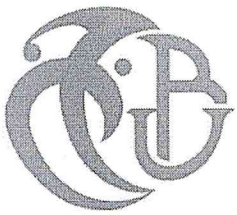
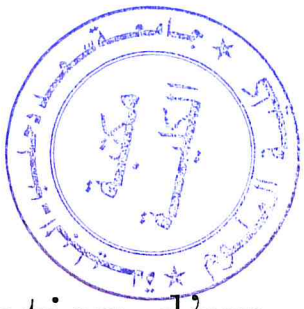
UNIVERSITÉ SAAD DAHLAB – BLIDA

FACULTÉ DES SCIENCES – DÉPARTEMENT D'INFORMATIQUES





Conception et implémentation d'un algorithme de réduction de données d'apprentissage pour la classification d'alertes à base de Knn

Algorithme Mémétique

Présenté par:

BOUZOURINE Hassan-El Fadhle

ZAMMOUCHI Ayoub

Promoteur : M.BABA-ALI Riadh

Encadreur : Mme MILOUD-AOUIDAT Amal

Juin 2013

Remerciments

Nous louons Allah, notre guide et notre force et la raison de notre existence.

Nous tenons à remercier M. BABA-ALI Riadh pour son accueil chaleureux et sa bonne humeur qui nous a permis de se sentir à l'aise et d'être motivés pour travailler,

Un grand merci à Mme. MILOUD-AUOIDAT Amal pour son aide précieuse et sa grande patiente et de nous avoir fait partager ses connaissances et d'être présente pour répondre à nos questions

Un grand merci à nos parents qui sont toujours là pour nous, que dieu nous les gardes en bonne santé.

Nous tenons à remercier nos amies pour leurs soutiens et leurs encouragements.

Résumé

Un système de détection d'intrusions IDS permet de capter et analyser les diffèrent parquets qui circulent sur un réseau, chaque paquet peut appartenir à une des deux classes Normal ou Attaque. Pour classer ses paquets, le système de détection d'intrusion a besoin d'une base d'apprentissage, le problème majeur de cette base est sa taille qui est très grande, le nombre important des données redondantes ainsi que la présence des instances bruits. Avec une base d'apprentissage pareille, le système de détection d'intrusions aura tendance à mal classer les instances captées. Pour éviter ce genre de problèmes il faudrait qu'une meilleure base d'apprentissage soit fournie à l'IDS. La création d'une base d'apprentissage plus performante à partir de la base d'apprentissage original est un problème d'optimisation, plusieurs algorithmes peuvent proposer une solution optimal à ce genre de problèmes, parmi eux on trouve Les algorithmes mémétiques qui sont une hybridation entre les algorithmes génétiques et les méthodes de recherches locale. Cette hybridation permet de créer un algorithme qui profite de la diversité qu'offre l'algorithme génétique et de la recherché de la solution optimal dans un espace étroit que propose les méthodes de recherché locale. L'algorithme mémétique s'appliquera sur la base de données initial, le résultat doit être une base d'apprentissage réduite mais possède les même performances que celle de la base initial. Cette base d'apprentissage permettra à l'IDS de mieux classer les instances capter avec un temps de réponse petit.

Table des matières

Table des matières iv
Chapitre 1 : Contexte général 2
1 Situation actuelle 2
2 Énoncé du problème 2
3 Objective général 3
4 Organisation du mémoire 4
Chapitre 2 : État de l'art 6
1 Introduction 6
2 Les Systèmes de détection d'intrusion 6
2.1 Définition 6
2.2 L'intrusion 7
2.3 Détection d'intrusion 7
2.4 Architecture d'un IDS 7
2.4.1 Le capteur 8
2.4.2 L'analyseur 8
2.4.3 Le manager 9
2.5 Les types d'IDS 10
2.5.1 Système de détection d'intrusion basé réseau (NIDS) 10
2.5.2 Système de détection d'intrusion basé hôte (HIDS) 11
2.5.3 Comparaison entre NIDS et HIDS 12
2.5.4 Les Système de détection Hybride 13
3 Apprentissage automatique 13
3.1 L'apprentissage supervisé 14
3.1.1 Classification 14
3.1.2 Régression 14
3.1.3 Séries temporelles 14
3.2 L'apprentissage non-supervisé 14

3.3 Méthode du plus proche voisin 15
3.3.1 Présentation de l'algorithme 15
3.3.2 Algorithme de base 15
3.3.3 Mesures de similarité 17
3.3.4 Inconvénients de la méthode 18
4 Approches de détection d'intrusions basées sur l'algorithme KNN 20
5 Les Métaheuristiques 22
5.1 Classification des métaheuristiques 22
5.2 Métaheuristiques à solution unique 25
5.2.1 La méthode de descente 25
5.2.2 La méthode de recuit simulé 26
5.2.3 La méthode Tabou 26
5.3 Métaheuristiques à population de solutions 27
5.3.1 Les algorithmes génétiques 28
5.3.2 Principe de l'algorithme génétique 29
5.3.3 Le codage 31
5.3.4 Fonction de fitness 31
5.3.5 La sélection 32
5.3.6 Le croisement 34
5.3.7 La mutation 35
5.3.8 Le remplacement 35
5.4 Algorithmes à solution Hybride 36
5.4.1 Algorithmes Mémétiques 37
5.4.2 Principe des algorithmes mémétiques 37
5.4.3 Algorithme Mémétique Basique 39
Chapitre 3 : Algorithme mémétique 41
1 Introduction 41
2 Algorithme mémétique pour la réduction de la taille de données d'apprentissage 42
2.1 Codage des individus 44
2.2 La population initial 45
2.3 La fonction de fitness 45
2.4 Paramétrage de l'algorithme 50
2.4.1 Opérateur de sélection 50

2.4.2 Opérateur de croisement 51
2.4.3 Opérateur de remplacement 52
2.4.4 Nombre de générations 54
2.4.5 Taille de la population initial 55
2.4.6 La recherche locale 56
3 Conclusion 60
Chapitre 4 : Implémentation et résultats 62
1 Introduction 62
2 Environment de travail 62
3 Présentation de l'application 64
3.1 Package Génétique 65
3.2 Package Threads 65
3.3 Package Selection 66
3.4 Package Croisement 67
3.5 Package Mutation 67
3.6 Package Remplacement : 68
3.7 Package LocalSearch 68
3.8 Interface utilisateur 69
4 Description du KDD'99 70
4.1 Présentation et structure du KDD'99 70
4.2 Catégories des attaques 71
4.3 Classification des attributs 71
4.4 Problèmes liés au KDD CUP 99 74
5 Tests et résultats 75
5.1 Test : Réduction de l'ensemble « Train » 75
5.2 Test : Classification de « TestCorrected » 76
5.3 Test : Variation des classificateurs 77
5.4 Test : Classification en temps réel 80
6 Conclusion 81
Conclusion général 83
Bibliographie 84

Chapitre 1

Contexte général

Chapitre 1 : Contexte général

1 Situation actuelle

Avec l'apparition du concept de connectivité et du partage d'informations, Les réseaux informatiques se sont développés très rapidement Le réseau le plus populaire aujourd'hui c'est l'internet. À son lancement il était utilisé comme un réseau privé connectant les gouvernements militaires et les chercheurs académiques. C'est pourquoi il n'y avait pas besoin d'une politique de sécurité.

Depuis qu'il est devenu ouvert à tout le monde, sa taille à augmenter et il permettait beaucoup plus de service comme, le partage de ressources, la communication entre personnes et la communication entre processus. Ceci à générer une nouvelle tendance, celle du piratage des informations. Rapidement différente techniques de piratage en vue le jour comme les virus, les vers, les chevaux de Troie, le phishing ainsi que les spam.

Le motive des différentes attaques est non seulement le goût de la réussite de l'exploit, mais aussi le gain d'argent, les révélations politiques, la réalisation de certains buts militaires et la curiosité. La cause d'une faille dans un système peut être due à un manque de budget, de temps d'installation, de personnes qualifiées, de politique de sécurité Néanmoins, de nos jours, le coût d'une attaque et de sa réparation peuvent être très élevés. Il est très fréquent que l'on apprenne que telle entreprise ou tel institut a essuyé de lourdes pertes financières en raison d'une déficience de la sécurité de son système d'information. C'est pourquoi les entreprises et l'ensemble des organisations s'intéressent de près à la sécurité informatique.

Parmi les techniques utilisées pour assurer la sécurité d'un système informatique on trouve les détecteurs d'intrusion (IDS), apparus dans les années 1980 comme résultat aux travaux de recherches de James Anderson qui a souhaité améliorer les équipements d'audit et les capacités de surveillance des systèmes de d´détection d'intrusions, Leurs buts c'est la protection de documents confidentiels, trouver de nouvelles attaques, satisfaire des contraintes légales tel que la confidentialité. Ils fonctionnent comme un système d'alarme, une fois qu'une intrusion est détectée, l'alarme est déclenchée pour avertir le gestionnaire du réseau. Des réponses aux attaques peuvent être effectuées par la suite pour réduire l'apparition de pénétrations en améliorant la prévention et la détection du système, ainsi que de réparer les dommages générés par l'intrusion.

2 Énoncé du problème

Comme cité précédemment les IDS ont pour rôle de détecter tout type d'intrusion possible dans le réseau, pour y parvenir ses systèmes ont besoin d'un mécanisme d'apprentissage capable de les faire détecter les différentes attaques dites « connu » ainsi de découvrir de nouvelles modèles d'attaques.

Ce mécanisme s'appuie sur une base d'apprentissage dans laquelle un ensemble de modèles d'attaques sera stocké. L'IDS se basera ensuite sur cette base pour classer ce qui est considérer comme un comportement normal ou anormal. Pour déployer ce mécanisme il faut prendre en considération trois facteurs, La taille de la base d'apprentissage, le niveau de précision de détection et le temps de réponse.

Avec une base d'apprentissage contenant un grand nombre de modèles d'attaques, certes le niveau de précision de détection sera élevé mais la taille de la base d'apprentissage sera énorme du coup le temps de détection sera affecté et le taux du faux-positif [1]peut augmenter. On minimisant le nombre de modèles d'attaques on gagne en taille de la base et en temps de réponse mais le niveau de précision se dégradera ainsi le taux du faux-négatif [2] sera très important.

3 Objective général

Une solution judicieuse consiste à utiliser une base d'apprentissage de taille optimal qui garanti un niveau de précision qui permet de détecter la majorité des attaques et découvrir d'autres nouvelles, ceci en se basant sur les modèles d'attaques déjà présents dans la base d'apprentissage.

Ceci est considéré comme un problème d'optimisation qui n'a pas de solution final mais plutôt une solution optimal.

Parmi les algorithmes qui sont utilisés pour résoudre ce genre de problème on trouve les algorithmes génétiques, les méthodes de recherche locale, ainsi que les algorithmes mémétiques qui sont une hybridation entre les deux méthodes.

Pour résoudre ce problème nous allons démarrer à partir d'une base d'apprentissage initial, en appliquant un algorithme mémétiques nous allons essayer d'extraire une seconde base de données de taille inférieur à celle de l'initial, cette nouvelle base doit respecter les contraintes suivantes :

- Le niveau de précision doit être supérieur ou égal à celui de la base initial
- Diminuer le taux de faux positif généré par l'IDS
- Assurer un meilleur temps d'exécution

L'algorithme mémétique se basera sur les méthodes de classement à base de voisinage. L'utilisation de l'Algorithme des K plus proches voisins pondérés (KNN) va nous permettre de classifier les modèles d'attaques selon leurs plus proches voisins, d'une autre manière, soit x le nouveau modèle détecté, pour classifier x il suffit de faire voter les plus proches voisins de x. La classe de x (comportement normal ou anormal) est alors déterminée en fonction de la classe majoritaire parmi les k plus proches voisins de x.

---

[1] Une alerte provenant d'un IDS mais qui ne correspond pas à une attaque réelle

[2] Une intrusion réelle qui n'a pas été détectée par l'IDS

En utilisant un tel algorithme de classification nous allons essayer de fournir à l'IDS **le maximum des cas d'intrusion possible avec le minimum de données**, ce qui va nous permettre d'optimiser la taille de la base d'apprentissage et du coup le temps de réponse et ainsi de garder un niveau optimal de précision.

4 Organisation du mémoire

Le mémoire se réparti en trois grandes partie. On commence tout d'abord par l'état de l'art, on y détaillera ce que c'est un système de détection d'intrusion tout d'abord puis nous verrons les différents types d'apprentissage automatique, nous entamerons les différentes approches de détection d'intrusion basées sur l'algorithme KNN et on parlera ensuite des métaheuristiques dans lesquels deux grands familles sont présentées. Les métaheuristiques à population de solution et les métaheuristiques à solution unique.

Nous présenterons par la suite l'approche que nous avons essayé d'adopter pour résoudre le problème de réduction de la taille de la base de données d'apprentissage qu'utilisent les systèmes de détection d'intrusion en utilisant les algorithmes mémétiques, ou chaque étape de notre algorithme sera détaillée.

À la fin on finira par des séries de test pour vérifier et valider l'algorithme ainsi qu'une conclusion général.

Chapitre 2

État de l'art

Chapitre 2 : État de l'art

1 Introduction

Dans ce chapitre nous allons présenter les systèmes de détection d'intrusion en détaillons leurs architecture est leurs fonctionnement, nous verrons après l'apprentissage automatique et la méthode du plus proche voisin ainsi que les différentes approches basées sur cette méthode. On détaillera à la fin les différentes familles des métaheuristiques et comment les algorithmes mémétiques sont nés. On terminera par une conclusion du chapitre.

2 Les Systèmes de détection d'intrusion

Les systèmes d'information sont aujourd'hui de plus en plus ouverts sur Internet. Cette ouverture, a priori bénéfique, pose néanmoins un problème majeur : il en découle un nombre croissant d'attaques.

La mise en place d'une politique de sécurité autour de ces systèmes est donc primordiale.

Outre la mise en place de pare-feu et de systèmes d'authentification de plus en plus sécurisés, il est nécessaire, pour compléter cette politique de sécurité, d'avoir des outils de surveillance pour auditer le système d'information et détecter d'éventuelles intrusions.

Afin de détecter les attaques que peut subir un système, il est nécessaire d'avoir un logiciel spécialisé dont le rôle serait de surveiller les données qui transitent sur ce système, et qui serait capable de réagir si des données semblent suspectes.

2.1 Définition

Plus communément appelé IDS (Intrusion Detection Systems), les systèmes de détection d'intrusions se compose de composants logiciels et matériels dont la fonction principal est d'analyser et de détecter tout tentative d'effraction ou tout comportement anormal dans un système informatique [1], ceci en examinant les données fournis par les audits de sécurité fournis par le système d'exploitation ou les outils de contrôle du trafic réseau.

Son rôle consiste à compléter la sécurité du pare-feu et empêcher tout accès malveillant au côté saint du système informatique. Par exemple, dans une voiture c'est la serrure qui protège la voiture contre le vol. Mais c'est le système d'alarme qui détecte que la serrure a été forcée et déclenche l'alarme.

2.2 L'intrusion

Nous appellerons intrusion toute utilisation d'un système informatique à des fins autres que celles prévues. L'intrus est généralement une personne étrangère au système informatique qui a réussi à en prendre le contrôle [2].

2.3 Détection d'intrusion

En sécurité informatique, la détection d'intrusion est l'acte de détecter les actions qui essaient de compromettre la confidentialité, l'intégrité ou la disponibilité d'une ressource. La détection d'intrusion peut être effectuée manuellement ou automatiquement. Dans le processus de détection d'intrusion manuelle, un analyste humain procède à l'examen de fichiers de logs à la recherche de tout signe suspect pouvant indiquer une intrusion. Par contre, dans une détection d'intrusion automatisée on fait appel à des systèmes de détection d'intrusion (IDS) [2].

Lorsqu'une intrusion est découverte par un IDS, les actions typiques qu'il peut entreprendre sont d'enregistrer l'information pertinente dans un fichier ou une base de données, de générer une alerte par e-mail ou un message sur un pager[3] ou un téléphone mobile. Déterminer quelle est réellement l'intrusion détectée et entreprendre certaines actions pour y mettre fin ou l'empêcher de se reproduire, ne font généralement pas partie du domaine de la détection d'intrusion. Cependant, quelques formes de réactions automatiques peuvent être implémentées par l'interaction de l'IDS et des systèmes de contrôle d'accès tels que les pares-feux.

2.4 Architecture d'un IDS

Un IDS se compose principalement de trois éléments de base: Le capteur qui a pour rôle d'intercepter les activités et les flux de données circulant sur un réseau ou une machine ; L'analyseur qui traite ensuite ces données collecté par le capteur, il joue le rôle le plus important dans la détection d'intrusion ; et le manager qui est chargé de la réaction à adopter contre une intrusion.

[3] Appareille permettent de capter des messages radios

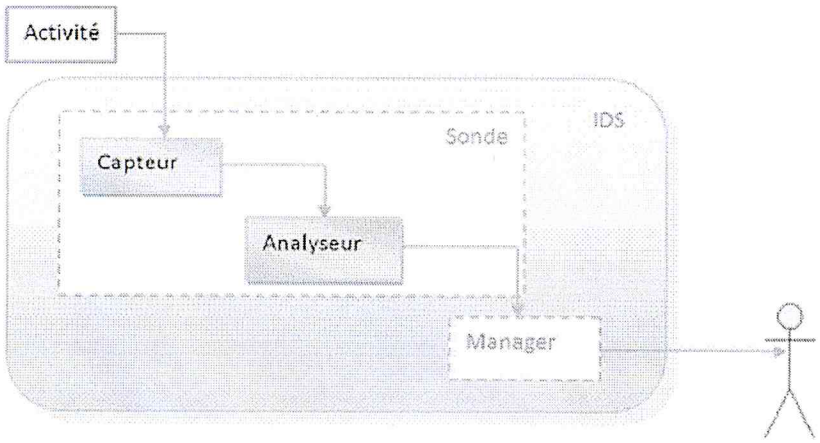


Figure 1 : Architecture de base d'un system de détection d'intrusion

2.4.1 Le capteur

Le capteur observe l'activité du système à travers une source de données, il est chargé ensuite de fournir à l'analyseur l'état du système sous forme de séquences d'événements. En générale le capteur s'occupe de filtrer les données en éliminant celles qui sont considérées comme non pertinentes pour limiter la quantité de données à traiter par l'analyseur, il met aussi les informations collecté sous forme d'événements afin de les présenter à l'analyseur. On peut distinguer trois types de capteurs :

1. **Les capteurs systèmes :** se basent sur les données produites par les systèmes d'exploitation des machines, les journaux d'audit systèmes, les appels système invoqués par les applications. Les IDS utilisant ce type de capteurs sont les HIDS (Host-based IDS).

2. **Les capteurs réseau :** Les informations sont collectées en écoutant le trafic réseau (Les paquets transmis entre les machines du réseau). Les IDS utilisant ce type de capteur sont les NIDS (Network-based IDS)

3. **Les capteurs applicatifs :** Les équipements tels un serveur web ou un serveur de base de données ont certaines fonctionnalités qui leurs permettent de fournir des données à propos de leurs état, les capteurs applicatifs sont désignés pour collecter ces données.

2.4.2 L'analyseur

Comme son nom l'indique, il est chargé d'analyser le flux d'événements fourni par le capteur, son objectif est de déterminer si ce flux contient des éléments caractéristiques d'une activité malveillante.

Deux grandes approches de détection d'intrusions ont été proposées, l'approche par scénario et l'approche comportementale :

2.4.2.1 *L'approche par scénario (Basée signatures)*

Dans l'approche par scénario (basée-signatures), le système de détection d'intrusion possède une base de signatures qui modélise les différentes attaques connues. L'analyse consiste à rechercher l'occurrence d'un motif caractéristique d'une attaque dans le flux d'événements. Cette analyse est efficace si le type d'attaque est déjà connu mais inefficace dans le cas contraire.

Plusieurs techniques ont été proposées qui reposent en général sur des mécanismes de reconnaissance de motif ou *pattern matching*.

Le pattern matching est une méthode simple à mettre en œuvre. Cependant, la difficulté vient de la définition des motifs. En effet, ceux-ci doivent être suffisamment précis pour pouvoir discriminer les différents types d'attaques, mais également suffisamment génériques pour pouvoir détecter les différences variantes d'un même type d'attaque. Une signature trop générique conduira à l'augmentation du nombre de faux positif, diminuant ainsi la faisabilité. La technique de détection par scénario nécessite en outre une maintenance active du système pour mettre à jour régulièrement la base de signatures. En effet, le système ne peut détecter que des attaques connues a priori, il faut donc pouvoir réactualiser cette connaissance. Ceci implique notamment un coût de maintenance important.

Les outils commerciaux ou libres ont évolué pour proposer une personnalisation de la signature afin de faire face à des attaques dont on ne connaît qu'une partie de leurs éléments [3].

2.4.2.2 *L'approche comportemental (Détection d'anomalies)*

Cette approche se base sur l'hypothèse que l'exploitation d'une faille du système nécessite une utilisation anormale de ce système. Une intrusion est alors détectée en fonction du comportement passé de l'utilisateur. Pour cela, il faut préalablement dresser un profil dit « normal » à partir des habitudes passées de l'utilisateur et déclencher une alerte lorsque des événements hors profil se produisent. On cherche donc à répondre à la question « le comportement actuel de l'utilisateur ou du système est-il cohérent avec son comportement passé ? »

Plusieurs techniques sont utilisées dans cette approche parmi elles l'observation de seuils, ou on fixe le comportement normal d'un utilisateur par la donnée de seuils de certaines mesures (par exemple, le nombre maximum de mots de passe erronés). On a ainsi une définition claire et simple des comportements non acceptés. Il est cependant difficile de caractériser un comportement intrusif en termes de seuils, et on risque beaucoup de fausses alarmes ou beaucoup d'intrusions non détectées sur une population d'usager non uniforme.

Cette approche peut être appliquée non seulement à des utilisateurs mais aussi à des applications et des services [3].

2.4.3 *Le manager*

Le manager s'occupe de la présentation des alertes à l'opérateur. Il réalise les fonctions de corrélation d'alertes et peut assurer le traitement d'incidents, selon les fonctions suivantes :

1. confinement de l'attaque, qui a pour but de limiter les effets de l'attaque.

2. éradication de l'attaque, qui tente d'arrêter l'attaque.

3. recouvrement, qui est l'étape de restauration du système dans un état sain.

4. diagnostic, qui est la phase d'identification du problème, de ses causes et qui peut.

Les réactions sont rarement automatisées, car elles peuvent se traduire par un déni de service en cas de réaction à des faux positifs.

2.5 Les types d'IDS

Depuis le premier modèle de systèmes de détection d'intrusion générique qui a été proposé par Dr Dorothy Denning en 1987, plusieurs travaux et recherches ont été effectués sur cet axe. Aujourd'hui on peut distinguer trois grandes familles :

- **Système de détection d'intrusions basé réseau (NIDS)** qui analysent les flux en transit sur le réseau et détectent les intrusions en temps réel.
- **Système de détection d'intrusion basé hôte (HIDS)** qui analysent le flux et l'activité d'une seule machine en temps réel ainsi que les fichiers journaux.
- **IDS Hybrides** ils utilisent les avantages du NIDS et du HIDS pour générer des alertes plus pertinentes.

2.5.1 Système de détection d'intrusion basé réseau (NIDS)

Un NIDS écoute et analyse le trafic passant sur le réseau, il peut étudier les trames réseau à tous les niveaux (couche réseau, couche de transport, couche applicatif). Les NIDS sont capables d'analyser les paquets pour comprendre les protocoles et chercher les signes d'attaques à différents endroits. Un NIDS peut être déployé soit sur la machine cible, et va donc surveiller son propre trafic, soit sur une machine indépendante et là il surveillera tout le trafic réseau. [4]

Un NIDS surveille les paquets sur le réseau et tente de découvrir s'il y a une tentative d'intrusion ou de déni de service. Un NIDS peut être déployé soit sur la machine cible, et va donc surveiller son propre trafic, soit sur une machine indépendante et là il surveillera tout le trafic réseau.

Un IDS basé-réseau (NIDS) surveille le trafic en des points choisis sur un réseau ou un ensemble interconnecté de réseaux. Le NIDS examine paquet par paquet le trafic en temps réel ou en temps quasi-réel, pour tenter de d´détecter les modèles d'intrusions.

Le NIDS peut examiner le protocole au niveau réseau, transport et/ou au niveau application. Une installation typique de NIDS comprend un certain nombre de capteurs pour la surveillance du trafic des paquets, un ou plusieurs serveurs pour les fonctions de gestion et une ou plusieurs consoles de gestion pour l'interface avec l'homme. L'analyse des modèles de trafic pour détecter les intrusions peut être faite au niveau du capteur, au niveau du serveur de gestion, ou dans une combinaison des deux. [5]

2.5.1.1 *Architecture*

L'implantation d'un NIDS sur un réseau se fait de la façon suivante : des capteurs sont placés aux endroits stratégiques du réseau et génèrent des alertes s'ils détectent une attaque. Ces alertes sont envoyées à une console sécurisée, qui les analyse et les traites éventuellement. Cette console est généralement située sur un réseau isolé qui relie uniquement les capteurs à la console.

Un NIDS fonctionne toujours en mode espion, ceci garantit un fonctionnement furtif qui lui permet d'analyser tous les paquets passant par le lien en question.

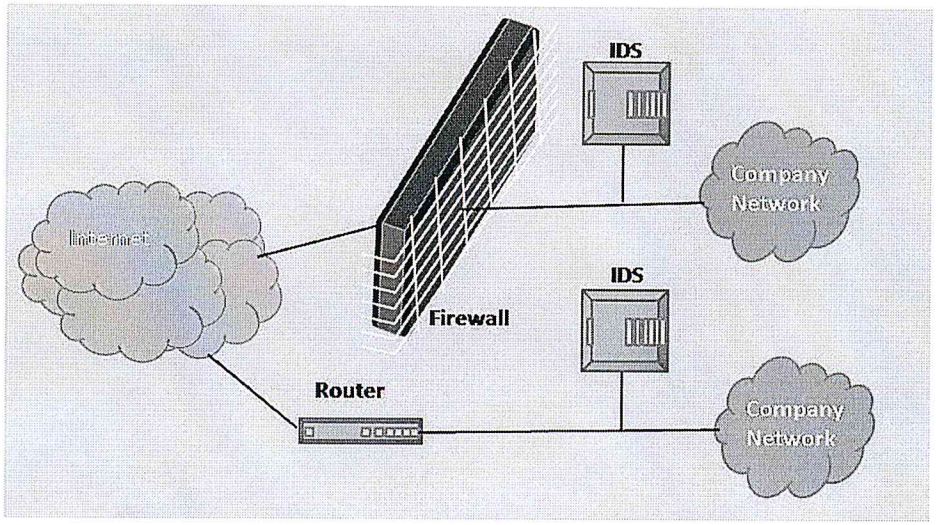


Figure 2 : Emplacement des NIDS dans un réseau

2.5.1.2 *Avantages*

- Les capteurs sont bien sécurisés puisqu'ils se contentent d'observer le trafic.
- La détection est facile grâce aux signatures .
- Possibilité de filtration du trafic.

2.5.1.3 *Inconvénients*

- La probabilité de faux négatifs (attaques non détectées) est élevée et il est difficile de contrôler le réseau entier.
- Ils doivent principalement fonctionner de manière cryptée.
- À l'opposé des IDS basés sur l'hôte, ils ne voient pas les impacts d'une attaque .

2.5.2 *Système de détection d'intrusion basé hôte (HIDS)*

Les systèmes de détection d'intrusion basés sur l'hôte analysent exclusivement l'information concernant un hôte. Comme ils n'ont pas à contrôler le trafic du réseau mais uniquement les activités d'un hôte ils se montrent habituellement plus précis sur les types d'attaques.

Un HIDS se compose d'un agent sur un hôte, qui identifie les intrusions par des analyses d'appels système, des journaux d'application, des modifications du système de fichiers et des états d'activités de l'hôte. [4]

Les HIDS sont en général placés sur des machines sensibles, susceptibles de subir des attaques et possédant des données sensibles pour l'entreprise. Les serveurs, web et applicatifs, peuvent notamment être protégés par un HIDS.

2.5.2.1 Avantages

- L'impact de l'attaque est directement constaté, donc on peut mieux réagir.
- Détection d'attaques impossibles à détecter avec des IDS réseau puisque, car le trafic y est souvent crypté.
- Possibilité d'observer les activités sur l'hôte avec précision.

2.5.2.2 Inconvénients

- Ils sont plus vulnérable aux attaques de type déni de service.
- L'analyse des traces d'audits du système est très contraignante en raison de la taille de ces derniers .
- Ils consomment beaucoup de ressources CPU.

2.5.3 Comparaison entre NIDS et HIDS

Malgré la grande différence de déploiement entre le NIDS et le HIDS, leur comparaison reste possible, et, sur beaucoup de points, tels que la protection sur et hors LAN, l'exigence en bande passante sur le LAN et sur internet, etc.

- Selon la nature des systèmes à surveiller et de l'activité qui leur est liée, un choix judicieux sur un site ne le sera pas sur un autre.
- Corréler les informations fournies par les deux types de sondes fournit une valeur ajoutée non négligeable.

Les contraintes d'exploitation ne permettent cependant pas toujours d'optimiser la surveillance .

- Utilisation des ressources pour les HIDS.
- Coûts au niveau de l'infrastructure réseau pour les NIDS.

Les rôles des NIDS et HIDS sont largement complémentaires, Une architecture idéale consiste à allier les deux.

2.5.4 *Les Système de détection Hybride*

Généralement utilisés dans un environnement décentralisé, ils permettent de réunir les informations de diverses sondes placées sur le réseau. Leur appellation « hybride » provient du fait qu'ils sont capables de réunir aussi bien des informations provenant d'un système HIDS qu'un NIDS.

Les systèmes HIDS et NIDS se complètent l'un l'autre. Ainsi, nous pouvons remarquer que la combinaison des deux IDS permet d'éviter presque toutes les menaces d'attaques. Sachant que les systèmes hybrides sont basés sur une architecture distribuée ce qui oblige chaque composant d'unifier son format d'envoi d'alertes pour éviter tout sort de confusion.

3 Apprentissage automatique

L'apprentissage automatique[4] est une discipline scientifique faisant référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine ou un automate d'évoluer grâce à un processeur d'apprentissage, et par conséquent de remplir des taches qu'il est difficile ou impossible de résoudre par des moyens algorithmiques classiques. En règle générale, l'utilisation des techniques d'apprentissage automatique nécessite une base de données de départ, permettant à un algorithme de construire un modèle optimal. L'algorithme se fonde donc sur des observations et optimise un modèle selon un critère de performance donné. Les techniques d'apprentissage automatique sont utilisées dans de nombreux domaines : reconnaissance de formes, de visages, de caractères de manuscrites, de la parole, aide au diagnostic médical et bien d'autres encore.

On peut distinguer deux grandes familles d'apprentissage automatique :

- L'apprentissage supervisé
- L'apprentissage non supervisé

[4] Anglais : Machine Learning

3.1 L'apprentissage supervisé

Dans ce type d'apprentissage automatique, les classes sont prédéterminées. La base de données d'apprentissage contient alors un ensemble de couples (entrée, sortie), le processus se passe en deux phases. Lors de la première phase (apprentissage), il s'agit de déterminer un modèle de données étiquetées (pour chaque type d'entrée on associe une sortie). La seconde phase (test) consiste à construire un modèle permettant de classer au mieux de nouvelles données, non présentes dans la base de données d'apprentissage. Connaissant préalablement les classes associées à toutes les entrées. [6]

On distingue en général trois types de problème auxquels l'apprentissage supervisé est appliqué. Ces taches diffèrent essentiellement par la nature des données.

3.1.1 Classification

Dans les problèmes de classification, l'entrée correspond à une instance, et la sortie qui y est associée indique la classe de cette instance. Par exemple pour un problème de détection d'intrusion dans un système informatique, l'entrée serait l'état du système et son comportement lors d'une période, et la sortie indiquerait si ce comportement et considérer comme normal ou non [7].

3.1.2 Régression

Dans le problème de régression, l'entrée n'est pas associée à une classe, mais dans le cas général, à une ou plusieurs valeurs réelles (un vecteur). Par exemple pour une expérience de biochimie, on pourrait vouloir prédire le taux de réaction d'un organisme en fonction de taux de différentes substances qui lui sont administrées [7].

3.1.3 Séries temporelles

Dans le problème de série temporelle, il s'agit typiquement de prédire les valeurs futures d'une certaine quantité connaissant, des valeurs passées ainsi que d'autres informations. Par exemple le rendement d'une action en bourse [7].

3.2 L'apprentissage non-supervisé

Les classes ne sont pas connues à l'avance. Dans ce cas, l'algorithme tente de découvrir par lui-même la structure des données. Un tel algorithme essaie d'identifier des groupes d'entrées relativement homogènes sur la base des valeurs des attributs des éléments de la base de données. L'objectif est d'obtenir le modèle le plus général possible afin de classer de nouvelles données.

3.3 Méthode du plus proche voisin

La règle de classification du K-plus proche voisin[5] (KNN) se classe dans l'apprentissage supervisé [8], c'est une méthode de classification très puissante qui permet la classification d'un prototype inconnu grâce à un ensemble de prototypes d'apprentissage. Quoiqu'une conséquence inévitable de la grande taille des ensembles de prototypes, due au fait que les bases de données utilisées, dans certains domaines, tel que la détection d'intrusion, sont en croissance constante et dynamique en vue du nombre important de mises à jour quotidiennes, est la charge de calcul qu'implique ce problème de recherche. Cela constitue l'un des principaux inconvénients de la méthode KNN. Un autre inconvénient très important vient du fait que les prototypes d'apprentissage peuvent contenir des prototypes bruyants ou mal étiquetés qui peuvent influer sur les résultats et par conséquent les fausser [8].

3.3.1 Présentation de l'algorithme

L'apprentissage supervisé est basé sur un ensemble de classes ou prototypes définis préalablement, le K-NN se base sur un type d'apprentissage basé sur les instances, appelé aussi ***memory-based*** ou ***lazy-learning***, car il n'y a pas d'apprentissage réel, les exemples d'apprentissage sont juste stockée en mémoire et réutilisées lors de la classification.

L'algorithme K-NN est une méthode de classification d'objets, son principe est très simple. L'algorithme prend comme paramètres un ensemble de données d'apprentissage, une fonction de calcule de distance et un entier k. Pour tout nouveau point de teste (instance) pour lequel il doit prendre une décision, l'algorithme recherche dans l'ensemble de données d'apprentissage les k points les plus proches en termes de la distance, et attribue la classe qui est la plus fréquente parmi les classes de ses k voisins. La distance la plus souvent utilisée est la distance Euclidienne.

3.3.2 Algorithme de base

L'algorithme de base de la méthode du plus proche voisins a était présenté sous le format suivant [9]

[5] Anglais : K-Nearest Neighbor

O : *L'observation à classer*

X : *Ensemble d'attributs du prototype*

Y : *Classe du prototype*

D : *L'ensemble des n prototypes et leur étiquette, notés respectivement* (Xi, Yi)

k : *Le nombre de plus proches voisins à considérer*

Li : *Une liste contenant les plus proches voisins de O avec leurs distances*

$Vote(liste)$: *Une fonction de vote qui retourne la classe la plus représentée dans une liste et qui départage arbitrairement les cas d'égalité*

$d(a, b)$: *Une fonction qui calcule la distance selon une métrique (Souvent la distance euclidienne)*

Algorithme K-Plus proche voisins

Début

$i \leftarrow 0$

Répéter

$i \leftarrow i + 1$

$dist \leftarrow d(Xi, O)$

$Li \leftarrow \{dist, Yi\}$

Jusqu'à $i = k$

Trier la liste L par ordre croissant des distances

$PPVi \leftarrow Li \mid i \in \{1, \ldots, k\}$

Fin

Return $Vote(PPV)$

Figure 3 : Algorithme K plus proche voisins

3.3.3 Mesures de similarité

Les méthodes de classification se basent sur le concept de similarité entre objets ; les mesures de similarité sont donc fondamentales pour la plupart des algorithmes tels que le K-NN. Le choix de la mesure de similarité doit être réalisé en prenant en compte toutes les informations disponibles sur l'ensemble des données.

3.3.3.1 Distance euclidienne et métrique de Minkowski

La plus simple et la plus populaire des mesures de similarité entre des données multi variées est la distance euclidienne qui est un cas particulier de la famille des métriques de Minkowski (quand l'exposant de la métrique $r = 2$). [10]

Les métriques de Minkowski sont définis par :

$$D_r(X_i, X_k) = \left(\sum_{j=1}^{d} |x_{ij} - x_{kj}|^r \right)^{1/r}$$

D'où la distance euclidienne est définie par :

$$D_2(X_i, X_k) = \sqrt{\sum_{j=1}^{d} (x_{ij} - x_{kj})^2}$$

Tel que :

X : Ensemble de données multi variées

r : Exposant de la métrique

d : Dimension de données (le nombre d'attributs d'un ensemble des données)

x_i : Scalaire représentant une mesure d'un vecteur de données

La distance euclidienne, $r = 2$ est la plus connue ; couramment utilisée dans des espaces à 2 ou 3 dimensions, cette métrique donne des bons résultats si l'ensemble des données présente des classes compactes et isolées [10].

3.3.3.2 *Distance de Manhattan*

La distance de Manhattan, c'est aussi un cas spécial des métriques de Minkowski tel que $r = 1$, [10] plus appropriée pour mesurer la similarité entre des données multi variées. Défini par :

$$D_1(X_i, X_k) = \sum_{j=1}^{d} |x_{ij} - x_{kj}|$$

3.3.4 *Inconvénients de la méthode*

La méthode connait trois inconvénients majeurs.

3.3.4.1 *Définition de la valeur du k*

La valeur de k est un des paramètres à déterminer lors de l'utilisation de ce type de classificateur car des résultats très différents résultent de sa valeur [11].

La figure 1 montre un exemple où on doit classer un nouvel élément x. Si on choisit $k = 1$, x sera classé dans C_2 , Si $k = 5$, le même x sera classé dans C_1 car c'est la classe la plus fréquente parmi ses 5 plus proches voisins. On voit donc que le choix de k reste très important dans le résultat final.

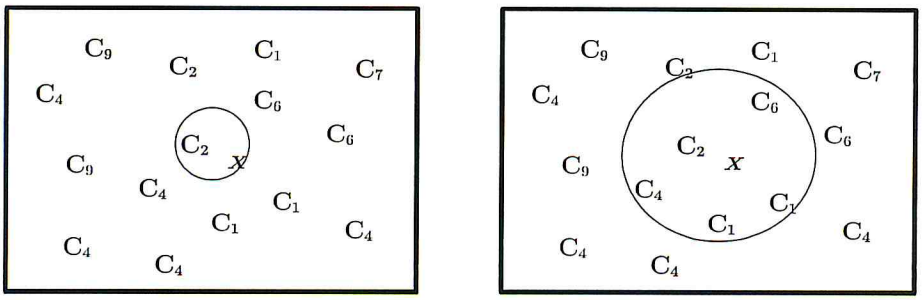


Figure 4: Exemple de classification avec différentes valeurs de k

Souvent on procède à une optimisation de ce paramètre à l'aide de tests sur une portion de l'ensemble d'apprentissage.

À mesure que k augmente, La probabilité que la classe Ci soit la meilleure à être attribuée à x augmente, On préférera donc des valeurs élevées de k pour augmenter la précision de l'estimation, d'un autre côté, il faut s'assurer que les k voisins x' de l'élément inconnu x soient assez près de sorte que la probabilité que x appartient à la classe Ci de x' soient considérable, d'une autre manière plus les voisins diffèrent de l'élément à classer, moins ils ont d'impact sur la prise de décision. Cependant, il demeure nécessaire de limiter le nombre de voisin pour s'en tenir à un temps de calcul raisonnable, d'où le choix de k résulte donc d'un compromis et doit être ajusté à chaque problème.

3.3.4.2 *La taille de l'ensemble d'apprentissage*

La taille de l'ensemble d'apprentissage est un facteur très important dans les résultats fournis par l'algorithme [11], avec le temps de nouvelles instances sont détectés et classées et plusieurs classes peuvent apparaitre d'où la tache de l'algorithme devient de plus en plus complexe, d'un côté le temps de calcul des distances entre les instances de l'ensemble d'apprentissage et la nouvelle observation sera couteux, surtout si la valeur du k et très importante. D'un autre côté, avec le temps des prototypes d'apprentissage bruyants apparaissent dans l'ensemble d'apprentissage, ce sont des prototypes qui ne participent pas au classement des nouvelles observations ou qui ont un effet très léger sur le résultat final de l'algorithme ce qui nous mène vers la sélection des prototypes. Une tâche qui n'est pas facile à réaliser à cause de sa complexité algorithmique.

3.3.4.3 *La complexité algorithmique*

Supposons n protoypes d'apprentissage dans un espace de caractéristique de dimension d. Le calcul de la distance (en général euclidienne) entre une observation à classer et un prototype est de complexité $O(d)$. Pour trouver le prototype le plus proche d'une observation, la méthode la plus simple consiste à calculer la distance entre cette observation et les n prototypes et à comparer chaque distance obtenue avec celle que l'on considère la plus proche. La complexité est donc d'ordre $O(d \times n2\ )$ [11] .

Il existe néanmoins des techniques pour réduire la complexité de l'algorithme, mais il faut noter que souvent ces techniques n'assurent pas de trouver le ou les plus proches voisins. La diminution de la complexité peut donc signifier l'augmentation de la probabilité d'erreur du système.

4 Approches de détection d'intrusions basées sur l'algorithme KNN

La méthode proposée par Law et Kwok [12] consiste à déterminer si les séquences d'alarmes entrantes sont différentes de la situation normale, si oui, une alerte peut être signe d'attaque et une investigation plus approfondie est nécessaire, sinon le risque d'être attaqué est faible. Etant donné un grand nombre d'alarmes générées par un IDS dans un environnement sans attaques avec N attributs dans un espace représentant ce nombre d'alarmes de types différents dans une fenêtre temporelle. Pour détecter les modèles D'alertes anormales nouvellement arrivées, de nouveaux points de données sont créés pour ces nouvelles alarmes. La distance de ces points par rapport aux points normaux indique leur déviation de la situation normale. Le classificateur KNN a été utilisé pour classer les données et savoir si un point est normal ou non. KNN a d'abord été utilisé dans la zone de détection d'intrusions pour la détection d'anomalies d'apprentissage et la découverte du comportement du programme d'intrusion à partir des données d'audit. Les auteurs ont proposé de l'étendre à l'utilisation pour la détection de fausses alarmes en se basant sur la distance Euclidienne, qui représente la similitude entre deux points. Plus cette distance est petite plus les points sont similaires. Le processus de détection d'alarmes a utilisé les alarmes normales (alarmes relevées par l'IDS en cas d'aucune attaque) pour construire le modèle de fausses alarmes. Les alarmes entrantes sont filtrées en permanence par une procédure de filtrage utilisant ces modèles comme patrons. La modélisation proposée par les auteurs ainsi que les procédures de filtrage ont été indépendantes du processus de détection d'intrusions. Par conséquent, ce modèle peut être appliqué à la plupart des IDS commerciaux sans rien changer dans leur configuration.

Yang Li et Li Guo [13] ont proposé une nouvelle méthode de détection d'intrusions basée sur l'algorithme d'apprentissage automatique TCM-KNN, ainsi qu'une méthode de sélection de données d'apprentissage basée sur l'apprentissage actif. Cette approche a été la première qui inclut le TCM-KNN dans la détection d'intrusions. Pour cette utilisation les auteurs ont optimisé le TCM-KNN sur deux aspects.

La première amélioration est l'introduction d'une méthode d'apprentissage actif afin de sélectionner un petit nombre de données mais de bonne qualité pour l'apprentissage, ce qui permet d'alléger la quantité de données étiquetées et de réduire la taille de l'ensemble d'apprentissage, et donc de réduire aussi le coût de calcul du TCM-KNN.

Le second aspect a été la proposition d'une méthode de sélection d'attributs les plus importants et nécessaires pour le TCM-KNN. Les auteurs ont attribué à chaque point une mesure appelée mesure d'étrangeté individuelle. Cette mesure définie l'étrangeté du point par rapport au reste des points. La mesure d'étrangeté utilisée a été le rapport de la somme des k distances les plus proches de la même classe à la somme des k distances les plus proches de toutes les autres classes. Cette mesure coût lorsque la distance entre les points de la même classe augmente ou lorsque la distance entre les autres classes devient plus petite.

L'Algorithme comprend deux phases importantes : la phase de formation et la phase de détection. Dans la première phase trois actions principales doivent être considérées : la collecte

des données pour la modélisation, la sélection d'attributs et enfin la modélisation par l'algorithme TCM-KNN et donc la construction du classificateur de détection d'intrusions. Pour la phase de détection toutes les données recueillies en temps réel du réseau doivent être prétraitées en étant vectorisées suivant les attributs sélectionnés.

Sur la base de leur précédente proposition (TCM-KNN), les mêmes auteurs ont présente un mécanisme de sélection d'instance pour TCM-KNN basée sur EFCM (Extended CMeans Fuzzy) pour la détection d'anomalies. Ils ont introduit un algorithme de clustering [7], visant à limiter la taille des données de formation, réduisant ainsi le coût de calcul du TCM-KNN et améliorant le rendement de sa détection. Tout d'abord, les auteurs ont proposé de classer l'ensemble de données d'entrainement normal en introduisant trois classes : les données notables, des données obscures et les données redondantes. Les données notables ont été définies comme celles appartenant à une série de clusters. Ces données représentent ces clusters et y sont les plus centrées. Les données obscures désignent celles qui appartiennent à un cluster avec de très petits grades d'appartenance calculés par EFCM. Les données redondantes comprennent, quant à elles, les données restantes qui n'appartiennent à aucune des deux classes précédentes, et qui sont essentiellement les données qui se trouvent le long des limites des clusters. Ils ont ensuite proposé de former l'algorithme TCM-KNN avec les données notables et obscures, afin que ce dernier puisse offrir le meilleur usage possible en permettant de distinguer les anomalies des cas normaux avec un taux de d´détection élevé et un faible taux de faux positifs, ainsi qu'un coût de calcul réduit.

Liwei Kuang and Mohammad Zulkernine [**14**] ont proposé une méthode de détection d'anomalies: CSI-KNN en combinant l'étrangeté et la mesure d'isolation. L'algorithme de détection d'intrusions analyse différentes caractéristiques des données du réseau en employant deux mesures: l'´étrangeté et l'isolation.

La métrique d'étrangeté mesure si le comportement inconnu est plus semblable aux comportements normaux ou aux comportements anormaux. Elle effectue une détection d'anomalies en analysant la distribution d'étrangeté des données normales mais qui traite également les attributs d'anomalies en employant des données d'attaque. La métrique d'isolation identifie la similarité par rapport aux classes normales et peut détecter les attaques anormales. C'est une méthode pure de détection d'anomalies qui n'utilise que les données normales.

Basée sur ces mesures, une unité de corrélation soulève les alertes d'intrusions et les estimations de confiance associées. Multiples classificateurs CSI-KNN travaillent en parallèle afin de traiter différents types de services réseau. Le composant de corrélation agrège les résultats de l'étrangeté et les modèles d'isolement et fournit une décision finale. En outre, l'algorithme fournit une confiance graduée pour chaque alerte d'intrusion.

5 Les Métaheuristiques

Le mot *métaheuristique* est dérivé de la composition de deux mots grecs :

- *Heuristique* qui vient du verbe heuriskein et qui signifie « trouver »
- *Meta* qui est un suffixe signifiant « au-delà », «dans un niveau supérieur »

Plusieurs définitions ont été proposées pour expliquer clairement ce qu'est une métaheuristique. Aucune de ces définitions n'est universellement reconnue :

> « A methaeuristic is formally defined as an iterative generation process which guides a subordinate heuristic combining intelligently diffrent concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions » [**15**].

> « A methaeuristic in an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a simple local search or just a constructive methode » [**15**].

Pour résumer ces définitions, on peut dire que les propriétés fondamentales des métaheuristiques sont les :

- Les métaheuristiques sont des stratégies qui permettent de guider la recherche d'une solution optimale.
- Le but visé par les métaheuristiques est d'explorer l'espace de recherche efficacement afin de déterminer des solutions (presque) optimales.
- Les techniques qui constituent des algorithmes de type métaheuristiques vont de la simple procédure de recherche locale à des processus d'apprentissage complexes.
- Les métaheuristiques sont en général non-déterministes et ne donnent aucune garantie d'optimalité.
- Les métaheuristiques peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace de recherche.
- Les concepts de base des métaheuristiques peuvent être décrits de manière abstraite, sans faire appel à un problème spécifique.
- Les métaheuristiques peuvent faire usage de l'expérience accumulée durant la recherche de l'optimum, pour mieux guider la suite du processus de recherche.

5.1 Classification des métaheuristiques

On peut faire la différence entre les métaheuristiques qui s'inspirent de phénomènes naturels et celles qui ne s'en inspirent pas. Par exemple, les algorithmes génétiques et les algorithmes des fourmis s'inspirent respectivement de la théorie de l'évolution et du comportement de fourmis à la recherche de nourriture. Contrairement à la méthode Tabou. Une telle classification ne semble

dépendent pas très utile et est parfois difficile à réaliser, En effet, il existe de nombreuse métaheuristiques récentes qu'il est difficile de classer dans l'une des deux catégories.

Une autre façon de classifier les métaheuristiques et de distinguer celles qui travaillent avec une **population de solutions**, le cas des algorithmes génétiques ou les colonies de fourmis. Celles qui ne manipulent qu'une seule solution à la fois et qui tentent itérativement de l'améliorer sont appelée **méthodes de recherche locale** ou **méthodes de trajectoires**, la méthode Tabou, le Recuit Simulé et la recherche à voisinage variables sont des exemples typiques.

La figure suivante décrit la classification des métaheuristiques.

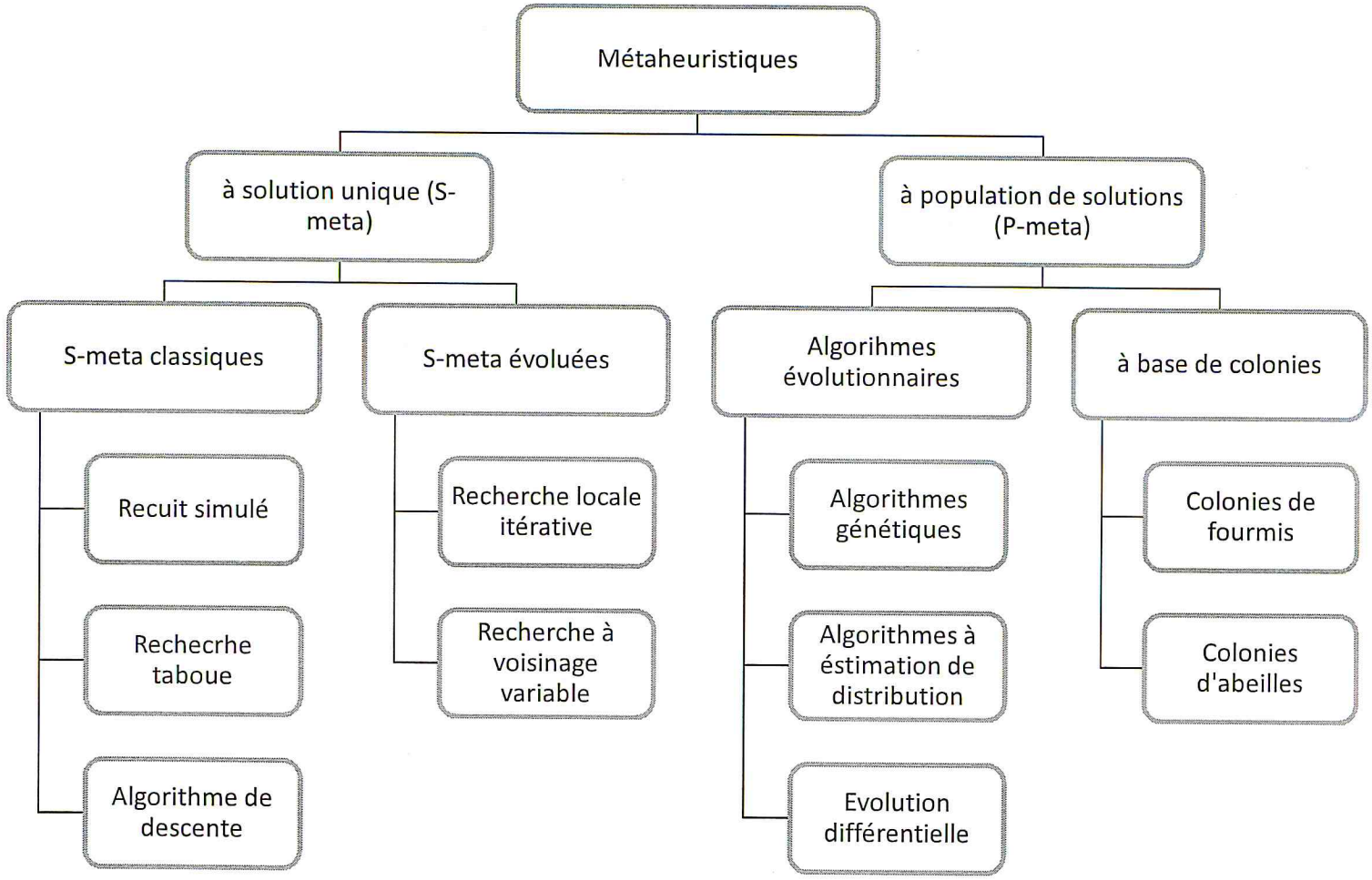


Figure 5: Classification des Métaheuristiques

5.2 Métaheuristiques à solution unique

Ces algorithmes partent d'une solution initiale (obtenue de façon exacte, ou tirage aléatoire) et s'en éloignent progressivement, pour réaliser une trajectoire, un parcours progressif dans l'espace de solutions, le terme de recherche locale est de plus en plus utilisé pour qualifier ces méthodes. Dans cette section nous allons citer trois algorithmes. [16]

5.2.1 La méthode de descente

Le principe de la méthode de descente (dite aussi *basic local search*) consiste de démarrer d'une solution s et de choisir une solution s' dans un voisinage de s, telle que s' améliore la recherche (généralement telle que $f(s') < f(s)$). [17]

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

La méthode de recherche locale la plus élémentaire est la *méthode de descente.* On peut la schématiser par le pseudo code suivant :

Procédure descente_simple (solution initiale s)

Répéter :

Choisir s' dans N(s)

Si f(s') < f(s) alors s← s'

Jusqu'à ce que f(s') f(s) , ∀ s'∈S

Fin

On remarque que la solution finale est bien meilleure que la solution initiale mais elle n'est pas la solution optimale dans l'espace S. parce que l'algorithme d'arrête des que la solution s'commence à se dégrader.

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

5.2.1.1 *Avantages et inconvénients*

Le principal avantage de la recherche locale simple est évidemment sa grande simplicité de mise en œuvre: la plupart du temps, elle ne fait que calculer $f(s+i)-f(s)$, où i correspond à un déplacement élémentaire.

Par contre l'efficacité des méthodes de recherche locale simples (descente, ou plus grande descente) est très peu satisfaisante. D'abord, par définition, la recherche s'arrête au premier minimum local rencontré, c'est là leur principal défaut. Pour améliorer les résultats, on peut lancer plusieurs fois l'algorithme en partant d'un jeu de solutions initiales différentes, mais la performance de cette technique décroît rapidement.

5.2.2 *La méthode de recuit simulé*

Comme on vient de voir, le principal défaut de la méthode de descente est de s'arrêter au premier minimum trouvé. En revanche, autoriser de temps à autre une certaine dégradation des solutions trouvées, afin de mieux explorer tout l'espace des configurations, a conduit au développement des deux méthodes à savoir le recuit simulé et la méthode Tabou. [17]

5.2.3 *La méthode Tabou*

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

A l'inverse du recuit simulé qui génère de manière aléatoire une seule solution voisine $s' \in N(s)$ à chaque itération, Tabou examine un échantillonnage de solutions de $N(s)$ et retient la meilleure s' même si $f(s')>f(s)$. La recherche Tabou ne s'arrête donc pas au premier optimum trouvé. [17]

Le danger serait alors de revenir à s immédiatement, puisque s est meilleure que s'. Pour éviter de tourner ainsi en rond, on crée une liste T qui mémorise les dernières solutions visitées et qui interdit tout déplacement vers une solution de cette liste. Cette liste T est appelée liste Tabou.

Mémoriser les configurations telles quelles peut s'avérer extrêmement gourmant en mémoire, et on préférera plutôt conserver une caractéristique, un attribut de chaque configuration, typiquement, la valeur de la fonction objectif prise en chaque configuration. Mais il serait absurde d'interdire définitivement une valeur. Plusieurs configurations différentes peuvent en effet donner la même valeur de fonction objective, et l'on risquerait ainsi de s'interdire des chemins qui permettraient de s'échapper vers autre vallée, plus profonde et prometteuse. Les solutions ne demeurent donc dans T que pour un nombre limité d'itérations.

Le choix de T doit être fait judicieusement si le nombre T est trop petit, il y a risque de boucle infinie, mais s'il est trop grand, cela peut perturber la recherche en empêchant l'algorithme d'exploiter en profondeur une vallée. [17]

Par ailleurs, après un certain temps d'exécution on peut aboutir à une solution meilleure que toutes celles déjà visitées, mais elle est présente dans la liste Tabou. Pour éviter cela, on incorpore dans l'algorithme des *critères d'aspiration*, qui autorisent certains mouvements, bien qu'interdits par la liste Tabou, parce qu'on suppose qu'ils vont améliorer la recherche. Typiquement, une solution *s*'est toujours autorisée si sa valeur *f(s)* est meilleure que toutes les solutions rencontrées jusqu'ici.

Le pseudo code suivant peut représenter l'algorithme général de la méthode :

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

5.2.3.1 Avantages et inconvénients

La méthode Tabou est une méthode de recherche locale, et la structure de son algorithme de base est finalement assez proche de celle du recuit simulé, donc on passera donc de l'un à l'autre facilement, avec l'avantage, par rapport au recuit simulé, d'avoir un paramétrage simplifié : dans un première temps, le paramétrage consistera d'abord à trouver une valeur indicative T d'itérations pendant lesquelles les solutions sont interdits. Il faudra également décider d'une stratégie de mémorisation à long terme – sur la qualité des solutions, sur leur récence, ou sur leur qualité...

En revanche, la méthode Tabou exige une gestion de la mémoire de plus en plus lourde à mesure que l'on voudra raffiner le procédé en mettant en place des stratégies de mémorisation complexe.

5.3 Métaheuristiques à population de solutions

Elles constituent à travailler avec un ensemble de solutions simultanément, que l'on fait évoluer graduellement. L'utilisation de plusieurs solutions simultanément permet naturellement d'améliorer l'exploration de l'espace des configurations. On recense dans cette catégorie les algorithmes évolutionnaires.

Les algorithmes évolutionnaires sont des algorithmes d'optimisation stochastique[6] fondés sur un parallèle grossier avec l'évolution darwinienne des populations biologiques [17].

Un algorithme évolutionnaire a pour but d'optimiser une fonction réelle. La fonction à optimiser, appelée aussi performance, est définie sur un espace de recherche Ω. L'algorithme fait évoluer une population, un sous-ensemble de l'espace de recherche. Cette évolution résulte d'une part de la théorie de Darwin, qui se manifeste par la sélection des individus et le remplacement. D'autre part de l'effet du hasard, qui s'exprime dans l'initialisation des individus et les opérateurs de variation.

L'idée fondamental est que la sélection favorise les individus qui optimisent la performance et que les variations font apparaitre dans la population sélectionnée des individus que l'on peut espérer meilleurs au regard de la performance. Dans cette évolution les générations successives de la population restent à taille constante et l'aspect stochastique ne dépend que de la génération précédente.

La mise en place d'un algorithme évolutionnaire est complexe et le cout de calcul est important [17]. De tels algorithmes sont donc destinés à traiter des problèmes qui n'ont pas de solutions classiques.

Trois types d'algorithmes évolutionnaires ont été développés isolément et à peu près simultanément, par différents scientifiques :

- La programmation évolutionnaire (L. Fogel 1966)
- Les stratégies d'évolution (J.Rechenberg 1973)
- Les algorithmes génétiques (J.Holland 1975)

Dans les années 90n ces trois champs ont été regroupés sous le terme anglo-saxon *Evolutionnary Computation* [18].

Dans notre cas nous allons traiter seulement les algorithmes génétiques.

5.3.1 *Les algorithmes génétiques*

Les algorithmes génétiques ont été inspirés de la théorie d'évolution présentée par Darwin et ka théorie de la génétique moderne. [18] Le vocabulaire employé est donc directement calqué sur ces deux théories. Nous parlons donc de :

- **Individu :** c'est l'élément essentiel, il représente une solution (complète, ou partielle) de l'espace de recherche, la structure de l'individu diffère d'un problème à un autre.
- **Codage :** il définit la structure de l'individu, le codage est généré généralement après une phase de modélisation mathématique du problème traité. Le codage binaire a été utilisé au début, aujourd'hui les codages réels sont largement utilisés.

[6] Se dit de phénomène qui, partiellement relèvent du hasard et qui font l'objet d'une analyse statistique.

- **Population** : c'est un ensemble d'individus qui sera évolué durant un certain nombre de génération, la solution à un problème peut être la population finale obtenue ou un individu parmi cette population.
- **Génération** : c'est une étape durant laquelle la population sera évoluée, elle subira plusieurs opérations comme la sélection, croisement et mutation.
- **Opérateurs** : ils permettent de diversifier la population au cours des générations et d'explorer l'espace d'état de façon à favoriser l'émergence de meilleurs individus. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace de solutions.

5.3.2 *Principe de l'algorithme génétique*

L'algorithme génétique repose sur une boucle qui enchaîne des étapes de *sélections* et des étapes de *croisements*.

Initialiser la population initiale P

Évaluer P

Tant que (Condition d'arrêt non satisfaite) faire

P'= Sélection des individus autorisés à se reproduire (les parents)

P'= Appliquer opérateur de croisement sur P'

P'= Appliquer opérateur de mutation sur P'

P'= Remplacer les anciens individus de P par leurs descendants de P'

Évaluer de nouveau P

Fin tant que

Dans un premier temps, une population initiale P de α individus est générée à travers un mécanisme de génération de population. Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. La performance de cette population est ensuite évaluée, grâce à une fonction dite (*fitness*).

Une procédure de *sélection* est appliquée sur les individus de la population initiale pour désigner ceux autorisés à se reproduire. Ils formeront la population P'

Les opérateurs de croisement et de mutation sont appliqué sur P' de façon à obtenir une population d'enfants, la performance de ces dernies est évaluée, et l'on désigne, dans la population totale résultante (parents + enfants), les individus autorisés à survivre, de telle manière que l'on puisse repartir d'une nouvelle population de α individus. La boucle est bouclée, et l'on recommence une phase de sélection pour la reproduction, une phase de mutation, et ainsi de suite.

La condition d'arrêt diffère selon le problème à résoudre et selon le résultat qu'on cherche à obtenir par exemple :

- Un taux minimum qu'on désire atteindre d'adaptation de la population au problème.
- Un certain nombre de génération.
- Un certain temps de calcul à ne pas dépasser.
- Non amélioration de la valeur retournée par la fonction de fitness après un certain temps d'exécution ou un certain nombre de génération.
- Ou une combinaison des points cités.

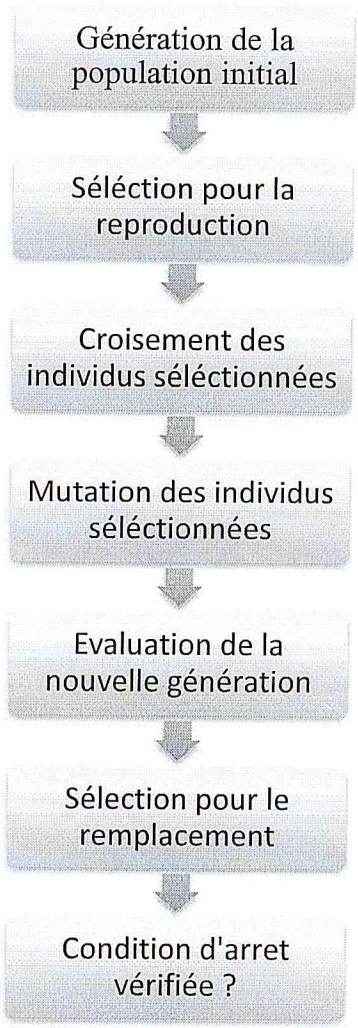


Figure 6 : Etapes de l'algorithme génétique

5.3.3 *Le codage*

Historiquement le codage utilisé par les algorithmes génétiques était le codage binaire [**19**]. Dans ce codage, chaque paramètre d'une solution est assimilé à un gène, chaque gène dispose du même alphabet binaire {0,1} et elle est représentée par un entier long de 32bits, les chromosomes sont représentés par des tableaux de gènes et les individus sont représentés par des tableaux de chromosomes.

Ce type de codage a pour intérêt de permettre de créer des opérateurs de croisement et de mutation simples, cependant il n'est pas toujours efficace car on utilise souvent la distance de Hamming comme mesure de la similarité entre deux élément de population, cette mesure compte les différences de bits de même rang de ces deux séquences. Mais, deux éléments voisins en termes de distance de Hamming ne codent pas nécessairement deux éléments proches dans l'espace de recherche. Cet inconvénient peut être évité en utilisant le codage de Gray qui considère que deux éléments sont voisins dans l'espace de recherche si seulement un bit diffère [**18**].

Le codage réels évitent ce genre de problèmes, ils sont utiles notamment dans le cas où l'on recherche le maximum d'une fonction réelle.

5.3.4 *Fonction de fitness*

C'est le cœur des algorithmes génétique, la fonction de fitness attribue à chaque individu une valeur dite *performance* ou *fitness*, elle reflet à quel point l'individu est utile et important dans la population, cette valeur est utilisé dans certains étape de l'algorithme génétique par exemple dans la sélection par tournois qu'on va aborder par la suite, ou l'individu qui possède la meilleur performance gagnera le tournois, aussi dans l'étape de remplacement pour connaitre quelle sont les enfant à remplacer dans la population initial.

Il n y a pas de méthode précise pour définir la fonction de fitness, cela diffère d'un problème à un autre, par exemple :

Soit une population P contenant des chaines de caractère d'une longueur de 11 caractères généré aléatoirement à partir de l'alphabet {A, B, C, ..., Z, + le caractère espace}

Exemple : Chaine 1 = « HYGEKNGEDCH » Chaine 2 = « HU_LONODRDL »

Notre objectif c'est d'obtenir la chaine cible « HELLO_WORLD ». On peut définir notre fonction de fitness de façon à ce qu'elle Il assigne simplement un point pour chaque caractère de la chaîne candidat qui correspond au caractère à la position correspondante dans la chaîne cible. L'algorithme de la fonction est le suivant.

```
Cible : la chaine cible « HELLO_WORLD »
Candidate : La chaine candidate
Algorithme
        Pour chaque chaine candidate faire
                Pour chaque position i faire
                        Si (candidate(i)= cible (i))
                                Performance ++
                        FinSi
                Fait
        Return performance
```

La valeur maximal de *performance* qu'on peut obtenir ici c'est 11 ce qui veut dire que tous les caractères de la chaine candidate corresponds à celles de la chaine cible.

La *fitness* ou la *performance* d'un individu peut être interpréter différemment, dans certains cas on préfère maximiser la valeur de fitness comme celle de l'exemple précèdent, dans d'autres on préfère la minimiser.

5.3.5 La sélection

C'est l'étape où l'on sélectionne certains individus à partir de la population initiale P pour être dupliquer dans une nouvelle population P' et vont servir de parents pour la prochaine génération. C'est une étape crucial de l'algorithme génétique car elle permet aux individus d'une population de survire, de se reproduire ou de mourir, la survie d'un individu est liée directement à son efficacité au sein de la population .

On trouve essentiellement quatre méthodes de sélection différentes :

- Sélection Roulette
- La méthode Élitiste
- Sélection par tournois
- Sélection universelle stochastique

5.3.5.1 Sélection roulette

Pour utiliser l'image de la roue ans cette méthode, chaque individu se voit attribué un secteur dont l'angle est proportionnel à son adaptation, sa performance, sa *fitness*. On fait tourner la roue et quand elle cesse de tourner on sélectionne l'individu correspondant au secteur désigner

par une sorte de *curseur*. Donc plus l'individu est adaptés au problème, plus l'angle de son secteur est grand alors plus de chance d'être sélectionnés.

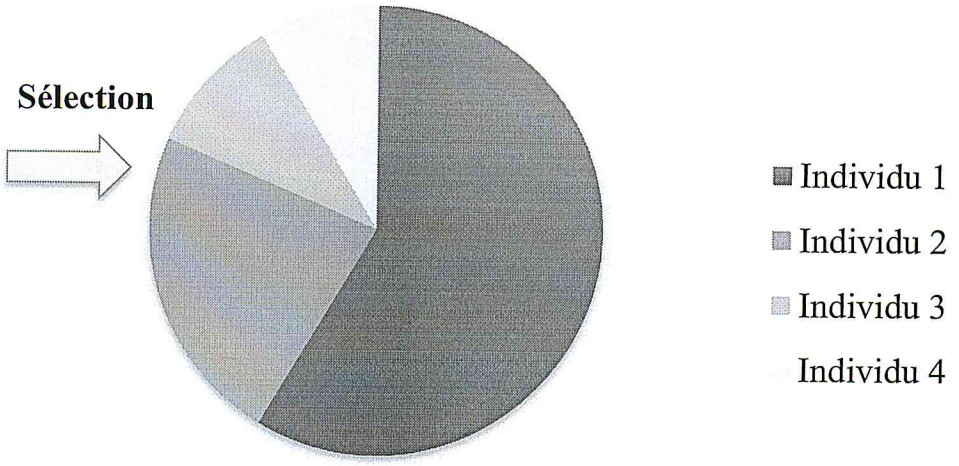


Figure 7: Sélection Roulette

Cette méthode est largement répondu, malgré tout elle possède pas mal d'inconvénients.

Il n'est pas possible que sur n sélection successives, la quasi-totalité ou bien pire la totalité des individus sélectionnés auront une fitness mauvaise, ce qui empêchera par la suite d'obtenir des individus de meilleur fitness, a l'inverse on peut arriver à une domination écrasante d'un individus de fitness « localement supérieur » au reste de la population, ceci affectera la diversité des individus dans les prochaines générations et l'évolution de l'algorithme se met donc à stagner et on restera bloqué sur un optimum local.

5.3.5.2 La méthode Élitiste

Cette méthode consiste à trier la population initiale de manière décroissante (selon la fitness des individus) et choisir les n meilleurs individus pour être sélectionner dans la nouvelle population. Il est bien claire que la diversité des individus sélectionnés et presque nulle, cette méthode soufre du même problème que celui de la méthode précédente, mais encore pire car la probabilité de stagner sur un optimum local converge de manière plus rapide et plus sure. Du moins le peu de diversité qu'il pourrait y avoir résultera du croisement et des mutations.

5.3.5.3 Sélection par tournois

Le principe de cette méthode est d'effectuer un tirage avec ou sans remise de n individus de la population initial, un combat est ensuite effectuer entre les n individus sélectionnés (n= 2, 4, 6..etc), celui qui possède la meilleure fitness remportera le combat et sera sélectionner dans la nouvelle population. Une valeur p est assigné à cette méthode, elle est comprise entre 0.5 et 1 et

détermine la probabilité pour que le meilleur individu remporte le combat, le fait d'augmenter ou de diminuer la valeur de p permet respectivement d'augmenter ou de diminuer la précision de la fonction.

Cette méthode est vue comme celle avec laquelle on obtient les résultats les plus satisfaisants car elle possède une variance très élevée.

5.3.6 Le croisement

Cet opérateur est inspiré du mécanisme qui permet, dans la nature, la production de chromosomes qui héritent partiellement des caractéristiques des parents.

Après l'étape de sélection présenté précédemment on se retrouve avec une population P' de taille $n/2$, l'opérateur de croisement nous permet de doubler ce nombre pour que la nouvelle génération soit complète.

À partir de la population P' on choisit au hasard deux individus qui représentent les parents pour les faire se reproduire. Les chromosomes (ensemble de paramètres) des parents sont copiés et recombiner de façon à former deux descendent possédant des caractéristiques issues des deux parents. Pour le faire, un ou plusieurs points de croisement sont déterminé au hasard, chaque chromosome se trouver alors réparti en plusieurs segments, puis chaque segment du premier parent est échangé avec son homologue du deuxième parent selon une probabilité p. On obtient alors deux fils pour chaque couple de parents.

L'opérateur de croisement favorise l'exploitation de l'espace de recherche, et il permet d'obtenir rapidement de nouveaux individus qui combinent les meilleurs caractéristiques de chaque parent et donc des individus plus adapté au problème, de même qu'on peut obtenir de nouvelles individus qui sont moins adapté au problème en combinant les mauvaise caractéristiques de chaque parent.

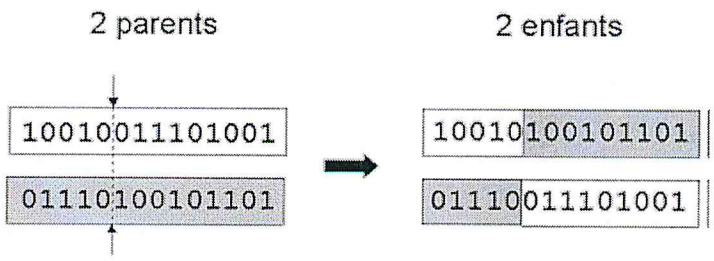


Figure 8 : Croisement parraport à un point unique

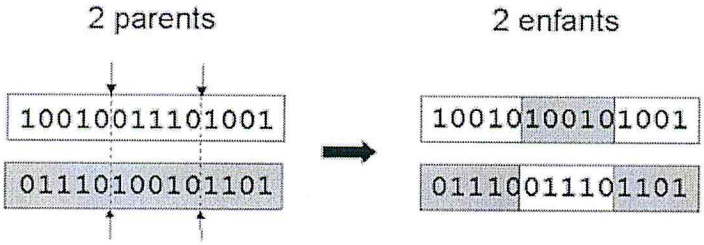


Figure 9 : Croisement multipoints

5.3.7 *La mutation*

La mutation c'est le changement de la valeur allélique d'un gène du chromosome (un bit) déterminer de manière aléatoire avec une probabilité P_m très faible, généralement comprise entre 0.01 et 0.001.

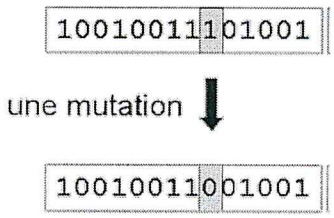


Figure 10: La mutation dans l'algorithme génétique

Cet opérateur jour le rôle d'un élément perturbateur, il introduit du bruit au sein de la population et garantie la diversité de la population, ce qui est primordial pour les algorithmes génétiques. Il permet de mieux exploiter l'espace de recherche et aussi de limiter le risque d'une convergence prématurée causée par exemple par les méthodes de sélection élitiste et Roulet Wheel, ou on se retrouve avec une population dont tous les individus sont identiques, ceci ne peut être résolu par les différentes méthodes de croisement, car l'échange d'informations entre deux éléments identiques est bien sur totalement sans conséquences, Du coup la mutation entrainant des inversions de bits de manière aléatoire et permet de réintroduire des différence entre les individus, bien sur la probabilité d'une mutation est très faible donc il est plus intelligent de ne pas utiliser de méthodes de sélection qui entrainent ce genre de problème.

5.3.8 *Le remplacement*

C'est l'étape ou les descendants obtenus dans la population P' par application successive des opérateurs de sélection, de croisement et de mutation sont réintroduit dans la population de leurs parents P. On trouve deux différente méthode de remplacement

- **Le remplacement stationnaire** : ou les enfants remplacent leurs parents sans tenir compte de leurs performance, soit de complètement c'est-à-dire que la totalité de la population P sera remplacer par P' ou partiellement c'est-à-dire une certaine proportion d'individus (30%, 40%, ...etc.) de P' sont choisi pour remplacer leurs parents.
- **Le remplacement élitiste** : dans ce cas on garde au moins l'individu possédant les meilleures performances d'une génération à la suivante. Un nouvel individu prend place ai sein de la population que s'il remplit le critère d'être plus performant que le moins performant des individus de la population précédente.

Le remplacement stationnaire engendre une population ayant une grande variation mais dans certains cas, un enfant ayant une faible performance peut remplacer un parent qui possède une meilleure performance, ce qui nous empêchera d'atteindre la meilleure solution. Quant au remplacement élitiste, il améliore les performances de l'algorithme mais présente aussi un désavantage en augmentant le taux de convergence prématuré.

5.4 Algorithmes à solution Hybride

L'hybridation est une tendance observée dans de nombreux travaux réalisés sur les métaheuristiques ces dix dernières années. Elle permet de tirer profit des avantages cumulés de différentes métaheuristiques cité précédemment.

Les métaheuristiques hybrides sont apparues en même temps que le paradigme lui-même, mais la plupart des chercheurs n'y accordaient que peu d'intérêt [**20**], Elles gagnent maintenant en popularité, car les meilleurs résultats trouvés pour plusieurs problèmes d'optimisation ont étés obtenus avec des algorithmes hybrides.

Selon Talbi [**20**] Les méthodes hybrides peuvent être devisées en deux classes :

- Hybridation de bas niveau .
- Hybridation de haut niveau.

On a une hybridation de bas niveau lorsqu'une fonction d'une métaheuristique est remplacée par une autre métaheuristique. On obtient une hybridation de haut niveau lorsque deux métaheuristiques sont hybridées sans que leur fonctionnement interne ne soit en relation .

Chacune des deux classes d'hybridation précédentes se subdivise en deux autres classes :

- À relais
- Co-évolutionnaire

Lorsque les métaheuristiques sont exécutées de façon séquentielle, l'une utilisant le résultat de la précédente comme entrée, on a une hybridation à relais, L'hybridation Co-évolutionnaire se fait lorsque des agents coopèrent en parallèle pour explorer l'espace de solutions.

Pour résumer, la combinaison de toutes les classes nommées précédemment donne quatre classes différentes :

- Hybridation de bas niveau à relais .
- Hybridation de bas niveau Co-évolutionnaire.
- Hybridation de haut niveau à relais.
- Hybridation de bas niveau Co-évolutionnaire.

5.4.1 *Algorithmes Mémétiques*

Les algorithmes mémétiques sont introduits pour la première fois par Moscato [**21**]. Fondamentalement, ils hybrident les algorithmes génétiques avec des méthodes de recherche locale. Pour cette raison, certain chercheurs les ont vus comme des algorithmes génétiques hybrides, ils ont aussi reçu la dénomination d'algorithme génétiques parallèles ou algorithmes de recherche local génétique.

Les algorithmes génétiques peuvent être une bonne solution pour résoudre des problèmes d'optimisation combinatoire. Cependant, un inconvénient d'un algorithme génétique et que les opérateurs standard de croisement et de mutation ne permettent pas d'intensifier suffisamment la recherche [**22**], L'opérateur de mutation apporte une légère modification à l'individu. Son rôle est de favoriser la diversification des individus alors que la sélection se charge de conserver les meilleurs. C'est pourquoi les algorithmes génétiques sont souvent hybridés avec des méthodes de recherche locale.

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

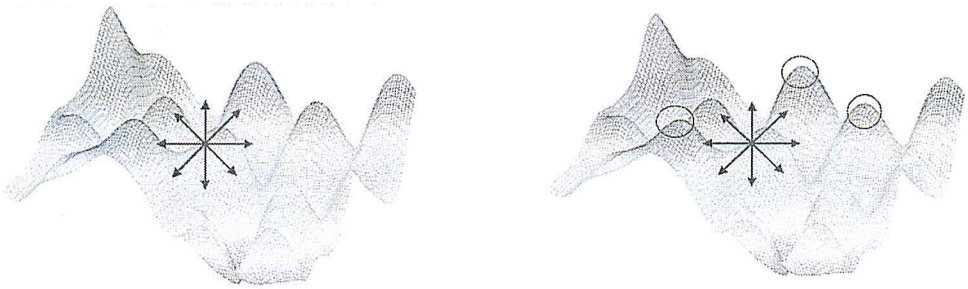


Figure 11: Application de la recherche locale dans un espace de solutions

5.4.2 *Principe des algorithmes mémétiques*

Les algorithmes mémétiques sont une hybridation entre les algorithmes de recherche locale est les algorithmes génétiques. Le principe générale est le même que pour les algorithmes génétiques mis à part l'ajout d'un opérateur de recherche locale qui remplace ou succède l'opérateur de mutation. La partie génétique de ces algorithmes peut être vue comme une forte diversification alors que la partie recherche locale correspondait à une forte intensification accompagné d'une faible diversification.

Les composants principaux qui jouent un rôle important dans l'algorithme mémétique sont principalement les composants de l'algorithme génétique5.3.1 , il vient s'ajouter à eux l'opérateur de recherche locale pour améliorer les nouveaux individus obtenu.

Il existe de multiples façons de concevoir un algorithme génétique, les méthodes de recherche locale sont aussi très nombreuses. L'hybridation de ces deux approches permet d'envisager un nombre considérable de combinaisons, que ce soit dans le choix des algorithmes a utilisé ou l'emplacement de la recherche locale au niveau de l'algorithme génétique. Tout cela dépend du problème à optimiser.

Dans ce qui suit nous allons présenter l'algorithme mémétique basique, on expliquera par la suite notre approche.

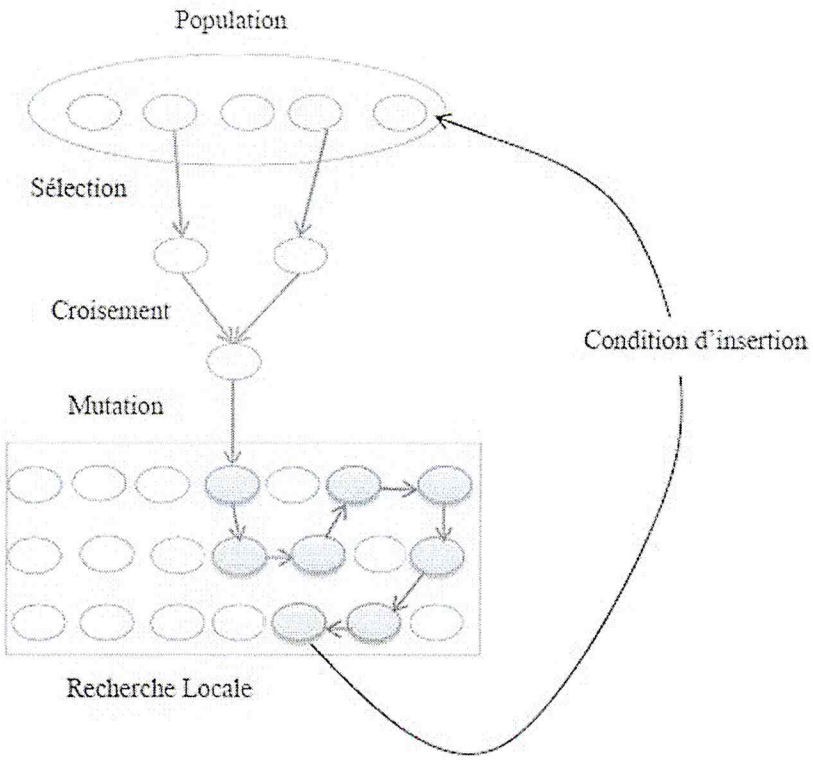


Figure 12 : Étapes de l'algorithme mémétiques

5.4.3 *Algorithme Mémétique Basique*

Initialisation : Générer une population (Pop) de solutions (S) initial de taille = n

Améliorer chaque solution (S) de (Pop) par la méthode recherche locale

Répéter

Sélectionner deux solutions S1 et S2 avec la technique de sélection

Croiser les deux parents S1 et S2 pour obtenir deux enfants E1 et E2

Pour chaque enfant E

Améliorer : E avec la méthode de recherche locale

Muter E

Si (E satisfait la condition de remplacement)

Remplacer une solution S de Pop par E

Fin Si

Fait

Jusqu'à (critère d'arrêt)

L'algorithme commence par la construction d'une population initiale d'individus de manière aléatoire, l'opérateur de sélection est ensuite appliqué pour choisir les parents. L'étape suivante consiste à appliquer l'opérateur de croisement sur les parents choisit, à cette étape la méthode de recherche locale est appliqué sur les deux enfants résultant et renvoyer le meilleur résultat, la mutation est ensuite appliqué sur ce résultat pour introduire de nouveau gène dans la population initial. À la fin une mise à jour est effectuée sur la population initiale qui consiste à remplacer le mauvais parent par le nouvel enfant.

Les algorithmes mémétiques ont permis de produire d'excellents voire les meilleurs résultats sur des problèmes bien connus. C'est le cas par exemple du problème du voyageur de commerce TSP , de la coloration de graphes, et de l'étiquetage de graphes .

Chapitre 3

Algorithme mémétique

Chapitre 3 : Algorithme mémétique

1 Introduction

Un IDS est un outil qui a pour vocation la surveillance d'un ou plusieurs réseaux de machine, Il permet de détecter des attaques ou des évènements suspects, pour réaliser sa tâche, l'IDS se base sur une base de données, elle contient différent types d'attaques, identiques ou proche de ceux qui sont détecter par l'IDS.

Une base de données est un ensemble de vecteurs de connexion « Instance » signé comme normal ou attaque, chacun de ces vecteurs contient 41 attributs, ils représentent les caractéristiques d'une connexion normal ou anormal.

La base de données connait des défauts comme la redondance des données et la non régularité de distribution d'attaques, ceci pousse l'IDS à générer des taux élevés de faux positifs et de faux négatif.

Dans cette partie nous allons présenter notre proposition qui permet d'améliorer cette base de données. Trois critères doivent alors être respectés :

- **La taille de la base de données :** Pour une détection rapide et efficace des attaques, la base de données doit contenir les données les plus significatifs et surtout éliminer ceux qui sont redondantes
- **Le taux de classification d'attaques :** il doit être supérieur ou égal à celui de la base de données original
- **Le taux de faux positifs :** ceci doit être inférieur à celui de la base de données original.

Pour réaliser cela nous avons opté pour un algorithme mémétique qui est une hybridation entre un algorithme génétique et une méthode de recherche locale, les algorithmes de ce type ont prouvé leurs efficacités dans plusieurs problèmes d'optimisation combinatoire. Comme le problème de tournées des véhicules. Ils profitent de la diversification de solution que peut fournir l'algorithme génétique et de la recherche raffinée sur l'ensemble de ces solutions que peut fournir les méthodes de recherche locale.

2 Algorithme mémétique pour la réduction de la taille de données d'apprentissage

Notre approche consiste à travailler sur la base de données qu'utilise l'IDS pour classer les différents paquets interceptés, en améliorant cette base de données. l'IDS aura tendance à minimiser les erreurs et le temps de classification.

L'idée principale consiste à prendre la base de données initiale et essayer d'en extraire une nouvelle base ne contenant que les instances les plus significatives. Pour réaliser cela la base de données initial sera deviser en deux sous-ensembles, le 1er on va le nommer « Test ». Il contiendra des instances choisis aléatoirement et sans remise de la base de données initial, le 2èm qu'on va le nommer « Train » contiendra le reste des instances.

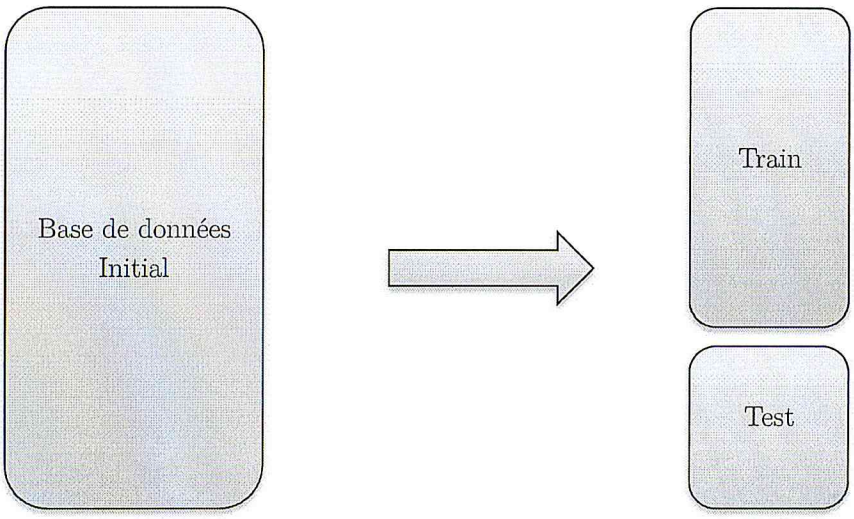


Figure 13: Répartition de la base de données initial en deux sous-ensembles Train et Test

Nous allons essayer de réduire l'ensemble « Train » en utilisant l'apprentissage supervisé 3.1 ci-dessus Pour faire, on génère aléatoirement différentes combinaisons d'instances de l'ensemble « Train », ces combinaisons représentant les solutions initiales de l'algorithme mémétiques. En d'autres termes ils constituent la population initiale de l'algorithme. Chaque combinaison sera ensuit évaluée comme montre la figure suivante

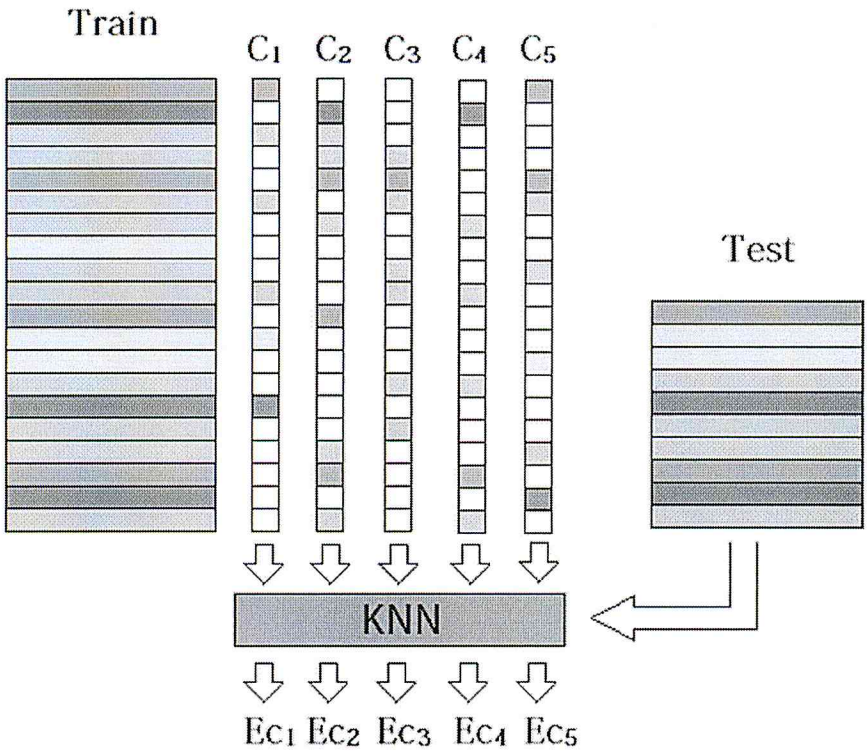


Figure 14 : Évaluation des combinaisons d'instances

L'évaluation de la combinaison se fait par la classification des instances de l'ensemble « Test » en utilisant l'algorithme K – Plus proches voisins1 ci-dessus 3.3 3.3 ci-dessus, ceci nous permettre de connaitre quels sont les meilleurs combinaisons. Ensuite les opérateurs de sélection, de croisement et de mutations sont appliqués sur ces combinaisons pour entrainer plus de diversité à la population.

Pour mieux raffiner les solutions obtenues après application des opérateurs, on effectue une recherche locale sur les meilleures combinaisons obtenues, cette recherche va nous permettre d'éliminer quelques instances bruyantes, et du coup diminuer le taux des faux positifs.

Au cœur d'un algorithme mémétique on trouve un algorithme génétique amélioré avec une méthode de recherche local, donc évidement il doit être établi en prenant en considération les paramètres suivants :

- Un codage pour les individus de la population
- Une population initiale
- Une fonction de fitness pour évaluer les individus
- Un opérateur de sélection
- Un opérateur de croisement

- Un opérateur de mutation
- Un opérateur de remplacement

En général, L'apport de la recherche local consiste à ajouter une méthode de recherche dans l'algorithme génétique ou remplacer l'opérateur de mutation par cette dernière. De plus, pour obtenir de bons résultats il faut choisir quelle méthode de sélection, de croisement, de remplacement et de mutation utiliser, car le résultat de l'algorithme est fortement lié à ces opérateurs.

Dans ce qui suit nous allons détailler tout d'abord le codage de chaque individu, le format de la population initial et la fonction de fitness, on donnera une forme initial de notre algorithme, on entamera par la suite le paramétrage qui est sous forme d'une série de tests, au quel, selon les résultats obtenu, on pourra choisir quelle méthode il faudra utiliser pour chaque opérateur dans l'algorithme.

2.1 Codage des individus

Le codage des individus diffère d'un problème à un autre, pour notre problème nous avons décidé d'utiliser le codage suivant.

L'ensemble « Train » que nous avons extrait à partir de la base de données initial est composé de n instances, notre objectif est de réduire la taille de cet ensemble et d'éliminer les instances bruyantes, mais le problème qui se pose est comment choisir les instances à éliminer ?

Puisque nous n'avons aucune information à propos de l'importance d'une instance, que cette instance apporte une amélioration ou non, qu'elle peut générer un faux positif ou pas. La seule solution est de générer des combinaisons aléatoires d'instances, cette combinaison est représentée sous forme d'un vecteur, sa taille est identique à la taille de l'ensemble « Train », il est composé d'une suite de 1 et de 0. Pour obtenir un ensemble réduit de « Train » (une solution) .On ne prend de l'ensemble « Train » que les instances qui correspondent aux positions des 1 dans l'individu.

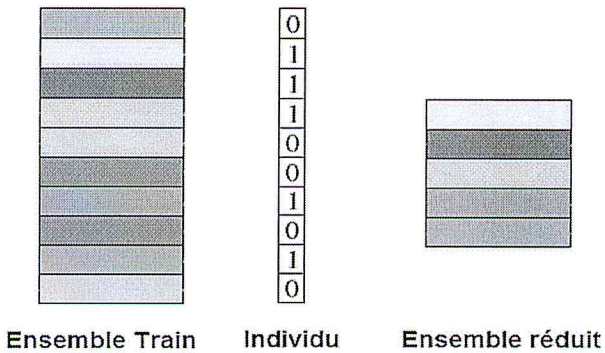


Figure 15 : Représentation du codage d'un individu

2.2 La population initial

Comme chaque algorithme génétique, La population initiale doit être variantes pour assurer la diversité des solutions. La taille de la population joue aussi un rôle très important car, plus la population est grande plus le nombre d'individus est grand et plus la population sera variée.

Notre population initiale est sous forme d'une matrice, chaque ligne de la matrice représente un individu. Le nombre de lignes de la matrice représente la taille de notre population.

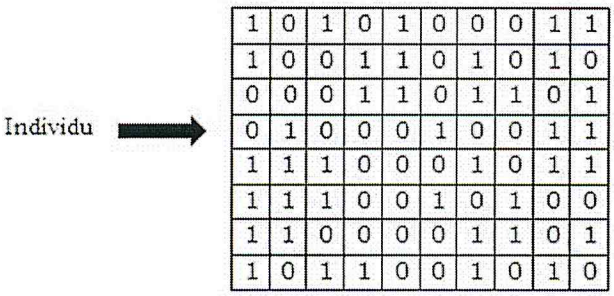


Figure 16 : Format de la population initial

2.3 La fonction de fitness

La fonction de fitness joue le rôle le plus important dans l'algorithme, elle est appliqué sur chaque individus, son résultat représente la valeur, le poids, l'importance de l'individu dans la population.

Chaque instance qui est représentée par la présence de 1 dans l'individu est composée essentiellement de deux parties :

- **Les attributs qui définissent l'instance :** ce sont les 41 premiers attributs qui sont présentent dans les instances de l'ensemble « Train »
- **La classe de l'instance :** c'est l'attribut numéro 42 de l'instance, il indique le types de l'instance (Normal ou Attaque)

```
0,tcp,http,SF,411,247,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,33,0,0.06,0,0,1,0,0.18,12,255,1,0,0.08,0.01,0,0,0,0,normal
0,icmp,ecr_i,SF,520,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0,0,0,0,1,0,0,255,255,1,0,1,0,0,0,0,attack
```

Figure 17: Format des instances présentes dans l'ensemble Train

La fonction de fitness est basée sur l'algorithme des k – Plus proche voisins (KNN) qui a était présenté dans 3.3.2 ci-dessus Son rôle donc est de **classer** chacune des instances de l'ensemble « Test » avec ceux qui sont présentent dans l'individu.

Il y a deux paramètres importants à définir dans l'algorithme :

- La fonction de distance à choisir
- La valeur de k

T_i : une instance de « Test »

N_j : une instance de l'individu

d_{ij} : la distance euclidienne entre T_i et N_j

n : la taille de l'ensemble « Train »

m : la taille de l'ensemble « Test »

Pour T_1 calculer (d_{11}, d_{12}, d_{13}, ... d_{1n}) et retourner les k plus petites distances

Pour T_2 calculer (d_{21}, d_{22}, d_{23}, ... d_{2n}) et retourner les k plus petites distances

.....

Pour T_n calculer (d_{n1}, d_{n2}, d_{n3}, ... d_{mn}) et retourner les k plus petites distances

Figure 18 : Calcul des k plus petites distances

La fonction de distance la plus utilisé pour définir les voisins c'est la distance euclidienne, elle considère chaque prototype comme un vecteur de nombre réelle de taille fixe. Dans notre cas nous allons procéder comme suit :

La distance euclidienne sera calculé en utilisant les 41 attributs de chaque instance comme vecteurs de nombres réelle, plus la valeur de la distance est petite entre les deux instances, plus ils ont considéré comme plus proche voisins.

La valeur de k détermine le nombre de voisins à rechercher. Il est très fréquent de choisir une valeur très petite et impaire de k, principalement (1, 3 ou 5). À la fin, on choisit les k plus petites distances donc les k plus proche voisins.

Pour **classer** l'instance T_i L'algorithme choisit la **classe d'instance** la plus fréquente entre ces plus proches voisins (d_{ij}). Cette dernière sera alors attribuée à T_i

Le 1-NN est la base de K-NN, il classifie tout simplement les instances de l'ensemble « Test » comme leur meilleur plus proche voisin, on choisissant le $k = 1$ nous allons essayer de trouver la classe la plus proche possible.

On considère une classification comme correcte si la classe du plus proches voisin est la même que celle de l'instance à classer.

Pour mesurer la fitness de l'individu, nous avons besoins des paramètres suivants :

- Le taux de classification : il représente le pourcentage des instances de l'ensemble « Test » classé correctement.
- Le taux de réduction : représente le pourcentage de réduction de l'ensemble « Train » .
- Deux coefficient 0.9 et 0.1 sont utilisées pour équilibrer l'équation, nous avons choisis de donner plus d'importance au taux de classification qu'au taux de réduction de l'individu, car parmi les contraintes que l'ensemble réduit doit satisfaire est que le taux de classification reste le même ou s'améliore par-rapport au taux de l'ensemble initial. Un individu qui possède un taux de classification élevé est plus privilégier que celui qui possède un très haut taux de réduction avec une mauvaise classification.

tc : taux de classification

tr : taux de réduction

f : la fitness de l'individu

n : la taille de l'ensemble test

m : la taille de l'ensemble train

cmpt : une compteur incrémenté à chaque bonne classification

nbins : représente le nombre d'instances présent dans l'individu (nombre de 1)

$$tc = (cmpt \times 100) \div n$$

$$tr = (nbins \times 100) \div m$$

$$f = tc \times 0.9 + tr \times 0.1$$

Figure 19 : Calcul de la fitness de l'individu

Algorithme Classer (individu)

Début

Pour chaque instance T_i de l'ensemble test

$d_{final} \leftarrow null$

$N_{final} \leftarrow null$

Pour chaque instance N_j de l'individu
d_{ij} = *distance euclidienne (T_i et N_j)*

Si ($d_{ij} < d_{final}$ *ou* $d_{final} = null$)

$d_{final} \leftarrow d_{ij}$

$N_{final} \leftarrow N_j$

Fin si

Fait

Si (classe de N_{final} = classe de T_i)

cmpt ++

Fin *Fin si*

Figure 20 : Algorithme de la méthode Classer

Algorithme Fitness (individu)

Début

cmpt = Classer (individu)

tc = (cmpt × 100) ÷ n

tr = (nbins×100) ÷ m

f = tc × 0.9 + tr × 0.1

Return f

Fin

Figure 21 : Algorithme de la fonction de fitness

À cette étape, le codage de l'individu, la population initial ainsi que la fonction de fitness sont bien définit, on peut donner une forme à notre algorithme, il prendra comme paramètres d'entrée les deux ensembles « Train » et « Test », il s'exécutera pendant un nombre défini de génération et il donnera comme résultat l'ensemble réduit « RTrain ». Ceci ne sera pas considérer comme algorithme final, il sera de plus en plus détaillé par la suite.

Algorithme Mémétique-1NN (Ensemble Train, Ensemble Test)

Tableau : FitnessVect

Debut

Générer une population initiale de manière aléatoire

Pour chaque individu D_i de la population initial

Stocker Fitness (D_i) dans fitnessVect

Fait

Pour chaque génération

Appliquer l'opérateur de sélection

Appliquer l'opérateur de croisement sur les éléments sélectionnés

Appliquer l'opérateur de mutation

Remplacer dans la population initiale

Fait

Retourner le meilleur individu de la matrice de population

Fin

Remarque : Le meilleur individu retourné sera considérer comme l'ensemble réduit « Rtrain »

Figure 22 : Algorithme Mémétique - 1NN non paramétré

2.4 Paramétrage de l'algorithme

Dans un algorithme mémétique comme dans l'algorithme génétique, la bonne combinaison des opérateurs est très importante, pour cela nous avons procéder en une série de tests.

Six opérateurs doivent être fixés :

- Opérateur de sélection : il faudrait choisir parmi les différentes méthodes présentées dans 5.3.5 ci-dessus Élitiste, Tournois, Sélection Roulette
- Opérateur de croisement : il faudrait choisir parmi la méthode de croisement multipoints ou en point unique 5.3.6 ci-dessus.
- Opérateur de remplacement : il faudrait choisir parmi les différentes méthodes de remplacement 5.3.8 ci-dessus.
- Le nombre de génération pendant lequel l'algorithme va tourner.
- La taille de la population initiale.
- Opérateur de Mutation : la probabilité de mutation doit être très petite. **Cet opérateur va être remplacé par une méthode de recherche locale .**

À chaque étape nous allons varier un seul opérateur, et on observera l'évolution de l'algorithme.

2.4.1 Opérateur de sélection

Pour commencer nous allons varier l'opérateur de sélection et fixer les autres, nous avons choisi de commencer par la sélection car le reste des opérateurs vont être appliquées sur le résultat de cette dernière, c'est pour ça une bonne méthode de sélection doit être choisit avant tout.

Les méthodes de sélection testées sont :

- Sélection élitiste
- Sélection roulette
- Sélection par tournois de 2 (avec / sans) remise
- Sélection par tournois de 4 (avec / sans) remise
- Sélection par tournois de 8 (avec / sans) remise
- Sélection par tournois de 16 (avec / sans) remise

La remise dans une sélection par tournois veut dire si les éléments déjà sélectionnés vont être remis dans la population initiale pour être sélectionné la prochaine fois

Dans notre cas, la méthode de sélection va choisir des individus à partir de la matrice de population 5.3.2 ci-dessus, le nombre d'individus varie selon la méthode utilisée.

Pour les tests, les paramètres de l'algorithme ont était fixé comme suit

- Méthode de croisement : Un seul point de croisement
- Méthode de remplacement : remplacement élitiste
- Taille de la population : ¼ la taille de l'ensemble « Train »
- Nombre de génération : 10
- Probabilité de mutation : 0.1

Les résultats obtenus sont résumé dans le tableau suivant :

Tc : Taux de classification

Tr : Taux de réduction

f : Fitness de l'individu

Méthode de sélection	TC %	TR %	La fitness (f) de l'individu
Sélection roulette	91.0	85.0	90.7
Sélection élitiste	92.0	82.0	91.0
Tournois (2) avec remise	93.0	80.0	91.7
Tournois (2) sans remise	96.0	83.0	94.7
Tournois (4) avec remise	94.0	77.0	92.3
Tournois (4) sans remise	94.0	84.0	93.0
Tournois (8) avec remise	95.0	83.0	93.8
Tournois (8) sans remise	93.0	81.0	91.8
Tournois (16) avec remise	93.0	84.0	92.1
Tournois (16) sans remise	94.0	76.0	92.2

Tableau 1: Résultat de paramétrage de la méthode de sélection

Les données présentent dans le tableau ci-dessus représente le meilleur résultat dans chaque test, le meilleur individu est celui qui possède la plus grande valeur de f.

On remarque tout de suite que le meilleur résultat est retourné par la méthode de sélection par tournois de 2 sans remise. Ce qui est évident car elle permet de faire introduire toujours de meilleurs individus dans la population initial, les autres méthodes de sélection par tournois retournent aussi de bon résultat mais plus le nombre de candidats dans la sélection augmente, plus on remarque une détérioration dans la fitness.

D'après ces résultats, notre opérateur de sélection sera désormais la sélection par tournois de deux sans remise.

2.4.2 Opérateur de croisement

Une fois la méthode de sélection fixée, on peut maintenant entamer le choix de la méthode de croisement. Celle-ci aura pour rôle d'apporter plus de diversification à la population, ce qui fait la force des algorithmes génétiques.

Deux principales méthodes de croisement sont utilisées

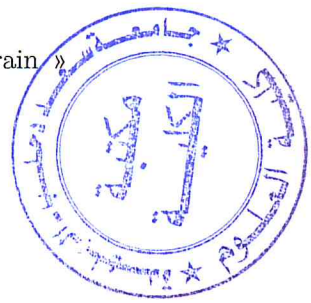
- Croisement en un point unique
- Croisement en plusieurs points

Chacune des deux méthodes a était détaillée dans 5.3.6 ci-dessus

Comme dans le test précédents nous allons variée la méthode de croisement et fixé les autres paramètres comme suit :

- Méthode de sélection : Tournois (2) sans remise
- Méthode de remplacement : remplacement élitiste
- Taille de la population : ¼ la taille de l'ensemble « Train »
- Nombre de génération : 10
- Probabilité de mutation : 0.1

Les résultats obtenus sont résumé dans le tableau suivant :



Méthode de croisement	TC %	TR %	La fitness (f) de l'individu
Un seul point	95.0	87.0	94.2
Multipoints	95.0	85.0	94.0

Tableau 2: Résultat de paramétrage de la méthode de croisement

On remarque que la méthode de croisement en un seul point est légèrement meilleure que celles de plusieurs points, nous allons opter alors pour la méthode de croisement en un seul point car c'est la plus facile à la plus rapide.

2.4.3 Opérateur de remplacement

Pour la méthode de remplacement nous allons procéder de deux manières :

- **Remplacement Élitiste :** À chaque fois que deux individus parents sont croisés, on obtient deux individus enfant, ces deux individus sont évaluer avec la fonction de fitness, puis pour chaque enfant, si ça fitness est supérieur à celle du pire individu dans la matrice de population (celui qui possède la fitness la plus faible), alors il le remplacera.
- **Un Remplacement par génération :** Dans chaque génération, on évalue les tous les individus enfants et on garde que les deux meilleurs, ces deux dernières remplacerons les deux pire parents dans la matrice de population

La fonction de remplacement affecte de manière très important l'évolution de la population initial, plus la génération évolue, plus elle sera rempli de bon individu, plus la diversité diminue. Une bonne méthode de remplacement assure un équilibre entre l'évolution et la diversité de la population.

Pour notre algorithme nous avons testé les deux méthodes citées en dessus, l'algorithme prendra les paramètres suivants :

- Méthode de sélection : Tournois (2) sans remise
- Méthode de croisement : Un seul point de croisement
- Taille de la population : ¼ la taille de l'ensemble « Train »
- Nombre de génération : 500
- Probabilité de mutation : 0.1

Les résultats sont présentés dans le tableau ci-dessous :

Méthode de remplacement	TC %	TR %	La fitness (f) de l'individu	Génération
Remplacement élitiste	91.0	86.0	90.5	1
Remplacement élitiste	95.0	84.0	93.9	20
Remplacement élitiste	95.0	91.0	94.6	50
Remplacement élitiste	95.0	91.0	94.6	100
Remplacement élitiste	95.0	91.0	94.6	200
Remplacement élitiste	95.0	91.0	94.6	500
Un remplacement par génération	90.0	86.0	89.0	1
Un remplacement par génération	92.0	91.0	91.9	20
Un remplacement par génération	92.0	91.0	91.9	50
Un remplacement par génération	93.0	91.0	92.8	100
Un remplacement par génération	95.0	90.0	94.5	300
Un remplacement par génération	95.0	90.0	94.5	500

Tableau 3: Résultat de paramétrage pour la méthode de remplacement

D'après le tableau on remarque qu'avec la méthode de remplacement élitiste on atteint un taux de classification de 95.0% avec une fitness de 94.6 dans 50 générations, ceci s'explique par le nombre important de remplacement dans une seule génération alors que la méthode d'un remplacement par génération prends beaucoup plus de temps pour atteindre un taux de 95.0% est ne réalise aucune amélioration durant 200 générations.

Notre choix est alors bien clair, la méthode de remplacement qui sera utilisée est la méthode de remplacement élitiste.

2.4.4 *Nombre de générations*

Le nombre de génération détermine le temps pendant lequel l'algorithme va tourner, durant ces génération la population va évoluer, mais comme il est connu chez les algorithmes génétiques, à un certain moment l'algorithme stagnes sur une valeur, et l'évolution de la population à partir de cette étape sera nulle ou d'une valeur négligeable.

Pour résoudre cette contrainte nous avons décidé de faire tourner l'algorithme durant différent nombre de générations, est en déduire le nombre de générations optimal, ceci nous permettra d'assurer la bonne évolution de la population et de minimiser le temps d'exécution.

Le test ont était effectuer avec 5, 10, 50, 100, 500 générations

L'algorithme prendra les paramètres suivants :

- Méthode de sélection : Tournois (2) sans remise
- Méthode de croisement : Un seul point de croisement
- Méthode de remplacement : remplacement élitiste
- Taille de la population : ¼ la taille de l'ensemble « Train »
- Probabilité de mutation : 0.1

Les résultats sont présentés dans le tableau ci-dessous :

Nombre de générations	TC %	TR %	La fitness f de l'individu	Point de stagnation
5	93.0	90.0	92.0	Aucun
10	94.0	88.0	93.4	Aucun
50	95.0	91.0	94.6	Génération N° 42
100	95.0	91.0	94.6	Génération N° 45
500	95.0	91.0	94.6	Génération N° 46

Tableau 4: Résultat de paramétrage pour le nombre de génération

Dans ce test nous nous sommes basée sur l'évolution du taux de classification, car la fonction de fitness varie selon le taux de réduction et le taux de classification, elle ne nous permettra pas d'avoir un bon point de repère puisque le taux de réduction n'est pas une priorité. Un individu avec un taux de classification élevé et un nombre d'instances assez élevée et privilégier qu'un individu qui contient peu d'instance mais un taux de classification médiocre.

Le point de stagnation représente la génération à partir de la quel le taux de classification n'a pas était améliorer.

On remarque que l'algorithme stagne entre les générations 42 et 46 avec un taux de classification de 95%, un taux de réduction de 91%, on peut conclure que le nombre de génération idéal sera 50 générations.

2.4.5 *Taille de la population initial*

La taille de la population représente le nombre d'individus que prend l'algorithme au départ, plus le nombre est grand, plus la diversité est bonne, une très grande population affecte le temps d'exécution de l'algorithme car le nombre de fois que les opérateurs de croisement, sélection et remplacement sont appliquées augmente.

Nous avons présenté précédemment notre population, elle est sous la forme d'une matrice ou chaque ligne représente un individu, dans tous les tests précédents le nombre de ligne a été fixé à ¼ de la taille de l'ensemble « Train ». Maintenant nous allons le varier le et nous observerons le comportement de l'algorithme.

Le nombre de lignes va prendre les valeurs suivantes :

- ¼ la taille du fichier « Train »
- ½ la taille du fichier « Train »
- ¾ la taille du fichier « Train »
- La même taille que celle du fichier « Train »
- 2 fois la taille du fichier « Train »

L'algorithme prendra les paramètres suivants :

- Méthode de sélection : Tournois (2) sans remise
- Méthode de croisement : Un seul point de croisement
- Méthode de remplacement : remplacement élitiste
- Nombre de génération : 50
- Probabilité de mutation : 0.1

Taille de population	TC %	TR %	La fitness (f) de l'individu
¼ la taille de « Train »	96	91	95.5
½ la taille de « Train »	96	94	95 .8
¾ la taille de « Train »	97	91	96.4
Taille de « Train »	98	92	97.4
2 fois la taille de « Train »	97	93	96.6

Tableau 5: Résultats de paramétrage pour la taille de population

D'après le tableau on remarque que plus la taille s'agrandit, plus la fitness du meilleur individu s'améliore, elle démarre de 95.5 et atteint 97.4 comme valeur maximal, ainsi le taux de classification qui commence à 96% et atteint 98%, ceci est dû au taux de diversité que comporte la population à chaque fois que sa taille s'agrandit, la taille idéal de la population sera alors la même taille que celle de l'ensemble « Train » comme indique le tableau.

2.4.6 La recherche locale

À cette étape notre algorithme à bien était paramétré, la méthode de sélection, de croisement, de remplacement ainsi que le nombre de génération et la taille de la population ont bien était fixé. On peut l'écrire sous le format suivant (les changements apportés à l'algorithme sont écrit en gras)

Algorithme Mémétique-1NN (Ensemble Train, Ensemble Test)

n : taille de l'ensemble Train

Début

Générer une population initiale de ***n individu*** *de manière aléatoire*

Pour chaque individu N_i de la population initial

Stocker Fitness (D_i) dans fitnessVect

Fait

Pour ***50*** *générations*

Tournois (Population initial, 2, sans remise)

Croisement (Parent 1, Parent 2, un seul point)

Appliquer l'opérateur de mutation

Remplacer (Population initial, enfant 1, enfant 2)

Fait

Retourner le meilleur individu de la matrice de population

Fin

Figure 23 : Algorithme mémétique 1-NN Paramétré

Parmi les problèmes liés à la base de données qu'utilisent les IDS c'est le taux élevé des instances en double et des instances bruits qui poussent l'IDS à générer des faux *fn* négatif et des faux positif *fp*. Ce dernier est généré quand une instance de type normal est classée comme une attaque (rep faux négatif). La présence des instances bruits en grand nombre oblige le classificateur à les prendre en considération et du coup la classification sera alors falsifier. Pour une meilleure classification, ces instances bruits doivent être éliminées de la base de données.

La différence entre un algorithme génétique et un algorithme mémétique c'est la méthode de recherche locale, cette recherche permet d'explorer des solutions voisines proches du résultat fourni par l'opérateur de croisement de l'algorithme génétique dans un espace plus étroit.

La mutation dans l'algorithme génétique consiste à muter un seul gène dans les enfants résultants de chaque opération de croisement, avec une probabilité très petite.

La méthode de recherche locale que nous allons implémenter va **remplacer l'opérateur de mutation**, elle s'appliquera toujours sur les enfants résultant de chaque opération de croisement. Mais pour chaque enfant elle va essayer de muter un nombre aléatoire de gènes actifs (qui possède la valeur 1), cette mutation aura pour effet d'essayer d'éliminer les instances bruit encore restant dans l'ensemble réduit « RTrain ».

Une nouvelle fonction MemeticFitness sera ajouté, elle nous permettra de calculer f_m cette valeur représente la fitness de l'individu en prenant en considération deux paramètres, le taux de classification tc et le pourcentage du faux positif fp (contrairement à f qui prenait tc et tr comme paramètres).

Soit :

nbrNormal : nombre d'instances de class normal classées comme attaque

TestNormal : nombres d'instances de type normal présents dans le fichier test

$$fp = (nbrNormal \div TestNormal) \times 100$$

$$tc = (cmpt \times 100) \div n$$

$$f_m = tc \times 0.8 + fp \times 0.2$$

Figure 24 : Calcule de *fitness*

Chaque enfant résultant d'un croisement sera tout d'abord évalué avec MemeticFitness, ensuit la recherche local s'appliquera sur ce dernier (mutation des gènes), puis il sera réévaluer avec MemeticFitness, cette procédure sera répété tant que la valeur de f_m reste la même ou n'a pas été améliorée par rapport à sa valeur initial, sinon on s'arrête.

```
Algorithme
MemeticFitness(individu)

Début

        cmpt = Classer (individu)

        tc = (cmpt × 100) ÷ n

        fm = tc × 0.8 + fp × 0.2

        Return fm

Fin
```

Figure 25 : Algorithme de MemeticFitness

La probabilité de mutation dans la recherche locale est très petite, parce que les individus sur les quels la recherche locale s'applique comportent déjà peu d'instances, si la probabilité est trop élevée, on risque de perdre les instances significative.

La recherche locale conclu le paramétrage de l'algorithme, tout est bien définie. On peut maintenant écrire l'algorithme final en remplaçant l'opérateur de mutation par la méthode de recherche locale.

```
Algorithme RechLocale ( individu )
nbrins : nombre d'instance de l'individu (nombre de 1)
Saveindividu , Copyindividu
Début
        individuCopy = individu
        fm1 = MemeticFitness (individuCopy)
        Boolean = True
        Tant que (Boolean)
                random1 = nombre aléatoire entre 1 et nbrins //nombre de mutation à
                                                              effectuer
                pour i←0, i < random1, pas = 1
                        random2 = nombre aléatoire entre 1 et nbrins // indice de mutation
                        Mutation (individuCopy[random2 ] ,0 .05) //changer la valeur de 1 à 0
                                                              avec une probabilité de 0.05
                Fait
                fm2 = MemeticFitness (individuCopy)
                Si ( fm2 >= fm1)
                        fm1 ← fm2
                        Saveindividu ← individuCopy
                Sinon
                        Boolean ← False
                Fin si
        Fait
        Return individuCopy
Fin
```

Figure 26 : Algorithme de RechercheLocale

Algorithme Mémétique-1NN (Ensemble Train, Ensemble Test)

n : taille de l'ensemble Train

FitnessVect

Début

Générer une population initiale de n individu de manière aléatoire

Pour chaque individu N_i de la population initial

Stocker Fitness (N_i) dans FitnessVect

Fait

Pour 50 générations

Tournois (Population initial, 2, sans remise)

Croisement (Parent 1, Parent 2, un seul point)

Pour chaque enfant

enfant = RechLocale (enfant)

Remplacer (MatricePopulation, enfant)

Fait

Retourner le meilleur individu de la population final

Fin

Figure 27 : Algorithme Mémétique 1-NN final

3 Conclusion

Dans ce chapitre nous avons présenté les différentes étapes que nous avons passé pour écrire notre algorithme, chaque étape est important c'est pour ça que nous avons effectué des séries de tests pour choisir la meilleure combinaison d'opérateur à utiliser dans l'algorithme.

À la fin nous avons introduit une nouvelle méthode recherche local, elle est adaptée au problème que nous essayons de résoudre, ça fonction est d'essayer d'éliminer les instances bruyants de l'ensemble réduit pour diminuer le taux du faux positif est améliorer le taux de classification.

Reste maintenant à tester l'efficacité de l'algorithme, et l'amélioration que peut porter sur la base de données qu'utilisent les IDS pour la classification des attaques .

Chapitre 4

Implémentation et résultats

Chapitre 4 : Implémentation et résultats

1 Introduction

Les IDS ont pour but de de classifier les trafics d'un réseau, cette classification nécessite une base de données d'apprentissage, la performance de l'IDS dépend fortement de cette dernière, l'amélioration de la base d'apprentissage assure la diminution du taux de faux positif (i.e. les comportements normaux classées comme intrusions) et du taux de faux négatifs (i.e. les intrusions classées comme comportement normaux). Ainsi la diminution de temps de détection d'attaque surtout si elle se produit en temps réel.

Cette expérimentation a pour but de montrer la nécessité de la réduction de la taille de la base de données d'apprentissage, pour une meilleure classification, ceci en utilisant les algorithmes mémétiques.

L'expérimentation sera effectuée à l'aide du classificateur KNN. Pour valider l'algorithme. Nous avons travaillé avec les données de la base KDD'99, ces données permettent aux algorithmes de classification d'être appliqué sur des données non erronées.

En première partie, nous présentons l'environnement de travail, les outils de développement utilisées, la présentation de la base KDD'99 et ainsi l'application que nous avons développée pour réduire cette dernière. Nous nous attarderons sur les éléments entourant l'évaluation des performances du classificateur. Finalement nous expliquerons la partie expérimentale de cette validation avec une analyse des résultats et une conclusion générale.

2 Environment de travail

Le choix du langage de programmation représente une étape très importante dans la réalisation de n'importe quelle application. C'est dans cette étape qu'on fait la correspondance entre les solutions que le langage nous offre et les résultats souhaités.

Pour implémenter notre application de réduction de la base de données, nous avons utilisé :

- **Java (Version 1.7.0.21- Dolphin) :** Java est un langage centré complètement sur les objets et fournis un ensemble prédéfinie de classes facilitant la manipulation des entrées-sorties, de la programmation réseaux et de l'interface utilisateur. Cette dernière version est disponible depuis le 7juillet 2011, elle contient 8000 classes et interfaces [**23**], son nom de code est *Dolphin*. Il s'agit de la première version sous licence GPL. Cette version n'est pas supportée par Mac OSX v10.6, certains Api requises ayant été ajoutées dans Mac OS

X 10.7.3, qu'Apple n'as pas prévu de rendre disponibles dans les anciennes versions de Mac OS [**24**]. Nous avons utilisé *Eclipse IDE (Integrated Developement Enviroment) for java developers, Version : Juno Release* comme outil de développement, On peut y écrire, compiler et exécuter les programmes, dont le but est de fournir une plate-forme modulaire, et aussi un outil libre permettant potentiellement de créer des projets de développement mettant on œuvre n'importe quel langage de programmation. Cette version peut être téléchargée gratuitement à partir du lien suivant : http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junosr1

- **La librairie Jpcap :** Jpcap est une bibliothèque open source composée d'un ensemble de classes java pour capturer et envoyer les paquets du réseau, elle est basée sur Libpcap/Winpcap. Elle facilite :
  - La capture de paquets bruts directement du réseau.
  - Sauvegarde les paquets capturés dans un fichier, et lire les paquets capturés à partir d'un fichier.
  - Identifier automatiquement les types de paquets et produits des objets correspondants à java (Pour l'Ethernet, les paquets IPv4, IPv6, TCP, UDP et ICMPv4).
  - Filtre les paquets selon des règles personnalisées par l'utilisation avant de les expédier à l'application.

  La librairie Jpcap doit être installée sur les deux machines, afin de capturer l'ensemble des paquets réseau (sonde et manager).
  Les deux bibliothèques peuvent être téléchargées gratuitement à partir des liens suivant :
  Winpcap : http://www.winpcap.org/install/default.htm
  Jpcap : http://sourceforge.net/projects/jpcap/
- **Weka (Version 3.6.2) :** Weka est un logiciel libre et gratuit, développé sous Java, à l'université de Waikato en nouvelle Zélande, Il est utilisable soit via une interface, soit via des lignes de commandes, ou encore à travers son API Java, Il propose beaucoup d'algorithmes classiques en apprentissage automatique (supervisé et non supervisé), avec les fonctionnalités de prétraitement des données, analyse et évaluation des résultats. Il intègre aussi plusieurs classificateurs comme kNN, NaiveBayes, RandomForest, ...etc. il permet aussi de crée des échantillons à partir des ensembles de données de grande taille avec la même distribution d'instances et visualiser l'ensemble de données à travers des graphiques très simples.

  Il comporte son propre format de fichier appelé ARFF, mais peut aussi traiter des données issues de bases relationnelles. Ce format ARFF précise essentiellement trois paramètres :

  - **Relation :** association d'un nom à l'échantillon de données
  - **Attribute :** spécification du nom et du type d'attribut de chaque instance dans l'échantillon de données
  - **Data :** une ligne indiquant le début de la description des données.

Sous éclipse pour utiliser l'API java de weka il faudra ajouter le Jar correspondant. Le logiciel ainsi que l'Api Java peuvent être téléchargé gratuitement à partir du lien suivant : http://www.cs.waikato.ac.nz/ml/weka/index.html

3 Présentation de l'application

Dans cette section nous allons détailler le code de l'application qui nous avons développé pour réduire la taille de la base de données sous format de schémas UML, nous expliquerons aussi le rôle de chaque classe et nous présenterons à la fin l'interface graphique.

L'application a été développée en java sous Eclipse IDE, dans cet environnement l'utilisation des packages est très utile pour organiser le code source, la figure suivante représente les packages présent dans l'application et les liaisons entre eux, nous détaillerons par la suite le contenu de chaque package.

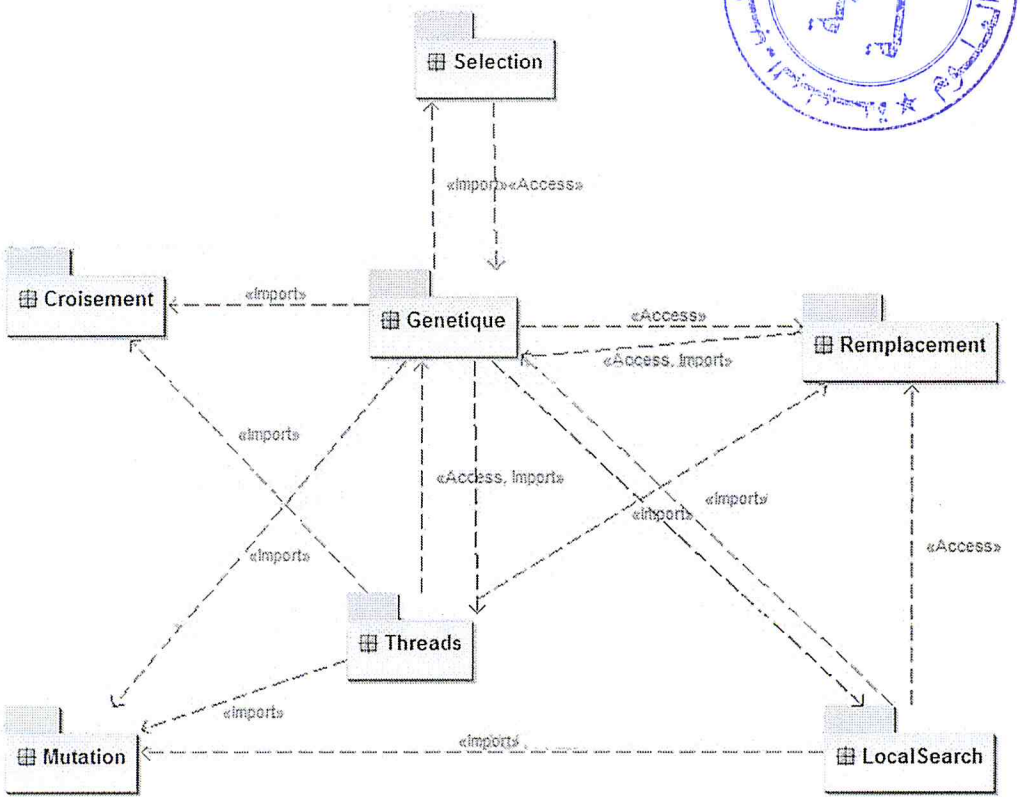


Figure 28 : Liaison entre les différentes packages de l'application

Le package « Genetique » représente le cœur de l'algorithme, c'est dans ce package que tous l'appelles aux autres méthodes que nécessite l'algorithme mémétique est effectuer.

3.1 Package Génétique

Il comporte deux classes :

- **Genetique_Operation :** cette classe regroupe tout l'algorithme génétique, elle contient la méthode *firtsPopulation*, celle-ci nous permet de générer notre population initial, et *CreatGeneration* qui fait appel aux différentes méthodes de sélection, croisement et mutation ainsi que la méthode de recherche local.
- **FitnessFunction :** elle contient la méthode *Evaluation* qui nous permet d'évaluer chaque individu (calculer sa fitness)

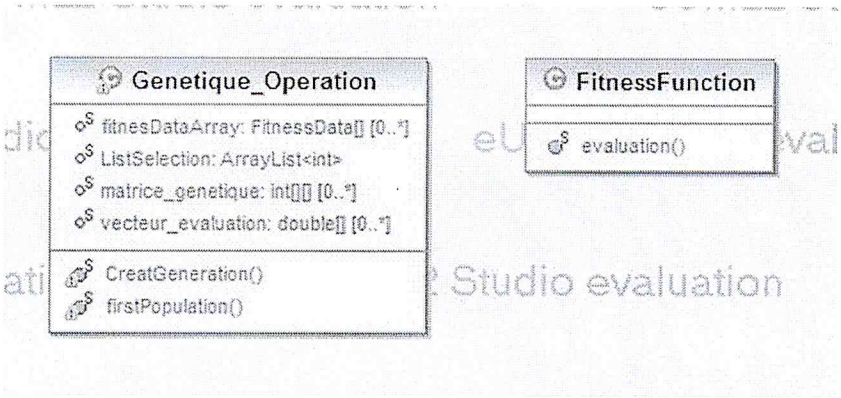


Figure 29: Contenu du package genetique

3.2 Package Threads

Il est très important de prendre le facteur du temps d'exécution on considération, en plus les bases de données à réduire sont de taille important et cela peut prendre un temps d'exécution considérable, pour remédier à ce problème nous avons profité de la force du langage Java dans la gestion du parallélisme, ceci en utilisant les threads.

Deux class deux threads sont présentes dans le package

- **FirstPopulationThread :** ce type de thread est utilisé pour l'évaluation de la population initiale. Chaque individu de la population est généré et évaluer dans un thread séparé.
- **GeneticOperationThread :** ce types de thread est utilisé au cœur de l'algorithme génétique, chaque thread prend en charge deux individu résultant de l'étape de sélection, il effectue le croisement de ces deux, la mutation et la recherche locale ainsi que le remplacement dans la population initial.

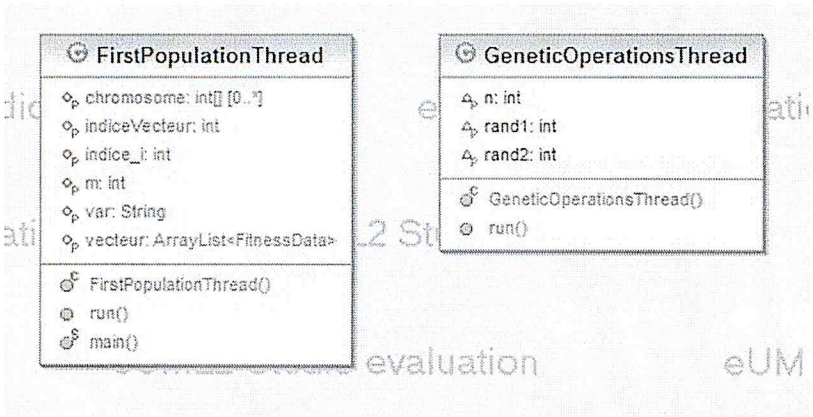


Figure 30: Contenu du package Threads

3.3 Package Selection

Dans ce package on trouve les différent méthodes de sélection que peut prendre l'algorithme génétique, chaque méthode implémente l'interface *Selection* qui contient la fonction abstraie *Select,* cette dernière sera alors défini dans chaque méthodes de sélection selon les besoins.

- **EletisteSelection** : pour la méthode de sélection Elitiste.
- **TournementSelection** : pour la méthode de sélection par tournois.
- **WheelSelection** : Pour la méthode de sélection Roulette.
- **Interface Selection** : Interface qui contient une méthode abstraite *select()*.

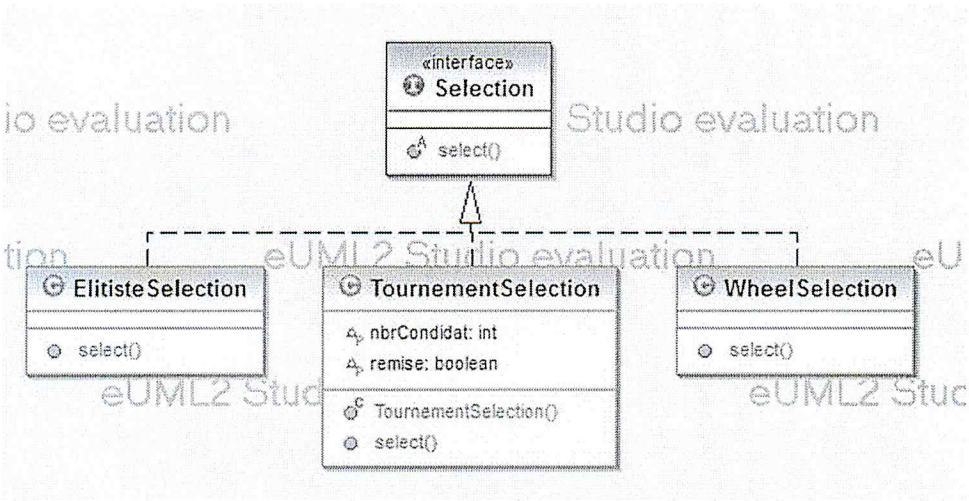


Figure 31: Contenu du package Selection

3.4 Package Croisement

Dans ce package on trouve les différentes méthodes de croisement, chaque méthode de croisement implémente l'interface *CrossOver*, est défini ensuit selon ses besoins la méthode *Cross*. Deux méthodes de croisement sont présents dans le package :

- **MultiPointCrossOver :** elle permet de définir plusieurs points de croisement sur les deux individus
- **OnePointCrissOver :** les deux individus seront croisés en se basent sur un seul point de croisement

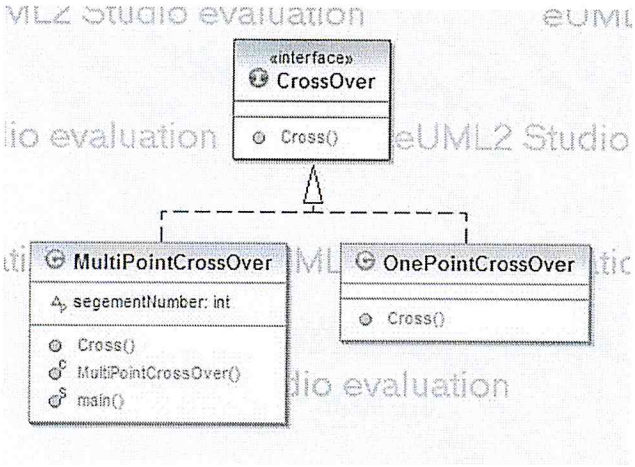


Figure 32: Contenu du package croisement

3.5 Package Mutation

Pour ce package on trouve une seule class *Mutation*, elle est sollicitée à chaque fois qu'une mutation est nécessaire durant l'exécution de l'algorithme. Deux méthodes sont définies dans la classe :

- **GeneticMutation :** Cette fonction est utilisée dans l'opérateur de mutation de l'algorithme génétique, avec une probabilité très petite
- **LocalSearchMutation :** Cette méthode est utilisée, durant la recherche locale appliquée sur chaque enfant résultant d'un croisement, la recherche consiste à modifier les gènes de l'individu qui porte l'indice 1 en 0 .

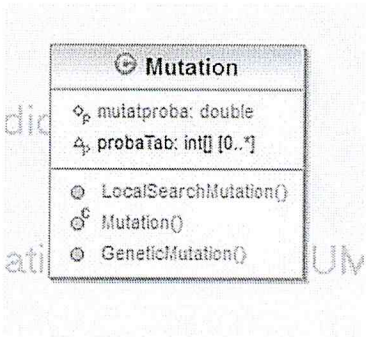


Figure 33: Contenu du package Mutation

3.6 Package Remplacement :

Ce package contient les différents méthodes de remplacement que peut utiliser un algorithme génétique, Une interface *Remplacement* est définie, elle contient la fonction abstraite *Remplacement( )*, qui sera défini différemment dans chaque méthode de remplacement selon les besoins . Deux classes sont définis dans le package

- **RemplacementElitiste :** cette classe définie la fonction abstraite *Remplacement()* de manière à ce que chaque nouveau enfant résultant de l'opération de croisement sera remplacer dans la population initial s'il répond aux critères exigées.
- **OneReplaceInGeneration :** cette classe définie la fonction abstraite *Remplacement()* de manière à ce que à chaque fin de génération, les deux meilleurs enfants résultants parmi tous les autres, seront remplacer dans la population initial.

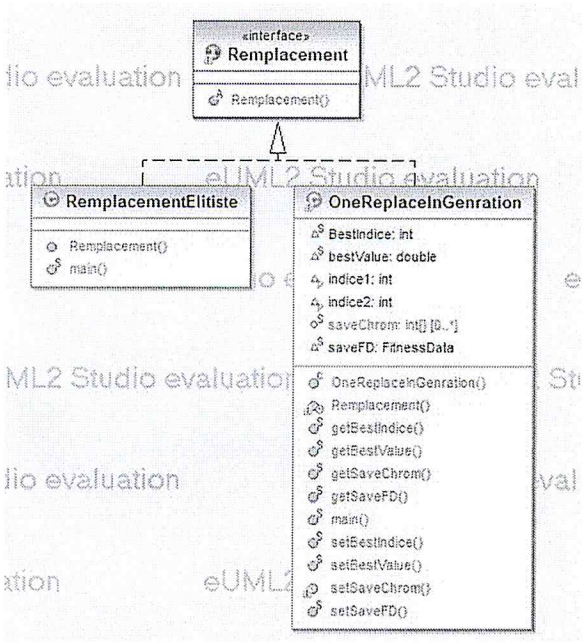


Figure 34: Contenu du package remplacement

3.7 Package LocalSearch

C'est le package qui contient la méthode de recherche local que nous avons implémenté, La classe *LocalSearch* présente dans ce package, contient la méthode *ControledSearch()* qui utilise LocalSearch*Mutation()* du package *Mutation* pour assurer la procédure de mutation des gènes de l'individu qui portent l'indice 1 en 0

ControledSearch() vérifie aussi si l'individu a était améliorer par la recherche local ou non et décide si ce dernier sera remplacer dans la population.

Figure 35: Contenu du package locale Search

3.8 Interface utilisateur

L'interface utilisateur à était conçu de manière simple, et offre la possibilité de visualiser les résultats de réduction, la sauvegarde de l'ensemble réduit au format .ARFF pour le visualiser sur Weka.

- **Section 1 :** Permet à l'utilisateur de fournir le fichier « Train » à réduire et le fichier « Test » utilisé pour l'apprentissage au format Arff.
- **Section 2 :** On trouve ici des informations concernant la taille des deux fichiers ainsi que la taille de la population généré, le temps 'd'exécution, le taux de classification, réduction initiaux.
- **Section 3 :** Pour lancer l'algorithme, sauvegarder les résultats affichés dans la Section 4 sous un fichier Xls et sauvegarder l'ensemble réduit comme un fichier au format Arff.
- **Section 4 :** Pour l'affichage des résultats de la réduction.

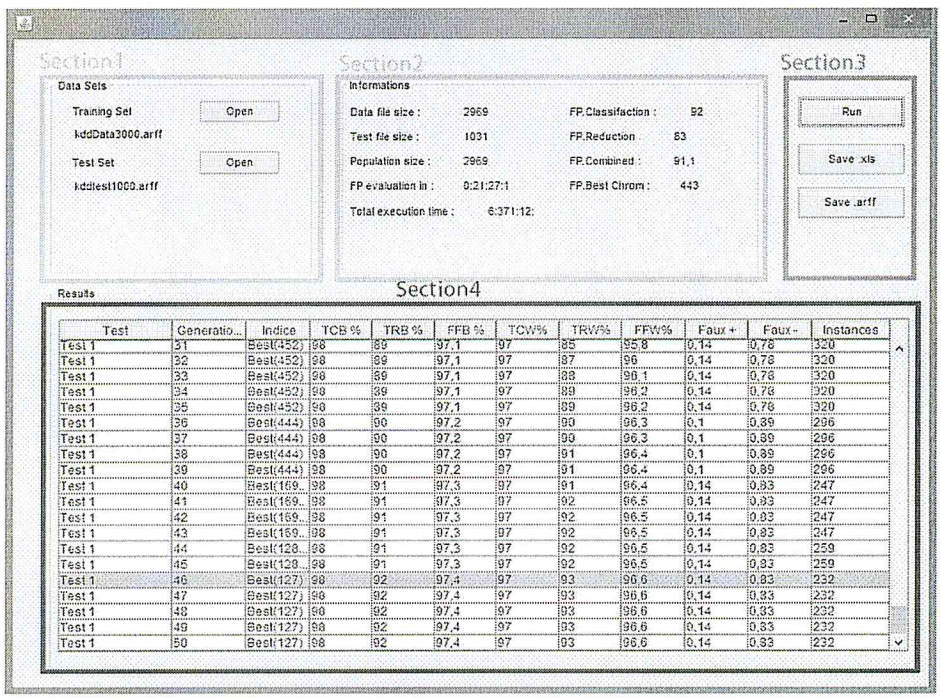


Figure 36 : Interface Utilisateur de l'application

4 Description du KDD'99

L'une des étapes les plus importantes dans le domaine du *machine leraning* c'est la phase d'apprentissage, ou un ensemble de données est utilisé pour entrainer un algorithme.

Parmi les célèbre ensembles on trouve le KDD'99, c'est un benchmark utilisé dans les recherches pour évaluer le taux de classification des algorithmes de classification, il est utilisé surtout quand il s'agit de la sécurité informatique, puisque il contient des types d'attaques réel sur un réseau LAN des US .Air Force.

Le KDD'99 connait aussi certain problèmes comme le taux élevé des données redondantes ce qui peut nuire à la classification ainsi qu'à l'évaluation des données, c'est pour ça que de nombreux recherches ont été réaliser pour corriger et améliorer cet ensemble qui reste largement très utilisé.

4.1 Présentation et structure du KDD'99

En 1998, un programme de recherche concernant la détection et l'évaluation des intrusions (DARPA'98) a était lancé par MIT Lincoln Labs. L'objectif était l'étude et l'évaluation de la recherche dans ce domaine [**25**].

Dans ce projet, un environnement a était créé pour acquérir pendant 9 semaines les données des paquets TCP qui peuvent circuler dans un réseau LAN. L'environnement crée simulé le réseau LAN des U.S. Air Force Surchargé avec différente types d'attaques.

Les données collectées dans ce projet sont sous forme binaire et ils ont une taille approximative de 4Go. Un record de 5 millions connexions a était enregistré pendant 7 semaines du projet, 2 semaines ont était consacré pour générer un ensemble de test qui contient plus de 2 millions connexions.

KDD'99 est un ensemble qui regroupe une version des données collectées dans DARPA'98, il contient 4.900.000 vecteurs de connexion signé comme normal ou attaque, chacun de ces vecteurs contient 41 attributs tel que la durée de connexion, le type du protocole, etc. ils représentent les caractéristiques d'un état normal ou anormal (le cas d'une attaque) du système. Il inclut 24 types d'attaques et sa taille est aux environs de 700Mo.

L'ensemble de test est utiliser pour mesuré le taux de classification du KDD'99 et le niveau de généralité de ces éléments. Il contient 2984154 vecteurs de connections avec une taille de 400Mo, Il faut noter que l'ensemble de teste ne possède pas la même probabilité de distribution que le KDD'99, il inclut les 24 types d'attaques présents dans KDD'99 plus 14 autres attaques spécifique qui ne sont pas présente dans le KDD'99 ce qui rend la tâche plus réaliste. Certain expert dans le domaine d'intrusion estime que la plus part des nouvelles attaques sont des variantes des anciennes attaques et les signatures des anciennes attaques sont parfois suffisante pour découvrir de nouvelles variantes d'attaques.

Une version réduite du KDD'99 est aussi disponible, elle contient 494020 instances (10% de la taille original), la distribution des différent types d'attaques est la même que celle de la base de

données original. Pareille pour l'ensemble de test, une version réduite est disponible, elle contient 311029 instances (10% de la taille original) avec la même distribution d'attaques.

KDD'99 ainsi que l'ensemble de test peuvent être téléchargés dans leurs format complet ou réduit à partir du site web de l'université de Californie – Irvine. Elle est en extension *.data* est peut être visualisé comme un fichier *.txt*

http://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/

4.2 Catégories des attaques

Chaque attaque présente dans l'ensemble KDD CUP 99 appartient à l'une des catégories suivantes :

- **Denial of Service Attack (Dos) :** c'est un type d'attaques ou l'attaqueur rend certaines ressources de calcul ou de mémoire trop occupé, donc incapable de répondre aux requêtes légitimes ou empêcher l'accès à ces ressources par les utilisateurs légitimes.
- **User to Root Attack (U2R) :** c'est une attaque qui commence par l'obtention d'un accès à un compte utilisateur légitime, l'attaquer utilise ensuite ce compte pour exploiter certaine vulnérabilité et essayer de créer une porte ouverte sur cette machine
- **Remote to Local Attack (R2L) :** se produit lorsque un attaqueur envoie des paquets à une machine à travers un réseau, mais il n'a pas le droit d'accès à la machine et il essaye d'exploiter quelques vulnérabilités pour gagner un accès local et se faire passer comme un utilisateur de la machine.
- **Probing Attack (Attaques de sondage) :** c'est une tentative de récolter des informations à propos d'un réseau d'ordinateurs dans le but évident de contourner ses contrôles de sécurité.

4.3 Classification des attributs

Les données dans l'ensemble KDD'99 sont stocké sous forme de vecteurs signés comme attaque ou normal, chaque vecteur contient 41 attributs qui représente les caractéristiques de la signature (Normal/Attaque) du vecteur. Ces attributs peuvent être regroupés dans 3 groupes :

- **Caractéristiques principal :** cette catégorie encapsule tous les attributs qui peuvent être extraites d'une connexion du type TCP/IP, la plut part de ces caractéristiques ont un impact implicite sur le délai de détection.
- **Caractéristiques du trafic :** cette catégorie inclut des caractéristiques qui sont calculé dans un laps de temps précis, elle se divise en deux groupes :
  - **Caractéristiques du même hôte :** seulement les connections qui se sont produits dans les 2 dernières secondes et qui ont comme destination le même hôte que celui de la connexion courante qui sont examinées.

- **Caractéristiques du même service :** seulement les connections qui se sont produits dans les 2 dernières secondes et qui ont le même service que celui de la connexion courante sont examinées.

  Les deux catégories (a et b) cité sont dites « time-based » mais il existe différente attaques « lentes » qui scannent les hôtes ou les ports de connexion sur un laps de temps supérieur à 2 secondes. Pour résoudre ce problème, les caractéristiques a et b sont recalculées après chaque 100 connexions effectuées au lieu de chaque 2 secondes, ces caractéristiques sont dites **caractéristiques du trafiques basé sur le nombre de connexions.**

- **Caractéristiques du contenu :** les attaques de types **R2L** et **U2R** ne possèdent pas de scénarios d'attaques fréquent, contrairement au attaques Dos ou Probing attacks qui invoques plusieurs connexions sur certaines hote(s) dans un laps de temps très court. Les attaques de types **R2L** et **U2R** sont encastré dans les trames de données des paquets et n'implique qu'une seule connexion, pour détecter ce genre d'attaques on doit être capable de repéré les comportements suspect dans les trames de données, comme le nombre de fois ou la connexion d'un utilisateur a échoué.

Le tableau suivant englobe la description des 41 attributs que peut contenir un vecteur de connexion.

Attribut	Description
duration	durée de la connexion (nb de secondes)
protocol_type	type du protocole, ex. tcp, udp, icmp...
service	service réseau (destination) ex. http, telnet
flag	statut de la _ connexion _ (normal ou erreur)
src_bytes	nb de données (en octets) de la source vers la destination
dst_bytes	nb de données (en octets) de la destination vers la source
land	1 si la _ connexion _ est de/vers le même hôte/port ; 0 sinon
wrong_fragment	nb de fragments « erronés »
urgent	nb de paquets urgents
hot	nb d'indicateurs « hot »
num_failed_logins	nb d'essais login ratés
logged_in	1 si succès du login ; 0 sinon
num_compromised	nb de conditions de « compromis »

root_shell	1 si la racine shell est obtenue ; 0 sinon
su_attempted	1 s'il y a tentative de la commande _ racine su _ ; 0 sinon
num_root	nb d'accès à la « racine »
num_file_creations	nb de créations d'opérations de fichiers
num_shells	nb de shell prompts
num_access_files	nb d'opérations sur les fichiers de contrôle d'accès
num_outbound_cmds	nb de commandes outbound dans une session ftp
is_hot_login	1 si le login appartient à la liste _ hot _ ; 0 sion
is_guest_login	1 si le login est login _ invité _ ; 0 sinon
count nb de connex.	pour le même hôte
srv_count	nb de connex. pour le même service
serror_rate	% de connex. pour le même hôte ayant l'erreur « SYN »
srv_serror_rate	% de connex. pour le même service ayant l'erreur « SYN »
rerror_rate	% de connex. pour le même hôte ayant l'erreur «REJ »
srv_rerror_rate	% de connex. pour le même service ayant l'erreur « REJ »
same_srv_rate	% de connex. pour le même hôte utilisant le même service
diff_srv_rate	% de connex. pour le même hôte utilisant différents services
srv_diff_host_rate	% de connex. pour le même service utilisant différents hôtes
dst_host_count	nb de connex. pour le même hôte
dst_host_srv_count	nb de connex. pour le même hôte utilisant le même service
st_host_same_srv_rate	% de connex. pour le même hôte utilisant le même service
dst_host_diff_srv_rate	% de connex. pour le même hôte utilisant différents services
dst_host_same_src_port_rate	% de connex. pour le même hôte ayant le port src
dst_host_srv_diff_host_rate	% de connex. pour le même hôte et le même service utilisant différents hôtes
dst_host_serror_rate	% de connex. pour le même hôte ayant l'erreur «SYN »

dst_host_srv_serror_rate	% de connex. pour le même hôte et le même service ayant l'erreur « SYN »
dst_host_rerror_rate	% de connex. pour le même hôte ayant l'erreur « REJ »
dst_host_srv_rerror_rate	% de connex. pour le même hôte et le même service

4.4 Problèmes liés au KDD CUP 99

Comme mentionné précédemment, KDD'99 a été construit en se basent sur les données de DARPA'98, qui a était critiqué par McHug [**26**], principalement en raison de la caractéristiques des données synthétique. Comme résultat, certain erreurs du DARPA'98 demeurent toujours dans KDD'99.

- Les collecteurs de trafic comme TCPdump[7], qui a été utilisé dans le projet DARPA'98 ont tendance d'être surcharger durant les point de pic du trafic, et de rejeté certain paquets, cependant aucune mesure a été prise pour remédier à ce problème
- Il n'y a pas de définition exacte pour les attaques, par exemple **probing** n'est pas considéré comme attaque seulement si le nombre d'itération dépasse un seuil spécifique, un paquet qui cause un dépassement en mémoire ne représente pas toujours une attaque.
- Portony et AL. [**27**] ont divisé l'ensemble KDD en 10 sous-ensembles, chacun d'eux contenait approximativement 490,000 instances ou 10% de l'ensemble de test complet. Ils ont pu remarquer que la distribution des attaques dans le KDD est très irrégulière, ce qui a rendu la validation croisée des données une taches très difficile, plusieurs sous-ensemble ne contenait qu'une seule instance d'attaque.
- l'un des problèmes majeurs de l'ensemble KDD c'est la redondance des données, dans [**28**] les auteurs ont mentionnés deux types d'attaques *smurf* et *neptun*, ces deux attaques de types Dos constituent 71% de l'ensemble de test, ce qui affecte complètement l'évaluation des données. Aussi ça pousse les algorithmes d'apprentissage de se baser sur les instances les plus fréquente, et cela les empêchent d'apprendre les données moins fréquente qui sont très utile dans la détection au niveau des réseaux.

[7] C'est un analyseur de paquets en ligne de commande. Il permet d'obtenir le détail du trafic visible depuis une interface réseau.

5 Tests et résultats

Pour valider l'algorithme que nous avons présenté dans 2.4.62.4.6 ci-dessus nous allons effectuer une série de tests, ça nous permettra de vérifier si l'algorithme donne de bon résultat ou non

L'algorithme présenté consiste à réduire la taille de la base de données utilisé par l'IDS, en proposant un ensemble plus petit avec des performances égales ou supérieures à ceux de l'ensemble initial.

La réduction de l'ensemble KDD'99 en entier est impossible sur un simple ordinateur, sa taille est trop volumineuse (700 Mo), ceci nécessite une machine très performant, avec une grande puissance de calcul et de mémoire.

l'ensemble de 10% (494020 instances) reste aussi difficile à réduire, avec les limitations que nous impose l'environnement de développement, en utilisant la machine virtuelle de Java, la taille maximal que peut être allouer pour une application Java sous un système 32bits est de 1Go et 5Go sur un systèmes de 64bits , ceci reste très peu pour tenter de réduire l'ensemble de 10%.

Le logiciel Weka 2 ci-dessus, propose une solution pour ce problème, il permet de crée de petites échantillons d'un ensemble donné en gardant la même distribution d'attaques que celle de l'ensemble initial.

En utilisant Weka, nous avons créé trois sous-ensembles :

- **Train :** Un ensemble de 4942 instances, il représente 1% de l'ensemble (10% KDD'99), ceci sera l'ensemble que nous allons essayer de réduire
- **Test :** Un ensemble de 1000 instances, tiré aléatoirement de l'ensemble (10% KDD'99) et qui ne sont pas présentent dans l'ensemble **Train,** cet ensemble sera utiliser dans la phase d'apprentissage.
- **TestCorrected :** Un ensemble de 2996 instances, il représente 1% de l'ensemble (10% Corrected) , cet ensemble contient 24 types d'attaques qui sont présent dans **Train** et 14 autres non disponible, ceci permettra de tester si l'ensemble réduit gardera un bon taux de classification quand il sera exposé à de nouvelle attaques totalement inconnue.

5.1 Test : Réduction de l'ensemble « Train »

Tout d'abords nous allons essayer de générer quatre différents ensembles réduits « **RTrain** » ceci en appliquant l'algorithme Mémétique-1NN quatre fois en utilisant les ensembles « **Train** » et « **Test** » comme paramètres d'entrée. Les résultats des quatre ensembles résultant vont être comparés avec ceux de l'ensemble initial par la suite.

Ensemble	TC %	FP %	FN %	ICC	IMC	TR %	Instances
Train	99.09	1.57	0.00	990	9	0.00	3943
RTrain 1	97.497	0.52	0.74	974	25	94.0	236
RTrain 2	96.997	0.00	1.23	969	30	93.0	276
RTrain 3	98.898	0.52	0.12	988	11	89.0	433
RTrain 4	98.398	0.00	0.49	983	16	90.0	354

Tableau 6: Résultats de la classification de Test

Le tableau ci-dessus indique les résultats de classification de l'ensemble « Test » avec les quatre ensemble réduit ainsi que l'ensemble « Train », on voit clairement que le taux de classification des quatre ensemble réduit n'as pas gravement détérioré, on atteint 98.898 avec RTrain2.

Le taux de faux positif a été amélioré, l'ensemble initial affiche 1.57% alors que **RTrain2** et **RTrain4** affichent tous les deux 0.00 de faux positif.

Le taux de réduction atteint 94.0 (236) comme valeur maximal, ce qui confirme la présence des instances doubles dans l'ensemble **Train**.

À partir de ces résultats on peut confirmer que l'algorithme réussi à générer des ensembles réduit du fichier Train sans trop détériorer le taux de classification ainsi que le taux du faux positif.

5.2 Test : Classification de « TestCorrected »

Dans ce test nous allons essayer de classé les instances de **TestCorrected** avec les ensemble réduit et l'ensemble **Train**, comme cité précédemment, **TestCorrected** contient 14 autres types d'attaques qui n'existent pas dans **Train**, donc forcément elles ne seront pas présents dans les ensemble réduit, ce test va mesurer la capacité des ensembles à détecter de nouvelles attaques inconnus.

Ensemble	TC %	FP %	FN %	ICC	IMC	TR%	Instances
Train	93.591	1.87	5.44	2804	192	0.00	3943
RTrain 1	92.923	2.55	4.69	2784	212	94.0	236
RTrain 2	92.923	2.38	4.90	2784	212	93.0	276
RTrain 3	93.424	3.06	4.40	2799	197	89.0	433
RTrain 4	93.491	2.04	5.19	2801	195	90.0	354

Tableau 7: Résultats de classification de TestCorrected

La différence est très clair par-rapport aux résultats du test précédent, on voit bien que le taux de classification descend à 93.59 pour l'ensemble **Train** et 93.49 pour **RTrain4**, ceci est à fausse classification de certain nouvelle types d'attaque, malgré ça on remarque que il n'y a pas une grande différence entre le taux de classification pour **Train** et **RTrain4** dème pour le taux de faux positif. Si on prend en considération le taux de réduction l'avantage sera pour **RTrain** car il arrive à rivaliser **Train** avec seulement 354 instances.

5.3 Test : Variation des classificateurs

Pour valider le nouvel ensemble réduit, il est nécessaire de le tester avec plusieurs classificateurs, Nous avons choisis sept classificateurs parmi ceux que propose Weka. Nous avons choisi les classificateurs suivants :

- 1-NN
- J48
- NaiveBase
- NBTree
- RandomForest
- Multi-Layer Perceptron

Les résultats de chaque classificateur sont présentés par un histogramme :

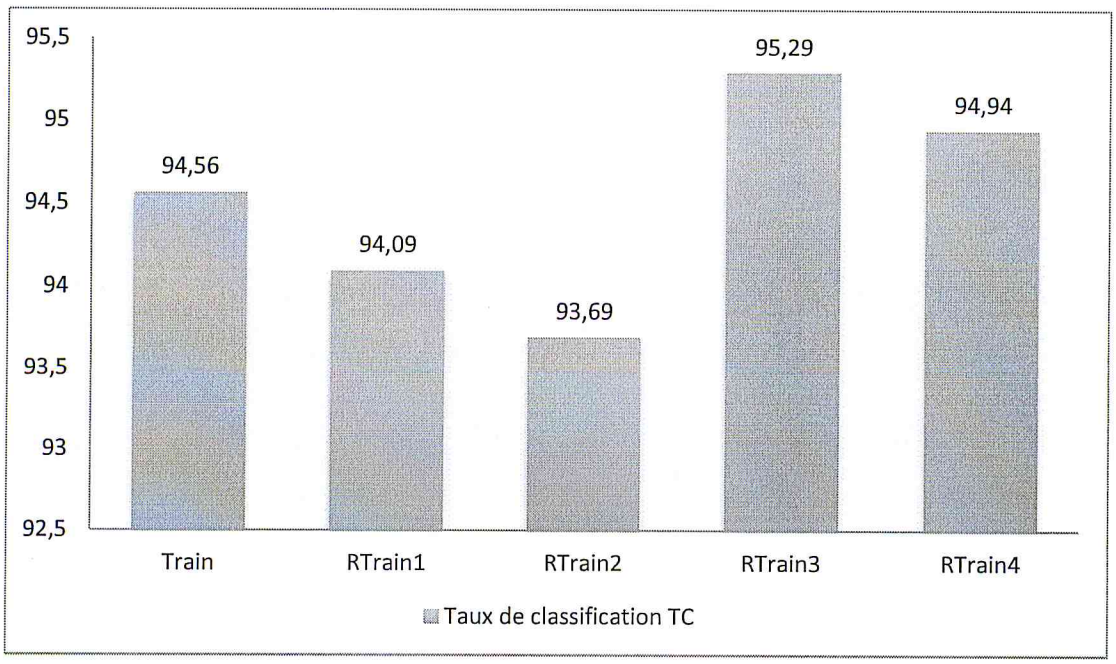


Figure 37 : Résultats du classificateur 1-NN

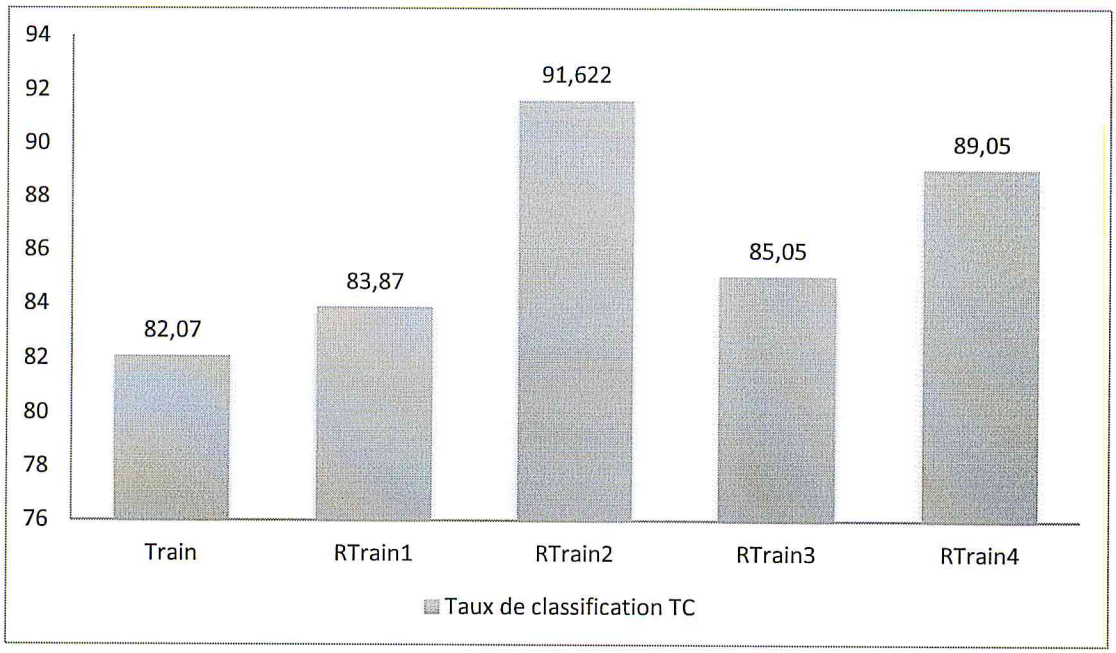


Figure 38: Résultats du classificateur J48

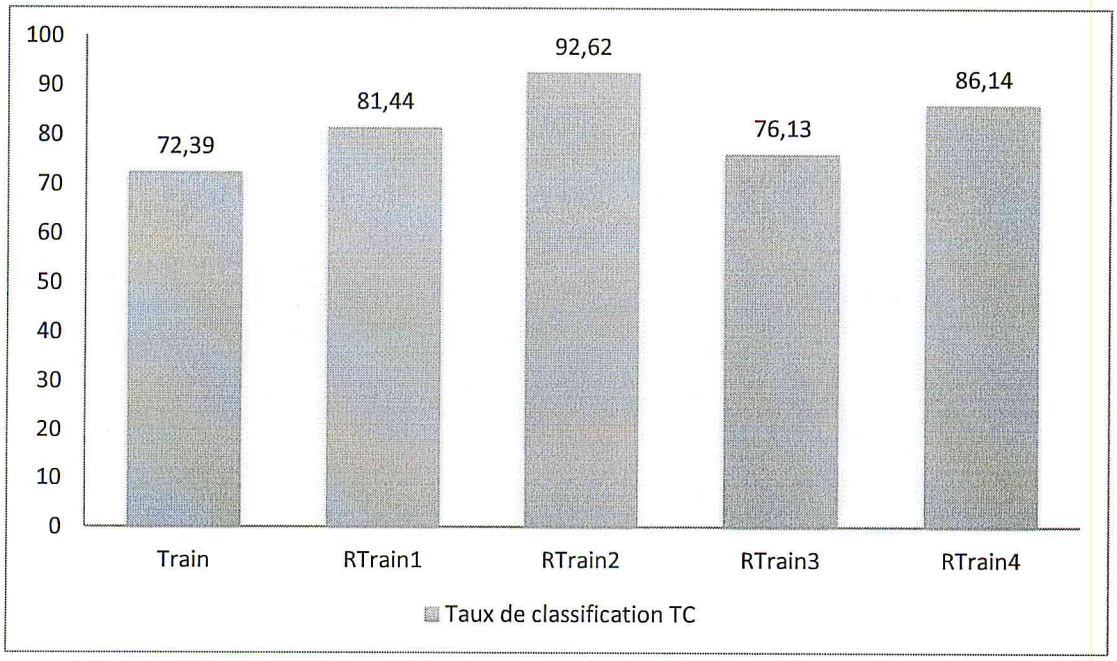


Figure 39 : Résultats du classificateur NaiveBase

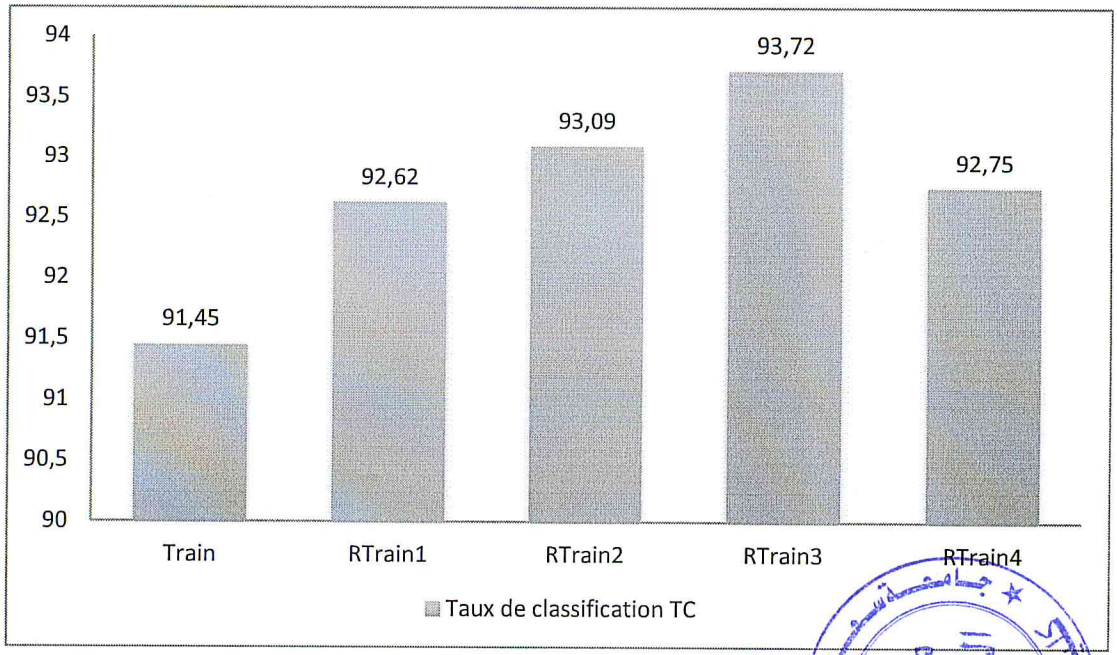


Figure 40 : Résultats du classificateur NBTree

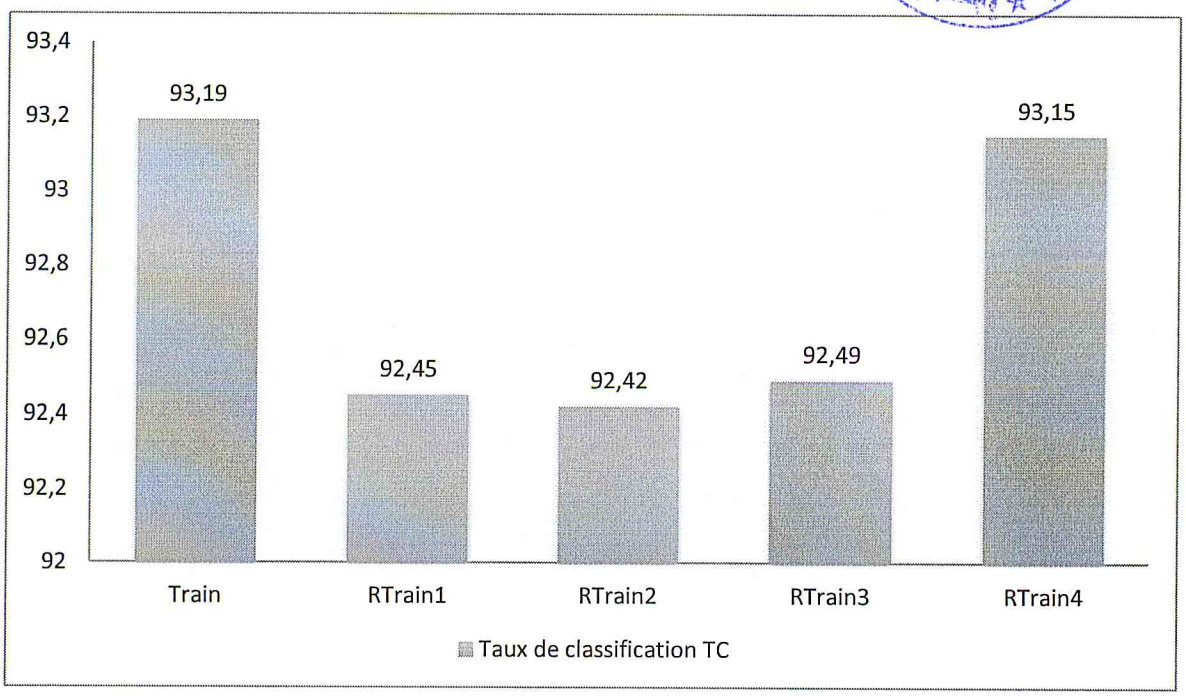


Figure 41 : Résultats classificateur Multi-Layer Perceptron

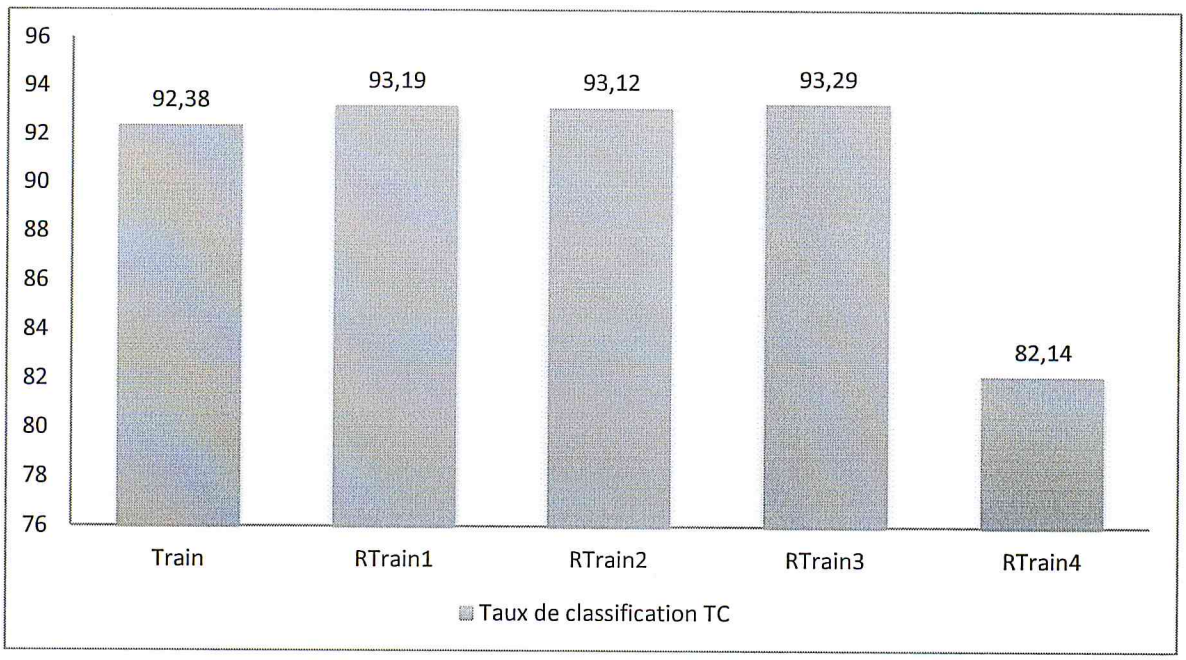


Figure 42 : Résultats du classificateur RandomForest

On remarque que pour la plus part des classificateurs, l'ensemble qui possède le meilleur taux de classification est souvent l'un des ensemble réduit RTrain, ceci confirme que l'ensemble réduit peut être utilisé avec d'autres classificateurs mais les résultats dépend de la manière auquel le classificateur procède.

5.4 Test : Classification en temps réel

La classification en temps réel exige que la taille de l'ensemble soit petite, car plus la taille est grande plus le classificateur met beaucoup de temps pour classer une instance et du coup, d'autres instances passes inaperçu ou seront détruite.

Pour simuler une classification en temps réel nous allons envoyer sur un réseau LAN à partir d'un ordinateur A, les instances de l'ensemble **TestCorrected** dans un paquet TCP, une après l'autre. De l'autres coté l'ordinateur B jouera le rôle d'un IDS, il recevra les instances une après l'autre et il essayera de les classer en utilisant l'ensemble réduit **RTrain.**

Le tableau ci-dessus représente les résultats obtenu, la vitesse de réception pour tous les cas est de 8kb/s , le taux de détection indique le t'aux de paquets d'attaque détecté :

Base de données	TC %	FP %	FN %	Détection
RTrain1	94.09	2.55	6.73	93.27
RTrain2	93.09	4.42	6.77	93.23
RTrain3	95.29	0.68	5.69	94.31
RTrain4	94.49	2.55	6.23	93.77

Tableau 8: Résultats de classification en temps-réel

6 Conclusion

Dans ce chapitre nous avons réalisé une série de tests pour tester et valider l'algorithme réalisé, nous avons pu voir que malgré que les ensembles réduit contient moins d'instances que l'ensemble initial mais, ils arrivent toujours à garder un bon taux de classification et de faux positif, ceci renforce l'idée de la de la réduction de la taille de la base de données pour améliorer les performances de l'IDS.

Malheureusement nous n'avons pas pu appliquer notre algorithme sur des ensemble de grande taille comme le 10%KDD'99 ou KDD'99 mais cela n'empêche de dire qu'il ne fonctionnera pas, puisque on arrive à extraire les meilleurs instances à partir d'un ensemble petit, ce serait encore possible de les extraire avec un ensemble plus grand et non pas le contraire.

Conclusion général

Conclusion général

La sécurité des réseaux informatiques courts chaque jour de nouveaux risques, et de plus en plus de politiques de sécurité sont mis en place pour corriger les failles. La possibilité pour un système informatique d'être sans failles est quasiment nulle, ce qui est le cas des systèmes de détection d'intrusion.

Au cours de ce projet, nous avons essayé d'apporter une amélioration pour les systèmes de détection d'intrusions, cela en se basent sur la base de données qu'ils utilisent pour reconnaitre les différents types d'attaques, nous avons proposé un algorithme qui consiste à réduire la taille de cette base de données tout en la gardent aussi performent que possible, par conséquent le système de détection d'intrusion aura tendance à minimiser les fautes de classification qu'il génère et aussi d'être plus rapide .

Pour réaliser cette tache nous avons opté pour les algorithmes mémétiques, qui ont prouvé leur efficacité dans plusieurs problèmes d'optimisation combinatoire, et nous avons pu obtenir des résultats convainquant sur un petit échantillon de la base de données KDD'99 qui est très utilisé dans le domaine de détection d'intrusions.

On espère par la suite que notre algorithme sera appliqué sur la base KDD'99 pour tester son efficacité, et essayer de l'améliorer.

L'apport que nous avons essayé d'apporter au domaine de détection d'intrusion est très petit, ce domaine est très vaste es nécessite la collaboration de plusieurs travaux pour proposer des solutions aux différents problèmes existant.

Bibliographie

[1] K. J, *Integreating Artificial Immune Algorithms for Intrusion Detection.* London: University College, 2002.

[2] P. Biondi, "Architecture expérimental pour la détection d'intrusions dans un système informatique," in , 2001.

[3] G. Hiet, "Détection d'intrusion paramétrée par la politique de sécurité grace au controle collaboratif des flux d'information au sein du système d'exploitation et des applications: mise en oueuvre sous Linux pour les programmes Java," in , Rennes, 2008.

[4] V. Glaume, "Détection d'intrusion réseau et système," in , 2008.

[5] M. A. Ibrahim and H. Tebourbi, "Installation et configuration d'un système de détection dintrusion (IDS)," in , Carthage, 2009.

[6] R. Simon, *Catégorisation automatique de textes et coocurrence de mots provenant de documents non étiquetées.* Université Laval, 2005.

[7] O. Julien, "Caractérisation d'un air de musique et reconnaisance avec un son dans une base de données, Annexe apprentissage automatique," 2011.

[8] P. Vincent, "Modèles à noyaux à structure locale," 2003.

[9] T. Guilaume, *Optimisation d'ensembles de classifieurs non paramétriques avec apprentissage par représentation partielle de l'information.* Ecole de téchnologie supérieur du Quebec, 2004.

[10] L. Cosmin, "Méthodes non supervisées pour l'analyse des données multivariées," 2008.

[11] M.-A. Amal and B.-A. Ahmed Riadh, "Survey of nearest neighbor condensing techniques," 2008.

[12] L. Yang and G. Li, *Tcm-Knn algorithm for supervised network intrusion detection coputers & security.* 2007.

[13] Y. Li, *Optimizing network anomaly detection sheme using instance selection mechanism*, G. T. Conference, Ed. 2009.

[14] L. Kwok Ho and K. Lam For, *Ids false alarm filtering using knn classifier. In WiSA'04 Proceedings of the 5th international conference on Information Security Applications.* 2005.

[15] S. Vob, S. Martello, I. H. Osman, and C. Roucairol, *Meta-Heurestics-Advances and Trends in Local Search Paradigms for Optimization*, Dordecht, Ed. The Netherlands: Kluwer Academic

Publisher, 1999.

[16] O. I H and L. G, *Metaheuristcis: a bibliography.* Annals of Operations Research, 1996.

[17] N. Sébastien, *Métaheuristiques hybrides pour la résolution de problème d'ordonnacement de voitures dans une chaine d'assemblage automobile.* Chicoutimi, Québec: Université de Québec, Montréal, 2007.

[18] A. Souquet and F.-G. Radet, *Algorithmes Génétiques.* 2004.

[19] A. Jean-Marc and D. Nicolas, *Algorithmes génétiques.* 2005.

[20] C. Cotta, Talbi, and E. Alba, *Parallel Hybrid Métaheuristics, in Parallel Métaheuristics: A New Class of Algorithms.* 2005.

[21] M. P, *On evolution, search, optimization, genetic algorithms and martial arts Towards memetic algorithms.* Pasadena: California Institue of Technologie, 1989.

[22] H. H. H and S. T, *Stochastic local search: Foundations and applications*, M. Kauf-mann, Ed. 2004.

[23] Oracle. Oracle Documentation. [Online]. http://docs.oracle.com/javase/7/docs/api

[24] Oracle. Informations et configuration minimale requise pour l'installation et l'utilisation de Java 7 pour Mac . [Online]. http://www.java.com/fr/download/faq/java_mac.xml#java%20available

[25] T. Mahbod, B. Ebrahim, L. Wei, and A. G. Ali, ""A Detailed Analysis of the KDD CUP 99 Data Set," Proceeding of the 2009 IEEE Symposium on Computational intelligence in Security and Defense Applications (CISDA 2009)," 2009.

[26] J.McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," in *ACM Transaction on information and system security, vol. 3, no. 4,*, 2000, vol. 3, pp. 262-294.

[27] L. Portnoy, E. Eskin, and S. Stolfo, ""Intrusion detection with unlabeled data using clustering" Proceedings of ACM CSS Workshop on data Mining Applied to Security," 2001.

[28] K.Leung and C. Leckie, ""Unsupervised anomaly detection in network intrusion detection using clusters" ," in *Proceedings of the twenty-eighth Australasian conference on Computer Science*, 2005, vol. 38, pp. 333-342.