OPTIMISATION COMBINATOIRE

3.1 Introduction

Les problèmes d'optimisation occupent actuellement une place importante dans la communauté scientifique où il est nécessaire de trouver des solutions optimales pour des problèmes très compliqués et difficiles à résoudre. Parmi les méthodes les plus utilisées pour la résolution de tels problèmes, on trouve les métaheuristiques qui englobent un ensemble d'algorithmes capables de résoudre des problèmes assez compliqués en s'inspirant d'analogies avec la physique (recuit simulé), avec la biologie (algorithmes évolutionnaires)ou encore l'éthologie (colonies de fourmis, essaims particulaires).

Dans ce chapitre, nous avons présenter le concept d'optimisation combinatoire et ses différentes techniques, où nous avons concentrer sur les métaheuristiques vu que c'est la méthode d'optimisation que nous allons utiliser pour générer notre modèle de détection d'intrusions. Plus particulièrement, la méthode de recuit simulé et celle de recherche tabou qui sont utilisées dans ce travail conjointement avec le réseau de neurones multicouches pour optimiser les poids de ce dernier afin de perfectionner notre modèle de détection d'intrusions.

3.2 Optimisation Combinatoire

3.2.1 Définition

L'optimisation combinatoire recouvre toutes les méthodes qui permettent de déterminer l'optimum d'une fonction avec ou sans contraintes. En théorie, un problème d'optimisation combinatoire est défini par un ensemble d'instances. A chaque instance du problème est associé un ensemble discret de solutions S, un sous-ensemble X de S représentant les solutions admissibles (réalisables) et une fonction de coût f (ou fonction objectif) qui assigne à chaque solution $s \in$X le nombre réel (ou entier) $f(s)$. Résoudre un tel problème (plus précisément une telle instance du problème) consiste à trouver une solution $s^* \in$ X optimisant la valeur de la fonction de coût f. Une telle solution s^* s'appelle une solution optimale ou un optimum global.[44]

3.2.2 Classification des méthodes d'optimisation combinatoire

La résolution d'un problème d'optimisation combinatoire est réalisée à l'aide des méthodes illustrées dans la figure suivante :

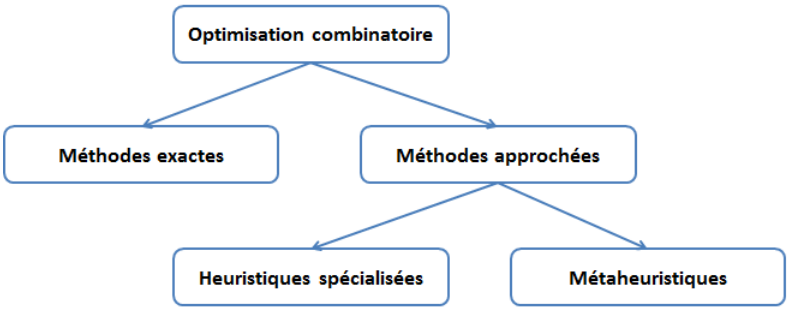

Figure 3.1 – Classification des méthodes d'optimisation
[45]

a) Méthodes exactes

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche. Pour améliorer l'énumération des solutions, une telle méthode dispose de techniques pour détecter le plus tôt possible les échecs (calculs de bornes) et d'heuristiques spécifiques pour orienter les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de séparation et évaluation (Branch and Bound) ou la programmation dynamique. Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable.[21]

b) Méthodes approchées

Lorsque l'on dispose d'un temps de calcul limité ou lorsqu'on est confronté à des problèmes difficiles ou de taille importante, on peut avoir recours aux méthodes approchées, en se contentant de rechercher une solution de bonne qualité. Dans ce cas le choix est parfois possible entre une heuristique spécialisée et une métaheuristique :[46]

- **Heuristique :** Le mot *heuristique* vient du grec « eurisko »qui signifie « je trouve »d'où la célèbre Eureka d'Archimède. Une heuristique, ou méthode approximative, est un algorithme qui fournit rapidement(en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation difficile. Une méthode heuristique est généralement conçue pour un problème particulier, en s'appuyant sur sa structure propre.
- **Métaheuristique :** Le mot *métaheuristique* est dérivé de la composition de deux mots grecs : méta signifiant « au-delà »et heuristique. En effet, ces algorithmes se veulent des méthodes génériques pouvant optimiser une large gamme de problèmes différents, sans nécessiter de changement profond dans l'algorithme employé.

3.3 Métaheuristiques

3.3.1 Définition

Contrairement aux méthodes exactes, les métaheuristiques permettent de s'attaquer aux instances problématiques de grande taille en fournissant des solutions satisfaisantes dans un délai raisonnable. Il n'y a aucune garantie de trouver des solutions globales optimales ou même des solutions limitées. Les métaheuristiques ont reçu de plus en plus de popularité au cours des 20 dernières années. Leur utilisation dans de nombreuses applications montre leur efficacité pour résoudre des problèmes vastes et complexes. Parmi ces domaines, on peut citer les suivants : [45]

- Conception technique, optimisation de la topologie et optimisation structurelle en électronique et VLSI, aérodynamique, dynamique des fluides, télécommunications, automobile, et la robotique.
- Apprentissage automatique et fouille de données en bioinformatique et biologie computationnelle, et les finances.
- Simulation et identification en chimie, physique et biologie, traitement du signal et de l'image...etc.
- Planification des problèmes de routage, planification des robots, problèmes d'ordonnancement et de production, logistique et transport, gestion de la chaîne d'approvisionnement...etc.

3.3.2 Concepts fondamentaux des métaheuristiques

Les métaheuristiques ne nécessitent pas une connaissance particulière sur les problèmes d'optimisation à résoudre. Il suffit d'associer une ou plusieurs variables à une ou plusieurs solutions (optimum).

Il existe deux points critiques pour toute métaheuristique :[47]

- **Diversification :** Le principe de diversification d'une méthode d'optimisation donnée correspond à sa capacité de parcourir aisément l'espace de recherche pour obtenir des solutions très différentes les unes des autres. Autrement dit, c'est une mécanisme pour une exploration assez large de l'espace de recherche.

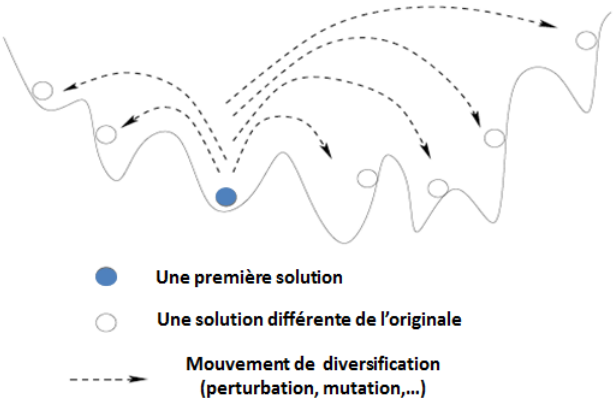

Figure 3.2 – Processus de diversification d'une solution

- **Intensification :** Autant le principe de diversification essaye de déplacer les solutions dans d'autres zones de l'espace de recherche, autant le processus d'intensification vise à forcer une solution donnée à tendre vers l'optimum local de la zone à laquelle elle est attachée. En effet, Elle permet une exploitation de l'information accumulée durant la recherche.

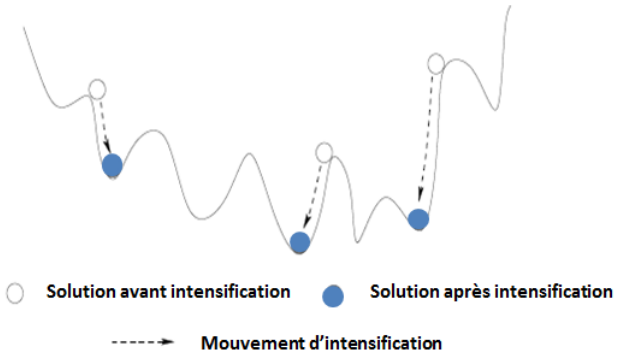

Figure 3.3 – Processus d'intensification de plusieurs solutions

3.3.3 Classification des métaheuristiques

Une manière de classifier les métaheuristiques est de distinguer celles qui travaillent avec une population de solutions de celles qui ne manipulent qu'une seule solution à la fois(méthodes de trajectoire).

Voici une figure qui montre cette classification avec des exemples typiques pour chaque catégorie :

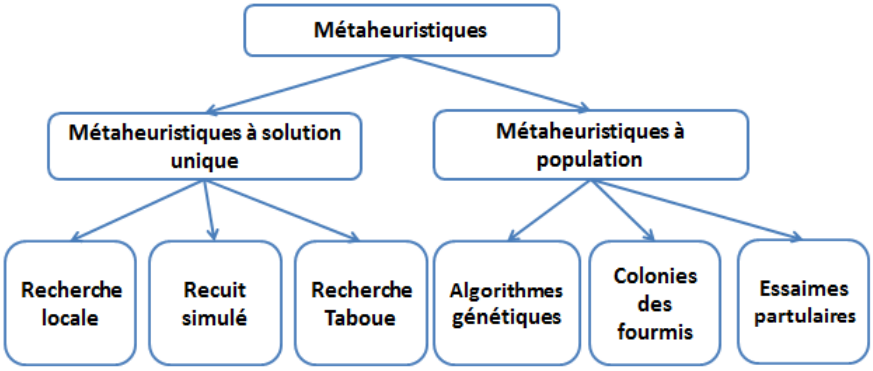

Figure 3.4 – Classification des métaheuristiques

a) Métaheuristiques à solution unique

Les méthodes itératives à solution unique sont toutes basées sur un algorithme de recherche de voisinage qui commence avec une solution initiale, puis l'améliore pas à pas en choisissant une nouvelle solution dans son voisinage[48].

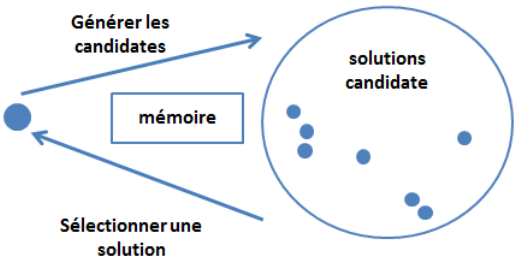

Figure 3.5 – Principe des métaheuristiques à solution unique

Il existe plusieurs méthodes pour cette classification, les plus connues sont :

- **La recherche locale :** La recherche locale itérative est un modèle de recherche locale qui améliore le principe de recherche locale multi-départ (« multiple start local search ») dans lequel des méthodes de descente sont lancées successivement sur des solutions initiales générées aléatoirement pour contrer l'aspect déterministe de la descente. Le principe est simple, une recherche par descente est appliquée sur une solution initiale pour générer une meilleure solution. On applique ensuite une nouvelle recherche par descente sur cette nouvelle solution après l'avoir « perturbée ». La solution obtenue est comparée avec la solution initiale pour savoir si elle la remplace ou non. Tout cela représente une itération de la recherche locale itérative.[47]
- **Le recuit simulé :** Le recuit simulé est une technique d'optimisation de type Monte-Carlo généralisé à laquelle on introduit un paramètre de température qui sera ajusté pendant la recherche. Elle s'inspire des méthodes de simulation de Metropolis (années 50) en mécanique statistique. L'analogie historique s'inspire du recuit des métaux en métallurgie : un métal refroidi trop vite présente de nombreux défauts microscopiques, c'est l'équivalent d'un optimum local pour un problème d'optimisation combinatoire. Si on le refroidit lentement, les atomes se réarrangent, les défauts disparaissent, et le métal a alors une structure très ordonnée, équivalente à un optimum global. Cette technique sera détaillée dans la suite de ce chapitre vu que c'est l'une des méthodes utilisées pour réaliser ce travail.[49]
- **La recherche tabou :** La recherche Tabou a été introduite par F. Glover et a montré sa performance sur de nombreux problèmes d'optimisation. Le principe de l'algorithme est le suivant : à chaque itération, le voisinage (complet ou sous-ensemble de voisinage) de la solution courante est examiné et la meilleure solution est sélectionnée. En appliquant ce principe, la méthode autorise de remonter vers des solutions qui semblent moins intéressantes mais qui ont peut être un meilleur voisinage. Cette méthode sera détaillée dans la suite de ce chapitre.[48]

b) Métaheuristique à population

Dans cette classe les métaheuristiques utilisent la notion de population : elles manipulent toutes un échantillonnage de la fonction objectif, via des processus communs. Autrement dit, les méthodes d'optimisation à population de solutions améliorent, au fur et à mesure des itérations, une population de solutions. L'intérêt de ces méthodes est d'utiliser la population comme facteur de diversité.

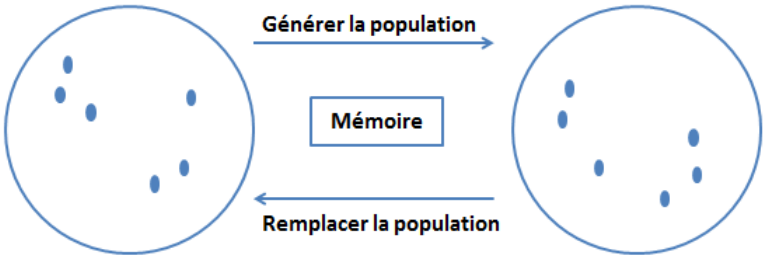

Figure 3.6 – Principe des métaheuristiques à population

Parmi les algorithmes inclus dans cette classification on peut citer les suivants :

- **Les algorithme génétiques :** Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle : croisements, mutations, sélection, etc. Introduits par J.H. Holland au début des années 1970. Ils sont appliqués dans diverses domaines : l'économie, l'optimisation de fonctions, la finance, en théorie du contrôle optimal, théorie des jeux répétés et différentiels.
- **Les colonies des fourmis :** L'algorithme de colonies de fourmis a été à l'origine principalement utilisé pour produire des solutions quasi-optimales au problème du voyageur de commerce, puis, plus généralement, aux problèmes d'optimisation combinatoire. On observe, depuis ses débuts, que son emploi se généralise à plusieurs domaines, depuis l'optimisation continue jusqu'à la classification, ou encore le traitement d'image.[50]
- **Les algorithmes à essaim de particules :** Les algorithmes d'optimisation par essaim de particules (PSO) ont été introduit en 1995 par Kennedy et Eberhart comme une alternative aux algorithmes génétiques standards. Ces algorithmes sont inspirés des essaims d'insectes (ou des bancs de poissons ou des nuées d'oiseaux) et de leurs mouvements coordonnés. En effet, tout comme ces animaux se déplacent en groupe pour trouver de la nourriture ou éviter les prédateurs, les algorithmes à essaim de particules recherchent des solutions pour un problème d'optimisation. Les individus de l'algorithme sont appelés particules et la population est appelée essaim. [48]

3.4 Recuit simulé

3.4.1 Principe de base

La méthode de recuit simulé ou « simulated annealing »[51, 52] est un algorithme d'optimisation souvent utilisé lorsque le calcul de la solution optimale exacte demanderait un temps de calcul trop important.

Historiquement, cette technique tire son nom et son inspiration de pratiques issues de la thermodynamique et plus particulièrement, de la façon dont les métaux sont chauffés puis refroidis. Ce processus utilisé en métallurgie pour améliorer la qualité d'un solide cherche un état d'énergie minimale qui correspond à une structure stable du solide. En partant d'une haute température à laquelle le solide est devenu liquide, la phase de refroidissement conduit la matière liquide à

retrouver sa forme solide par une diminution progressive de la température. Chaque température est maintenue jusqu'à ce que la matière trouve un équilibre thermodynamique. Quand la température tend vers zéro, seules les transitions d'un état à un état d'énergie plus faible sont possibles.

Metropolis et al. [53] furent les premiers à implémenter, dès 1953, ce type de principe dans le calcul numérique. Ils utilisent une méthode stochastique pour générer une suite d'états successifs du système en partant d'un état initial donné. Tout nouvel état est obtenu en faisant subir un déplacement (une perturbation) aléatoire à un atome quelconque. Soit ΔE la différence d'énergie occasionnée par une telle perturbation. Le nouvel état est accepté si l'énergie du système diminue ($\Delta E \leq 0s$). Sinon, il est accepté avec une probabilité définie par : $p(\Delta E, T) = e^{(-\Delta E/(C_b * T))}$ où T est la température du système et C_b une constante physique connue sous le nom de *constante de Boltzmann*.

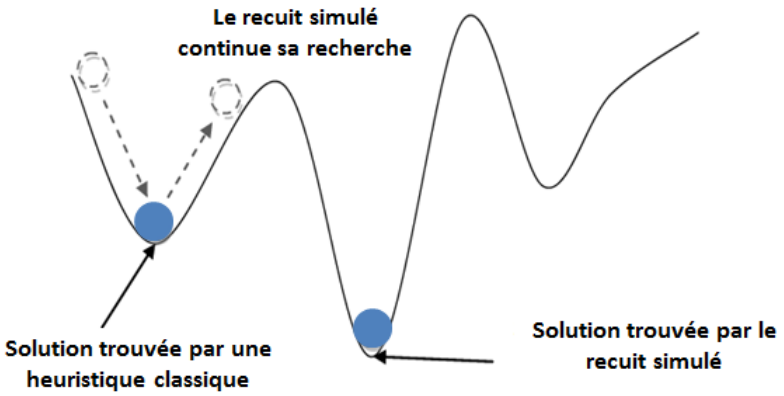

Figure 3.7 – Comparaison entre le recuit simulé et une heuristique classique

3.4.2 Algorithme

Le recuit simulé applique itérativement l'algorithme de Metropolis [1], pour engendrer une séquence de configurations qui tendent vers l'équilibre thermodynamique :

1. l'algorithme a été nommé d'après Nicholas Metropolis, qui avec Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller et Edward Teller rédigea l'article fondateur de 1953, « *Equations of State Calculations by Fast Computing Machine* »proposant l'algorithme pour le cas spécifique de la distribution de Boltzmann. Keith W. Hastings l'étendit au cas plus général en 1970.

engendrer une configuration (solution) initiale s0 de s: S←S0
Initialiser la température T
Répéter
    Voisinnage s' de s
    Calculer ΔE=f(s)-f(s') /* f fonction objective à maximiser
    Si ΔE<0 alors
        s←s'
    Sinon
        tirer u∈[0,1]
            si u< exp(-ΔE/T) alors s←s'
    Fsi
    Réduire la température T (T ←αT) (0.5<α<0.9999)
Jusqu'à(critère d'arrêt)
Retourner(S)

Figure 3.8 – Algorithme de recuit simulé

Une recherche de recuit simulé accepte n'importe quelles nouvelles solutions qui sont évaluées comme des solutions supérieures, mais elle accepte aussi les changements négatifs de qualité avec une probabilité qui dépend de la taille de diminution dans la qualité et la valeur courante de la température. La température commence à une haute valeur, idéalement assez haute que n'importe quelle solution inférieure aura presque une chance de 100 % d'être acceptée, et le processus suit son cours.[54]

3.4.3 Pseudo-code

La figure suivante donne un pseudo-code du recuit simulé :

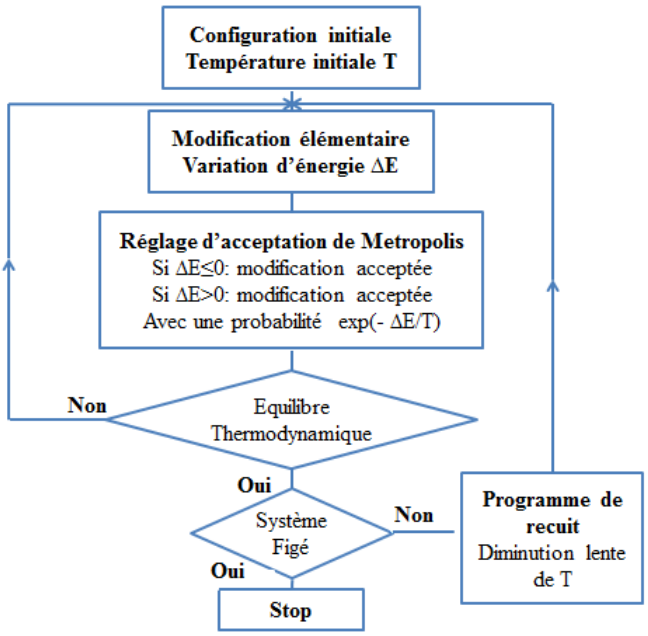

Figure 3.9 – Organigramme de l'algorithme du recuit simulé

3.4.4 Domaines d'application

la méthode du recuit simulé peut être utilisée pour résoudre plusieurs problèmes d'optimisation. En effet, par rapport aux autres algorithmes d'optimisation globale, il est perçu que cette méthode est plus facile à mettre en œuvre et moins encline à être piégée dans les optimums locaux.
Parmi les domaines d'applications les plus connus on peut citer :

- La conception des circuits intégrés (problème de placement et de répartition).
- Le routage des paquets dans les réseaux.
- La segmentation d'images.
- Le problème du voyageur de commerce.
- Le problème du sac à dos.
- ...etc.

3.4.5 Avantages et Inconvénients

Comme pour toute métaheuristique, le recuit simulé offre plusieurs avantages, et au même temps souffre de plusieurs défauts. Dans cette section on cite quelques avantages et inconvénients de cette méthode d'optimisation combinatoire.[55]

a) Avantages

Les principaux avantages de l'algorithme de recuit simulé (SA) peuvent être résumés comme suit :

- il peut traiter des systèmes arbitraires et des fonctions objectives, en termes de modèles hautement non linéaires et multimodaux, données bruitées ou soumises à de fortes contraintes.
- il converge vers un optimum global lorsque le nombre d'itérations tend vers l'infini (assez grand).
- il fournit des bonnes solutions pour la majorité des problèmes d'optimisation.
- il est relativement facile à mettre en œuvre et moins sensible à la taille de problème.
- c'est une technique d'optimisation généralisée, car elle ne repose pas sur toutes les propriétés restrictives du modèle.

b) Inconvénients

Il existe plusieurs défauts, parmi eux on peut citer les suivants :

- nombre important de paramètres(température initiale, taux de décroissance de la température, durée des paliers de la température, critère d'arrêt du programme).
- réglage souvent empirique des paramètres.
- temps de calcul excessif dans certaines application (problème de performance).
- l'impossibilité de savoir si la solution trouvée est optimale.

3.5 Recherche tabou

3.5.1 Principe de base

La méthode de recherche tabou ou « Tabu Search (TS) »[56] est une métaheuristique qui guide une procédure de recherche locale pour explorer l'espace de la solution au-delà de l'optimalité locale.

Cette méthode de recherche est basée sur deux principes : la mémoire adaptative et l'exploration réactive :[57]

- La fonction de mémoire adaptative de TS permet la mise en œuvre de procédures capables d'exploiter l'espace de solution de manière économique et efficace et cela en évitant les retours en arrières. Comme les choix locaux sont guidés par les informations recueillies lors de la recherche, TS contraste avec les conceptions sans mémoire qui s'appuient fortement sur des processus semi-aléatoires qui implémentent une forme d'échantillonnage.
- L'accent mis sur l'exploration réactive dans TS, que ce soit dans un déterministe ou probabiliste mise en œuvre, découle de la supposition qu'un mauvais choix stratégique peut souvent fournir plus d'informations qu'un bon choix aléatoire.

Contrairement aux autres procédures de recherche locales simples, TS est déterministe [2] où, à chaque itération, la meilleure solution de voisinage non interdite de la solution actuelle est sélectionnée, même si elle conduit à une pire solution par rapport à la fonction objectif. TS peut ainsi échapper aux optima locaux. La recherche s'arrête après un nombre fixe d'itérations ou un nombre maximum d'itérations continues sans améliorations de la solution la plus connue.[58]

3.5.2 Algorithme

L'algorithme de recherche tabou peut être résumé comme suit :

```
engendrer une configuration (solution) initiale s0 de s: S←S0
Répéter
    Voisinage s' de s n'appartient pas à la liste tabou
    Calculer ΔE=f(s)-f(s')  /* f fonction objective à maximiser
    Si ΔE<0 alors
        s←s'
    Fsi
    mettre à jour la liste tabou
Jusqu'à(critère d'arrêt)
Retourner(S)
```

Figure 3.10 – Algorithme de recherche tabou

2. Un algorithme est dit déterministe, s'il se comporte toujours de la même façon lorsqu'il est appliqué sur une instance donnée du problème

3.5.3 Pseudo-code

La figure suivante illustre la structure générale de la méthode recherche tabou :

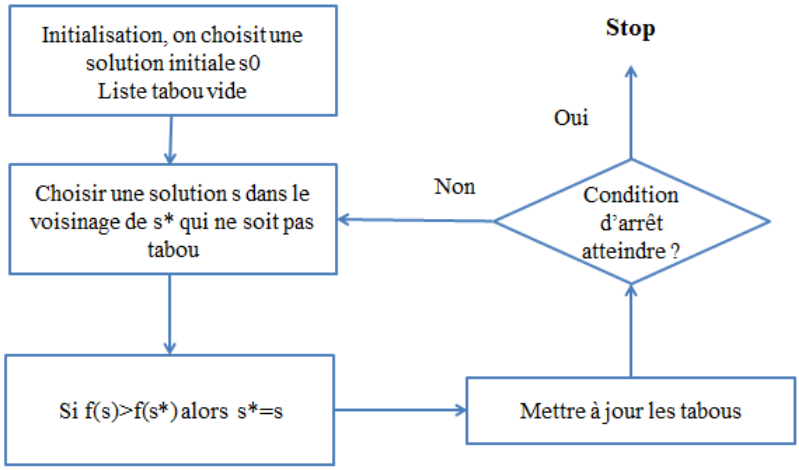

Figure 3.11 – Structure générale de la méthode recherche tabou

3.5.4 Domaines d'application

La méthode de recherche tabou est utilisée pour résoudre des problèmes complexes et/ou de très grande taille (NP-complet). Vu l'importance de cette technique de recherche, il existe plusieurs domaines qui s'intéressent à celui-ci. Parmi eux, on peut citer les suivants :

- Problèmes de transport.
- Planification et ordonnancement.
- Optimisation de la production et gestion des inventaires.
- Optimisation de graphes.
- Logique et intelligence artificielle.
- Télécommunications.
- Optimisation de structures.
- ...etc.

3.5.5 Avantages et Inconvénients

La méthode de recherche tabou fournit plusieurs avantages mais elle souffre aussi de beaucoup d'inconvénients. Les avantages et les inconvénients les plus importants sont les suivants :

a) Avantages

On peut résumer les avantages de la technique de recherche tabou dans deux points principaux qui sont :

- Grande efficacité.
- Fonctionnement simple à comprendre.

b) Inconvénients

Il existe plusieurs inconvénients pour la méthode RT, parmi eux :

- Paramètres peu intuitifs.
- Demande en ressources importantes si la liste des tabous est trop imposante.
- Aucune démonstration de la convergence.

3.6 Conclusion

A travers ce chapitre, nous avons abordé les définitions essentielles à la compréhension de travail du point de vue optimisation. Nous avons pu voir qu'un grand nombre de méthodes d'optimisation existe pour résoudre un problème combinatoire. Ces méthodes peuvent être exactes ou bien approchées où on distingue deux grandes familles : les heuristiques et les métaheuristiques qui sont détaillées dans la deuxième partie de ce chapitre.

Enfin, nous avons présenté deux techniques largement utilisées pour résoudre les problèmes d'optimisation qui sont le recuit simulé et la recherche tabou. Alors, on a donné pour chacune de ces méthodes sa définition, son algorithme, ses domaines d'application et nous avons terminé par ses avantages et inconvénients. Ces méthode seront utilisées dans ce travail pour optimiser les valeurs des poids qui relient les neurones d'une couche à l'autre dans le but d'augmenter la performance de classification du modèle MLP.

CHAPITRE 4

NOTRE MODÈLE DE DÉTECTION D'INTRUSIONS

4.1 Introduction

En fait, les réseaux de neurones sont considérés comme étant l'une des techniques les plus puissantes et les plus utilisées en classification des données, notamment le modèle MLP qu'on a déjà décrit dans le deuxième chapitre. Cette technique qui a la possibilité de classifier les données en classes, chacune comporte les objets les plus similaires entre eux, sera utilisée dans notre modèle pour classifier le trafic du réseau TCP/IP en trafic normal ou en trafic malveillant. La génération de notre modèle est basée sur la base NSL-KDD issue du jeu de données DARPA 1998.

Dans ce chapitre, nous présentons les différentes étapes du développement de notre travail en commençant par une simple description de la base de données NSL-KDD. Où, nous allons détailler les différentes phases de prétraitement qu'on a appliqué sur cette base, notamment la numérisation, la normalisation et la sélection des meilleurs attributs. Ensuite, nous allons présenter les techniques utilisées pour entrainer et tester notre modèle tout en illustrant les techniques d'optimisation que nous avons appliqué pour augmenter la performance de ce dernier.

4.2 Présentation de la base NSL-KDD

4.2.1 Historique

Le dataset NSL-KDD repose sur un autre dataset populaire, le KDD Cup 99, qui est à son tour extrait d'une autre base de données d'évaluation de systèmes de détection d'intrusions, DARPA [1]. Le KDD99 fut créé en 1999 pour une compétition de machine learning [60]. Le but de cette compétition était de classifier correctement des connexions réseau en 5 catégories : normal, déni de service (DoS), sonde réseau (probe), distant à local (R2L – Remote to Local) et utilisateur à root (U2R – User to Root).

NSL-KDD a été créé en 2009 pour résoudre certains problèmes inhérents à KDD Cup 99 [61]. Il reprend ainsi les mêmes données que ce dernier, mais le modifie grandement pour apporter ses corrections. Ainsi, les connexions redondantes ou dupliquées, qui composaient de 75%

1. Le premier événement IDS parrainé par la DARPA a été réalisé par le MIT Lincoln LAB en 1998 [59]. Dans cet événement DARPA, un scénario d'attaque à la base de l'Air-Force est simulé

à 78% du dataset, ont été supprimées.

Figure 4.1 – La relation entre les datasets

4.2.2 Description du dataset NSL-KDD

Bien qu'il soit assez vieux et non une représentation parfaite des réseaux réels existants, il est en continu un indice qui est utilisé pour comparer les systèmes de détection d'intrusions dans les recherches communes. Dans la littérature la plus récente [62, 63, 64], tous les chercheurs utilisent le NSL-KDD comme ensemble de données de référence.[65]

La base de données NSL-KDD contient des enregistrements de connexion TCP/IP (125973 pour l'apprentissage et 22544 pour le test), dont chaque enregistrement est constitué de 41 attributs caractérisant la connexion, et un attribut indiquant la nature de connexion s'il s'agit d'une attaque ou non. Les quatre catégories d'attaques existantes dans cette base sont :

- Dénis de Services
- Probing
- User to Root
- Remote to User

Ces types d'attaque sont détaillés dans la section 1.2.7 et le tableau suivant montre l'ensemble des attaques peuvent être inclut dans chaque type.

La classe	*Types d'attaques*
DoS	Apache2, Back, Land, Mailbomb, Neptune, Pod, Processtable, Smurf, Teardrop, Udpstrom
Probe	Mscan, Ipsweep, Nmap, Portsweep, Saint, Satan
R2L	Ftp_write, Guess_passwd, Imap, Multihop, Named, Phf, Dict, Snmpguess, Spy, Sqlattack, Warezclient, Warezmaster, Xlock, Xsnoop, Guest
U2R	Buffer_overflow, Httptunnel, Loadmodule, Xterm, Perl, Ps, Rootkit

Tableau 4.1 – Regroupement des attaques dans la base NSL-KDD

La distribution des connexions de la base NSL-KDD est illustrée dans le tableau ci-dessous :

Type	*Base d'apprentissage*		*Base de test*	
	Nbr connexions	*Pourcentage*	*Nbr connexions*	*Pourcentage*
Normal	67343	53.46%	9711	43.07%
DoS	45927	36.46%	7591	33.67%
Probe	11656	09.25%	2421	10.74%
R2L	995	00.79%	2754	12.22%
U2R	52	00.04%	67	00.30%
Total	125973	100.0%	22544	100.0%

Tableau 4.2 – Distribution des données dans la base NSL-KDD
[65]

4.2.3 Contenu de la base de données NSL-KDD

Cette base contient quatre groupes de données qui sont : [66][65]

- **KDDTrain^{+}** : qui contient toutes les données d'apprentissage du dataset NSL-KDD.
- **KDDTrain^{+}_20Percent** : qui représente seulement 20% des données d'apprentissage.
- **KDDTest^{+}** : ce groupe de données représente les données du test de la base NSL-KDD.
- **KDDTest^{-21}** : c'est un sous-ensemble du $KDDTest^{+}$ qui n'inclut pas les enregistrements ayant un niveau de difficulté de 21 sur 21.

Type de données	KDDTrain^{+}	KDDTrain^{+}_20Percent	KDDTest^{+}	KDDTest^{-21}
Nbr d' enregistrements	125,973	25192	22,554	11850

Tableau 4.3 – Contenu de la base NSL-KDD

4.2.4 Attributs de la base NSL-KDD

Dans la base NSL-KDD, chaque enregistrement est constitué de 41 attributs qui sont illustrés dans le tableau A.1 de l'annexe. Ces attributs peuvent être classés en quatre catégories :[66]

- **Les attributs de base :**qui sont les attributs de base des connexions TCP, telles que la durée, les hôtes source et destination, port et flag.
- **Les attributs du trafic :**qui sont les attributs calculés à l'aide d'une fenêtre de temps de deux secondes, tels que le nombre de connexions vers la même machine.

- **Les caractéristiques du contenu :**ces attributs sont construits à partir de la charge utile (Data) des paquets du trafic tels que le nombre d'échec de connexion et le nombre d'accès aux fichiers de contrôle.
- **Les caractéristiques de l'hôte :**ce sont les attributs conçus pour évaluer les attaques qui durent plus de deux secondes.

4.3 Processus de génération du modèle de classification

Dans ce travail, où on souhaite élaborer un modèle de détection d'intrusions puissant capable de classifier les connexions TCP/IP en deux catégories : normale ou attaque, on doit passer par les étapes principales que tout modèle de classification doit les suivre. Ces étapes sont résumées dans trois phases principales illustrées dans la figure 4.2 qui sont : le prétraitement, l'apprentissage et enfin la phase de test.

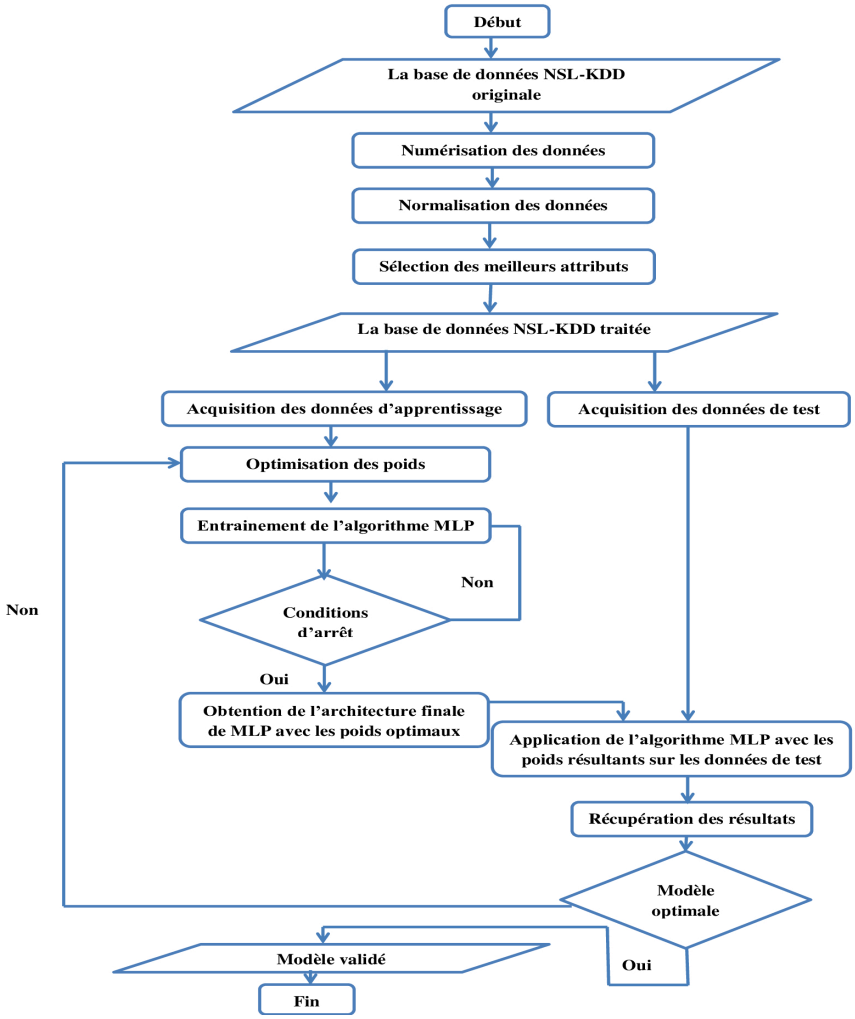

Figure 4.2 – Organigramme de fonctionnement de notre modèle de détection d'intrusion

4.3.1 Prétraitement des données

Le processus de prétraitement modifie le dataset pour le rendre lisible par certains algorithmes, comme le réseau de neurones. Cette phase est donc très importante pour générer un modèle de classification fiable et cohérent. En effet, trois étapes ont été réalisées sur la base NSL-KDD avant de l'exploiter. Ces étapes sont : la numérisation, la normalisation et enfin la sélection des attributs.

a) Numérisation des données

Afin d'adapter le dataset NSL-KDD avec les modèles de réseaux de neurones qui n'acceptent que des attributs numériques, et comme cette base contient 3 attributs alphabétiques qui sont : protocol_type, service et flag parmi ses 41 attributs, il est nécessaire de transformer toutes ces données catégoriques en données numériques via un encodage précis. Dans notre cas, chaque valeur alphabétique est remplacée par son entier équivalent, c'est-à-dire, s'il existe N valeurs possibles pour l'attribut X, ces valeurs sont remplacées par des valeurs entières compris entre 0 et N-1.

Si on prend par exemple le cas de l'attribut *protocol-type* qui peut prendre trois valeurs : tcp, udp ou bien icmp, le résultat de numérisation de cet attribut sera comme suit :

Avant numérisation	*Après numérisation*
TCP	0
UDP	1
ICMP	2

Tableau 4.4 – Exemple de numérisation

b) Normalisation des données

Pour garantir l'efficacité et améliorer les performances du modèle généré, il est très important d'ajuster les valeurs numériques obtenues après la phase de numérisation, puisqu'elles sont très variées et constituent un grand intervalle. Par exemple, certains attributs comme src_bytes et dst_types prennent des grandes valeurs tandis que d'autres comme serror_rate et same_srvrate ne prennent que des petites valeurs. Donc, pour éviter ce genre de problème, on est obligé d'effectuer une opération de transformation sur les données de la base NSL-KDD en utilisant une fonction bien choisie. Dans notre cas, la fonction utilisée est la fonction Min-Max décrite comme suit :

$$val_{nouv} = \frac{val_{anc} - Min_{anc}}{Max_{anc} - Min_{anc}}$$

Où :

— val_{anc} :est la valeur à normaliser.

— val_{nouv} :est la valeur après la normalisation.

— Min_{anc} :est la limite inférieure de l'intervalle à que val_{anc} appartient.

— Max_{anc} :est la limite supérieure de l'intervalle à que val_{anc} appartient.

En appliquant cette formule sur les données de la base NSL-KDD, on obtiendra une base normalisée dont toutes ses valeurs sont compris entre 0 et 1.

c) Sélection des meilleurs attributs

Comme nous avons vu précédemment, la base NSL-KDD contient 41 attributs qui sont censés être dans le cas normal les entrées de notre modèle neuronal. Cependant, si on utilise tout ces attributs, on risque d'affecter les performances de notre modèle de détection d'intrusions en terme de ressources utilisées ainsi que le temps de réponse. Donc, pour garantir l'efficacité de notre modèle, on doit sélectionner les attributs les plus significatives et les plus pertinents parmi tout ces attributs.

Il existe plusieurs techniques pour réaliser cette tâche, parmi-elles, celle qu'on a utilisé dans ce travail où on a choisi les attributs ayant les meilleurs gains d'informations parmi l'ensemble totale d'attributs et cela en suivant les étapes suivantes :

Étape 1 : Calcul de l'entropie pour chaque attribut en utilisant la formule suivante :

$$H(x_i) = -\sum_{j=1}^{n} p(x_j|c_1)\log_2 p(x_j|c_1) + p(x_j|c_2)\log_2 p(x_j|c_2)$$

Où :

— c_1, c_2 : dénotent les deux classes de classification (normale, attaque).

— x_i : représente un attribut.

— x_j : représente une valeur particulière de l'attribut x_i.

— n : dénote le nombre de valeurs de l'attribut x_i.

— p : la probabilité.

Étape 2 : Calcul de gain pour chaque attribut à l'aide la formule suivante :

$$Gain = Entropie_E - H(x_i)$$

Sachant que :

$$Entropie_E = \frac{nb_{normal}}{nb_{connexion}}\log_2(\frac{nb_{normal}}{nb_{connexion}}) + \frac{nb_{attack}}{nb_{connexion}}\log_2(\frac{nb_{attack}}{nb_{connexion}})$$

Où :

— nb_{normal} :présente le nombre des connexions qui sont classifiées comme normal.

— nb_{attack} :présente le nombre des connexions qui sont classifiées comme une tentative d'attaque.

— $nb_{connexion}$:présente le nombre total des connexions dans la base d'apprentissage..

Étape 3 : Élimination des attributs ayant un gain inférieur à un seuil donné (0.5 dans notre cas).

Étape 4 : Génération du modèle de classification en se basant sur l'ensemble des attributs restants après la troisième étape. Les attributs sélectionnées dans notre cas sont les suivants :

Numéro	*Attribut*	*Gain*
3	Service	0.672
4	Flag	0.519
5	Src-bytes	0.827
6	Dst-bytes	0.648
29	Same-srv-rate	0.510
30	Diff-srv-rate	0.519

Tableau 4.5 – Résultat de la phase de sélection d'attributs

4.3.2 Apprentissage et génération du modèle de classification

L'objectif de ce travail est d'établir un modèle de détection d'intrusions comportemental qui se base sur le perceptron multi-couches (MLP). Ce type de réseaux de neurones, qui contient un ensemble de neurones répartis en couches, utilise comme algorithme d'apprentissage celle de rétro-propagation du gradient, et puisque on a besoin de classifier nos données seulement en deux classes qui sont attaque ou normale, la couche de sortie va comporter un seul neurone qui peut prendre deux valeurs :

— 0 : trafic normal.

— 1 : attaque qui regroupe tous les types d'attaques : DOS, U2R, Probe et R2L.

La phase d'apprentissage consiste à entrainer le MLP en utilisant une base de données dite d'apprentissage. Dans notre cas, on utilise la base KDDTrain^{+}_20 Percent qui contient 25192 enregistrements comme base d'apprentissage. Ce processus se répète jusqu'à l'obtention d'une erreur quadratique inférieure à un seuil donné. A ce moment là, les poids optimaux obtenus après les mises à jour effectuées pendant l'entrainement seront utilisés pour calculer la valeur de sortie pour chaque enregistrement de la base de test.

4.3.3 Test et évaluation du modèle généré

Dans cette étape, on veut estimer la qualité de notre modèle de détection d'intrusions par rapport aux autres modèles déjà réalisés en se basant sur une base de données dite de test où toutes les instances sont déjà classées. En effet, pour comparer notre modèle avec les autres on a besoin de calculer quelques métriques lors de la phase de test telles que la matrice de confusion, la précision, le rappel...etc. Ces métriques sont définies comme suit : [67] [68]

- **La matrice de confusion :**est une matrice qui rassemble en lignes les observations et en colonnes les prédictions. Les éléments de la matrice représentent le nombre d'exemples correspondants à chaque cas.

		Classe détectée (prédite)	
		Normale	**Attaque**
Classe réelle	**Normale**	Vrai négatif **TN**(True Negative)	Faux positif **FP** (False Positive)
	Attaque	aux négatif **FN** (False Negative)	Vrai positif **TP** (True Positive)

Tableau 4.6 – Matrice de confusion

- Un vrai négatif (TN) est une activité normale correctement classée lors de test.
- Un faux positif (FP) est une activité normale mal classée pendant le test, c'est-à-dire, elle est considérée comme attaque.
- Un faux négatif (FN) est une attaque non détectée pendant le test, c'est-à-dire, elle est considérée comme trafic normal.
- Un vrai positif (TP) est une attaque correctement détectée.

- **L'exactitude (Accuracy) ou le taux de réussite :** c'est le rapport entre les enregistrements bien classés et la totalité d'enregistrements de test. Il sert à indiquer la façon dont la technique de détection est correcte.

$$Exactitude = \frac{TP + TN}{TP + TN + FP + FN} * 100\%$$

- **Le rappel :** c'est le taux des intrusions correctement détectées par rapport au nombre total d'intrusions. Il est calculé à partir de la formule suivante :

$$Rappel = \frac{TP}{TP + FN} * 100\%$$

- **La précision :**appelée aussi le taux de reconnaissance, est la proportion de prédictions de positifs qui sont en effet des positifs.

$$Prcision = \frac{TP}{TP + FP} * 100\%$$

- **Le taux des fausses alertes (taux des faux positifs) :** est la proportion des négatifs incorrectement détectés comme des positifs.

$$Taux - des - fausses - alertes = \frac{FP}{FP + TN} * 100\%$$

- **F-mesure (Moyenne harmonique) :** c'est une métrique qui combine la précision et le rappel en un nombre compris entre 0 et 1. Elle donne une évaluation de synthèse de la classification.

$$F - mesure = \frac{2 * Prcision * Rappel}{Prcision + Rappel}$$

4.3.4 Optimisation du modèle

Dans cette phase, nous nous intéressons par l'optimisation des poids du réseau en utilisant les algorithmes cités dans le troisième chapitre (le recuit simulé et la recherche tabou). Le principe de ces algorithmes est de générer une solution optimale à partir d'une solution initiale choisie aléatoirement en se basant sur le principe de voisinage qui est implémenté dans notre cas à l'aide de la méthode java *shuffle* qui fonctionne en permutant les éléments d'une liste spécifiée de telle sorte qu'on a obtenu une nouvelle solution extraite de la solution initiale.

a) Objectif

Dans ce travail, nous cherchons à maximiser le taux de réussite du modèle de détection d'intrusions établi, donc la fonction objectif sera comme suit :

$$f = Max(taux - de - russite)$$

b) Solution initiale

Dans notre cas, la solution initiale c'est un tableau des poids comme il est illustré dans cette figure :

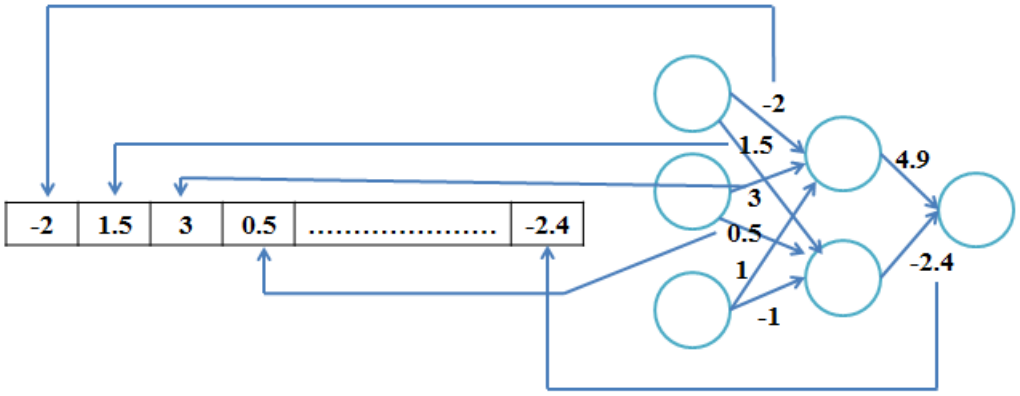

Figure 4.3 – Préparation de la solution initiale

c) Algorithme détaillé

Pour atteindre notre objectif, nous utilisons l'algorithme suivant :

```
Début
    Acquisition des données ;
    Numérisation ;
    Normalisation ;
    Sélection d'attributs ;
    Solution_initiale  S0=P0P1P2…….Pn ;  // n : le nombre des arcs du réseau
    Solution_finale=S0 ;
    f= l'exactitude (S0) ;
    Répéter
        S'= shuffle(Solution_finale) ; // pour la recherche tabou on doit vérifier
                                       que cette solution n'appartient pas à la liste taboue
        ΔE= f(Solution_finale)-f(S') ;
        Si (ΔE <0)
            Solution_finale= S' ;
        Sinon
            Tirer un nombre aléatoire u ∈ [0,1] ;
                Si  u<exp (-ΔE/T) alors ; //T : température initialisée à 1000 ;
                    Solution_finale=S' ;
                FSi
        FSI
        T=T*α ; // refroidissement
    Jusqu'à (critère d'arrêt) ;  // T<0.001(RS) ou bien Nbr_itération>MAX (RT)
    Entrainer _modèle (base d'apprentissage) ; //On utilise le vecteur des poids optimaux
    Tester_modèle(base de test) ;
    Extraction des résultats (matrice de confusion, rappel, exactitude,….) ;
Fin
```

Pour le recuit simulé

Figure 4.4 – Algorithme détaillé

4.4 Conclusion

Dans ce chapitre, nous avons introduit la base NSL-KDD qui a été utilisée pour entrainer et aussi pour tester notre modèle de détection d'intrusions qui est construit à base d'un réseau de neurones multi-couches. Tout d'abord, nous avons présenté l'historique de cette base ainsi que sa description, son contenu et ses différents attributs. Ensuite, nous avons détaillé notre processus de génération du modèle de classification en expliquant ses différentes phases notamment le prétraitement, l'apprentissage et le test.

Enfin, nous avons terminé ce chapitre avec une illustration de la phase la plus importante dans notre travail qui est l'optimisation du modèle. Où, nous avons montré l'utilisation du recuit simulé et recherche tabou dans notre problème de détection d'intrusions à base de réseau de neurones tout en donnant l'algorithme général utilisé.