et quand l'application d'un meme ne résulte aucune amélioration, le prochain meme qui existe dans la permutation sera utilisé.

4.2.2. <u>Gourmande (Greedy)</u> :

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

4.2.3. <u>Fonction de choix (Choice function)</u> :

Cette stratégie intègre plusieurs paramètres, elle est utilisée pour évaluer le degré d'efficacité d'un meme, en se fondant sur l'état actuel des connaissances sur la région de l'espace de solution dans l'exploration.

La fonction de choix contient trois composants :

a. ce composant est représenté par f_1 qui reflète les améliorations récentes produites par un meme et exprime l'idée que, si un meme a récemment obtenu de bons résultats, il est susceptible de continuer à être efficace.

b. Ce deuxième composant est f_2, il décrit les améliorations contribuées par les paires consécutives des memes.

c. f_3 enregistre la période écoulée depuis qu'un meme a été utilisé pour la dernière fois.

5. <u>Conclusion</u> :

Nous avons vu que les algorithmes mémétiques utilisent les méthodes de recherche locale à l'intérieur des algorithmes génétiques dans le but de trouver des solutions qui sont

inaccessibles par un AG. Pour cela, il faut prendre en considération ces différents points pour intégrer les méthodes de recherche locale dans un AG et aussi les stratégies utilisées.

Nous détaillons tout cela dans le chapitre conception où nous comptons présenter une modélisation de cette approche.

1. Introduction :

La fouille de données est définie comme un processus d'extraction de connaissances à partir de bases de données **[HOR 2007].** Parmi les fonctions de fouille de données, on distingue la classification. Dans cette fonction, le but est d'assigner à chaque cas (enregistrement de données), une classe choisie à partir d'un ensemble de classes prédéfinies, en fonction de la valeur de certains attributs **[PAR 2002].** En d'autres termes, étant donné un ensemble d'enregistrements de données appartenant chacun à une classe parmi un nombre de classes prédéfinies, le problème de la classification est celui de la découverte de règles qui permettent à des enregistrements de classes indéterminée, d'être correctement classés **[WAI 2003]** (voir ***figure 3.1***).

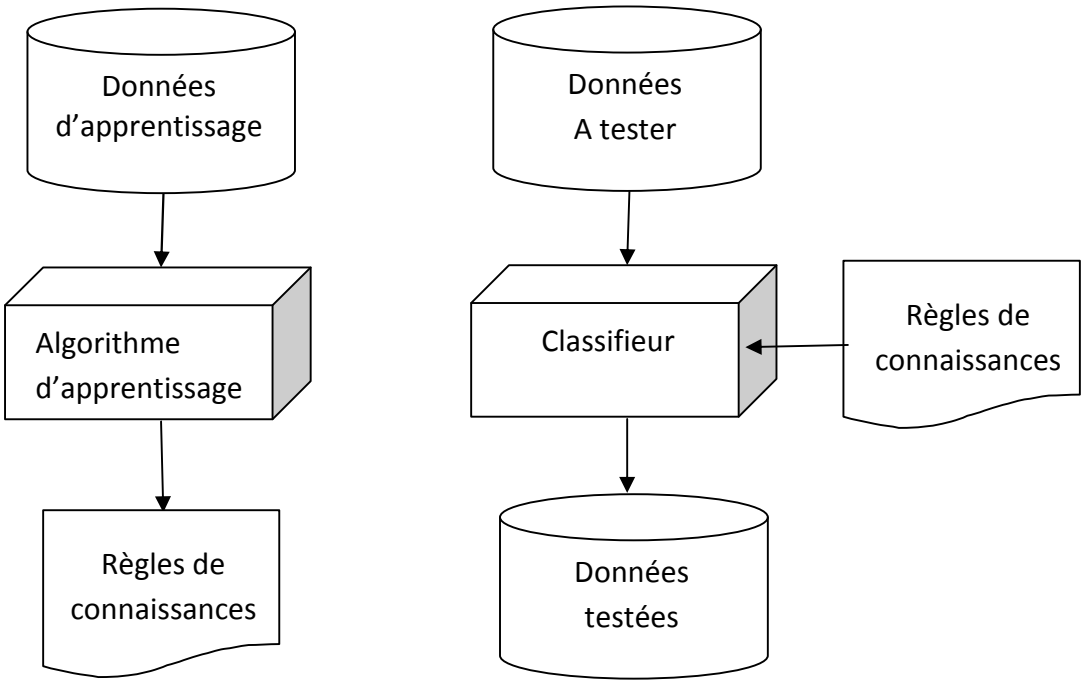
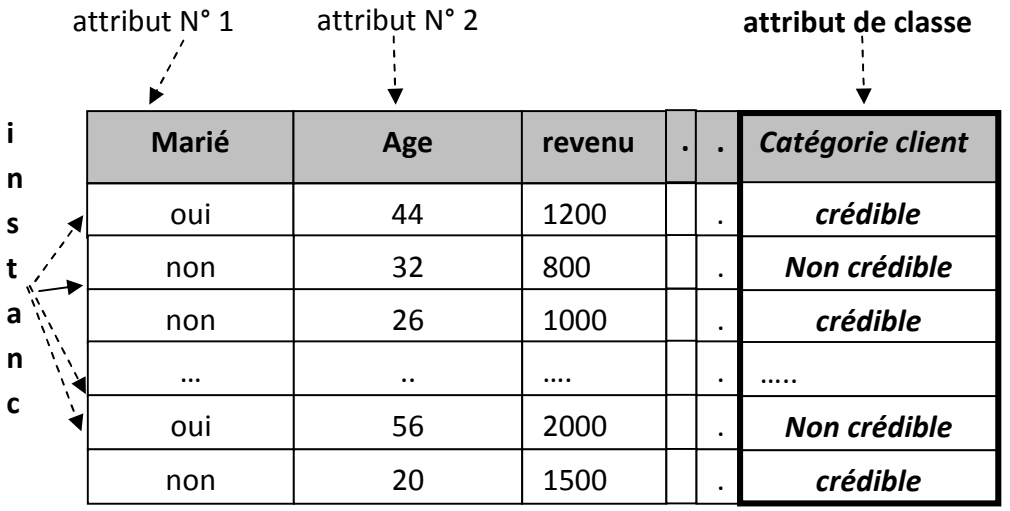


Figure 3.1. (a) Illustration du processus d'apprentissage pour la fouille de données (b) Illustration du processus de classification

2. Notions de données :

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



Marié	Age	revenu	.	.	*Catégorie client*
oui	44	1200		.	***crédible***
non	32	800		.	***Non crédible***
non	26	1000		.	***crédible***
…	..	….		.	…..
oui	56	2000		.	***Non crédible***
non	20	1500		.	***crédible***

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

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

Chaque colonne de la matrice représente toutes les valeurs possibles associées à un ***attribut***. Par exemple, un attribut peut être pour une base de données client: soit le statut marital, l'âge, son revenu annuel etc. Un attribut peut être numérique ou nominal. Un attribut nominal prend ses valeurs à partir d'un ensemble de choix discret. Enfin un attribut particulier appelé ***l'attribut de classe***, est utilisé comme attribut de classification. L'ensemble des valeurs de l'attribut de classe représente l'ensemble des catégories qu'on attribue à chaque instance. Dans notre cas, cela peut être par exemple des classes telles que : clients crédibles ou pas crédibles ou bien fraudeur ou pas fraudeur etc.

On parle de ***données d'apprentissage*** si toutes les données contiennent des valeurs pour l'attribut de classe (voir ***figure 3.1***). On parle de ***données de test*** si ces dernières doivent être classées et donc ne possèdent pas de classe ou en d'autres termes possèdent comme valeur d'attribut de classe, la valeur indéterminée.

3. Notions de modèle : [PAR 2002]

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

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

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

- **Exemple de terme** : *Marié=Oui*

Où "*Marié"* est un attribut de la table clients (voir figure 1), le signe "*=*" est l'opérateur de comparaison et "*Oui"* est la valeur de comparaison.

- **Exemple d'antécédent** : *Marié=Oui* ***ET*** *Age<30* ***ET*** *Revenu>1000*

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

- **Exemple de règle** : **SI** *Marié=Oui ET Age<30 ET Revenu>1000* **ALORS** *Crédible*

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

L'ensemble des règles forme un modèle de classification ou simplement un ***modèle*** **[WIT 2005]**.

- **Exemple de modèle :**

  SI *Marié=Oui ET Age<30 ET Revenu>1000* **ALORS** *Crédible*

  Sinon SI *Marié=Non ET Age>50 ET Revenu>2500* **ALORS** *Crédible*

  Sinon SI *Marié=Non ET Age<20 ET Revenu<500* **ALORS** *Non crédible*

4. <u>Evaluation d'une règle de classification</u> : [AGR 1994]

La qualité d'une règle, une fois établie, est à évaluer avec grand soin, car le plus important pour les utilisateurs de ces règles est qu'elles soient pertinentes, intéressantes et plus lisibles par rapport à son langage et plus précises par rapport à son domaine. Pour cette raison, ils existent de nombreux critères d'évaluation de qualité dont nous citerons quelques uns. Les critères pour mesurer la qualité d'une règle de classification peuvent être :

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

Supp (R) = {instance (B) vérifie au moins une partie A de la règle} / |B|.

Avec : |B |= nombre d'instances de la base de données B.

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

$$\textbf{Conf (I) = Supp (R) / Supp (A).}$$

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

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

$$\textbf{Sp (R) = TN/ (TN+FP).}$$

Où :

- **TN (True Négative) :** le nombre d'exemples n'appartenant pas à C et non prédits dans C par le modèle.
- **FP (False Positive) :** le nombre d'exemples n'appartenant pas à C et prédits dans C par le modèle.

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

4.6. Autres mesures : On définit aussi les deux mesures de qualité suivantes :

- **TP (True Positive) :** le nombre d'exemples appartenant à C et prédits dans C par le modèle.
- **FN (False Négative) :** le nombre d'exemples appartenant à C et non prédits dans C par le modèle.

Remarque :

D'après Alex A. Freitas **[FRE 2000]**, les 04 mesures peuvent être reformulées comme suit :

- TP est le nombre d'exemples qui satisfont A et C.
- FP est le nombre d'exemples qui satisfont A et non C.
- FN est le nombre d'exemples qui ne satisfont pas A mais satisfont C.
- TN est le nombre d'exemples qui ne satisfont ni A et ni C.

5. Les différentes approches des systèmes de classifieurs : [FRA 2003] [JOU 2003]

Il existe, dans les systèmes de classifieurs, deux approches principales pour la modélisation d'un individu dans l'application d'un algorithme génétique, qui sont :

5.1. Approche Michigan :

Dans cette approche, chaque règle est considérée comme un individu. Elle est donc représentée par un chromosome et la population représente l'ensemble de ces règles.

Le chromosome est composé d'un ensemble de gènes, où chaque gène représente une des valeurs d'un attribut.

La taille d'un chromosome est fixée : elle vaut le nombre d'attributs de la base de données +1, qui représente la classe, appelée aussi ***variable de décision*** (Voir ***figure 3.2***).

V Att_1	V att_2	V Att_3	…	V Att_n	Classe

Figure 3.2. Représentation d'une règle par l'approche de Michigan.

Chaque valeur de l'attribut est représentée par son index (valeur entière) suivant l'ordre de déclaration de chaque valeur de cet attribut.

5.2. Approche de Pittsburgh :

Ce système de classifieurs a été introduit par Smith **[SMI 1991].** Dans cette approche, chaque individu représente un ensemble de règles qui sont générées dans la même itération. Mais à chaque itération, le nombre de règles peut être différent mais les individus doivent avoir la même taille. Pour cela, on ajoute des variables comportant la valeur -1 pour représenter des règles inexistantes. La représentation d'un individu dans cette approche dans une itération donnée est illustrée par l'exemple de la ***figure 3.3***.

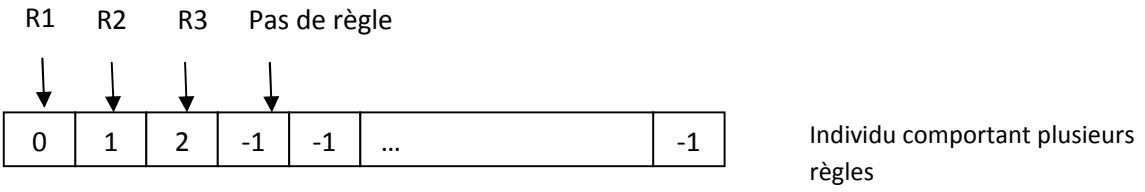


Figure 3.3. Représentation d'un classifieur par l'approche de Pittsburgh.

6. Complexité du problème :

Pour commencer, nous sommes d'abord amenés à situer la complexité de la classification. Pour cela, rappelons que le nombre de partitions non recouvrantes de n objets est le nombre de Bell donné par **[TUF 2005]**

$$B_n = \frac{1}{e} * \frac{\sum_{k=1}^{k=\infty} K^n}{K!}$$

e est un facteur de normalisation.

Par exemple, pour n= 4 objets a, b, c, d nous avons B =15 partitions telles que:

- 1 partition à une classe (abcd)

- 7 partitions à 2 classes (ab, cd), (ac, bd), (ad, bc), (a, bcd), (b, acd), (c, bad), (d, abc)

- 6 partitions à 3 classes (a,b,cd), (a, c, bd),(a, d, bc), (b, c, ad), (b, d, ac), (c, d, ab)

- 1 partition à 4 classes (a, b, c, d).

Pour n= 30 objets on a B = $8.47*10^{23}$ ce qui représente un nombre énorme.

De façon générale, B_n > exponentielle$^{(n)}$ ce qui nécessite la définition de critères de bonne classification et d'avoir des algorithmes performants puisqu'on ne pourra jamais tester toutes ces combinaisons.

7. <u>Apprentissage</u> :

Les méthodes de classification ont pour but d'identifier les classes auxquelles appartiennent des objets à partir de certains traits descriptifs (caractéristiques). Elles s'appliquent à un grand nombre d'activités humaines et conviennent en particulier au problème de la prise de décision automatisée. Donc, nous devons posséder une procédure de classification ou un système d'apprentissage qui sera extraite automatiquement à partir d'un ensemble d'exemples appelé données d'apprentissage.

7.1. Les types d'apprentissage :

7.1.1. L'apprentissage supervisé :

Les exemples d'apprentissage sont étiquetés afin d'identifier la classe à laquelle ils appartiennent. Le but de l'algorithme de classification est de correctement classifier les nouveaux exemples dans les classes définies dans la phase d'apprentissage.

7.1.2. L'apprentissage non supervisé:

L'algorithme d'apprentissage cherche à trouver des régularités dans une collection d'exemples, puisque dans ce type d'apprentissage on ne connaît pas la classe à laquelle les exemples d'apprentissage appartiennent. Une technique employée consiste à implémenter des algorithmes pour approcher les exemples les plus similaires et éloigner ceux qui ont le moins des caractéristiques communes. Ces groupes d'exemples similaires sont parfois appelés des ***prototypes*.**

7.2. Les méthodes de l'apprentissage supervisé :

Les méthodes les plus utilisées sont les arbres de décision et les réseaux de neurones

7.2.1. Les arbres de décision : [MIT 1997] [ADR 1996] [WIT 2005] [TUF 2005]

Un arbre de décision est un outil d'aide à la décision et à l'exploration de données. Il permet de modéliser simplement, graphiquement et rapidement un phénomène mesuré plus ou moins complexe. Sa lisibilité, sa rapidité d'exécution et le peu d'hypothèses nécessaires a priori expliquent sa popularité actuelle. Certains algorithmes utilisés pour répartir une population d'individus (de clients par exemple) en groupes homogènes, selon un ensemble de variables discriminantes (l'âge, la catégorie socio-professionnelle, ...) en fonction d'un objectif fixé et connu (chiffres d'affaires, réponse à un mailing, ...). Il s'agit de prédire avec le plus de précision possible les valeurs prises par la variable à prédire (objectif, variable cible, variable d'intérêt, attribut classe, variable de sortie, ...) à partir d'un ensemble de descripteurs (variables prédictives, variables discriminantes, variables d'entrées, ...). ***Voir figure 3.4.***

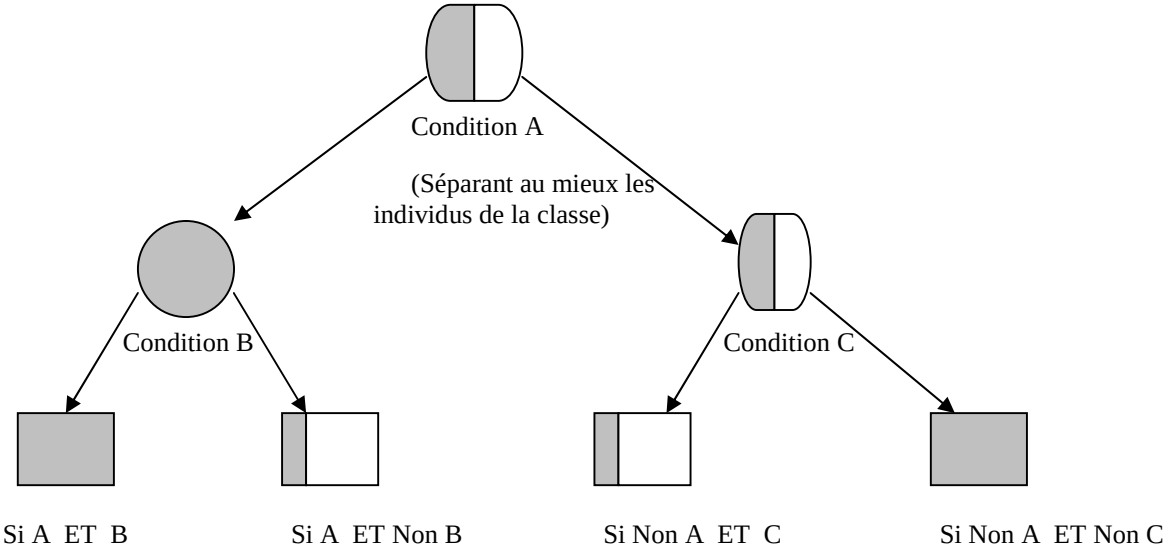


Figure 3.4. Illustration d'un arbre de décision

Mais la question qui se pose : Comment construire un arbre de décision ? Pour répondre à cette question, nous commençons d'abord par donner quelques notations puis nous passons au principe de construction de l'arbre.

Quelques notations :

Avant d'entamer les algorithmes d'apprentissage par arbres de décision, nous introduisons les notations suivantes :

Soient :

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

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

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

Exemple introductif et préliminaire :

CIEL	TEMPERATURE	HUMIDITE	VENT	JOUER
Ensoleillé	Elevé	Forte	Non	Non
Ensoleillé	Elevé	Forte	Oui	Non
Couvert	Elevé	Forte	Non	Oui
Pluvieux	Moyenne	Forte	Non	Oui
Pluvieux	Basse	Normale	Non	Oui
Pluvieux	Basse	Normale	Oui	Non
Couvert	Basse	Normale	Oui	Oui
Ensoleillé	Moyenne	Forte	Non	Non
Ensoleillé	Basse	Normale	Non	Oui
Pluvieux	Moyenne	Normale	Non	Oui
Ensoleillé	Moyenne	Normale	Oui	Oui
Couvert	Moyenne	Forte	Oui	Oui
Couvert	Elevé	Normale	Non	Oui
Pluvieux	Moyenne	Forte	Oui	Non

Tableau 02 : Base de données «Weather»

Principe de construction :

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

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

L'arbre obtenu est :

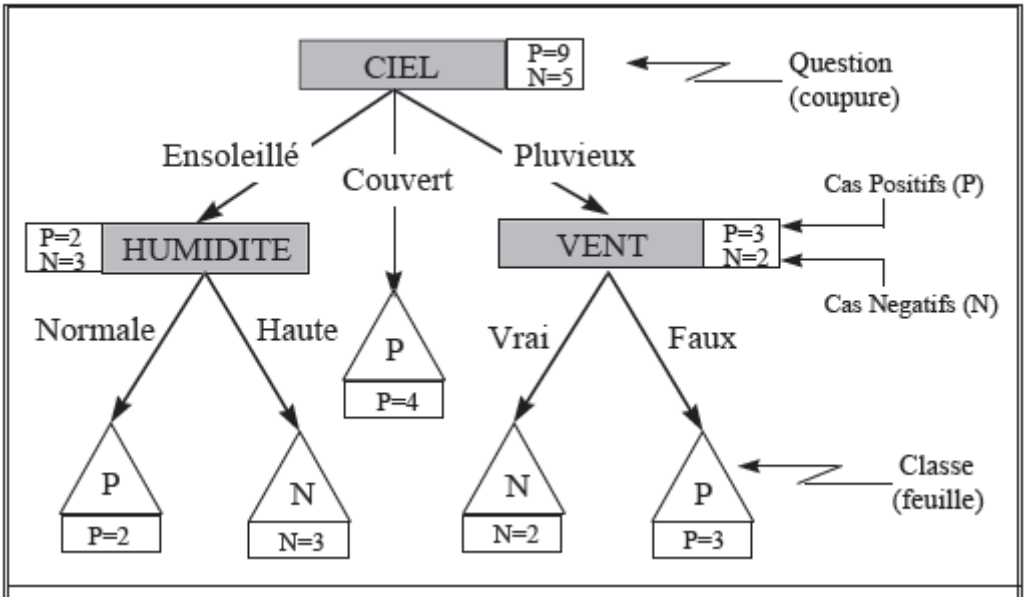


Figure 3.5. Exemple d'arbre de décision simple

Remarque :

L'ordre dans lequel ces attributs sont sélectionnés est primordial pour la construction de l'arbre. On peut avoir différents arbres résultants de l'apprentissage sur la même base d'exemple. Comme montre la ***figure 3.5*** l'arbre est simple mais si on commence avec un autre attribut, on obtiendra l'arbre complexe suivant **(*voir figure 3.6*).**

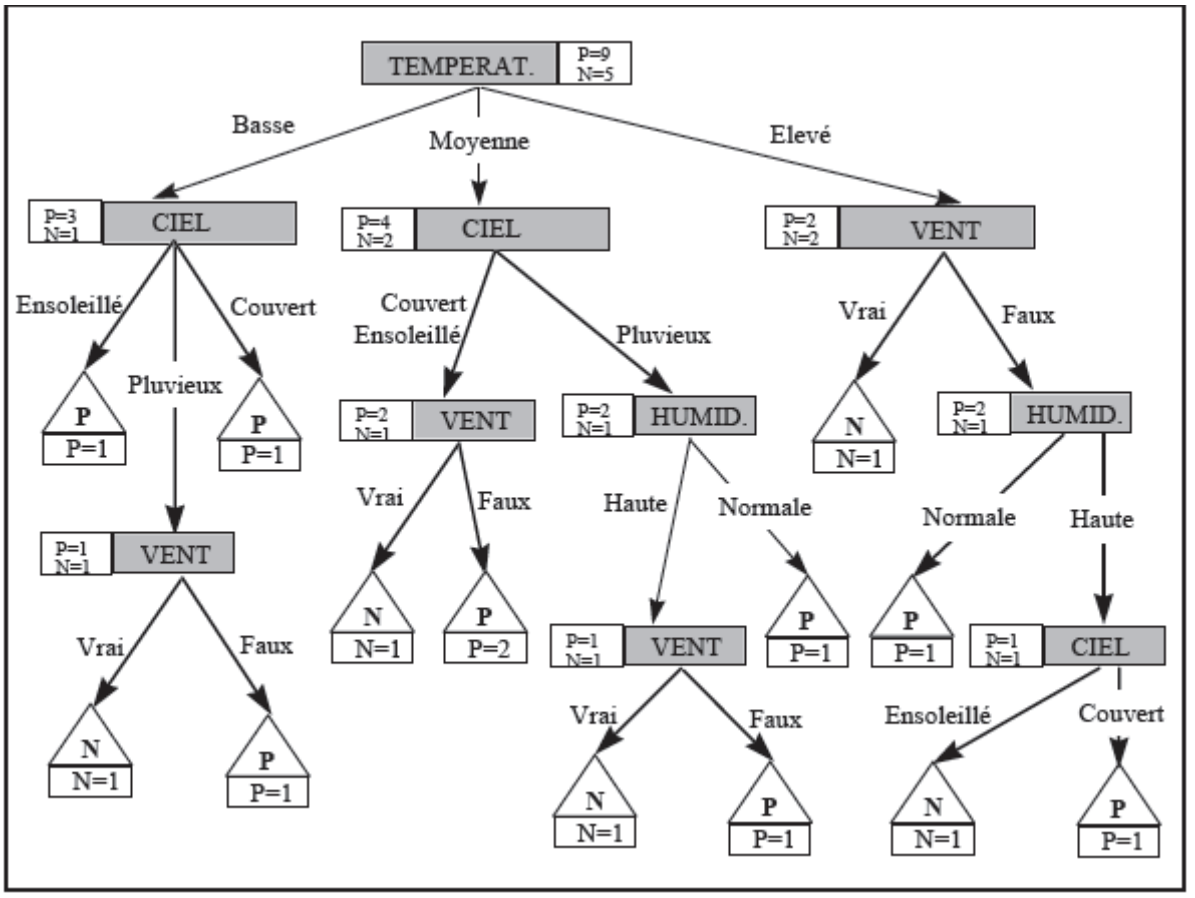


Figure 3.6. Exemple d'arbre de décision complexe.

Le problème de la taille est donc posé et qui dépend du choix de l'attribut. Quinlan a proposé une méthode pour traiter ce problème. Il utilise une technique fondée sur la théorie de l'information de Shannon **[QUI 1993]**. L'idée consiste à appliquer une méthode qui permet de maximiser **le gain d'information** apporté par chaque test.

Gain d'information :

Sa formule est donnée par :

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

Où :

- **test :** l'attribut choisi où n est son arité (les valeurs possibles que peut prendre cet attribut).
- $\mathbf{P_j}$ **:** la proportion d'éléments qui satisfont la $j^{ième}$ branche du test.

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

- **Entropie (p)=** $-\sum_{k=1}^{c} P\left(\frac{k}{p}\right) * \log\left(P\left(\frac{k}{p}\right)\right)$
- **Gini (p) = 1-** $\sum_{k=1}^{c} P\left(\frac{k}{p}\right)^2$

Remarque :

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

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

Règle majoritaire : *Cmaj* est associe à une feuille la classe k de {1,2,…, c} telle que P(k) soit maximum.

Exemple : en appliquant la règle majoritaire sur la ***figure 3.7*** à droite, on obtient alors la figure 3.6 à gauche.

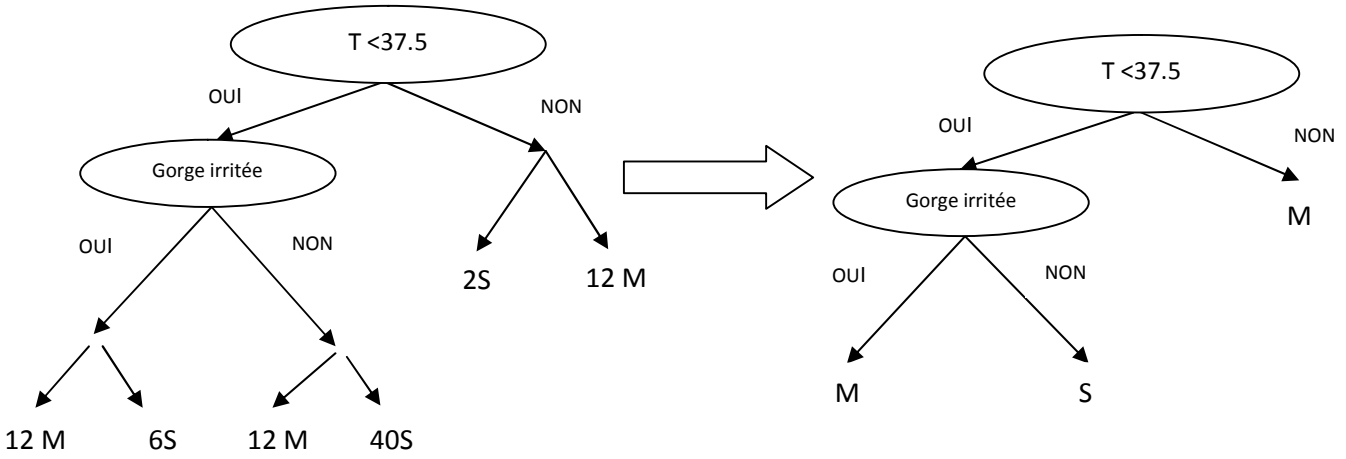


Figure 3.7. Exemple d'application de la règle majoritaire

Donc, pour toutes les méthodes, nous trouvons les 03 opérateurs de construction suivants :

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

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

Algorithme d'apprentissage générique :

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

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

L'idée serait de trouver un critère qui permet d'arrêter la croissance de l'arbre au bon moment. Malheureusement, dans l'état actuel des recherches, un tel critère n'a pu être trouvé. De plus, le risque d'arrêter trop tôt la croissance de l'arbre est plus important que de l'arrêter

trop tard. Par conséquent, les méthodes utilisées précédemment sont divisées souvent en deux phases :

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

Un premier algorithme : CART [BRE 1984]

Cette méthode permet d'inférer des arbres de décision binaires. Nous supposons prédéfini un ensemble de tests binaires. Pour définir l'algorithme, nous allons définir les 03 opérateurs utilisés par la méthode CART pour calculer un bon arbre de décision (phase d'expansion), puis nous verrons la phase d'élagage. Nous nous plaçons dans le cas d'un échantillon S « assez grand » qui peut être découpé en un ensemble d'apprentissage A et un ensemble test T.

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

1. **Décider si un nœud est terminal :** Un nœud p est terminal si **Gini(p) ≤ i_o** ou **n(p) ≤ n_0**, où **i_0** et **n_0** sont des paramètres à fixer.
2. **Sélectionner un test à associer à un nœud :** Soit **p** une position et soit ***test*** un test. Si ce test devient l'étiquette du nœud à la position p, alors on appelle $\mathbf{P_{gauche}}$ (respectivement $\mathbf{P_{droite}}$) la proportion d'éléments de l'ensemble des exemples associés à **p** qui vont sur le nœud en position $\mathbf{p_1}$ (respectivement $\mathbf{p_2}$). La réduction d'impureté définie par le test ***test*** est identique au gain est définie par :

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

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

Phase d'élagage :

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

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

2.

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

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

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

Où :

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

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

<u>Un deuxième algorithme</u> : C4.5 [QUI 1993]

1. Décider si un nœud est terminal : Un nœud p est terminal si tous les éléments associés à ce nœud sont dans une même classe ou si aucun test n'a pu être sélectionné.

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

une fonction qui pénalise les tests qui répartissent les éléments en un trop grand nombre de sous-classes. Cette mesure de la répartition est nommée ***SplitInfo*** et est définie par :

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

Où :

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

Remarques :

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

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

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

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

Phase d'élagage :

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

CHAID :

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

7.2.2. Les réseaux de neurones : [HON 1996]

Définition d'un neurone biologique :

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

Définition d'un neurone formel :

Le neurone formel est une modélisation mathématique qui reprend les principes du fonctionnement du neurone biologique, en particulier la sommation des entrées. Un neurone est une fonction non linéaire paramétrée. Les variables sur lesquelles opère le neurone sont appelées les entrées du neurone, la sortie de neurone n'est qu'une valeur de la fonction. On peut le définir par cinq éléments :

1. La nature de ses entrées $x_1, x_2, \ldots, x_n$.
2. La fonction d'entrée totale V qui définit le prétraitement effectué sur les entrées :
   $$V = W0 + \sum_{i=1}^{n} w_i x_i$$
3. La fonction d'activation ou état du neurone ***f***.
4. La fonction de sortie **y**, qui calcule la sortie en fonction de son état d'activation : $y = f(V)$.
5. La nature de la sortie.

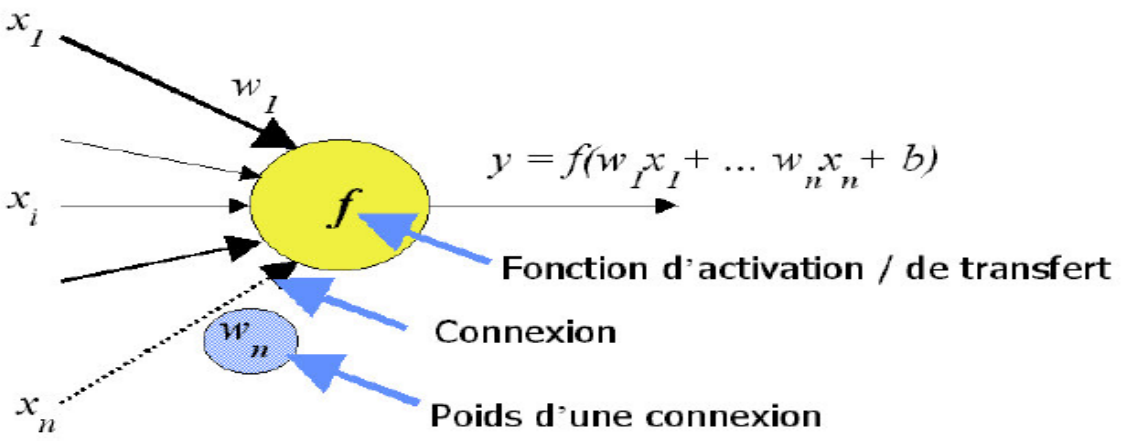


Figure 3.8. Représentation graphique d'un neurone formel.

Définition de réseau de neurones formel :

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

Les réseaux de neurones dans l'apprentissage supervisé :

Dans ce cas, on s'intéresse aux résultats obtenus en fonction des entrées. Pour cela, on initialise les poids à des valeurs quelconques et on calcule la sortie puis on fait une comparaison entre les résultats obtenus et les résultats désirés. Cette comparaison détermine donc ***l'erreur de réseau*.** Il s'agit alors de répartir cette erreur sur chaque poids de réseau de manière à réduire l'erreur. La procédure s'arrête jusqu'à ce que l'erreur soit nulle.

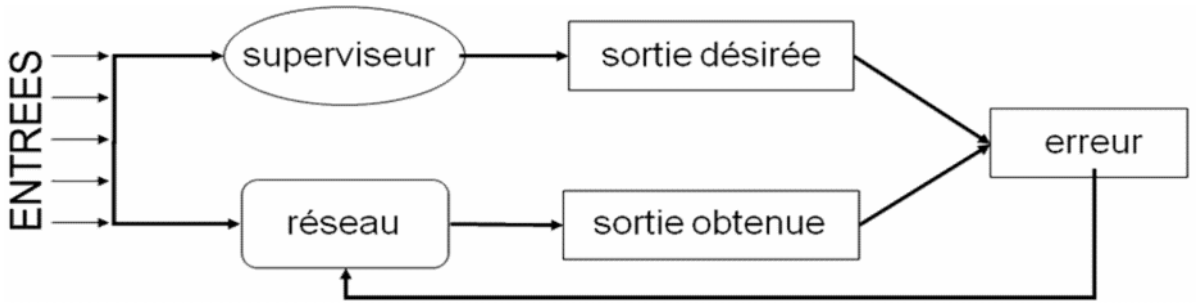


Figure 3.9. Illustration de l'apprentissage supervisé par les réseaux de neurones.

7.2.3. Les algorithmes génétiques :

Ce type d'apprentissage est détaillé dans ***le chapitre 2, section 2.***

7.3. Les méthodes de l'apprentissage non supervisé :

Les méthodes les plus utilisées sont les réseaux de neurones et les clusters.

7.3.1. Les réseaux de neurones :

Dans ce cas, on présente une entrée au réseau et on le laisse évoluer librement jusqu'à ce qu'il se stabilise. Donc, la règle d'apprentissage n'est pas en fonction du comportement de la sortie du réseau, mais plutôt du comportement local des neurones. Cela simplifie considérablement le problème du choix des poids synaptiques appropriés. Donc l'apprentissage non supervisé modifie les poids du réseau en fonction d'un critère interne, indépendant de l'adaptation entre le comportement du réseau et la tache qu'il doit effectuer.

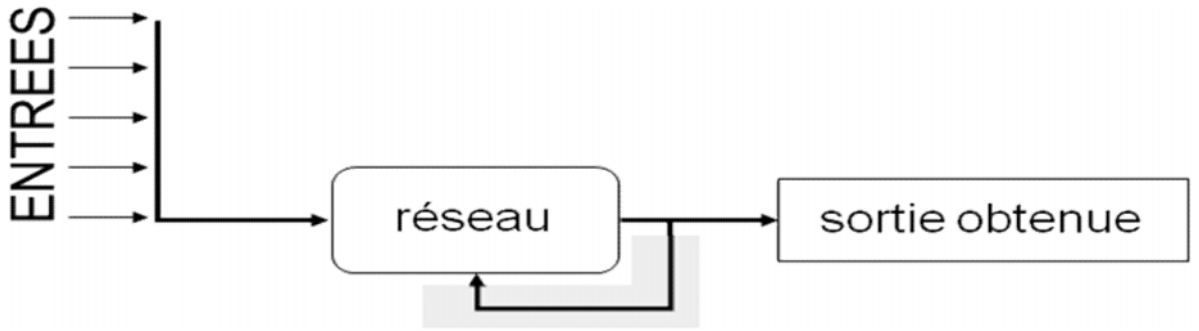


Figure 3.10. Illustration de l'apprentissage non supervisé par les réseaux de neurones.

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

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

7.3.2.1. Les méthodes de partitionnement :

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

Principe :

- ✓ Création d'une partition initiale de K classes,

✓ Puis itération d'un processus qui optimise le partitionnement en déplaçant les objets d'une classe à une autre.

Elles contiennent plusieurs types de méthodes, nous citons : les méthodes approchées/heuristiques qui contiennent aussi d'autres méthodes, nous citons :

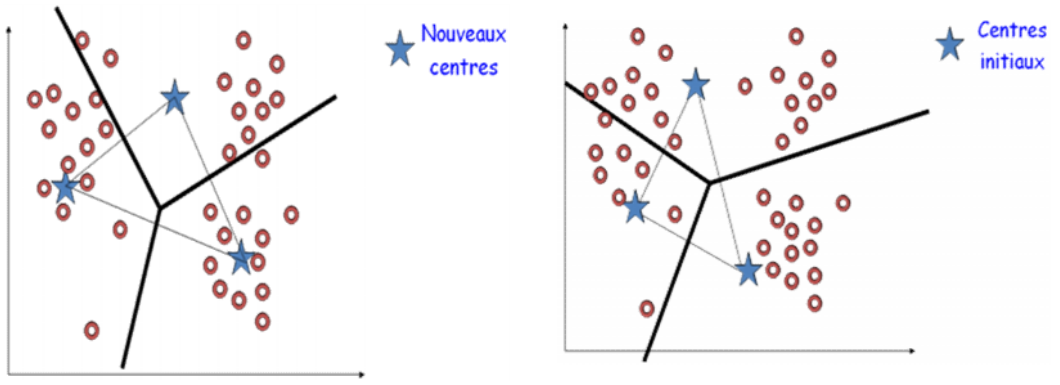
7.3.2.1.1. La méthode de K-Moyennes (centres mobiles) :

Principe :

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

Algorithme :

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



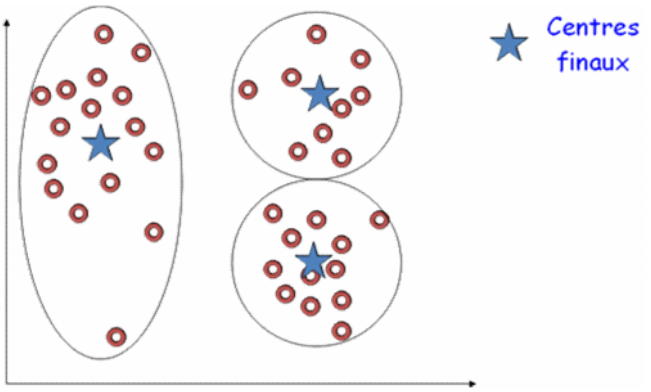


Figure 3.11. Exemple de la méthode K-Moyennes

Points forts :

- Simple à implémenter et à comprendre.
- Passage à l'échelle :
  - Complexité en O(T.K.N) où T est le nombre d'itérations et N le nombre d'exemples.
  - En général K et T sont nettement inférieurs à N.
- Relativement extensible dans le traitement d'ensemble de taille importante.

Points faibles :

- Nécessite de spécifier le nombre de classes à l'avance.
- Utilisable seulement quand la notion de moyenne existe (donc pas sur des données nominales).
- Sensibilité aux conditions d'initialisation (on n'obtiendra pas forcément les mêmes classes avec des partitionnements initiaux différents).
- Les points isolés sont mal gérés (doivent-ils appartenir obligatoirement à un cluster).

7.3.2.1.2 La méthode de K-Medoids :

Même principe que la précédente, on utilise à la place de la moyenne le medoid.

Un medoid est l'objet le plus central dans la classe.

7.3.2.2. Les méthodes hiérarchiques :

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

7.3.2.2.1. Approche ascendante (ou agglomération) :

• Commencer avec un objet dans chaque classe,

• Puis fusionner successivement les 2 classes les plus proches,

• S'arrêter quand il n'y a plus qu'une classe contenant tous les exemples.

Algorithme général de la classification ascendante hiérarchique :

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

Remarque :

Les mesures de similarité pour calculer D (h_i, h_j) sont nombreuses :

1) **Lien simple :** distance des deux objets les plus proches des clusters.
2) **Lien complet :** distance des deux objets les plus éloignés dans les deux clusters.
3) **Lien des moyennes :** la distance entre deux objets est donnée par la distance des centres.
4) **Lien moyen:** distance moyenne de toutes les distances entre un membre d'un cluster et un membre de l'autre.

Exemple Stratégie de lien simple :

Elle mesure la distance euclidienne entre les deux points les plus proches, l'un dans un bloc de partition, l'autre dans le second.

D (h_i,h_j) = Minimumx $\Delta(x,y)$; x et y appartiennent à h_i et h_j respectivement.

Quand les blocs sont réduits à un élément, cet indice mesure la distance euclidienne entre ces deux éléments. La hiérarchie indicée trouvée est monotone.

Reprenons un ensemble à six exemples et donnons la table des instances euclidiennes entre les exemples. Cette dernière est aussi la table TD, qui est identique sur les couples d'objets. Comme cette table est symétrique, seule une moitié est représentée.

Initialement :

$h_1 = \{a\}$, $h_2 = \{b\}$, $h_3 = \{c\}$, $h_4 = \{d\}$, $h_5 = \{e\}$, $h_6 = \{f\}$,

	a	b	c	d	e	f
a	0	**1.1**	**1.1**	3	6	5
b		0	**1.1**	4	5.5	4.2
c			0	3	6.5	5.3
d				0	9	8
e					0	1.9
f						0

(1)

	{a, b}	{c}	{d}	{e}	{f}
{a, b}	0	**1.1**	3	5.5	4.2
{c}		0	3	6.5	5.3
{d}			0	9	8
{e}				0	1.9
{f}					0

(2)

	{a, b, c}	{d}	{e}	{f}
{a, b,c}	0	3	5.5	4.2
{d}		0	9	8
{e}			0	**1.9**
{f}				0

(3)

	{a, b, c}	{d}	{e, f}
{a, b,c}	0	**3**	4.2
{d}		0	8
{e, f}			0

(4)

	{a, b, c, d}	{e, f}
{a, b, c, d}	0	**4.2**
{e, f}		0

(5)

	{a, b, c, d, e, f}
{a, b, c, d, e, f}	0

(6)

Donc, on trouve la hiérarchie suivante (voir figure 3) :

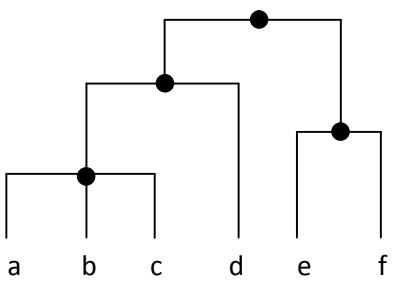


Figure 3.12. Stratégie de lien simple.

7.3.2.2.2. Approche descendante (ou par division) :

• Tous les objets sont initialement dans la même classe,

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

Intérêt des méthodes hiérarchiques :

- Facile à implémenter.
- Fournir une structure facile et compréhensible pour une analyse détaillée.

Difficultés des méthodes hiérarchiques :

- Passage à l'échelle difficile :
  Complexité en $O(N^2)$.
- Méthodes non extensibles pour des ensembles de données de grande taille.

8. Les algorithmes d'extraction des règles de classification : [COL 2001]

Les algorithmes d'extraction de règles sont des algorithmes évolutifs, leurs buts sont de trouver des règles justes et précises, qui par la suite nous donnent certaines informations et connaissances concernant l'analyse de la base de données.

Il existe plusieurs algorithmes d'extraction de règles, nous avons étudiés trois d'entres eux : X2R, OneR et les tables de décision.

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

8.1. L'algorithme X2R : [HUA 1995]

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

***Étape 1* : Génération de la règle**

Elle choisit l'instance qui revient le plus fréquemment dans le jeu d'instances pour générer la règle, puis la prochaine instance, de cette façon, X2R peut manipuler des bruits dans les données, ces derniers sont des instances qui n'ont pas de valeur de classe, ou certains attributs n'ont pas de valeurs.

Le cœur de cette étape est un algorithme avide (glouton) qui trouve la plus petite règle basée sur un premier ordre d'information (une seule condition), ceci peut différencier les instances en prenant en considération l'instance des autres classes.

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

***Etape 2* : Regroupements ou segmentation de règle**

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

***Etape 3* : Nettoyage des données**

A chaque regroupement de règle, les règles redondantes (qui se répètent) sont éliminées, puis les règles spécifiques sont remplacées par des règles plus générales.

Les avantages :

- Les règles sont courtes ce qui implique plus de compréhensibilité.
- Le nombre de ces règles est réduit.
- On obtient un taux de précision de 80% à 100%.

Les inconvénients :

- Les règles générées sont de premier ordre.
- Plusieurs contraintes sont éliminées ce qui implique la perte de certaines informations.
- Le taux d'erreur est élevé par rapport aux taux d'incohérence dans les données d'origine.

8.2. L'algorithme OneR : [HOL 1993]

Est un algorithme de classification simple. Il produit des règles simples basées seulement sur un attribut qui sont de la forme :

Si attribut=valeur alors classe

Son principe est donné par l'algorithme suivant :

Début

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

Fin.

Exemple :

1. **Cas des valeurs alphabétiques :**

CIEL	Non Jouer	Jouer
Ensoleillé	**3**	2
Couvert	0	**4**
Pluvieux	2	**3**

Si ciel = ensoleillé alors jouer = Non

Si ciel = couvert alors jouer = Oui

Si ciel = pluvieux alors jouer = Oui

2. **Cas des valeurs numériques :**

Supposons que la température a les valeurs suivantes :

64	65	68	69	70	71	72	72	75	75	80	81	83	85
oui	non	oui	oui	Oui	non	non	oui	oui	oui	non	oui	oui	non

Dans ce cas, nous devons suivre les étapes suivantes :

1. Partitionner la séquence en plaçant des séparateurs quand la classe change.

64	65	68	69	70	71	72	72	75	75	80	81	83	85
O	N	O	O	O	N	N	O	O	O	N	O	O	N

2. Regrouper les séquences adjacentes qui ont un nombre minimal de la classe majoritaire.

64	65	68	69	70	71	72	72	75	75	80	81	83	85
O	N	O	O	O	N	N	O	O	O	N	O	O	N

3. Regrouper les séquences adjacentes qui ont la même classe majoritaire.

64	65	68	69	70	71	72	72	75	75	80	81	83	85
O	N	O	O	O	N	N	O	O	O	N	O	O	N

4. Les règles générées seront :

Si température $\leq$ 77.5 alors jouer = OUI

Si température > 77.5 alors jouer = NON.

8.3. <u>Les tables de décision</u> : [KOH 1995]

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

Les tables de décision s'appliquent cependant difficilement aux décisions découlant d'un nombre important de conditions, puisque les possibilités de combinaisons connaissent alors une croissance exponentielle. De même, elles ne permettent pas de représenter clairement les décisions comportant des boucles. Pour appliquer la technique, on doit :

1. Identifier les différentes **conditions** influant sur la décision. En programmation, ces conditions découleront des attributs pertinents des données (par exemple, le sexe ou l'âge d'un individu); les conditions sont en fait tous les "Si" (ou les *"if"* en programmation) reliés à la décision.

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

9. <u>Conclusion</u> :

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

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

1. Introduction :

Dans ce chapitre, nous allons donner les différentes étapes nécessaires pour la conception d'une méthode mémétique qui a pour but la construction d'un classifieur pour l'extraction de connaissances à partir d'un ensemble d'apprentissage sous forme de ***règles de classification*** pour attribuer des classes aux nouveaux éléments. Afin d'atteindre ce but, nous allons diviser ce chapitre en deux sections :

Section 1 : nous allons présenter un état de l'art pour l'extraction des règles de classification.

Section 2 : nous présentons notre conception qui est divisée en deux parties principales :

Partie 1 : contient la construction d'un classifieur en utilisant une approche basée sur les algorithmes génétiques purs.

Parie 2 : le classifieur est construit à l'aide d'une approche basée sur les algorithmes mémétiques (les algorithmes génétiques hybridés par les méthodes de recherche locale).

Et cela, dans le but de répondre à la question suivante :

Que nous apporte l'approche hybride par rapport à l'approche non hybride sur la qualité du classifieur construit ? Ou qu'elle est l'utilité d'intégrer les méthodes de recherche locale dans les algorithmes génétiques ?

2. Section 1 : Etat de l'art

Dans cette section, nous allons faire un état de l'art sur les différentes méthodes d'extraction des règles de classification qui peuvent être classées en 03 catégories selon le type de l'algorithme utilisé :

2.1. Par les algorithmes d'extraction :

Ce point est détaillé dans le ***chapitre 3, section 7***.

2.2. Par les algorithmes génétiques :

Parmi ces algorithmes, nous allons parler de GAssist et de Hider.

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

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

$$MDL = W * TL + EL$$

Où :

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

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

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

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

Où :

- **r** : est un individu.

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

Le pseudo code de Hider est le suivant :

Procédure Hider

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

Procédure AlgEvo(T)

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

Remarque 1:

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

Remarque 2 :

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

2.3. Par les algorithmes mémétiques: [BAC 2004] [BAC 2006]

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

1. Méthode Rule set-wise (MPLCS-RS):

Cette méthode utilise un opérateur de recherche locale et qui est intégré dans GAssist, il est défini par 03 phases :

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

Cet opérateur est intégré dans GAssist en utilisant deux stratégies :

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

2. Méthode Rule-wise (MPLCS-R):

Cette méthode utilise trois opérateurs de recherche locale suivants :

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

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

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

3. Section 2 : Présentation des étapes de conception

Dans cette section, nous allons présenter les étapes nécessaires pour la construction de notre classifieur.

3.1. Partie I : Construction du classifieur avec l'approche génétique

Dans cette partie, le classifieur est construit à l'aide des algorithmes génétiques (AG), dont le fonctionnement est le suivant :

- Initialiser les paramètres de l'AG (taille de la population, probabilité de croisement, probabilité de mutation, taille d'un individu, codage des règles).
- Créer la population initiale.
- Evaluer chaque individu de la population.
- Appliquer le cycle de l'AG.

3.1.1. Création de la population initiale :

La population initiale est générée de façon aléatoire mais avec un contrôle c'est-à-dire que nous devons vérifier à chaque fois que le chromosome produit donne une règle cohérente. La taille de cette population est déterminée par l'utilisateur selon les besoins et les contraintes et plus la taille de la population est grande, mieux l'espace des solutions sera exploré. Par conséquent, les résultats seront plus précis ; mais en contrepartie, le temps d'exécution sera plus long. Avant de générer cette population, nous devons d'abord représenter la structure d'un chromosome en utilisant le codage des règles suivant.

3.1.2. Codage des règles: [BEN 2007]

Pour pouvoir utiliser les règles de classification au sein d'un algorithme génétique, il faut les coder. Pour cela, il existe plusieurs approches. Les plus connues sont l'approche de Michigan et l'approche de Pittsburgh.

Dans la première approche, chaque règle est considérée comme un individu. Cependant, dans la seconde approche, un ensemble de règles générées dans la même itération est considéré comme un individu.

Dans notre étude, nous allons utiliser l'approche de Michigan car l'autre approche est coûteuse en espace mémoire **[LAE 2003].**

Le chromosome est donc composé de suite de gènes où chaque prémisse est codée sur 01 gène (contenant 03 champs): le 1^{er} champ indique si l'attribut correspondant à cette prémisse est actif ou non (prémisse existe), le $2^{ième}$ champ contient la valeur de l'opérateur de comparaison et le dernier champ donne la valeur correspondante à cet attribut.

Ces gènes sont rangés dans le même ordre de leur représentation dans la base de données, de sorte que la valeur d'index d'un gène, représente la position de l'attribut associé dans cette base de données. Le dernier gène (01 champ) du chromosome contient la valeur de la classe prédite. Donc, la taille d'un chromosome est :

((NB-1)*3)+1 ;

Où :

NB : est le nombre d'attributs de la base de données.

Le pseudo code suivant montre comment initialiser dans la population initiale des règles:

Procédure remplir Chromosome ( )

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

3.1.3. Evaluation de chaque individu :

Revient à attribuer à chaque individu une valeur calculée par la fonction d'adaptation (ou *d'évaluation qui est en fait la fonction à optimiser*).

Notons qu'il n'y pas de fonction précise qui nous permet de calculer avec exactitude l'adaptation d'un individu. Pour cela, nous avons opté pour les critères suivants :

1. Maximiser la couverture de la règle.
2. Maximiser le taux de précision de la règle.
3. Minimiser la taille de la règle car la compréhensibilité d'une règle est mesurée par son nombre d'attributs, une règle est moins compréhensible si sa taille est grande.

La fonction d'adaptation a la forme suivante :

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

Où :

λ_i est une valeur entière comprise entre 0 et 1, avec $\sum_{i=1}^{3} \lambda_i = 1$

Les pseudo-codes correspondants au calcul de la fitness sont :

1. **Fonction Couverture :** cette fonction calcule la couverture d'une règle j par rapport à la base de données d'apprentissage. Nous dirons que l'instance i est couverte par la règle j si la partie prémisse de la règle j est incluse dans la partie antécédente de l'instance i.

Début

Entier N=nombre d'instances de la base de données ;
Entier Nb=nombre d'attributs de la partie condition de la règle ;

```
Entier cpt=0 ;
Entier Couverture=0 ;
Pour i=0 à (N-1)
Faire
    Si(la ième  antécédent de l'instance satisfait le i éme attribut de la règle)
      Alors
          cpt++ ;
    Si(cpt==Nb) //la partie condition de la règle est incluse dans la partie antécédente
                // de l'instance
      Alors
          Couverture++;
Fait;
Retourner (Couverture);

Fin.
```

2. **TP (True positive) :** cette fonction calcule le TP d'une règle. La règle j a un TP=1 par rapport à l'instance i de la base d'apprentissage si la partie prémisse de la règle j est incluse dans la partie antécédente de l'instance i et qui ont la même classe.

```
Début
    Entier N=nombre d'instances de la base de données ;
    Entier Nb=nombre d'attributs de la partie condition de la règle+1;
    Entier cpt=0 ;
    double TP=0 ;
```

```
Pour i=0 à (N-1)
Faire
    Si(la i ième antécédent de l'instance satisfait le i éme attribut de la règle)
      Alors
          cpt++ ;

    Si(cpt==Nb) //la partie condition de la règle est incluse dans la partie antécédente
                // de l'instance et la classe de la règle est égale à la classe de l'instance
      Alors
          TP++;
Fait;
Retourner (TP);

Fin.
```

3. Fonction Taille : elle calcule le nombre d'attributs actifs d'une règle donnée.

3.1.4. Application du cycle de l'AG : Nous avons choisi l'algorithme génétique ***Steady State*** **[SYS 1991]**

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

Fin.
```

Les algorithmes correspondants à chaque opération du cycle sont les suivants :

3.1.4.1. La sélection:

Nous avons utilisé le principe de sélection par Tournoi ***« Tournament Selection »,*** le pseudo-code suivant illustre cette opération :

Fonction Individu Tournament Selection()

```
Début
    Entier p1, p2 ;
    Entier taille ; //taille représente la taille de la population
```

```
Répéter
        Tirer aléatoirement deux nombre p1 et p2 ;  //p1<=taille et p2<=taille
Jusqu'à (p1≠p2)
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
Fsi;
Fin.
```

3.1.4.2. Le croisement :

Nous avons choisi le croisement en un point, ce point est choisi aléatoirement, donc les prémisses entre deux règles vont être échangées.

Fonction Individu Croisement (Indiv1, Indiv2)

```
Début
    Entier k ; //k est le point de coupure
    Entier L ; //taille d'un chromosome
```

```
Double pm ; //La probabilité de croisement
k=fonction aléatoire (0, L) ;
Tant que ((k n'est pas multiple de 3)ou  (k<3) ou (k>L-4))
 Faire
      k=fonction aléatoire (3, L-3) ;
 Fait ;
 //Générer l'enfant 1
 //Copier le premier chromosome
Pour i=0 à k-1
       Indiv[i] :=Indiv1[i] ;
    //Copier le deuxième chromosome
Pour i=k à L-1
       Indiv[i] :=Indiv2[i] ;
//Générer l'enfant 2
//Copier le premier chromosome
Pour i=0 à k-1
       Indiv[i] :=Indiv2[i] ;
    //Copier le deuxième chromosome
Pour i=k à L-1
       Indiv[i] :=Indiv1[i] ;

Retourner le meilleur des 2 enfants ;
Fin.
```

3.1.4.3. La mutation :

La mutation consiste à changer la valeur d'une case ou plusieurs cases d'un chromosome selon la probabilité de mutation.

Fonction Individu Mutation (Indiv)

Début

Entier i, j ; i=0 ; j=0 ; entier L=taille du chromosome ;
double pm ;//Probabilité de mutation
Tant que (i<L-1)
Faire
Si(r<pm)
Alors
- Muter la case Active/Désactive ;
- i++ ;
- Muter la case opérateur dans le cas où l'attribut j est numérique mais avec une autre valeur de l'opérateur différente de la valeur actuelle ;
- i++ ;
- Muter la case valeur de l'attribut par une autre valeur parmi les valeurs possibles de l'attribut j (j=i/3) mais elle doit être différente de la valeur actuelle ;
- j++ ;

Fsi ;
Fait ;
Fin.

3.1.4.4. Remplacement : il existe plusieurs stratégies de remplacement, nous avons choisi de remplacer le mauvais individu de la population par l'individu résultant du croisement et de mutation.

3.1.5. Construction du classifieur: [BEN 2007]

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

Procédure Construction du classifieur

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

Remarque :

Le critère d'arrêt pour la construction de notre classifieur serait : Ou bien on arrête l'exécution de l'algorithme car la base de données est vide : toutes les instances ont été couvertes ou bien au bout d'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 données qui ne peuvent être couvertes par aucune règle.

Les fonctions utilisées dans la procédure de construction :

Nous avons utilisé dans cette procédure deux fonctions :

1. La fonction ***Décoder*** **:** cette fonction permet de transformer un individu en une règle, le pseudo-code ci-dessous représente son fonctionnement.

Fonction règle Décodage (Individu)

```
Début
    Entier i, j ; i=0 ; j=0 ; entier L=taille du chromosome ;
    Chaîne de caractère règle=" if " ;
    Tant que (i<L-1)
    Faire
        Si (chromosome[i]=1) //L'attribut est inactif
            Alors
                i=i+3 ; j++ ;;
            Sinon //L'attribut est actif
                i++ ;
        Fsi;
        Si(chromosome[i]=0)
                règle=règle+"=" ;
        Si(chromosome[i]=1)
                règle=règle+"≥" ;
        Si(chromosome[i]=2)
                règle=règle+"≤" ;
        i++ ;
        Si(type(attribut(j)est nominal))
            Alors
                règle=règle+valeur correspondante au contenu de Chromosome[i]+"and";
            Sinon //Le type de l'attribut est numérique
                règle=règle+ chromosome[i]+"and" ;
        Fsi;
        i++ ; j++ ;
        //Passer à l'attribut classe
        règle=règle+"then"+"class"+valeur correspondante à l'attribut classe ;

    Fait;
    Retourner(règle);
Fin.
```

2. La fonction ***Est_Couverte*** **:** permet de tester si une instance est couverte par la règle extraite par l'AG. On dit qu'une instance est couverte par la règle extraite si la partie prémisse de cette dernière est incluse dans la partie antécédante de l'instance. Le pseudo-code correspondant à cette fonction est le suivant :

Fonction booléen Est_Couverte(Règle_extraite, instance)

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

3.2. Partie II : Construction du classifieur avec l'approche mémétique

Dans cette partie, nous allons intégrer les deux méthodes suivantes de recherche locale dans le cycle d'un algorithme génétique :

1. Méthode de recherche locale simple **(RLS)**.
2. Méthode de recherche locale génétique **(RLG)**.

La description de ces deux méthodes ainsi que les pseudo-codes qui les correspondent sont représentés dans ce qui suit :

3.2.1. Méthode de recherche locale simple : [BEN 2008]

Cette méthode consiste à créer tous les voisins d'un individu (***x)*** qui diffèrent par un seul attribut et choisir le meilleur individu entre x et ses voisins. Le pseudo-code correspondant à cette méthode est le suivant :

Recherche locale simple (Individu *indiv*)

```
Début
    Créer k individus autours de l'individu indiv  par mutation d'un seul attribut;
    Evaluer indiv et les k individus obtenus ;
    Best :=indiv ;
    Pour i=1 à k
    Faire
        Si (indiv(i)) est meilleur que Best)
            Alors
                Best :=indiv(i) ;
        Fsi ;
    Fait ;
        Replacer indiv par Best ;
Fin
```

3.2.2. Méthode de recherche locale génétique : [BEN 2008]

Dans cette méthode, nous appliquons un AG sur un individu x et tous ses voisins et de choisir le meilleur entre eux. Son pseudo-code est le suivant :

Recherche locale génétique (Individu *indiv*)

```
Début
    Créer k individus par mutation de l'individu indiv ;
    Evaluer indiv et les k individus obtenus par mutation;
    AG (indiv, indiv1, indiv2, indiv3,… ; indivk) ;
    Best :=Individu généré par l'AG ;
        Si (indiv est meilleur que Best)
            Alors
                Best :=indiv;
        Fsi ;
    Remplacer indiv par Best ;
Fin.
```

3.2.3. Les stratégies d'application des méthodes de recherche locale :

Nous allons intégrer les deux méthodes précédentes dans un AG en appliquant les stratégies suivantes :

1. Appliquer la méthode de recherche locale simple après croisement, on la note **RLS(C).**
2. Appliquer la méthode de recherche locale simple après mutation, on la note **RLS(M).**

3. Appliquer la méthode de recherche locale simple après croisement et après mutation, on la note **RLS(C+M)**.
4. Appliquer la méthode de recherche locale génétique après croisement, on la note **RLG(C).**
5. Appliquer la méthode de recherche locale génétique après mutation, on la note **RLG(M)**.
6. Appliquer la méthode de recherche locale génétique après croisement et après mutation, on la note **RLG(C+M)**.

Dans le pseudo code suivant, nous montrons un exemple d'intégration de la méthode RLS dans un AG après croisement et après mutation :

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

4. Conclusion :

Nous avons vu dans ce chapitre les travaux faits pour l'extraction des règles de classification et nous avons présenté la conception du deux classifieurs, l'un est basé sur les algorithmes génétiques et l'autre sur les algorithmes mémétiques dont le but est de faire une étude comparative entre les deux. Pour atteindre ce but, nous devons procéder à la phase des tests dans le chapitre suivant en appliquant toutes les stratégies présentées ci-dessus sur les classifieurs construits.

1. Introduction :

La phase des tests consiste à appliquer le classifieur construit aux bases de données UCI. Pour cela, nous divisons ce chapitre en 03 sections :

Section 1 : nous présentons les outils utilisés pour l'implémentation de notre classifieur et de la mesure utilisée pour évaluer sa qualité.

Section 2 : dans cette section, nous parlons des paramètres utilisés par L'AG, et nous procédons aussi au réglage de certains paramètres.

Section 3 : cette section est divisée en 03 parties :

- La première partie donne les résultats obtenus par l'application du classifieur construit à l'aide d'un algorithme génétique pur sur quelques bases de données UCI.
- La deuxième partie montre les résultats obtenus par l'application du classifieur mémétique sur quelques bases de données UCI.
- Dans la 3^{ième} partie, nous allons faire une étude comparative entre les résultats obtenus par l'application de l'approche mémétique, l'approche génétique et quelques algorithmes de classification tels que J48, Table de décision, OneR.

2. Section 1 : Outils utilisés

Afin de pouvoir appliquer les différentes stratégies sur notre classifieur, il faut d'abord l'implémenter. Pour cela nous avons utilisé un langage de programmation, des bibliothèques externes et des bases de données.

2.1. Le langage de programmation utilisé :

Nous avons utilisé le langage de programmation JAVA (version 1.6.0) qui est un langage orienté objet, il a été développé par Sun Microsystems 1991, conçu sur le modèle du langage C++, simple, concis et portable sur toutes les plateformes et systèmes d'exploitation.

2.2. La bibliothèque externe :

Nous avons utilisé Weka (**W**aikato **E**nvironment for **K**nowledge **A**nalysis) **[WIT 2005]** qui est un logiciel écrit en java développé à l'Université de Waikato en nouvelle Zélande.

Weka est un ensemble d'outils permettant de manipuler et d'analyser des fichiers de données, implémentant la plupart des algorithmes d'intelligence artificielle, entre autres, les arbres de décision et les réseaux de neurones. Il se compose principalement :

- ✓ De classes Java permettant de charger et de manipuler les données.
- ✓ De classes qui implémentent pour les principaux algorithmes de classification supervisée ou non supervisée, ce qui facilite l'étude comparative entre notre approche et les différents algorithmes tels que J48, Id3, Table de décision, OneR,…
- ✓ D'outils de sélection d'attributs, de statistiques sur ces attributs.
- ✓ De classes permettant de visualiser les résultats.

2.3. Bases de données de l'UCI :

C'est une bibliothèque qui contient un grand nombre de BD (benchmarks) sélectionnées par l'**U**niversité de **C**alifornie **I**vrine) UCI **[INT 01]** et qui le met à la disposition de la communauté des chercheurs en DATA MINING. Les bases de données UCI sont largement utilisées dans le monde et considérées comme une référence, d'où l'importance de les utiliser pour l'évaluation des qualités d'un algorithme et de comparer leurs performances par rapport à d'autres algorithmes. Dans nos tests, nous allons utilisées les bases de données suivantes :

- **Iris :** elle contient des données sur les fleurs de certaines plantes, le but est de déterminer la classe d'appartenance d'Iris. Il faut dire qu'Iris est une des plus célèbres BD ; elle est très utilisée dans le domaine de recherche et souvent citée dans la littérature.
  - Nombre d'instances = 150.
  - Nombre d'attributs = 5.
  - Nombre de classes =3.
- **Diabètes :** c'est une base de données qui touche le domaine médical plus précisément la maladie du diabète. Le but est de déterminer le facteur de risque dans la population en diagnostiquant cette maladie sur les différents patients.
  - Nombre d'instances = 768.
  - Nombre d'attributs = 9.
  - Nombre de classes =2.
- **Mushroom :** Cette base de données contient les descriptions des échantillons hypothétiques correspondant à 23 espèces de champignons de la famille agaricacée. Chaque espèce est identifiée comme comestible ou toxique.

- Nombre d'instances = 8124.
- Nombre d'attributs = 23.
- Nombre de classes =2.

- **Kdd_train :** Cette base de données concerne la sécurité des réseaux : il s'agit de détecter des attaques sur les réseaux, selon différents facteurs : le protocole utilisé, le serveur (http, nntp…etc.), les adresses IP de destination…etc.
  - Nombre d'instances = 11 419.
  - Nombre d'attributs = 42.
  - Nombre de classes =2.

Le ***tableau 03*** résume les informations de ces bases de données.

Bases de données	**Nombre d'instances**	**Nombre d'attributs**	**Nombre de classe**
Iris	150	5	3
Diabètes	768	9	2
Mushroom	8124	23	2
Kdd-train	11419	42	2

Tableau 03 : Résumé sur les bases de données utilisées.

2.4. La mesure utilisée pour évaluer la qualité du classifieur :

Dans nos tests, nous allons évaluer la qualité d'un classifieur construit en utilisant une mesure appelée précision qui nous donne le pourcentage des instances bien classées.

3. Section 2 : Paramètres de l'AG

Notre algorithme génétique (AG) utilise les paramètres de dimensionnement suivants pour la construction de classifieur :

- Taille de la base d'apprentissage.
- La taille de la population.
- Le nombre d'itérations.
- La probabilité d'application des opérateurs génétiques (croisement et mutation).
- Les paramètres « λ_i » de la fonction objectif.

Certains de ces paramètres ont une influence sur la qualité de classifieur. Ces paramètres sont :

- La taille de la base d'apprentissage.
- Les paramètres « λ_i » de la fonction objectif.

3.1. Influence de la taille de la base d'apprentissage :

Pour voir l'influence de ce paramètre sur la précision du classifieur, nous prenons la base de données « diabètes » et nous choisissons des tailles différentes de la base d'apprentissage puis nous exécutons le classifieur génétique 10 fois successives pour chaque taille. Les résultats trouvés sont résumés dans le ***tableau 04***.

	Taille de la base d'apprentissage								
	10%	**25%**	**40%**	**50%**	**60%**	**75%**	**85%**	**90%**	**100%**
Précision du classifieur	75.65%	81.51%	96.61%	92.57%	80.33%	98.95%	98.43%	88.02%	90.75%
	90.49%	89.18%	43.35%	51.69%	52.99%	95.05%	83.98%	96.48%	98.17%
	87.10%	90.36%	98.56%	83.20%	96.61%	56.64%	95.57%	98.17%	93.48%
	36.97%	78.90%	65.36%	98.82%	98.56%	83.33%	100%	70.31%	91.01%
	40.23%	94.79%	99.6%	96.74%	95.31%	99.47%	85.15%	92.44%	98.95%
	93.09%	62.23%	98.56%	92.05%	97.78%	96.22%	80.59%	90.62%	92.57%
	87.36%	98.04%	60.93%	99.21%	84.11%	98.69%	86.06%	94.66%	95.31%
	93.33%	90.36%	97.13%	92.96%	98.82%	93.75%	86.84%	91.66%	88.15%
	86.19%	39.71%	88.15%	84.11%	95.96%	96.22%	99.60%	95.70%	81.38%
	81.77%	53.90%	79.42%	88.67%	95.57%	81.77%	98.95%	98.69%	91.14%
Précision Moyenne	***77.22%***	***77.9%***	***82.77%***	***87.96%***	***89.60%***	***90.01%***	***91.51%***	***91.67%***	***92.09%***

Tableau 04: Test sur la taille de la base d'apprentissage.

Ce tableau peut être schématisé par la ***figure 5.1***.

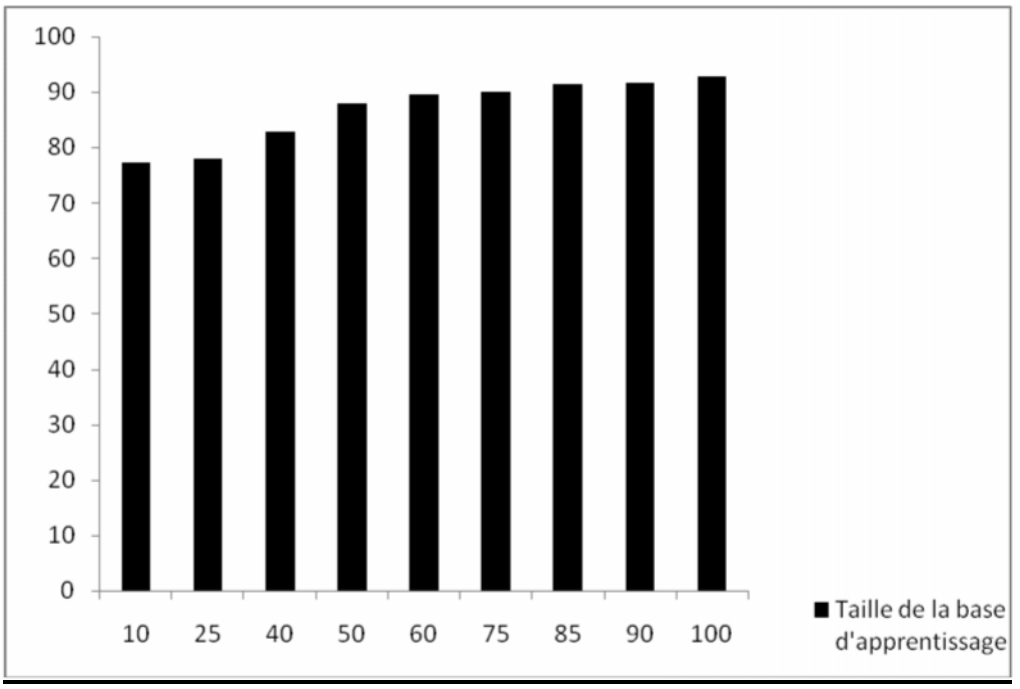


Figure 5.1. Influence de la taille de la base d'apprentissage sur la qualité du classifieur

Remarque : nous remarquons que la précision moyenne (donnée dans la dernière ligne du ***tableau 03***) du classifieur augmente si la taille de la base d'apprentissage augmente.

3.2. Influence des paramètres « λ_i » de la fonction objectif:

Nous allons présenter dans cette partie, un ensemble de tests d'exécution sur la base de donnée Diabètes, en modifiant les « λ_i » tel que :

- λ_1 : paramètre associé au taux de couverture de la règle.
- λ_2 : paramètre associé au taux de précision de la règle.
- λ_3 : paramètre associé à la taille de la règle (compréhensibilité).

Nous numérotons d'abord les différentes valeurs de λ_i :

- **Cas 1 :** 0.8, 0.2, 0.0.
- **Cas 2 :** 0.5, 0.5, 0.0.
- **Cas 3 :** 0.4, 0.6, 0.0.
- **Cas 4 :** 0.3, 0.7, 0.0.
- **Cas 5 :** 0.2, 0.8, 0.0.

Et nous exécutons le classifieur génétique 10 fois successives pour chaque cas, nous obtenus le ***tableau 05***.

	Cas 1	**Cas 2**	**Cas 3**	**Cas 4**	**Cas 5**
Précision du classifieur	99.47%	98.69%	52.47%	90.75%	90.75%
	98.43%	99.73%	98.82%	98.69%	98.17%
	38.28%	99.86%	90.49%	97.91%	93.48%
	81.77%	35.12%	99.60%	90.62%	91.01%
	85.41%	99.60%	97.26%	98.56%	98.95%
	55.98%	98.95%	97.26%	95.70%	92.57%
	56.38%	98.04%	95.31%	99.08%	95.31%
	37.76%	61.67%	98.95%	95.18%	88.15%
	99.47%	65.10%	69.01%	64.71%	81.38%
	93.61%	98.56%	92.57%	82.03%	91.14%
Précision Moyenne	***74.66%***	***87.36%***	***89.41%***	***91.32%***	***92.09%***

Tableau 05 : Influence des paramètres « λ_i » de la fonction objectif

Remarques :

1. Nous avons pris λ_3=0, car nous avons augmenté dans la génération des chromosomes la chance d'obtenir des attributs désactivés donc la taille de la règle diminue et par conséquence elle sera plus compréhensible et la précision augmente.
2. Nous remarquons que la précision moyenne du classifieur augmente si le paramètre $\boldsymbol{\lambda_2}$ associé au taux de précision augmente.

3.3. Conclusion : pour réaliser les tests sur les bases de données UCI, nous allons affecter des valeurs aux paramètres de l'AG cités ci-dessus. Le ***tableau 06*** nous donne ces valeurs :

Paramètres	Valeurs
Taille de la base d'apprentissage	100%
Taille de la population	50 individus
Nombre d'itérations	100 itérations
Probabilité d'application des opérateurs génétiques	• Probabilité de croisement (pc)=0.8 • Probabilité de mutation (pm)=0.1
Les paramètres « λ_i » de la fonction objectif	• λ1=0.2 • λ_2=0.8 • λ_3=0.0

Tableau 06: les paramètres utilisés par l'algorithme génétique.

4. Section 3 : Résultats obtenus

Dans cette section, nous allons montrer les résultats obtenus par l'application de classifieurs génétique et mémétique construits sur les bases de données UCI citées ci-dessus.

4.1. Partie 1 : Résultats obtenus par l'approche génétique

Pour réaliser ces tests, nous allons exécuter le classifieur génétique sur les bases de données UCI précédentes 10 fois successives et nous donnons à chaque fois la précision obtenue ainsi que le nombre de règles.

- **Premier test génétique:** Iris

	Précision	**Nombre de règles**
	66%	4
	60%	5
	84.66%	3
	69.33%	5
	64%	7
	86%	4
	67.33%	6
	88%	6
	86.66%	4
	90%	5
Précision moyenne et nombre moyen de règles	***76.19%***	***4***

Tableau 07: Résultats obtenus par l'approche génétique sur «Iris».

- **Deuxième test génétique: Diabètes**

	Précision	**Nombre de règles**
	90.75%	15
	98.17%	12
	93.48%	10
	91.01%	16
	98.95%	6
	92.57%	23
	95.31%	10
	88.15%	6
	81.38%	3
	91.14%	8
Précision moyenne et nombre moyen de règles	***92.09%***	***14***

Tableau 08: Résultats obtenus par l'approche génétique sur «Diabètes».

- **Troisième test génétique: Mushroom**

	Précision	**Nombre de règles**
	74.91%	14
	87.12%	8
	70.80%	31
	91.61%	13
	77.28%	11
	95.75%	17
	86.70%	24
	68.03%	16
	98.81%	22
	93.88%	9
Précision moyenne et nombre moyen de règles	***84.55%***	***16***

Tableau 09: Résultats obtenus par l'approche génétique sur «Mushroom».

- **Quatrième test génétique: Kdd-train**

	Précision	**Nombre de règles**
	97.36%	16
	83.67%	16
	99.66%	13
	84.67%	5
	94.81%	14
	95.41%	7
	96.14%	24
	99.73%	6
	83.67%	13
	86.37%	16
Précision moyenne et nombre moyen de règles	***92.14%***	***10***

Tableau 10: Résultats obtenus par l'approche génétique sur «Kdd-train».

4.2. Partie 2 : Résultats obtenus par l'approche mémétique

Dans cette partie, nous allons appliquer le classifieur construit à l'aide de l'approche mémétique sur les bases de données précédentes en utilisant les différentes stratégies citées dans ***le chapitre 4, section 3.2.3*** en appliquant le même processus que précédemment.

- **Premier test mémétique:** Iris

	Méthode de recherche locale simple (RLS)			**Méthode de recherche locale génétique (RLG)**		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Précision du classifieur	**100%**	**99.33%**	95.33%	**99.33%**	98.66%	96%
	96%	98.66%	**100%**	98.66%	**100%**	96%
	96.66%	**99.33%**	96%	**100%**	**99.33%**	**99.33%**
	98.66%	**100%**	98.66%	97.33%	**99.33%**	**99.33%**
	100%	**99.33%**	98.66%	**100%**	97.33%	96%
	98.66%	**100%**	**99.33%**	**99.33%**	**99.33%**	93.33%
	100%	96%	**99.33%**	96%	**100%**	**99.33%**
	98.66%	**99.33%**	**100%**	**99.33%**	**100%**	94.66%
	97.33%	**99.33%**	**99.33%**	**99.33%**	96.66%	**99.33%**
	100%	**100%**	96%	**99.33%**	98.66%	**99.33%**
Précision Moyenne	***98.59%***	***99.13%***	***98.26%***	***98.86%***	***98.93%***	***97.26%***

Tableau 11: Résultats obtenus par l'approche mémétique sur «Iris».

Le nombre de règles correspond à chaque valeur de précision du ***tableau 11*** est donné dans le ***tableau 12.***

	Méthode de recherche locale simple (RLS)			**Méthode de recherche locale génétique (RLG)**		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Nombre de règles	8	7	5	7	7	3
	4	7	8	10	9	4
	6	6	3	5	6	3
	4	5	7	5	4	5
	5	5	9	8	5	4
	6	6	7	7	3	6
	5	3	3	5	6	4
	6	8	5	3	5	3
	6	4	7	6	8	4
	3	7	4	4	6	6
Nombre Moyen de règles	***5***	***5***	***5***	***6***	***5***	***4***

Tableau 12: Nombre de règles obtenu par l'approche mémétique sur «Iris».

- **Deuxième test mémétique:** Diabètes

	Méthode de recherche locale simple (RLS)			Méthode de recherche locale génétique (RLG)		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Précision du classifieur	**100%**	**99.47%**	**99.73%**	98.69%	**99.86%**	98.95%
	94.40%	**100%**	**99.86%**	98.08%	**99.73%**	**99.37%**
	97.52%	97.65%	**100%**	94.14%	95.83%	98.56%
	97.26%	95.96%	96.22%	**99.34%**	**100%**	91.79%
	99.60%	96.35%	**100%**	92.83%	97.13%	**100%**
	100%	**99.08%**	92.70%	**99.86%**	97.26%	98.17%
	99.34%	**99.86%**	**99.34%**	98.43%	96.74%	**99.21%**
	98.95%	98.43%	98.82%	98.95%	**99.21%**	96.74%
	92.83%	**99.60%**	**100%**	95.70%	**99.60%**	96.22%
	95.70%	**99.73%**	94.14%	98.04%	97.26%	95.57%
Précision Moyenne	***97.56%***	***98.61%***	***98.08%***	***97.40%***	***98.26%***	***97.46%***

Tableau 13: Résultats obtenus par l'approche mémétique sur «Diabètes».

Le nombre de règles correspond à chaque valeur de précision du ***tableau 13*** est donné dans le ***tableau 14.***

	Méthode de recherche locale simple (RLS)			Méthode de recherche locale génétique (RLG)		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Nombre de règles	28	59	23	35	58	49
	30	50	15	27	18	17
	13	35	19	17	4	24
	36	13	16	20	9	17
	38	7	2	27	22	23
	60	49	40	37	10	58
	31	35	8	30	21	40
	37	39	18	36	9	16
	28	66	14	11	18	11
	22	34	8	36	9	22
Nombre Moyen de règles	***32***	***38***	***16***	***27***	***17***	***27***

Tableau 14: Nombre de règles obtenu par l'approche mémétique sur «Diabètes».

- **Troisième test mémétique: Mushroom**

	Méthode de recherche locale simple (RLS)			**Méthode de recherche locale génétique (RLG)**		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Précision du classifieur	97.20%	**99.85%**	**99.87%**	99.66%	97.64%	**99.51%**
	99.59%	**99.74%**	**99.90%**	99.77%	92.83%	**99.54%**
	91.54%	**100%**	**100%**	**96.47%**	**99.96%**	98.53%
	98.16%	**99.33%**	**100%**	**99.81%**	93.54%	97.84%
	96.77%	**99.37%**	**99.69%**	**99.34%**	**99.18%**	**99.88%**
	99.79%	98.58%	97.45%	**99.58%**	**99.13%**	97.88%
	99.98%	99.81%	**99.08%**	**100%**	**99.87%**	98.75%
	99.54%	**99.64%**	**100%**	**98.99%**	**99.87%**	98.01%
	100%	**99.67%**	**99.67%**	**98.53%**	**99.27%**	**99.71%**
	99.98%	**99.88%**	**100%**	**96.47%**	**99.77%**	**99.88%**
Précision Moyenne	***98.25%***	***99.58%***	***99.56%***	***98.86%***	***98.10%***	***98.91%***

Tableau 15: Résultats obtenus par l'approche mémétique sur «Mushroom».

Le nombre de règles correspond à chaque valeur de précision du ***tableau 15*** est donné dans le ***tableau 16.***

	Méthode de recherche locale simple (RLS)			**Méthode de recherche locale génétique (RLG)**		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Nombre de règles	17	17	19	25	7	24
	22	6	16	13	15	25
	9	12	16	21	26	8
	25	15	34	22	13	19
	12	21	14	10	19	19
	26	**5**	26	15	29	11
	10	10	24	20	25	28
	20	22	20	35	27	20
	31	15	12	33	18	12
	10	24	23	21	15	37
Nombre Moyen de règles	***18***	***14***	***20***	***21***	***19***	***20***

Tableau 16: Nombre de règles obtenu par l'approche mémétique sur «Mushroom».

- **Quatrième test mémétique: Kdd-train**

	Méthode de recherche locale simple (RLS)			**Méthode de recherche locale génétique (RLG)**		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Précision du classifieur	100%	**99.97%**	**99.90%**	**100%**	**100%**	**99.96%**
	96.47%	**99.86%**	96.81%	**99.97%**	**100%**	**99.92%**
	99.99%	**99.91%**	95.53%	98.23%	**99.70%**	**99.97%**
	94.12%	**99.93%**	**99.93%**	**100%**	**99.92%**	**99.15%**
	100%	**99.96%**	**99.59%**	**95.18%**	**99.13%**	**100%**
	99.31%	**99.92%**	90.47%	**99.96%**	**99.98%**	**99.15%**
	99.92%	**99.25%**	95.27%	**99.80%**	98.94%	**99.97%**
	99.82%	**100%**	**99.96%**	**93.18%**	**100%**	**100%**
	99.82%	**99.66%**	**99.71%**	**100%**	**99.95%**	**100%**
	100%	**99.88%**	**99.89%**	**100%**	**100%**	**99.76%**
Précision Moyenne	**98.94%**	**99.83%**	**97.70%**	**98.63%**	**99.76%**	**99.78%**

Tableau 17: Résultats obtenus par l'approche mémétique sur «Kdd-train».

Le nombre de règles correspond à chaque valeur de précision du ***tableau 17*** est donné dans le ***tableau 18.***

	Méthode de recherche locale simple (RLS)			**Méthode de recherche locale génétique (RLG)**		
	RLS(C)	**RLS(M)**	**RLS(C+M)**	**RLG(C)**	**RLG(M)**	**RLG(C+M)**
Nombre de règles	17	5	7	9	27	4
	22	7	3	4	20	8
	9	5	2	5	3	3
	25	2	8	14	9	4
	12	8	4	5	5	8
	26	6	3	8	8	4
	10	4	4	5	9	6
	20	12	9	5	8	7
	31	10	4	10	6	3
	10	11	4	6	8	5
Nombre Moyen de règles	***18***	***7***	***4***	***7***	***10***	***5***

Tableau 18: Nombre de règles obtenu par l'approche mémétique sur «Kdd-train».

4.3. Partie 3 : Etude comparative

Dans cette partie, nous procédons à une étude comparative de la précision et du nombre de règles obtenus par l'approche hybride, l'approche génétique et les autres algorithmes de classification.

4.3.1. Etude comparative de la précision entre l'approche mémétique, l'approche génétique et les autres algorithmes de classification:

Nous allons faire une étude comparative entre les résultats obtenus par l'application de l'approche mémétique, l'approche génétique et quelques algorithmes de classification.

	PRECISION			
	Iris	**Diabètes**	**Mushroom**	**Kdd-train**
RLS(C)	98.95%	97.56%	98.25%	98.94%
RLS(M)	**99.13%**	**98.61%**	**99.58%**	**99.83%**
RLS(C+M)	98.26%	98.08%	99.56%	97.70%
RLG(C)	98.86%	97.40%	98.86%	98.63%
RLG(M)	98.93%	98.26%	98.10%	99.76%
RLG(C+M)	97.26%	97.40%	98.91%	99.78%
AG	76.19%	92.09%	84.55%	92.14%
OneR	94%	73.04%	98.52%	98.67%
Table de Décision	92.66%	73.30%	**100%**	99.81%
J48	96%	73.82%	**100%**	99.88%

Tableau 19: Comparaison de qualité entre l'approche mémétique, l'approche génétique et les autres algorithmes de classification.

Remarques :

1. Nous remarquons que l'approche mémétique a donné une très bonne précision par rapport à l'approche génétique et cela est dû à l'intégration des méthodes de recherche locale.

2. Nous avons remarqué aussi que le temps d'exécution de l'approche génétique hybride est inférieur à celui de l'approche génétique pur et ça grâce aux méthodes de recherche locale qui accélèrent la convergence de l'algorithme génétique.
3. Nous remarquons que l'approche hybride donne de bons résultats en termes de précision par rapport aux algorithmes de classification, sauf dans le cas de la base de données *mushroom* où l'algorithme J48 et table de décision donne une précision égale à 100%, cependant nous avons obtenus une précision égale à 99.58%.

4.3.2. Etude comparative de nombre de règles obtenu entre l'approche mémétique, l'approche génétique et les autres algorithmes de classification:

	NOMBRE DE REGLES			
	Iris	**Diabètes**	**Mushroom**	**Kdd-train**
RLS(C)	5	32	18	18
RLS(M)	5	38	14	7
RLS(C+M)	5	16	20	4
RLG(C)	6	27	21	7
RLG(M)	5	17	19	10
RLG(C+M)	4	27	20	5
AG	4	14	16	10
OneR	3	8	9	8
Table de Décision	6	32	176	117
J48	5	39	30	26

Tableau 20: Comparaison de nombre de règles obtenues par l'approche mémétique, l'approche génétique et les autres algorithmes de classification.

Remarque :

Nous remarquons que le nombre de règles obtenus par l'approche hybride est généralement inférieur aux autres nombre obtenus par l'approche génétique et les autres algorithmes de classification ce qui permet la construction des classifieurs compréhensibles et par conséquence la classification d'un nouvel élément sera rapide et facile.

5. Conclusion:

Nous avons présenté dans ce chapitre les résultats obtenus par l'application des différentes stratégies de recherche locale sur le problème de classification.

Nous avons obtenus des classifieurs de qualité avec haute précision et avec un petit nombre de règle. Ainsi, l'étude comparative avec les classifieurs construits à l'aide des

algorithmes génétiques et avec d'autres algorithmes ont montré l'efficacité des algorithmes mémétiques et l'intérêt d'intégration des méthodes de recherche locale qui explorent bien l'espace de recherche.

CONCLUSION GENERALE ET PERSPECTIVES

Dans ce travail, nous avons abordé le problème de l'extraction de connaissances à partir de données volumineuses et notamment le problème d'extraction de règles de classification. Ce problème étant combinatoire difficile, classé NP-Complet. Il n'existe pas d'algorithmes exacts pour le résoudre. Nous avons présenté quelques approches de représentation et de modélisation de ce problème ainsi que sa formulation mathématique. Nous avons présenté quelques heuristiques trouvées dans la littérature qui ont été utilisées dans sa résolution.

Par ailleurs, nous avons présenté quelques approches qui nous ont semblé prometteuses dans une utilisation future pour résoudre ce problème. Pour cela, nous avons étudié les métaheuristiques connues telles que la recherche taboue, le recuit simulé, les algorithmes génétiques,… Dans un but d'hybridation de plusieurs méthodes, nous avons passé en revue des méthodes de recherche locale. En effet, l'hybridation de ces dernières avec des algorithmes évolutionnaires constitue actuellement une grande tendance pour résoudre des problèmes complexes tels que la classification. Cette tendance constitue ce qu'on appelle couramment les algorithmes mémétiques. Pour cela, nous avons étudié quelques approches mémétiques et des stratégies d'application de ces approches ainsi que toutes les questions auxquelles on est confronté lors de la conception d'une approche mémétique pour la résolution d'un problème complexe.

Donc, nous avons 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 des méthodes de recherche locale en appliquant des stratégies de remplacement dans le but de voir les performances de ces méthodes et leurs effets sur la qualité de classifieur obtenu.

Dans le but de montrer l'efficacité de notre approche, nous avons mené des tests qui consistent à construire des classifieurs sur quelques bases de données de l'UCI. D'après les résultats obtenus, l'étude comparative a montré que les classifieurs obtenus sont de qualité élevée dans la mesure où ils ont une très bonne précision comparée par rapport à d'autres algorithmes de classification (J48, OneR, les tables de décision). Ceci est principalement dû aux performances des méthodes de recherche locale qui améliorent l'exploration de l'espace de recherche. De plus, la vitesse de convergence a été améliorée par rapport à l'algorithme génétique pur.

Nos recherches futures visent essentiellement à réduire le temps d’exécution en utilisant le parallélisme.

REFERENCES BIBLIOGRAPHIQUES

[AAR 1997] E. Aarts, J.K. Lenstra, *Local Search in Combinatorial Optimization. Wiley-Interscience Series in Discrete Mathematics and Optimization,* John Wiley & Sons, Chichester, England, UK, 1997.

[ADR 1996] P. Adriaans and D. Zantinge. *Data Mining*. Addison-Wesley, 1996.

[AGR 1994] R.Agrawal et A. Srikant. *Fast algorithms for mining association rules*. Proc, pp 787-499, VLDB 1994.

[BAC 2003] Bacardit, J., & Garrell, J. M. *Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system*. In Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003 pp. 1818–1831. LNCS 2724, Springer, 2003.

[BAC 2004] Bacardit, J. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain, 2004.

[BAC 2006] Bacardit, J. and Krasnogor, N. *Smart crossover operator with multiple parents for a pittsburgh learning classifier system*. In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pages 1441–1448, New York, NY, USA. ACM Press, 2006.

[BÄC 1997] TH. BÄCK, D.B. FOGEL, AND Z. MICHALEWICZ, EDITORS. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.

[BEN 2007] S. Benkhider. *A new Generationless parallel evolutionary algorithm for combinatorial optimization*, 2007.

[BEN 2008] S. Benkhider. *A memetic approach for rule extraction problem*. Conférence sur les métaheuristiques, META 08, LIFL, USTL, 2008.

[BOJ 2000] C.Bojarczuk, H S. lopes, Alex Freitas: *Genetic Programming for Knowledge Discovery in Chast-Pain diagnosis*; IEEE; 2000.

[BRE 1984] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and regression trees*. Technical report, Wadsworth International, Monterey, CA, 1984.

[BUR 2003] E. K. Burke, G. Kendall and E. Soubeiga, *A Tabu Search hyperheuristic for TimeTabling and rostering*, Journal of Heuristics Vol. 9, No. 6, 2003.

[COL 2001] M. COLARD, D. FRANCISCI, *Evolutionary data mining: an overview of genetic based algorithms.* 8th IEEE International Conference on Emerging Technologies and Factory; October 2001.

[COU 2002] Yves Coueque- Julien Ohler- Sabrina Tollari, *Algorithmes génétiques pour résoudre le problème du commis voyageurs,* Avril 2002.

[COW 2000] P. Cowling, G. Kendall and E. Soubeiga, *A Hyperheuristic Approach to Scheduling a Sales Summit*, PATAT 2000, Springer Lecture Notes in Computer Science, No. 2079, pp. 176-190, Konstanz, Germany, August 2000.

[DEJ 1991] DeJong, K. A. and Spears, W. M, *Learning concept classification rules using genetic algorithms. In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 651–656. Morgan Kaufmann, 1991.

[DOR 1994] M. Dorigo and G. Di Caro. *The ant colony optimization meta-heuristic*. In D. Corne, M. Dorigo,and F. Glover, editors, New Ideas in Optimization.McGraw Hill, London, UK, pages 11-32,1999.

[EIB 2007] Eiben, A.E, SMITH. *Introduction to Evolutionnary Computing*, Springer 2007.

[FRA 2003] Dominique FRANCISCI; Laurent BRISSON, MARTINE COLLARD, *Extraction des règles selon des critères multiples l'art du compromis ;* Rapport de recherche interne, Mai 2003.

[FRI 1996] B. Friesleben, P. Merz. *A Genetic local search algorithm for solving the symetric and assymetric traveling salesman problem.* Proceeding of the {IEEE} Conference on Evolutionary Computation, IEEE Press, Pisctaway, NJ, USA 1996.

[FRE 2000] A.A. Freitas. *Understanding the crucial differences between classification and discovery of association rules,* a position paper. ACM SIGKDD Explorations (ACM 2000), pp 65-69, 2000.

[GLO 1989] F. Glover, M. Laguna, *Tabu Search*, ORSA Journal on Computing, 190-206, 1989.

[GOL 1989] D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning. Reading*, MA AddisonWesley, 1989.

[HAR 2003] Hart, W., De Laurentis, J., and Fergueson, L. *On the convergence of an implicitly self adaptive evolutionary algorithm on one-demensional unimodal problems*, IEEE Trans Evolutionary Computation, 2003.

[HOL 1962] John Holland, *Outline for a logical theory of adaptive systems.* Journal of the Association of Computing Machinery, 3, 1962.

[HOL 1993] R. Holte, *Very Simple Classification Rules Perform Well on Most Commonly used data sets,Machine Learning*, Kluwer Academic Publisher, Boston, Vol. 11, pp.63-91, 1993.

[HON 1996] HONGJUN Lu, Rudy Setiono, Effective *Data Mining Using Neural Networks*, IEEE Transactiuon on Knowledge and Data Engennering, VOL o, No 6, December 1996.

[HOR 2007] M.F Hornick, E. Marcadé and S. Venkayala, *Java Data mining: strategy, standard and practice*, Morgan Kaufmann publishers, 2007.

[HUA 1995] HUAN, Sun Teck Tan; X2R: *A Fast Rule Genarator*. Proceeding International Conference on Systems, Man and Cybernetics. University de Singapore; IEEE, 1995.

[IGL 2006] B. de la Iglesia, M.S. Philpot, A.J. Bagnal, V.J Rayward-Smith, *Data mining rules using Multi_Objective Evolutionary Algorithms,* Journal of Operational Research, 2006.
[INT 01] www.uci.edu/~mlearn/MLRepository.html.

[INT 02] Optimisation combinatoire - Définition - Encyclopédie scientifique en ligne.htm.

[JES 2003] Jesus S.Aguilar-Ruiz, J.C. Riquelme and M.Toro. *Evolutionary Learning of Hierarchical Decision Rules,* IEEE Transaction on Systems, Man, and Cybernetics, Part B: Cybernetics, ISSN:1083-4419, vol. 33, no. 2, pp.324-331, 2003.

[JOU 2003] Laetitia JOURDAN, *Méta heuristiques pour l'extraction de connaissances: application de la génomique,* Thèse de doctorat de l'U.S.T.L; Novembre 2003.

[KAL 2001] L. Kallel, B. Naudts and C. Reeves, *Properties of fitness functions and search landscapes, in Theoretical Aspects of evolutionary Computing*, L. Kallel, B. Naudts and A. Rogers, eds, Springer, Berlin, Heidelberg, New York, 2001.

[KEN 2002] G. Kendall, P. Cowling, E. Soubeiga, *Choice Function and Random HyperHeuristics* , Fourth Asia-Pacific Conference on Simulated Evolution and Learning, SEAL, Singapore, pp.667-71, November 2002.

[KOR 1987] A. CORANA, M. MARCHESI, C. MARTINI, S. RIDELLA, *Minimizing multimodal functions of continuous variables with the simulated annealing* , ACM transactions on mathematical software, Vol 13, n° 3, September 1987.

[KOH 1995] R. Kohavy, *The power of decision tables,* Machine Learning 1995: 8th European Conference on Machine Learning, Greece, Springr Verlag 1995.

[KRA 2000] N. Krasnoger and J. Smith, A memetic algorithm with self-adaptive local search: {TSP} as a case study, in: *Processingd of the Genetic and Evolutionary Computation Conference*, D. Whitley, D. Goldberg, E. Cantu-Paz, L.Spector, I.Parmee and H.G. Beyer, eds, Morgan Kaufmann, san Fransisco, pp. 432-439, 2000.

[KRA 2002] N. Krasnoger, B. Blackburne, J.D. Hirst and E.K. Burke N., *Multimeme Algotithms for the Structure Prediction and Structure Comparaison of Proteins,* Parallel Problem Solving From Nature, Lecture Notes in Computer Science, 2002.

[KRA 2004] N. Krasnoger, *Self-generating metaheuristics in bioinformatics: The protein structure comparison case, in: Genitic Programming and Evolvable Machines*, Kluwer academic Publishers, vol 5, pp. 181-201, 2004.

[LAE 2003] LAETITIA JOURDAN, *« Métaheuristiques pour l'extraction de connaissances : Applications au génome.* Thèse de Doctorat, USTL, Novembre 2003.

[LEF 2001] *R. Lefébure, G. Venturini, Data Mining: Gestion de la relation client et Personalisation de sites web, Editions EYROLLES, 2001.*

[LIN 1973] S. Lin, B.W. Kernighan, *An effective heuristic algorithm for the Traveling-Salesman Problem*. Operations Research 21, pp. 498_516. 1973.

[MER 1999] P. Merz and B. Freisleben, *fitness landscapes and memetic algorithm design, in: New Ideas in Optimization*, D. Corne, M. Dorigo and F.Glover, eds, McGraw Hill, London, 1999.

[MER 2000] P. Merz, *Memetic Algorithms for Combinatorial Optimization Problems : Fitness Landscapes and Efective Search Strategies*, PhD thesis, Departement of Electrical Engennering and Computer Science, University of Siegen, Germany, 2000.

[MIC 2001] Z. Michalewicz, David B. Fogel, *How to Solve It : Modern Heuristics*; second edition, 2001.

[MIT 1997] T. Mitchell, *Machine Learning*. New York: McGraw Hill, 1997.

[MOS 1989] P. Moscato, *On evolution search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Calteh Concurrent Computation Program, C3P Report 826, 1989.

[MOS 2001] P. Moscato, *Problemas de Otimizaço, Aproximabilidade e Comutaçao Evolutiva :* Da Pratica a Teoria, PhD thesis, Universidade estadual de Campinas, C3P Report 826, 2001.

[ONG 2004] Y.S. Ong and A. J. Keane, *Meta-Lamarckian in Memetic Algorithm*, vol. 8, No. 2, pp.99-110, IEEE Transactions On evolutionary Computation, April 2004.

[PAP 1982] C. H. Papadimitriou & K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Englewoods Cliffs, Prentice Hall, 1982.

[PAR 2002] R.S Parpinelli, H.S Lopes and A.A Freitas ”*Data mining with an Ant colony optimization algorithm*”, IEEE trans. on Evolutionary Computation, vol (6), pp 321-332, 2002.

[QUI 1993] J. R. Quinlan. C4.5: *Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[RIS 1978] Rissanen, J. *Modeling by shortest data description*. Automatica, vol. 14, 465–471, 1978.

[RUD 1996] Rudolph, G, *Convergence of evolutionary algorithm in general search spaces*, in Proceedings of the International Congress of Evolutionary Computation, pp 50-54, 1996.

[SMI 1991] P.Smith et H.M. Goodman, *Rule induction using information theory. Knowledge Discovery in Databases.* G. Piatesky-Shapiro et W.J. Frawley, editors, MIT Press, 1991.

[SMI 2002] J. E. Smith, *Co-evolution of memetic algorithms: Initial investigations*. Parallel problem solving from Nature-PPSN VII, Guervos et al. Eds., LNCS no. 2439, Springer Berlin, pp. 537-546, 2002.

[SMI 2003] J. E. Smith, *Co-evolving Memetic Algorithms : A learning approach to robust scalable optimization*, IEEE Congress on Evolutionary Computation, IEEE Press, pp. 498-505, 2003.

[SYS 1991] G. Syswerda, *A Study of reproduction in Generational and Steady-State Genetic Algorithms*. Proceedings of FOGA, pages 94-101, Morgan Kaufmann, 1991.

[TUF 2005] S. Tuffery , *Data mining et Statistique Décisionnelle : l'intelligence dans les bases de données,* Editions TECHNIPE, Paris 2005.

[ULU 1999] E. L. Ulungu, J. Teghem, P. H. Fortemps et D. Tuyttens, « *MOSA method: A tool for solving multi-objective combinatorial optimization problem* », journal of multi-criteria decision analysis, Anal 8: 221,236, 1999.

[VAN 2007] Michel. Van Caneghem, *Algorithmes pour la Recherche Locale Cours 1* – Le voyageur de commerce, Algorithme "branch and bound", Algorithme Glouton, Méthode de recherche locale ,Novembre 2007.

[VEN 1993] G. Venturini, SIA: *A supervised inductive algorithm with genetic search for learning attributes based concepts*, in Proc. Eur. Conf. Machine Learning, pp. 281–296, 1993.

[WAI 2003] Wai-Ho Au, K. C.C Chan and Xin Yao,"*A novel evolutionary data mining algorithm with application to churn prediction*", in IEEE trans. on Evolutionary Computation, pp 532-545, December 2003.

[WIT 2005] I. H. Witten, E. Frank, *Data Mining: Practical Machine Learning tools and Techniques with JAVA Implementations,* Morgan Kauffman Publishing, 2005.