N° d'ordre : 10/2012

UNIVERSITE IBN ZOHR

CENTRE DES ETUDES DOCTORALES IBN ZOHR

Formation Doctorale : Mathématiques, Informatique et Applications

Faculté des Sciences Agadir

Equipe Modélisation Mathématique et Simulations (EMMS)

THESE

Présentée par

Mohamed BENADDY

Pour l'obtention du grade de

Docteur de l'Université Ibn Zohr

Spécialité : Informatique

Modèlisation et prédiction de la fiabilité des logiciels et des réseaux

Directeur de thèse : **M. Mohamed WAKRIM**

Co-directeur : **M. Sultan Mohammed Hammadi Aljahdali**

Soutenue le 15 Décembre 2012

Devant la commission d'examen composée de :

M. Hassan DOUZI	Université Ibn Zohr -FS- Agadir	Président
M. Karim AFDEL	Université Ibn Zohr -FS- Agadir	Rapporteur
M. Chihab Dine CHERKAOUI	Université Ibn Zohr -ENCG- Agadir	Rapporteur
M. Mohammed BOUHORMA	Université Abdelmalek Essaâdi -FST- Tanger	Rapporteur
M. Mohamed WAKRIM	Université Ibn Zohr -FS- Agadir	Directeur de thèse
M. Sultan M. H. ALJAHDALI	Taif University - Saudi Arabia	Co-directeur de thèse
M. Mohamed HACHIMI	Université Ibn Zohr -FSJES- Agadir	Examinateur

A mes parents.
A ma femme.
A Adam.
A mes sœurs et frères.

Ce document a été préparé à l'aide de l'éditeur de texte GNU Kile™ et de l'environnement de composition typographique LaTeX 2_ε sous le système d'exploitation Linux Ubuntu™.

Remerciements

Arrivé au bout de cette thèse ne m'aurait pas été possible sans l'inestimable collaboration de tant de personnes qui m'ont soutenu et encouragé même dans les moments les plus difficiles. Il m'est aujourd'hui difficile de n'en oublier aucune. Je m'excuse donc par avance des oublis éventuels.

Les mots ne suffiraient pas pour exprimer ma sincère et profonde gratitude à mon directeur de thèse, Monsieur Mohamed WAKRIM, professeur à la faculté des sciences d'Agadir, qui par sa grande expertise, ses conseils judicieux et son appui moral, a su me donner de son savoir et de son mieux pour que je devienne un docteur. Sa grande disponibilité et sa patience infinie m'ont permis d'effectuer une formation doctorale enrichissante. Merci de m'avoir supporté.... Faire ma thèse sous votre direction a été pour moi un grand honneur.

De même je voudrais remercier Monsieur Sultan Mohammed Hammadi Aljahdali, Dean of College of Computers & Info. Systems, Taif University (Kingdom of Saudi Arabia) pour avoir co-encadré cette thèse. Qu'il trouve ici l'expression de ma profonde gratitude pour l'intérêt qu'il a porté à mes travaux, son soutien et ses encouragements. Qu'il trouve ici aussi mon profond respect pour son rigueur scientifique.

Que Monsieur Karim AFDEL, professeur à la Fculté des Sciences d'Agadir, Monsieur Chihab Dine CHERKAOUI, professeur à l'ENCG d'Agadir ainsi que Monsieur Mohammed BOUHORMA, professeur à la FST de Tanger, trouvent ici ma reconnaissance pour l'honneur qu'ils m'ont fait en acceptant d'être les rapporteurs de ce travail.

Je remercie également Monsieur Mohamed HACHIMI, professeur à la Faculté des sciences juridiques économiques et sociales d'Agadir, Monsieur Hassan DOUZI, professeur à faculté des sciences d'Agadir pour leur participation à l'examen et au jury de cette thèse.

Mes vifs remerciements vont également à Madame L. Amina Idrissi Hassani, la directrice du Centre des Etudes Doctorales Ibn Zohr (CEDoc Ibn Zohr) ainsi que Omar Hasnaoui, administrateur au CEDoc Ibn Zohr, sans oublié le reste du corps administratif du Centre.

Trouver un article publié n'est pas assez simple, merci à Monsieur Abderrahim GHADI pour votre soutien dans les moments difficiles.

Mes remerciements s'adressent à mes parents ma femme mes sœurs et frères, sans leur soutien moral, affectif je n'aurais pas pu achever ce travail. Je ne remercierai jamais assez ces personnes qui ont fait tout ce que je suis.

Merci à tous mes amis (Nouh Izem, Mohamed EL Hajji, Youssef Essaady, Mustapha Amrouche, Taher Zaki, Khalid Housni ...) qui m'ont accompagné et aidé avec leur esprit à la fois solidaire et critique. Leur soutien infini, leur amitié et pour les discussions sur plein de sujets...

Enfin, je ne pourrais terminer cette liste sans adresser des remerciements particuliers, à des personnes particulières qui m'ont soutenu dans l'ombre...

Résumé

Modélisation et prédiction de la fiabilité des logiciels et des réseaux

L'objectif de cette thèse est de proposer d'une part, des modèles de prédiction de la fiabilité des logiciels en terme du nombre de défaillances cumulé dans un logiciel et d'autre part des modèles pour l'évaluation de la fiabilité des réseaux ou tout autre système qui peut être modélisé sous forme d'un réseau orienté ou non.

Nous utilisons pour la prédiction de la fiabilité des logiciels ; les réseaux de neurones artificiels et le modèle d'auto régression linéaire, ces méthodes sont entraînées par un algorithme évolutionniste d'une part et d'autre part par le recuit simulé. Ces modèles hybrides sont qualifiés de modèles non paramétriques, et donnent des résultats satisfaisants comparés à d'autres modèles.

En ce qui concerne l'évaluation de la fiabilité des réseaux nous proposons deux algorithmes efficaces pour énumérer l'ensemble des chemins minimaux dans des réseaux orientés et des coupes minimales dans des réseaux non orientés. Ces algorithmes sont couplés avec la méthode d'inclusion-exclusion pour évaluer la fiabilité d'un réseau. Ces deux algorithmes sont testés avec un ensemble des réseaux proposés dans la littérature et donnent des résultats intéressants en terme d'exactitude des résultats et en terme du temps d'exécution.

Mots clés :

Fiabilité des logiciels, Fiabilité des réseaux, Réseaux de neurones, Modèle de régression linéaire, Algorithmes évolutionnistes, Recuit simulé, Chemins minimaux d'un graphe, Coupes minimales d'un graphe, Méthode d'inclusion-exclusion

Abstract

Modeling and Prediction of Software Reliability and Network Reliability

This dissertation research attempts to explore on the one hand, models for software reliability prediction in term of cumulative failure in the software, on the other hand, models for networks reliability evaluation or any other system which can be modeled as a network (directed or not).

Artificial neural networks and the Auto-regression methods have been used to predict the cumulative software failure. These methods are trained by evolutionary and simulated annealing algorithms. The developed approaches are qualified as non-parametric models. Numerical results show that both the goodness-of-fit and the next-step-predictability of our proposed approaches have greater accuracy in predicting software cumulative failure compared to other approaches.

For evaluating the reliability of the networks we propose two efficient algorithms. The first one for enumerating minimal pathsets in directed networks and the second one for enumerating minimal cutsets in non directed networks. These algorithms are coupled with the inclusion-exclusion principle for computing the reliability of a network. Both algorithms are tested with a set of networks proposed in the literature and give interesting results in terms of accuracy and execution time.

Keywords :

Software Reliability, Network Reliability, Neural Networks, Auto-regression, Evolutionary Algorithms, Simulated Annealing, Graph Minimal Pathsets, Graph Minimal Cutsets, Inclusion-Exclusion Principle.

مُلَخَّص

نمذجة وَ تنبُّؤ موثُوقيّة البرمجيّات وَ الشّبكَات

تَهدِفُ هذهِ الاطروحةُ الَى تَطوِيرِ نَمَاذِج لِلتَّنَبُّؤ بِمَوثُوقِيَّة البَرمَجِيَّات في إِطَار التَّعطُّل المُتَرَاكِم في البَرنَامَج ، هذَا مِن جَانِب، وَ مِن جَانبٍ آخَرَ، اِلَى تَطوِير نَمَاذِج لِتَقيِيم مَوثُوقِيَّة الشّبكَات أَو أَيّ نِظَام يُمكِن نَمذَجَتُه عَلَى شَكل شَبَكَاتٍ (مُوَجَّهَة أَو غَير مُوَجَّهَة).

وَ قَد استَخدَمنَا الشّبَكَات العَصَبِيَّة آلِاصطِنَاعِيَّة وَ نَمُوذَج التَّرَاجُع الذَّاتِيّ لِلتَّنَبُّؤ بِالتَّعطُّل المُتَرَاكِم في بَرنَامَج مُعَيَّنٍ. كَمَا استَخدَمنَا خَوَارِزمِيَّةَ التَّطوُّرِ مِن جِهَة، وَ مِن جِهَةٍ أُخرَى خَوَارِزمِيَّة مُحَاكَاةِ الصَّلبِ لِتَدرِيبِ هَذِهِ النَّمَاذِج. وَ تُعتَبَر هَذِهِ الطَّرَائِق الهَجينَة نَمَاذِجًا غَير بَرَامِتِرِيَة. وَ تُبَيِّنُ النَّتَائِجُ المُحَصَّل علَيهَا عَلَى أَنَّ هَذِهِ النَّمَاذِج جِدّ دَقِيقَة وَ تُعطي نتَائِج جِدّ حَسَنَة مُقَارَنَةً مَعَ نَمَاذِج أُخرَى.

لِتَقيِيمِ مَوثُوقِيَّة الشَّبَكَات اقتَرحنَا خَوَارِزمِيَّتَانِ فَعَّالَتَانِ، الخَوَارِزمِيَّةُ الأَولَى لِتِعدَادِ المَسَارَات الصُّغرَى في الشَّبَكَات المُوَجَّهَة، وَ الثَّانِيَةُ لِتِعدَادِ التَّقطيعَات الصُّغرَى في الشّبكَات غَير المُوجّهة، وَ قَد تَمَّ إِستِخدَامُ كِلَاهُمَا اَلَى جَانِبِ أُسلوبِ الإِدرَاجِ وَ الإِستِبعَاد لِتقيِيمِ مَدَى مَوثُوقِيَّةِ الشَّبَكَات. هَاتَانِ الخَوَارِزمِيَّتَانِ تمَّ إِختِبَارُهُمَا عَلَى مَجمُوعَة مِنَ الشَّبَكَات المُقتَرَحَة ضِمن بَنك مَعلومَات أَكَادِمِيَّة وَ خَلُصَت إِلَى نتَائِج مُثيرة لِلآهتِمَام مِن حَيثُ دِقَّتِهَا وَ مُدَّة التَّنفِيذ.

الكَلِمَاتُ الرَّئِسِيَّة

مَوثُوقِيَّة البَرمَجِيَّات، مَوثُوقِيَّة الشَّبَكَات، الشّبَكَاتُ العَصَبِيَّةُ آلِاصطِنَاعِيَّةُ، نَمُوذَجُ التَّرَاجُع الذَّاتِيّ، خَوَارِزمِيَّةُ التَّطوُّرِ، خَوَارِزمِيَةُ مُحَاكَاةِ الصَّلبِ، المَسَارَات وَ التَّقطيعَات الصُّغرَى في الشَّبَكَات، أُسلوبُ الإِدرَاجِ وَ الإِستِبعَادِ.

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction générale

Sommaire

1.1 La fiabilité des logiciels

La modernisation de notre société durant les dernières décennies a conduit à une intégration massive des techniques informatiques dans la plupart des domaines de la vie courante. Cette intégration nous rend vulnérable si les systèmes mises en cause tombent en panne. Par conséquent les systèmes informatiques doivent être très fiables. La réduction fréquente du coût du matériel a rendu les techniques de redondances possibles, de sorte que les pannes matérielles sont de plus en plus tolérées. Au contraire les logiciels sont des systèmes complexes, par conséquent, les systèmes informatiques sont très sensibles aux défauts dans le logiciel. En outre les logiciels utilisés dans les systèmes informatiques sont trop larges et complexes, il est tout à fait susceptible de souffrir des défauts. Ainsi la fiabilité du logiciel est devenue un enjeux important qui concerne tant les concepteurs, les réalisateurs et les fournisseurs que les utilisateurs des services délivrés par ces logiciels.

Dans la dernière décennie du $20^{\text{ème}}$ et le début $21^{\text{ème}}$ siècle, les logiciels sont devenus la principale source de pannes signalées dans de nombreux systèmes. La défaillance d'un logiciel peut avoir des impacts catastrophiques sur son environnement par des conséquences de nature humaine ou bien économiques. En conséquence la littérature récente est pleine d'histoires d'horreurs de projets qui ont échoués, généralement à cause des défauts dans les logiciels.

Les erreurs dans les logiciels ont causé des défaillances spectaculaires, certaines avec des conséquences désastreuses, comme les exemples suivants [2]. Le 31 Mars 1986, un Boeing 727 de Mexicana Airlines s'est écrasé dans une montagne parce que le système logiciel

n'a pas négocier correctement la position de la montagne. Le 26 juin 1988, le nouveau modèle A320 d'Air France, livré juste deux jours avant, s'est écrasé dans les arbres dans un spectacle aérien près de Mulhouse en France en raison d'une défaillance de logiciel informatique tout en effectuant un passage à basse altitude. Trois passagers ont été tués.

Durant la période de 2 à 4 novembre 1988, un virus informatique infecte les logiciels dans les centres de recherche universitaires et du département de défense aux Etats Unis, causant des défaillances du système. Le 10 décembre 1990, la navette spatiale Coulumbia a été forcée de revenir sur terre en raison de problèmes logiciels. Le 17 Septembre 1991, une panne d'électricité à l'installation de commutation d'AT & T à New York interrompe le service à 10 millions d'utilisateurs de téléphone pendant 9 heures. Le problème était dû à la suppression de trois bits de code dans une mise à jour logicielle et le défaut de n'avoir pas testé le logiciel avant son installation dans le réseau public. Le 26 octobre 1992, le système de répartition assistée par ordinateur du Service d'Ambulance de Londres, qui gère plus de 5000 demandes quotidiennes dans le transport des patients dans un état critique, a échoué après l'installation. Cela a conduit à de graves conséquences pour de nombreux patients en état critique.

En 2004, la France a été confrontée en moins de 6 mois à 4 pannes majeures de réseaux informatiques [3] :

- 15-18 Juillet : le système de réservation Mosaic de la SNCF a été totalement bloqué pendant près de 4 jours, empêchant la vente et la réservation de billets. La panne s'est déclenchée quelques heures après la mise en place d'une nouvelle version de Mosaic. La défaillance a été causée par un programme de contrôle installé ponctuellement pour s'assurer de la qualité des transmissions entre le central de réservation et les postes de travail. Son fonctionnement, combiné à la forte charge de transmission de données liée à la période de grands départs, a saturé le système et provoqué l'ensemble de ces perturbations (communiqué de la SNCF). Le bug a provoqué de longues files d'attentes devant les guichets, mais n'a affecté les finances de la SNCF, car 90% des voyageurs de cette période avaient acheté leurs billets à l'avance.
- 30 et 31 Octobre : une panne de plus d'une journée est survenue sur le réseau fixe de France Télécom. 26 commutateurs sur 600 ont été affectés dans la région parisienne, le nord et l'ouest de la France. Quelques milliers d'appels ont été rejetés ou ne sont pas parvenus à leurs destinataires. La panne provenait d'une anomalie logicielle dans un équipement de traitement de la voix sur IP (VoIP) situé à Reims. Cette anomalie logicielle a provoqué des anormalités dans le formatage de la numérotation de certains appels, qui ont déclenché des protections de sécurité pour éviter une panne du réseau (communiqué de France Télécom).
- 17-18 Novembre : une panne de plus d'une journée est survenue sur le réseau mobile Bouygues Télécom. 1.37 millions de clients n'ont pas pu du tout utiliser leur portable

et 5.6 millions ont eu des difficultés pour accéder au réseau. La facture de panne s'est élevée à 16 millions d'euros : une journée de chiffre d'affaires perdue et l'équivalent en non facturé aux clients en compensation du désagrément subi. La panne provenait du dysfonctionnement de la base de données qui sert à repérer le mobile du client, lui permet d'acheminer ses appels et d'en recevoir. Deux serveurs centraux jumeaux sont tombés en panne au même moment, dans deux endroits différents. Les deux serveurs sont du matériel classique et largement utilisé par de nombreux opérateurs de téléphonie mobile avec, jusqu'à ce jour, une fiabilité sans faille. En fonctionnement normal, ils se secourent en prenant le relais l'un de l'autre. La panne est de nature exceptionnelle (communiqué de Bouygues Télécom).

- 3 Décembre : 800 terminaux de vente de la SNCF (sur 4000) ont été paralysés pendant plus d'une journée. La panne provenait d'un algorithme défectueux qui a progressivement contaminé les terminaux de vente en gare. Cet algorithme a pour objectif de définir la zone de travail informatique de la transaction de paiement (communiqué de la SNCF). Un nouveau bug du même type est survenu en novembre 2009.

Dans ces 4 cas, les problèmes ont été causés par des défaillances logicielles. Le site http ://www5.in.tum.de/~huckle/bugse.html et le livre [2] recensent une impressionnante collection de défaillances logicielles. Parmi celles-ci :

- 4 Juin 1996 : la fusée Ariane 5 a explosée à cause d'une faute de conception logicielle, et plus précisément un problème de réutilisation. Un programme de contrôle d'un gyroscope de la fusée (en ADA) avait été transféré sans modification d'Ariane 4 à Ariane 5. Or les trajectoires d'Ariane 5 sont sensiblement différentes de celles d'Ariane 4. Cela a provoqué un dépassement de mantisse qui a induit en erreur le calculateur de pilotage et fini par faire exploser le lanceur. Cette erreur a coûté 500 millions de dollars uniquement en matériel, sans parler des retards engendrés et des conséquences sur l'image de marque d'Arianespase.
- 8 Octobre 2005 : un défaut dans le système de contrôle en vol de la fusée russe Rockot provoque la perte du satellite CryoSat chargé d'étudier l'influence du réchauffement climatique sur la fonte des glaces polaires. Coût de la mission : 136 millions d'euros.
- Septembre 2007 : un bug dans Excel 2007 provoque des résultats de multiplication fantaisistes comme 77.1x850=100000 (au lieu de 65535).
- 2007 : les retards de livraisons de l'Airbus A380 sont en partie dus à un problème logiciel dont l'origine tient au fait que les sites d'Hambourg et Toulouse ont utilisé deux versions différentes du logiciel de CAO de Dassault Systèmes CATIA. Les pénalités versées aux compagnies aériennes se chiffrent à plusieurs milliards d'euros.
- Mars 2008 : Chrysler rappelle 25000 Jeeps Cherokee et Commander pour corriger un défaut dans le logiciel de transmission automatique.

- Janvier 2010 : un bug dans le logiciel de lecture des puces électroniques des cartes bancaires fabriquées par Gemalto a empêché 30 millions d'allemands de se servir de leurs cartes de crédit pendant près d'une semaine. Pour certains, ce "bug de l'an 2010" a largement surpassé le fameux "bug de l'an 2000".
- Le 29 février 2012, durant presque 8 heures, une partie de Windows Azure a été inaccessible. C'est la première fois que Azure subit une panne aussi massive. Environ 4-5% des services hostées étaient concernés par les problèmes. La cause de la panne est une mauvaise gestion du calendrier. Cette année 2012, bissextile, ayant la mauvaise idée de comporter un 29 février. Comme quoi le bug de l'an 2000 n'a pas servi pleinement de leçon.
- Une "panne logicielle" aux conséquences exceptionnelles a fortement perturbé pendant plus de douze heures entre vendredi 6 et samedi 7 juillet 2012 le réseau de téléphonie mobile d'Orange, au point de pousser l'opérateur télécoms historique à envisager une indemnisation de ses clients.

Ces pannes majeures et répétées ont sérieusement entamé la confiance du public dans la fiabilité des réseaux de télécommunication et des systèmes informatiques en générale. Dans de nombreux secteurs, on a pu constater que les défaillances sont de plus en plus fréquentes. En effet, dans un contexte fortement concurrentiel, les entreprises cherchent à réduire leurs coûts et avancer les dates de lancement de leurs produits, au détriment en particulier de la fiabilité et de la sûreté de fonctionnement.

A cette fin, de nombreuses entreprises de l'industrie logicielle partagent le coût de développement du projet entre la conception, la mise en œuvre et l'assurance de fiabilité du logiciel, et ils reconnaissent un énorme besoin d'approches systématiques pour mesurer et assurer la fiabilité du logiciel. Selon IEEE la fiabilité est définie comme étant la probabilité qu'un logiciel fonctionne sans défaillance pendant une période donnée dans un environnement spécifique. Elle est l'un des attributs de la qualité du logiciel, et est généralement acceptée comme le principale attribut, puisqu'elle quantifie la défaillance du logiciel qui peut faire d'un système puissant un système inopérant. En conséquence, la fiabilité est un élément essentiel de satisfaction de la clientèle.

L'ingénierie de la fiabilité (Reliability engineering) est une discipline indépendante qui remonte aux années 1950 aux Etats-Unis [4]. Aujourd'hui, elle est devenue une profession reconnue. La Société Américaine de Contrôle de Qualité effectue des examens à l'issue desquels, les candidats peuvent devenir des ingénieurs certifiés dans le domaine de la fiabilité des logiciels [5].

L'ingénierie de la fiabilité des logiciels est l'étude quantitative du comportement opérationnel d'un système logiciel, en prenant en compte les besoins de l'utilisateur au sujet de la fiabilité. Elle est centrée sur la mesure, la prévision et la gestion de la fiabilité des logi-

ciels [6]. La mesure, l'estimation et la prédiction de la fiabilité du logiciel se fait à l'aide des modèles établies dans la littérature. Les attributs de fiabilité sont très importants dans la conception, dans le processus de développement, dans l'architecture du système et dans le profile opérationnel du logiciel. L'application des métriques de fiabilité des logiciels guide l'architecte du système logiciel dans le développement, le test et la maintenance.

Le problème de développement de logiciels fiables à un faible coût demeure un défi ouvert [7]. Pour développer un logiciel fiable, nous devons répondre à plusieurs questions. Il s'agit notamment des spécifications métriques de logiciels fiables, des méthodologies de développement fiables, des méthodes de test pour la fiabilité, des modèles de prédiction de croissance de la fiabilité et son estimation précise [8].

Dans cette thèse, nous allons aborder la question de la modélisation et de la prédiction de croissance de la fiabilité logicielle. L'une des préoccupations dans la recherche de fiabilité logicielle, est comment développer des modèles de prédiction généraux. Les modèles existants reposent généralement a priori sur des hypothèses à propos de la nature des défaillances et la probabilité individuelle d'occurrence des défaillances. En outre, ces modèles, appelés modèles paramétriques, tentent de capturer dans deux ou trois paramètres explicites toutes les hypothèses faites sur le processus de développement du logiciel et son environnement. Parce que toutes ces hypothèses doivent être faites avant de commencer le projet, et parce que de nombreux projets sont uniques, le meilleur que l'on peut espérer c'est des techniques qui permettent de prédire les défaillances futures en se référant à la base de données des défaillances à partir de l'historique des projets similaires. Bien que certains modèles sont mieux adaptés à certains types de projets de logiciels que d'autres modèles, la question de trouver un modèle commun pour tous les projets logiciels n'est pas encore résolue.

1.2 La fiabilité des réseaux

L'objet des études de la théorie de fiabilité d'un système est la performance globale d'un système comprenant des éléments sujets de défaillance. Typiquement, les composants du système ne sont pas parfaits à l'égard de leur fonctionnement et leur structure d'échec sous-jacent est supposée suivre certaines distributions probabilistes. Il est donc important de caractériser le comportement du système en fonction du comportement de ses composants.

La fiabilité d'un réseau est sa capacité d'être opérationnel pendant une période du temps t. Formellement, la fiabilité $R(t)$ d'un réseau est :

$$R(t) = Pr(\textit{le réseau est opérationnel dans } [0, t])$$

La fiabilité d'un réseau s'intéresse à l'interconnexion des différents éléments sous la forme d'un réseau, ou un graphe, comme le cas des réseaux de télécommunication, de distribution et les réseaux informatiques. Par exemple les nœuds d'un réseau informatique pour-

raient représentés les éléments physiques du réseau (ordinateurs, serveurs, commutateurs, routeurs, ...) et les arrêtes d'un tel réseau pourraient représenter les liens de communication existants entre ces nœuds. Chaque arrête ou nœud ou le groupe des deux ou le réseau peut être soit opérationnel ou non opérationnel. Le terme opérationnel signifie qu'un expéditeur spécifique et un récepteur spécifique sont capables de communiquer sur certains liens du réseau, tandis que la défaillance signifie qu'aucun chemin de communication n'est disponible.

Non seulement la fiabilité singulière des composants est importante, mais aussi la manière dont ils sont disposés peut avoir un effet significatif sur les performances globales de la fiabilité du système. Par exemple, Moore et Shannon [9] ont configuré des composants non fiables au moyen de la redondance pour obtenir un système fiable (hautement disponible).

Le défi de déterminer la fiabilité d'un système complexe, dont les composantes (nœuds et liens) sont sujets à des défaillances, a reçu une attention considérable dans l'ingénierie, la recherche opérationnelle, et dans les statistiques. Les réseaux sont largement utilisés pour la modélisation des systèmes complexes dont les composants sont sujets à des défaillances.

Le modèle des réseaux stochastiques a été utilisé pour la première fois pour analyser les effets de redondance d'un composant ou d'un module dans une variété de systèmes électroniques et mécaniques [10]. Des réseaux plus généraux ont été analysés après pour déterminer l'effet du blocage dans les circuits de commutation des systèmes téléphoniques. L'étude des systèmes de communications informatiques a suscité un intérêt marqué dans des réseaux dont les nœuds et les liens sont sujets de défaillance, que se soit les réseaux orientés ou non orientés, et dans les mesures de fiabilité des systèmes plus complexes que les système à deux-terminaux (two-terminal).

Dans le cas des réseaux probabilistes (où les nœuds et les liens échouent au hasard et indépendamment avec des probabilités connues), un certain nombre de mesures ont été exploré. Supposons un réseau orienté G avec deux nœuds s et t distingués. La fiabilité deux-terminaux $R_{st}(G)$ est la probabilité qu'il existe au moins un chemin de liens qui fonctionnent entre s et t dans G. La fiabilité tous-terminaux (all-terminal reliability) est la probabilité qu'il existe au moins un chemin reliant chaque paire de nœdus dans G ; ceci est équivalent à la probabilité que le graphe contient au moins un arbre couvrant (spaning tree). La fiabilité k-terminaux (k-terminal reliability) est la probabilité que pour k nœuds, il existe un chemin reliant chaque paire des k nœuds.

L'étude de fiabilité des réseaux peut être classée en deux catégories l'analyse et la synthèse. L'analyse concerne les calculs effectués pour évaluer la fiabilité ainsi que leur complexité. Il est démontré que les problèmes de fiabilité des réseaux sont tous des problèmes NP-difficile, pour les fiabilités deux-terminaux, k-terminaux et tous-terminaux dans les réseaux non orientés et tous-terminaux dans les réseaux orientés [9, 11]. Le problème de synthèse met l'accent sur la recherche d'une topologie du réseau qui répond à certains critères

déterministes ou probabilistes.

La plupart des méthodes telles que, la méthode d'inclusion-exclusion et la méthode des produits disjoints utilisées dans la littérature pour évaluer la fiabilité d'un réseau, utilisent la notion de chemins minimaux et/ou de coupes minimales. D'autres méthodes proposées utilisent ces deux notions sans avoir recours à les déterminer. Le meilleur que l'on peut éspérer c'est des méthodes qui permettent d'énumérer toutes les coupes minimales et ou tous les chemins minimaux d'un réseau orienté ou non.

1.3 Objectifs de la recherche

Notre objectif est de développer des modèles de prédiction du nombre de défaillances dans un logiciel, en utilisant les réseaux de neurones (RNA) et le modèle d'auto-régression (AR) linéaire. Nous focalisons notre apport sur la phase d'apprentissage dans laquelle nous proposons d'utiliser les algorithmes évolutionnistes (AE) et/ou le recuit simulé (RS) [12, 13, 14]. Ces modèles hybrides ont permet de surmonter les problèmes associés aux modèles des RNA et AR et fournissent de meilleurs résultats que les mêmes modèles entraînés à l'aide des algorithmes classiques. Les modèles développés sont utilisés dans la prédiction des défaillances résiduelles du logiciel et ont montré une capacité d'améliorer la qualité de détection des défauts dans les projets logiciels, pour lesquels un petit nombre d'observations antérieurs est disponible.

Le développement d'approches et de modèles pour calculer la fiabilité d'un réseau est aussi l'un des objectifs de cette thèse. En se basant sur la notion de chemins minimaux et coupes minimales, deux algorithmes sont proposés pour énumérer ces chemins et coupes dans des réseaux orientés et non orientés respectivement.

1.4 Organisation du mémoire

Ce mémoire est structuré en deux parties : la première partie est consacrée à la l'introduction de l'état de l'art et l'ensemble des méthodes que nous avons utilisé dans la construction des modèles de prédiction de la fiabilité des logiciels et dans l'évaluation de la fiabilité des réseaux. La deuxième partie présente l'ensemble de nos contributions dans le cadre de la fiabilité des logiciels et des réseaux.

La première partie est constituée de trois chapitres : dans le chapitre 2, les concepts de base et les modèles paramétriques et non paramétriques proposés dans la littérature pour la modélisation et la prédiction de la fiabilité des logiciels sont introduits. Les concepts relatifs à la fiabilité des réseaux sont ensuite introduits et les modèles d'évaluation de la fiabilité des réseaux sont présentés.

Le chapitre 3 est consacré aux réseaux de neurones. Nous y présentons d'abord l'élément de base des réseaux de neurones qui est le neurone formel. Ensuite les différentes fonctions de transfert associées aux réseaux de neurones sont introduites en se concentrant sur les fonctions d'activation les plus utilisées dans la littérature. Après nous y exposons l'architecture générale des réseaux de neurones artificiels, en particulier les réseaux de neurones multicouches. Enfin l'algorithme de rétro-propagation pour l'apprentissage des réseaux de neurones est introduit qui sera l'objet d'une critique dans cette thèse.

Le chapitre 4 s'intéresse aux algorithmes évolutionnistes et le recuit simulé. Nous commençons par introduire les algorithmes génétiques et le contexte de leur apparition ainsi que les opérateurs associés. Ensuite nous introduisons les algorithmes évolutionnistes et les opérateurs de croisement et de mutation associés. Enfin nous exposons l'algorithme de recuit simulé dont nous détaillons les concepts et le paramétrage.

La deuxième partie est composée de trois chapitres : dans le chapitre 5, nous exposons des modèles hybrides de prédiction de la fiabilité des logiciels basés sur les réseaux de neurones et la méthode de régression linéaire pour lesquels nous intervenons dans la phase d'apprentissage par un algorithme évolutionniste et/ou le recuit simulé, ces modèles hybrides permettent de prédire le nombre de défaillances résiduelles dans le logiciel. Nous introduisons tout d'abord les deux modèles. Puis nous détaillons les deux algorithmes d'apprentissage évolutionnistes et de recuit simulé avec leur paramétrage. Ensuite les résultats de test des différents modèles sur des projets réels sont présentés avec leur interprétation. Enfin nous terminons ce chapitre avec une conclusion. L'ensemble des travaux sur ces modèles ont été publiés dans [12, 13, 14] et un article publié dans IJSEIA (International Journal of Software Engineering and Its Applications) [15].

Dans le chapitre 6 nous détaillons une approche évolutionniste pour l'énumération des chemins minimaux dans un réseau orienté. Les chemins ainsi énumérés sont utilisés pour évaluer la fiabilité du réseau concerné en utilisant le principe d'inclusion-exclusion. Cette approche a fait l'objet d'une présentation dans la Conférence Méditerranéenne sur l'ingénierie sûre des systèmes complexes (MISC11) qui s'est déroulée à la ville d'Agadir entre le 27 et le 28 Mai 2011 [16].

Dans le chapitre 7, nous proposons un algorithme récursif qui permet d'énumérer toutes les coupes minimales dans un réseau non orienté. L'algorithme ainsi développé, s'est inspirer de l'idée que chaque coupe minimale dans un réseau non orienté peut être déduite d'une autre. Les détails de cet algorithme sont exposés avec les résultats de test sur des réseaux proposés dans la littérature. Les résultats d'évaluation de la fiabilité en se basant sur les coupes énumérées et le principe d'inclusion-exclusion sont aussi exposés et interprétés en fin du chapitre. Ce travail a fait l'objet d'un article publié [17].

1.5 Publications

L'ensemble des travaux de cette thèse ont fait l'objet des communications et publcations suivantes :

Communications

- M. Benaddy, M. Wakrim, and S. Aljahdali. Evolutionary neural network prediction for cumulative failure modeling. IEEE/ACS Int. Conf. Computer Systems and Applications AICCSA 2009,Rabat.
- M. Benaddy, M. Wakrim, and S. Aljahdali. Evolutionary regression prediction for software cumulative failure modeling : A comparative study. IEEE Int. Conf. Multimedia Computing and Systems ICMCS'09, Ouarzazate.
- M. Benaddy, S. Aljahdali, and M. Wakrim. Evolutionary prediction for cumulative failure modeling : A comparative study. IEEE Eighth Int Information Technology : New Generations (ITNG) Las Vegas, Nevada, USA (2011).
- M. Benaddy, M. Wakrim Enumération des chemins minimaux dans un réseau. In Conférence Méditeranéenne de L'ingénierie sûre des systèmes complexes (MISC'11) (2011),Agadir.

Publications

- Mohamed Benaddy and Mohamed Wakrim. Cutset enumerating and network reliability computing by a new recursive algorithm and inclusion exclusion principle. International Journal of Computer Applications 45(16), 22-25 May (2012). Published by Foundation of Computer Science, New York, USA.
- Mohamed Benaddy and Mohamed Wakrim. Simulated Annealing Neural Network for Software Failure Prediction. International Journal of Software Engineering and Its Applications (IJSEIA) Vol.6. No.4 October 2012.
- M. Benaddy, M. Wakrim, and S. Aljahdali. Evolutionary neural network prediction for cumulative failure modeling. In IEEE Conference Publications, pp 179-184 (2009).
- M. Benaddy, M. Wakrim, and S. Aljahdali. Evolutionary regression prediction for software cumulative failure modeling : A comparative study. In IEEE Conference Publications, pp 286-292 (2009).
- M. Benaddy, S. Aljahdali, and M. Wakrim. Evolutionary prediction for cumulative failure modeling : A comparative study. In IEEE Conference Publications, pp. 41-47 (2011).

Première partie

Etat de l’art et méthodes

CHAPITRE 2

Revue de la litterature

Sommaire

2.1 Introduction

Ce chapitre est consacré à donner un aperçu général sur la fiabilité des logiciels et des réseaux. Les concepts associés à la fiabilité des logiciels seront présentés, ainsi les modèles de prédiction seront introduits. De même les concepts de base relatifs à la fiabilité des réseaux seront examinés. Une revue des modèles d'évaluation de la fiabilité des réseaux est établi.

2.2 Fiabilité des logiciels

Ces dernières années, les problèmes liés aux systèmes informatiques ont vu leur importance. L'opinion publique accepte de moins en moins la notion de défaillance et demande des niveaux de fiabilité, de sûreté et de sécurité de plus en plus élevés. Les systèmes informatiques, devenus omniprésents dans tous les domaines, sont fatalement de plus en plus impliqués dans les problèmes de sûreté, en effet, la fiabilité est devenue un argument de vente impératif. Il faut être capable de l'évaluer ou la mesurer.

2.2.1 Terminologie

Fiabilité d'un logiciel : la fiabilité d'un logiciel est la probabilité qu'il fonctionne sans défaillances pendant une durée donnée et dans un environnement spécifié [18]. C'est donc une notion temporelle. Le temps considéré peut être le temps d'exécution CPU ou le temps calendaire. Notons que pour certains systèmes, le temps n'est pas l'élément primordial : ce qui compte, c'est qu'une exécution se déroule correctement. Alors la fiabilité est définie comme la probabilité qu'une exécution soit correcte. Nous ne sommes pas intéressés à ce type de système, et nous conservons la définition temporelle de la fiabilité.

Une défaillance : se produit quand le résultat fourni par le logiciel n'est pas conforme au résultat prévu par les spécifications [18]. Pour éclaircir cette notion, on peut considérer qu'un logiciel est un système qui, par l'intermédiaire d'un programme, transforme des données d'entrée en résultats ou données de sortie. L'exécution d'un programme peut donc être vue (voir figure 2.1) comme une application de l'ensemble des données de sortie.

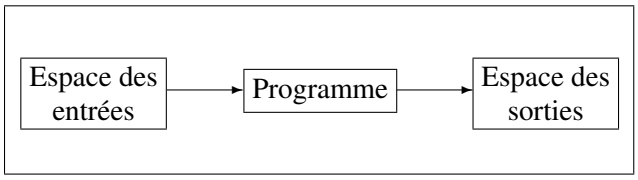


FIGURE 2.1 – L'exécution d'un programme

Les **spécifications** définissent quelle doit être la donnée de sortie pour chaque donnée d'entrée possible. Si, pour une donnée d'entrée particulière, la sortie fournie par le programme n'est pas celle prévue par les spécifications, il y a défaillance. On voit ainsi apparaître une relation forte entre donnée d'entrée et défaillance.

Une faute logicielle [19] : ou bug est un défaut du programme qui, exécuté dans certaines conditions, entraînera une défaillance. Une faute est un phénomène intrinsèque au programme, elle existe même quand le logiciel n'est pas utilisé. A l'inverse, une défaillance est un phénomène dynamique : le programme doit être exécuté pour qu'elle se manifeste. Une faute est crée suite à une erreur humaine de l'analyste, du concepteur ou du programmeur [4]. Aussi, on emploie le terme **faute de conception**. Les erreurs peuvent être de spécifications, de conception ou de codage. Il est également possible qu'une défaillance d'un système informatique soit due à un problème matériel. Ce type de défaillance est en général facilement identifiable.

Le profile opérationnel : définit le choix des entrées et la fréquence de sollicitation du logiciel, en associant à chaque entrée ou groupe d'entrée sa probabilité d'être fournie au programme à un instant donné. Le profil opérationnel est en général très différent en phase de test et en vie opérationnelle.

Quand une défaillance survient, on cherche à détecter la faute qui a provoqué cette défaillance et l'éliminer. On effectue alors une correction ou débogage. Parfois, un logiciel est amené à changer radicalement certaines de ses fonctionnalités. On procède alors à un changement de spécifications, qui va aboutir à une nouvelle version du logiciel. Les corrections et les changements de spécifications peuvent être interprétés de la même manière comme des modifications du programme. Une modification d'un logiciel est parfois appelée une maintenance logicielle. La correction a pour but de réduire l'occurrence des défaillances, donc elle devrait augmenter la fiabilité du logiciel.

Un système non réparable : est un système qui est mis au rebut dés qu'il tombe en panne. C'est le cas des petits systèmes (par exemple des ampoules) ou des systèmes qui coûtent plus cher a réparer qu'à remplacer.

Un système réparable : est un système qui, après sa défaillance, peut être remis en état de marche par des actions de réparation ou maintenance. C'est le cas de tous les systèmes complexes et en particulier des systèmes informatiques. Pour ces derniers, au lieu de réparation, on parle plutôt de correction, débogage ou mise a jour [19].
La maintenance des systèmes est de deux types :

- **La maintenance corrective** ou réparation remet en fonctionnement un système après sa défaillance.

- **La maintenance préventive** est effectuée alors que le système fonctionne et a pour but de retarder l'occurrence des défaillances futures.

Dans les études de fiabilité, on distingue les approches boîte noire et boîte blanche [19] :

- **Approche boîte blanche ou structurelle :** on considère qu'un système complexe est constitué de composants et que sa fiabilité dépend à la fois de la fiabilité de ses composants et de la façon dont le bon fonctionnement ou la panne de chaque composant influe sur le bon fonctionnement ou la panne du système tout entier. En particulier, on considère souvent qu'un système réparable est constitué de composants non réparables. Quand un composant tombe en panne, on le remplace par un neuf, mais le système complet, lui, n'est pas remis a neuf.
- **Approche boîte noire ou globale :** on considère le système comme un tout, qu'on ne cherche pas à décomposer en composants. On s'intéresse alors a la suite des défaillances et réparations successives du système.

2.2.2 Fiabilité des matériels versus logiciels

A priori, les défaillances des systèmes informatiques peuvent être soit d'origine matérielle, soit d'origine logicielle. En pratique, plus de 80 % [3] sont d'origine logicielle. C'est le cas de tous les exemples présentés dans l'introduction. On s'intéressera donc en priorité aux problèmes logiciels. Mais on peut noter qu'il existe des différences fondamentales entre la fiabilité des matériels et celle des logiciels [2].

- Les défaillances des matériels sont essentiellement dues à l'usure (ou vieillissement) et aux facteurs environnementaux, tandis que celles des logiciels sont dues à des fautes de conception (ou bugs), c-à-d à des erreurs humaines.
- Un matériel s'use, un logiciel ne s'use pas.
- La maintenance des matériels ralentit le vieillissement des systèmes mais ne l'empêche pas, tandis que la correction des logiciels augmente leur fiabilité.
- Un logiciel non utilisé ne tombe pas en panne (le terme panne est d'ailleurs peu utilisé pour le logiciel). Un matériel non utilisé peut tomber en panne du fait de l'usure ou des facteurs environnementaux.
- En logiciel, une faute bien répétée et corrigée est éliminée définitivement et ne peut se manifester. En matériel, on peut observer des défaillances répétées ou chroniques.
- La sensibilité d'un matériel à son environnement est assez forte, mais on peut néanmoins considérer qu'un matériel a une fiabilité en soi : les constructeurs quantifient la fiabilité d'une ampoule électrique quel que soit son environnement. En revanche, la sensibilité d'un logiciel a son environnement, à travers son profil opérationnel, est extrêmement forte. Un logiciel truffé de fautes peu très bien fonctionner sans défaillance si le profil opérationnel n'active pas ces fautes. Un matériel ayant beaucoup de défauts

ou très usé tombera fatalement en panne, quelle que soit la manière dont on l'utilise.
- Quand un matériel est en panne, il ne peut pas fonctionner tant qu'on ne l'a pas réparé. Au contraire, un logiciel peut être relancé immédiatement après une défaillance.

On voit que les différences sont nombreuses entre fiabilité des matériels et fiabilité des logiciels. On ne pourra donc pas traiter les deux aspects de la même manière. Historiquement, les concepts de la fiabilité on été introduis pour les matériels. Dans la suite on s'intéressera à la fiabilité des logiciels.

2.2.2.1 Fiabilité prévisionnelle ou fiabilité expérimentale

Que ce soit pour le matériel ou le logiciel, suivant les phases du cycle de vie où sont appliquées les techniques de fiabilité, elles sont qualifiées de prévisionnelles ou d'expérimentales [20].

1. **Les techniques prévisionnelles** se situent en amont dans les phases du cycle de vie et ont pour objectif principal de valider la conception du système par rapport aux objectifs de fiabilité visés. Dans le cas du logiciel, il n'existe actuellement pas de technique suffisamment mature pour évaluer la fiabilité prévisionnelle. Cette dernière relève donc encore du domaine de la recherche.
2. **Les techniques expérimentales** consistent à évaluer le niveau de fiabilité atteint par le système à partir des essais réalisés sur ce système dès qu'il peut fonctionner.

2.2.3 Le risque logiciel

Les défaillances des logiciels sont causées par des fautes dans les programmes. Or, d'une part, une étude [21] a montré qu'un programmeur professionnel fait en moyenne 6 fautes pour 1000 lignes de code (LOC) écrites, et d'autre part, la taille et la complexité des logiciels ne cessent d'augmenter.

Par exemple :

- La navette spatiale américaine a besoin pour voler de 500 000 LOC logiciel embarqué et 3 millions et demi de LOC au sol. Les réseaux téléphoniques utilisent des millions de LOC pour fonctionner.
- Le nombre de LOC est passé de moins de 5 millions pour Windows 95 à plus de 50 millions pour Windows Vista.
- Plus généralement, un logiciel commercial standard fait en moyenne 350 000 LOC.

Par conséquent, cela fait environ 2000 fautes potentielles pour un logiciel standard, 24 000 pour la navette spatiale et 300 000 pour Vista ! Evidemment, tout est fait pour éliminer ces fautes, essentiellement par le test du logiciel. Mais il est extrêmement difficile et coûteux de détecter et corriger des fautes dans un logiciel. En effet, une étude de Microsoft [21] a établi qu'il fallait en moyenne 12 heures de travail pour détecter et corriger une faute. Si un logiciel

contient 2000 fautes, il faut donc passer 24 000 heures pour le déboguer, soit presque 3 ans de travail cumulé. C'est pourquoi Microsoft emploie autant de personnel pour tester, vérifier et valider les programmes que pour les créer. Et malgré cela, chacun a pu expérimenter qu'il subsiste des erreurs dans les logiciels de Microsoft. Une étude plus récente [22] evalue a 60% du budget total d'un projet informatique le coût de la détection et correction des erreurs logicielles (ou maintenance logicielle). Malgré tous ses efforts, la complexité de la tâche fait qu'il reste toujours des fautes dans un logiciel. Comme partout, et peut être même moins que partout, le zéro défaut est impossible. Quand ces fautes résiduelles se manifestent, leurs conséquences peuvent aller du minime au franchement catastrophique.

Il est donc impératif de tout faire pour éviter que les pannes informatiques majeures se produisent. Pour cela on dispose de nombreuses méthodes dont le but est de produire des logiciels de fonctionnement sûr. On peut classer ces méthodes en 4 catégories [23, 24]

1. **La prévention des fautes :** ces méthodes ont pour objectif d'empêcher l'occurrence et l'introduction de fautes dès la conception du logiciel. Par exemple, on a de plus en plus souvent recours à des méthodes formelles pour développer les spécifications.
2. **L'élimination des fautes :** ces méthodes ont pour rôle de détecter les fautes dans un programme déjà écrit. Elles comprennent les preuves de programmes, les inspections, la vérification et surtout le test du logiciel.
3. **La tolérance aux fautes :** ces méthodes permettent au système de fonctionner correctement même en présence de fautes.
4. **La prévision des fautes :** ces méthodes essaient d'estimer la présence des fautes et de prévoir les défaillances futures du système.

En effet, il ne suffit pas d'avoir utilisé tous les moyens possibles pour développer un logiciel fiable, encore faut-il s'assurer qu'il l'est effectivement : il faut des méthodes pour atteindre des objectifs de fiabilité (les trois premières catégories, concernent le génie logiciel) et faire appel à d'autres méthodes pour savoir si ces objectifs sont atteints (la quatrième catégorie, qui fait intervenir des concepts probabilistes et statistiques). Par conséquent, il est très important de pouvoir prévoir l'occurrence des défaillances, et donc d'évaluer, mesurer ou quantifier la fiabilité d'un logiciel.

2.2.4 La fiabilité des logiciels selon les étapes du cycle de vie

Les méthodes d'évaluation de la fiabilité des logiciels varient suivant la nature des informations disponibles. Celles-ci sont étroitement liées au cycle de vie du logiciel, comme le présente le tableau 2.1 [2, 21].

Phase du cycle de vie	Pourcentage d'erreurs introduites	Pourcentage d'erreurs détectées
Analyse	55%	18%
Conception	30%	10%
Codage et test	10%	50%
Vie opérationnelle	5%	22%

TABLE 2.1 – Pourcentage d'erreurs introduites et détectées selon les phases du cycle de vie du logiciel.

Les types des erreurs dans les différentes phases sont les suivantes :

- **Analyse :** le logiciel ne répond pas à l'attitude des utilisateurs.
- **Conception :** mauvaise traduction des spécifications.
- **Codage et test :** erreurs de programmation ou de correction.
- **Vie opérationnelle :** erreur dans les mises à jour du système.

On constate que la majeure partie des erreurs sont introduites dans les premières phases du cycle de vie (85% en analyse et conception) et détectées dans les dernières phases (72% en codage, test et vie opérationnelle). Dans les phases d'analyse, conception et codage, le système n'est pas encore construit, donc il ne peut pas être utilisé et aucune défaillance n'est observée. Les éléments pouvant être utilisés pour des prévisions de fiabilité sont la structure du système et les métriques logicielles (nombre de lignes de code, nombre de cyclomatique du graphe de contrôle, mesures d'architecture et de spécifications, etc... [25]. A ce niveau, on peut évaluer la qualité du logiciel, mais pas sa fiabilité. Or on ne sait pas mesurer la corrélation entre qualité et fiabilité d'un logiciel. En phase de test et en vie opérationnelle, le système fonctionne, des défaillances sont observées et des corrections sont apportées au logiciel pour remédier aux fautes apparues. L'essentiel des méthodes d'évaluation de la fiabilité repose sur l'observation et l'analyse statistique de cette suite de défaillances et corrections successives.

Tout comme les matériels, les logiciels complexes sont constitués de modules unitaires que l'on assemble. Si on est capable d'évaluer la fiabilité de chaque module et d'analyser les liens entre les différents modules on peut appliquer les approches structurelles (boîte blanche) d'évaluation de la fiabilité. Ce n'est pas du tout facile en pratique. Aussi considère-t-on en général un logiciel comme un tout et on évalue sa fiabilité par une approche boîte noire. C'est ce que nous ferons dans cette thèse.

2.2.5 Utilisation des évaluations de fiabilité des logiciels

Dans un premier temps, les évaluations de la fiabilité permettent de quantifier la confiance d'un utilisateur envers un système informatique, c'est-à-dire d'évaluer quantitativement le

risque que l'on prend en le faisant fonctionner. Puis elle permettent de s'assurer que le logiciel a atteint un niveau de fiabilité conforme aux objectifs exprimés dans les spécifications. Un objectif de fiabilité est usuellement exprimé en terme de taux de panne ou taux de défaillance.

Pour les systèmes faisant l'objet d'une garantie, les évaluations de fiabilité permettent de déterminer la durée et le coût de la garantie. Si les mesures de fiabilité montrent que l'objectif n'est pas atteint, elle peuvent permettre d'évaluer l'effort de test à fournir pour atteindre l'objectif, et en particulier d'estimer le temps nécessaire pour y parvenir. Par conséquent, les mesures de fiabilité fournissent un critère d'arrêt des tests : on arrête les tests dès qu'on peut pouvoir, avec un niveau de confiance raisonnable, qu'un objectif donné de fiabilité a permis une réduction de 15% de la durée de la période de tests, ce qui a entraîné un gain de 4% sur le coût total du projet, alors que le surcoût du aux mesures n'a représenté que 0.2% de ce coût total [21]. D'autres exemples sont traités dans [26]. par ailleurs, une mesure de fiabilité est un moyen d'évaluer quantitativement la qualité d'une méthode de génie logiciel donnée. Elle peut aussi fournir un indicateur de performance d'un programmeur ou d'un testeur. Cette dimension humaine délicate est parfois un frein à l'utilisation effective des évaluations de fiabilité.

2.2.6 Gain de fiabilité par redondance

Le principe de redondance est valable à plus grande échelle. En pratique, le moyen le plus simple pour augmenter la fiabilité d'un système est d'ajouter des redondances, c-à-d de faire fonctionner plusieurs systèmes identiques en parallèle. Par exemple, on met deux phares aux voitures au lieu d'un. Evidemment, cela a un coût et il faut donc trouver un compromis entre le coût des redondances et le gain de fiabilité qu'elles entraînent.

Ce qui fait bien fonctionner la redondance pour les systèmes matériels, c'est l'indépendance entre les durées de bon fonctionnement des composants. Si on a deux composants de durées de vie continues et indépendantes X_1 et X_2, on a $P(X_1 = X_2) = 0$. Donc quand un des composants tombe en panne, l'autre fonctionne toujours et fait fonctionner le système. Ainsi, si on a un problème avec un phare, on peut toujours rouler en utilisant le deuxième. La probabilité que les deux phares tombent en panne en même temps est nulle (sauf si un événement externe perturbe l'ensemble de la voiture, comme une panne électrique générale ou un accident, qui provoque ce qu'on appelle une défaillance de cause commune).

Si on veut appliquer le principe de la redondance aux logiciels, on ne va évidemment pas faire fonctionner en parallèle deux copies du même logiciel, puisque leur fonctionnement sera identique. Il faut faire développer deux programmes ayant les mêmes spécifications par deux équipes différentes. Si on fait fonctionner les deux programmes en même temps avec les mêmes données d'entrée, on peut espérer que quand l'un aura une défaillance, l'autre

fonctionnera correctement. Mais l'expérience montre que ce n'est pas aussi simple [27] : les deux équipes butent sur les mêmes difficultés et auront tendance à faire des fautes aux mêmes endroits. Apparement, c'est ce qui s'est passé pour la panne géante du réseau mobile de Bouygues Télécom en novembre 2004 [3] : deux serveurs en parallèle sont tombés en panne en même temps.

En conclusion, l'indépendance des durées de bon fonctionnement, valable pour la plupart des matériels, ne l'est plus pour les logiciels. Donc la redondance logicielle est loin d'être aussi efficace que la redondance matérielle. Il reste que la redondance logicielle augmente quand même la fiabilité des logiciels, même si on ne sait pas quantifier cette augmentation. C'est donc une méthode souvent utilisée. Par exemple, Airburs utilise une redondance logicielle sur la chaîne de commande et surveillance des avions. A noter également que la redondance logicielle est très chère puisqu'il faut deux équipes de développeurs.

2.2.7 Modèles de fiabilité des logiciels

Les modèles de fiabilité du logiciel permettent de faire des calculs prévisionnels du nombre moyen de défaillances ou de la probabilité d'apparition de problèmes ultérieurs en fonctionnement opérationnel. L'intérêt de la prévision fournie par les modèles de fiabilité est multiple :

1. En cours de test, elle permet de dimensionner l'équipe de correction en conséquence. Dans le cas où la même équipe passe les tests et effectue les corrections, il est possible, grâce à cet indicateur, de savoir quelle charge affecter au test par rapport à la charge affectée à la correction ;
2. En fin de test, elle permet de savoir si les objectifs de fiabilité sont atteints et, si ce n'est pas le cas, de réviser la politique de test en conséquence ;
3. Enfin, au moment de la mise en service du logiciel, elle permet d'annoncer au client quel sera le comportement le plus probable du logiciel pendant ses premiers mois de fonctionnement.

Le domaine de l'évaluation de la fiabilité des logicels a donnée naissance, depuis le milieu des années 1970, à plusieurs modèles qui diffèrent essentiellement par les hypothèses établies concernant la nature et l'évolution des lois relatives aux variables caractérisant le processus de défaillance du logiciel. Le tableau 2.2 présente une liste non-exhaustive de ses hypothèses [28].

Hypothèses	Réalité	Effets sur le modèle
Quand une erreur est découverte elle est immédiatement corrigée	Les développeurs ont l'habitude en effet de faire en séquence, et non en parallèle, les tests et les corrections.	Pas de corrélation entre l'apparition des défauts et les tests.
La correction d'une erreur est parfaite	La correction d'un défaut peut introduire de nouvelles erreurs.	Le nombre des défauts augmente durant le test, au lieu d'être constant.
Aucun nouveau code n'est introduit durant le test.	De nouveaux lignes de codes sont introduites durant le test, afin, de corriger des défauts et d'introduire de nouvelles fonctionnalités.	Le nombre des défauts augmente durant le test, au lieu d'être constant.
Les défauts sont reportés par les testeurs.	Les défauts peuvent être reportés par des Bêta-testeurs, ou des développeurs qui travaillent sur une version ultérieur du système.	Pas de corrélation entre l'apparition des défauts et les tests.
Chaque test unitaire, pendant un temps d'exécution, est susceptible de trouver des défauts.	Certains tests sont écrits spécifiquement pour tester des parties du code rarement utilisées, donc ils ont peu de chances de trouver des défauts.	Pas de corrélation entre l'apparition des défauts et les tests.
Les tests représentent un profile opérationnel.	Il est difficile de déterminer un profile opérationnel approprié, car les utilisateurs exécutent des configurations et des applications variées.	La comparaison des résultats des différentes versions peut être non valide, car, les mêmes tests sont ré-exécutés sur les prochaines versions et il est peu probable de détecter plus de défauts, à moins qu'une erreur de régression a été faite.

TABLE 2.2 – Quelques hypothèses sur les modèles de fiabilité logicielle.

2.2.7.1 Classification des modèles

On dénombre actuellment plus de quarante modèles de croissance de fiabilité dont une revue détaillée peut se trouver dans [29]. Compte tenu du nombre important de modèles de croissance de fiabilité, plusieurs auteurs [18, 2, 21, 30, 31] ont procédé à une classification des modèles proposés par rapport à différents critères.

L'intérêt de procéder à une classification des modèles est, d'identifier ceux qui sont basés sur des hypothèses équivalentes afin d'éviter d'utiliser plusieurs modèles qui correspondent au même type de comportement du système considéré et qui conduisent par conséquent, à des résultats équivalents.

Dans la suite nous nous baserons sur la classification proposée dans [2] pour présenter brièvement ces modèles. Notre objectif n'est pas de présenter en détail tous les modèles existants, mais, de présenter ceux qui sont largement étudiés et qui sont en relation avec cette thèse.

2.2.7.2 Modèles à injection des erreurs

Dans ce groupe de modèles on estime le nombre des erreurs dans le programme par la technique d'échantillonnage à plusieurs degré. On distingue deux types d'erreurs, les erreurs résiduelles et les erreurs induites (erreurs injectées). En se basant sur les données de débogage, le nombre des futures erreurs résiduelles est estimé à partir des erreurs injectées et les erreurs résiduelles actuelles. Parmi les modèles de ce groupe :

- Le modèle de Mill d'essaimage des erreurs (1970) [32].
- Le modèle de Cai (1998) [33].
- Le modèle Hypergéométrique de Tohma (1991) [34].

2.2.7.3 Les modèles à taux de défaillance

Cette classe de modèles inclus les modèles qui étudient le taux de défaillance du système par rapport à son nombre d'erreurs résiduelles. Ce groupe englobe les modèles suivants :

- Modèle de Jelinski et Moranda (1972) [35].
- Modèle de Schick et Wolverton (1978) [36]
- Modèle géométrique de Jelinski et Moranda (1979) [37]
- Modèle Moranda Poisson géométrique proposé par Littelwood (1979) [38]
- Le modèle de Schick et Wolverton modifié (1977) [39]
- Le modèle de Goel Okumoto de l' imperfection du débogage (1979) [40]

Dans ces modèles on s'intéresse principalement aux études du changement des taux des défaillances durant les intervalles de ces défaillances. Comme le nombre des erreurs résiduelles change, en conséquence le taux de défaillance change aussi. A noter aussi que le nombre des erreurs dans le programme est une fonction discrète, le taux de défaillance l'est aussi avec des discontinuités au niveau des moments de défaillance.

2.2.7.4 Les modèles d'ajustement au courbe

Ces modèles utilisent des analyses statistiques de régression pour étudier la relation entre la complexité du logiciel, et le nombre des défauts dans le logiciel, ou le nombre des corrections du logiciel, ou bien son taux de défaillance. Les analyses de régression linéaire ou non linéaire ou bien les séries temporelles sont utilisées pour trouver les relations fonctionnelles d'indépendance ou de dépendance entre les variables régissant dans le modèle. Par exemple le nombre d'erreurs du programme est une variable dépendante, tandis que, le nombre de modules corrigés pendant la phase de maintenance est une variable indépendante. Ces modèles s'intéressent principalement à :

- L'estimation des erreurs.
- L'estimation de la complexité.
- L'estimation du taux de défaillance.

2.2.7.5 Les modèles de croissance de la fiabilité

Cette classe de modèles regroupe les modèles qui permettent de prédire et d'améliorer la fiabilité des programmes par le biais du processus de test. Un modèle de croissance représente la fiabilité ou le taux de défaillance en fonction du temps ou du nombre de test. Les modèles de ce groupe sont :

- Le modèle de Coutinho (1973) [41].
- Le modèle de Wall et Ferguson (1977) [42].

2.2.7.6 Les modèles de Markov

Le comportement futur d'un processus de Markov dépend seulement de son état actuel et est indépendant de son historique [2]. Les modèles de Markov présentent une façon générale pour représenter la défaillance du logiciel. Les modèles de ce groupe peuvent également être utilisés pour étudier la fiabilité et la relation entre les modules constituant le système. Il est supposé que les défaillances des modules sont indépendantes les uns des autres. Cette hypothèse semble raisonnable au niveau du module car il peut être conçu, codé et testé de manière indépendante, mais ne peut pas être vraie au niveau du système. Parmi ces modèles :

- Le modèle de Markov de débogage imparfait proposé par Goel (1979) [43].
- Le modèle Markovien de Littlewood (1979) [38].
- La sécurité du logiciel (1997) [44]

2.2.7.7 Les modèles de comptage des défaillances

Les modèles de cette catégorie, modélisent le nombre de défaillances observées ou les erreurs détectées durant la phase de test. Comme les erreurs sont corrigées, il semble que

le nombre de défaillances va diminuer en fonction du temps, par conséquent la courbe du nombre de défaillances cumulées en fonction du temps va éventuellement tendre vers une constante. L'intervalle du temps entre les défaillances peut être fixé à priori et le nombre de défaillances peut être traité comme une variable aléatoire. Parmi ces modèles :

- Le modèle des processus de Poisson non homogènes (NHPP) proposé par Goel-Okumoto (1979) [40].
- Le modèle des NHPP généralisé proposé par Goel (1982) [45, 46].
- Le modèle du temps d'exécution proposé par Musa (1976) [47].
- Le modèle exponentiel de Musa (1987) [7].
- Le modèle de croissance S-shaped proposé par Ohba et Yamada [48, 49, 50].
- Le modèle discret de croissance de la fiabilité proposé par Yamada (1985) [51].
- Le modèle généralisé des NHPP développé par Pham [2].

2.2.8 Modèles non paramétriques

Les modèles présentés dans la section 2.2.7 dépendent d'un ensemble d'hypothèses et de paramètres. Les hypothèses doivent être prises en considération avant d'utiliser un tel modèle et les paramètres doivent être définis avant d'appliquer un modèle. Donc on se confronte avec un problème d'adaptation au modèle ; un tel projet doit s'adapter au modèle et non le contraire. C'est dans ce sens que les modèles non paramétriques ont vue le jour, parmi les modèles non paramétriques on cite ; les réseaux de neurones artificiels et le modèle autorégressif. Ces modèles sont l'objet des deux sections suivantes.

2.2.8.1 Prédiction à l'aide des réseaux de neurones

Les réseaux de neurones artificiels (RNAs) utilisent le principe d'apprentissage pour s'adapter au problème, donc, avec un ensemble de données le RNA va ajuster ses paramètres internes. Les développements récents dans les RNAs ont montré qu'ils peuvent être utilisés pour résoudre une variétés de problèmes complexes. Par exemple, les RNAs sont utilisés dans les problèmes de classification, de reconnaissance de forme, de contrôle et commandement et dans la prédiction. Les RNAs sont devenues des alternatives dans la modélisation et la prédiction de la fiabilité des logiciels. En 1992 Karunanithi et al.[18, 52] ont proposé une première application des RNAs dans la prédiction de la fiabilité des logiciels. Adnan et al. [53, 54], Aljahdali et al.[1], Ho et al.[55], Park et al. [56] et Sitte [57], ont proposé des modèles non paramétriques de prédiction de la fiabilité des logiciels en se basant sur les RNAs, les résultats obtenues sont meilleures en les comparant à des méthodes traditionnelles. Nous avons aussi proposé un modèle hybride [12, 13] en utilisant les RNAs et les algorithmes évolutionnistes, nous avons obtenue de très bons résultats que ceux obtenus par Aljahdali et al.[1].

2.2.8.2 Prédiction à l'aide du modèle autorégressif

Le modèle de régression linéaire a de nombreuses applications pratiques. Il permet notamment de faire des analyses de prédiction. Après avoir estimé un modèle de régression linéaire, on peut prédire quel serait le niveau d'une donnée y pour des valeurs particulières de x. L'un des fameux modèles de régression est le modèle auto-régressif. Ce modèle peut être utilisé dans la prédiction de la fiabilité des logiciels comme il est décrit par l'équation 2.1.

$$Y(t+1) = a_0 + \sum_{i=1}^{n} a_i Y(t-i) \tag{2.1}$$

Avec $Y(t-i)$ le nombre de défaillances cumulé pendant n unités de temps considérées avant le temps $(t+1)$, les a_i sont les paramètres du modèle. Aljahdali et al.[1] ont proposé un modèle autorégressif d'ordre 4 ($n = 4$). Nous avons proposé un modèle hybride [14, 13] qui utilise les algorithmes évolutionnistes et le modèle autorégressif, les résultats obtenues sont très performants par rapport à ceux obtenus par Aljahdali et al.[1].

2.2.9 Les données de fiabilité

Les données des défaillances utilisées sont fournis par les services de DACS [58] au département de la défense Américaine (DoD). L'ensemble de données de la fiabilité des logiciels a été compilé par John Musa (Bell Telephone Laboratories). Son objectif était de rassembler des données sur l'intervalle des défaillances pour aider les chefs de projet logiciel dans le suivi des tests, dans la prédiction et pour assister les chercheurs dans la technologie logicielle pour valider leurs modèles de fiabilité des logiciels. L'ensemble de données se compose des données de défaillances de 16 projets. Les données ont été collectées soigneusement pour s'assurer qu'elles seraient de haute qualité au milieu des années 1970. Elles représentent des projets d'une variété d'applications, y compris les applications de commandement et de contrôle en temps réel, des applications de traitement de texte, des applications commerciales et des applications militaires. Les données de défaillances se composent de l'identification du projet, le numéro de la défaillance, le temps entre les défaillances et le jour d'occurrence de la défaillance. Le tableau 2.3 illustre la taille des données utilisées et le code attribué à chaque projet et le tableau 2.4 illustre les données de défaillance du projet 14C (Real Time), les données utilisées dans le cadre de cette thèse se trouvent dans l'annexe.

La première colonne (Failure) du tableau 2.4 représente le numéro de la défaillance. La deuxième (Failure Interval Length) représente le temps qui s'écoule entre la défaillance précédente et courante. Pour le projet 6 le temps est exprimé en temps CPU en secondes, tandis que pour le reste des projets le temps est exprimé en temps d'horloge. La dernière colonne (Day of Failure) indique le jour d'occurrence de la défaillance.

Projet	Désignation	Nombre de défaillances	Projet	Désignation	Nombre de défaillances
1	Real Time Command & Control	136	5	Real Time Commercial	831
14C	Real Time	36	6	Commercial Subsystem	73
17	Military	38	SS1A	Operating System	112
2	Real Time Command & Control	54	SS1B	Operating System	375
27	Military	41	SS1C	Operating System	277
3	Real Time Command & Control	38	SS2	Time Sharing System	192
4	Real Time Command & Control	53	SS3	Word Processing System	278
40	Military	101	SS4	Operating System	196

TABLE 2.3 – Taille des données des projets et leur designation.

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
1	19152	3	19	228315	66
2	2078820	27	20	51480	67
3	514560	33	21	44820	67
4	1140	33	22	850080	77
5	3120	33	23	361860	81
6	327480	37	24	39300	82
7	15420	37	25	545280	88
8	60000	38	26	256980	91
9	140160	39	27	396780	96
10	937620	50	28	91260	97
11	72240	51	29	1225620	111
12	737700	60	30	120	111
13	250680	63	31	1563300	129
14	2965	63	32	513000	135
15	196	63	33	177660	137
16	65173	63	34	2469000	165
17	2370	63	35	1678260	185
18	1581	63	36	170760	187

TABLE 2.4 – Données de fiabilité du projet 14C.

2.3 Fiabilité des réseaux

Plusieurs systèmes physiques (par exemple ; les réseaux informatiques et de télécommunications, les réseaux de distribution d'eau et du gaz, les réseaux électriques,...) peuvent être modélisés sous forme d'un réseau. Dans le contexte de cette thèse, un réseau est tout système physique qui peut être modélisé sous forme d'un graphe, qui, à son tour formé de nœuds et de liens (orientés ou non). On parle alors de graphe (ou réseau) orienté et non orienté. Ces systèmes ont souvent un nœud source et un nœud de destination ou ce qu'on peut appeler aussi des entrées et sorties, les opérations entres ces deux nœuds se basent sur les connexions entre eux. Les connexions sont mise en place par des liens d'interconnexion. Les liens fournissent un chemin sur lequel l'information peut circuler du nœud source vers la destination. Les liens sont sujet de défaillance, ce qui entraînera une coupure de connexion entre les deux nœuds communicants.

En outre la fiabilité d'un réseau peut être évaluée de plusieurs façons : les diagrammes de fiabilité, les arbres de défaillance, les graphes de markov, les diagrammes de décision binaire, les réseaux de petri, les réseaux bayésiennes, etc... Dans cette thèse nous nous intéresserons à la représentation physiques des réseaux sous forme de graphe.

2.3.1 Introduction sur les graphes

Un graphe $G = (V, E)$ est la donnée d'un ensemble $V = \{v_1, v_2, ..., v_n\}$ dont les éléments sont appelés sommets et d'une partie de E symétrique ($< u, v > \in A \Leftrightarrow < v, u > \in A$) dont les éléments sont appelés arêtes. En présence d'une arête $e =< u, v >$, on dit que u et v son les extrémités de e, que e est incidents en u et v, que v est un successeur ou voisin de u (et vice versa) et que u et v sont adjacents. On dit que le graphe G est sans boucle si E ne contient pas d'arête de la forme $< u, u >$, c-à-d joignant un sommet à lui même. Le nombre de sommets est appelé ordre du graphe. Un graphe ne possédant pas de boucle ni d'arêtes parallèles (deux arêtes distincts joignant la même paire de sommets) est appelé graphe simple. Graphiquement, les sommets peuvent être représentés par des points et l'arête $e =< u, v >$ par un trait reliant u à v (voir figure 2.2).

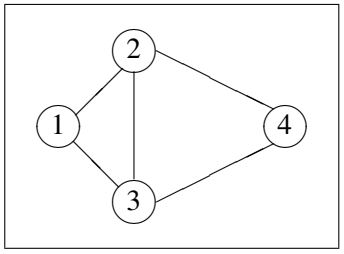


FIGURE 2.2 – Représentation graphique d'un graphe

On appelle un graphe orienté ou digraphe $G = (V, E)$ la donnée d'un ensemble $V = \{v_1, v_2, ..., v_n\}$ dont les éléments sont appelés sommets et d'une partie E de VxV dont les éléments sont appelés arcs ou arrêtes.

En présence d'un arc $e =< u, v >$, on dit que u est l'origine (ou extrémité initiale) et v l'extrémité (terminale) de e, que e est sortant en u et incident en v, et que v est un successeur de u tandis que u est un prédécesseur de v. On dit aussi que u et v sont adjacents.

2.3.1.1 Graphes et applications multivoques

L'ensemble des successeurs d'un sommet $v \in V$ est noté $\Gamma(v)$. L'application Γ, qui à tout élément de V, fait correspondre une partie de V (un élément de $P(V)$) est appelée une application multivoque. L'ensemble des prédécesseurs d'un sommet $v \in V$ peut alors être noté $\Gamma^{-1}(v)$ où Γ^{-1} est l'application (multivoque) réciproque de Γ.

2.3.1.2 Principales définitions

Les définitions qui suivent sont énoncées dans le cadre des graphes orientés. Elles peuvent être appliquées (si elles ont un sens) au cas des graphes non orientés.

- On appelle degré sortant ou demi-degré extérieur d'un sommet v le nombre d'arcs de la forme $e =< u, v >$ avec $u \neq v$, c-à-d le nombre d'éléments de $\Gamma(v) \setminus \{v\}$. On note $d_s(v)$ ce degré.
- On appelle degré entrant ou demi-degré intérieur d'un sommet v le nombre d'arcs de la forme $e =< u, v >$ avec $u \neq v$, c-à-d le nombre d'éléments de $\Gamma^{-1}(v) \setminus \{v\}$. On note $d_e(v)$ ce degré.
- On appelle degré de v la somme du degré entrant et du degré sortant.
- Un sommet de degré entrant non nul et de degré sortant nul est appelé puit, tandis qu'un sommet de degré entrant nul et de degré sortant non nul est appelé source.
- Un sommet n'ayant pas d'arcs incidents est appelé sommet isolé ; ces sommets ont un degré nul. Deux arcs adjacents sont dits "en série" si leur sommet commun est de degré égal à deux. Dans la définition d'un graphe, l'ensemble des arcs E peut être vide ; dans ce cas, on a affaire à un graphe nul. Tous les sommets d'un graphe nul sont donc isolés. En revanche, l'ensemble des sommets V ne peut être vide sinon le graphe correspondant n'existe pas. Cela signifie qu'un graphe comporte au moins un sommet.
- Un graphe est symétrique si, pour tout arc $e_1 =< u, v >$ appartenant à E, l'arc $e_2 =< v, u >$ appartient à E.
- Le concept de graphe symétrique est très proche de celui des graphes non orientés. En fait, à tout graphe symétrique, on peut associer un graphe non orienté en substituant aux arcs $e_1 =< u, v >$ et $e_2 =< v, u >$, une arête $e =< u, v >$.

- Soit un graphe $G = (V, E)$ et $V' \subset V$. Le sous graphe engendré par V' étant formé des arrêtes dont les deux extrémités sont dans V'.
- Si l'on se donne un sous ensemble E_1 de E, le sous graphe engendré par E_1 est $G_1 = (V, E_1)$. Dans certaines situations, les sommets de G_1 ayant un degré nul (sommets isolés n'ayant aucune arête incidente appartenant à E_1) peuvent être supprimés du sous graphe.

2.3.1.3 Mode de représentation d'un graphe

L'essor de la théorie des graphes est essentiellement dû à l'avènement de puissants calculateurs. Il est donc légitime de s'intéresser à la manière de représenter les graphes au sein d'un ordinateur. Plusieurs modes de représentation peuvent être envisagés selon la nature des traitements que l'on souhaite appliquer au graphe considéré.

2.3.1.4 Liste des successions

Un graphe peut être représenté à l'aide d'un dictionnaire ; il s'agit d'une table à simple entrée où chaque ligne correspond à un sommet et comporte la liste des successeurs ou des prédécesseurs de ce sommet. Considérons le graphe de la figure 2.3.

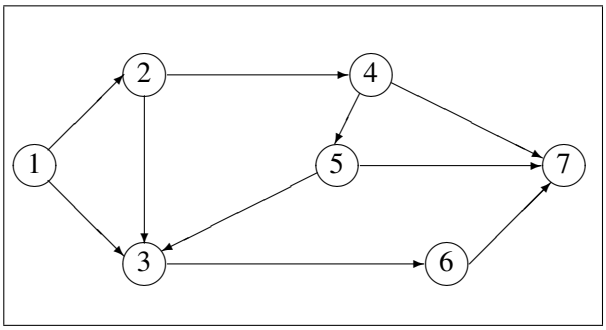


FIGURE 2.3 – Un exemple de graphe orienté

Celui-ci peut être représenté par la matrice suivante :

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 6 & 0 \\ 4 & 5 & 7 \\ 5 & 3 & 7 \\ 6 & 7 & 0 \\ 7 & 0 & 0 \end{pmatrix}$$

2.3.1.5 Matrice d'adjacence

Les outils classiques d'algèbre linéaire peuvent également être utilisés pour coder les graphes. La première idée consiste à considérer chaque arc comme un lien entre deux sommets. Considérons un graphe $G = (V, E)$ comportant n sommets. La matrice d'adjacence de G est égale à la matrice $U = (u_{ij})$ de dimension $n \times n$ telle que

$$u_{ij} = \begin{cases} 1 & si \ \ (i, j) \in E \\ 0 & sinon \end{cases} \tag{2.2}$$

Une telle matrice, ne contenant que des "0"et des "1"est appelée, de manière générale, une matrice booléenne.

Un graphe orienté quelconque a une matrice d'adjacence quelconque, alors qu'un graphe non orienté possède une matrice d'adjacence symétrique. L'absence de boucle se traduit par une diagonale nulle. La matrice d'ajacence du graphe de la figure 2.3 est la suivante :

$$U = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Ce mode de représentation engendre des matrices très creuses (i.e. comporenant beaucoup de zéros).

2.3.1.6 Etude de la connexité

Définitions

- Un chemin est une séquence finie d'arêtes qui relie deux sommets s et t, telle que chaque arête est incident avec les sommets qui l'encadre dans la séquence. La première arête de la séquence est sortant de s et la dernière arête est incidente à t.
- Un chemin minimal (Minimal Path, MP) est un chemin qui ne contient pas d'autres chemins entre s et t.
- Les deux sommets qui sont reliés par un chemin sont appelés, source s et terminal t.
- La longueur d'un chemin est égale au nombre d'arêtes qui le composent.

Toutes les définitions précédentes, s'appliquent au cas des graphes non orientés, peuvent être transposées au cas des graphes orientés.

2.3.1.7 Graphes et sous graphes connexes

Un graphe est connexe si l'on peut atteindre n'importe quel sommet à partir d'un sommet quelconque en parcourant différentes arêtes. De manière plus formelle on a :

Définitions

- Un graphe G est connexe s'il existe au moins un chemin entre une paire quelconque de sommets de G.
- La relation :

$$x_i \ R \ x_j \Leftrightarrow \begin{cases} soit \ \ x_i = x_j \\ soit \ \ il \ \ existe \ \ un \ \ chemin \ \ joignant \ \ x_i \ \ et \ \ x_j \end{cases} \tag{2.3}$$

  Est une relation d'équivalence (réflexivité, symétrie, transitivité). Les classes d'équivalence induites sur V par cette relation forment une partition de V en $V_1, V_2, ..., V_p$. Le nombre p de classes d'équivalence distinctes est appelé nombre de connexité du graphe.
- Une autre définition concernant la connexité d'un graphe peut être donnée. Un graphe est dit connexe si et seulement si son nombre de connexité est égal à 1.
- Les sous graphes $G_1, G_2, ..., G_p$ engendrés par les sous-ensembles $V_1, V_2, ..., V_p$ sont appelés les composantes connexes du graphe G. Chaque composante connexe est un graphe connexe.
- Une coupe est un ensemble d'arêtes dont la suppression divise le graphe en deux sous graphes connexes.
- Une coupe minimale (Minimal Cut : MC) est une coupe qui, ne contient pas d'autres coupes.
- Un graphe connexe, sans boucle et ayant plus d'un sommet, est appelé un réseau. Les sommets du réseau sont appelés des nœuds et les arêtes sont appelées des liens.

2.3.2 Evaluation de la fiabilité d'un réseau

Pour évaluer la fiabilité d'un réseau, on attribut à chaque élément (nœuds et liens) du réseau des probabilités de succès, qui reflète leur fiabilité. Dans la majorité des études de fiabilité des réseaux les hypothèses suivantes sont à prendre en considération :

- Tous les éléments (nœuds, liens) sont toujours en mode actif (pas de redondance).
- Chaque élément peut être représenté comme un dispositif à deux bornes.
- L'état de chaque élément et du réseau est soit fonctionnel (parfait) ou défaillant (imparfait).
- Les états de tous les éléments sont statistiquement indépendants.

- Le réseau est privé de boucles.

Ces hypothèses facilitent le développement des modèles souples.

Dans un réseau les liens comme les nœuds peuvent être sujets de défaillance, pour cela on peut les classer en trois catégories :

- Les réseaux dont les nœuds sont sujets de défaillance (les liens sont parfaitement fiables).
- Les réseaux dont les deux éléments (liens et nœuds) sont défaillants.
- Les réseaux avec la défaillance des liens uniquement (les nœuds sont parfaitement fiables).

Par la suite nous introduisons l'état de l'art de chacune des catégories. Dans cette thèse nous nous intéressons au cas des réseaux dont les liens sont sujets de défaillance. Pour laquelle nous développons les méthodes d'évaluation de fiabilité.

2.3.3 Les réseaux avec la défaillance des nœuds

Dans cette catégorie la défaillance du réseau se traduit par la défaillance de certains nœuds dont le fonctionnement est indispensable au réseau.

2.3.3.1 Le modèle de connectivité des nœuds résiduels

C'est l'un des anciens modèles largement étudié qui est introduit par Frank [59, 60]. Le réseau est représenté par un graphe G simple non orienté (sans boucle ni liens parallèles) avec un ensemble de nœuds V et un ensemble E de liens. Si un ensemble de nœuds sont défaillants, ils sont supprimés de G avec les liens incidents. Le sous graphe résultant est induit par les nœuds restants W, ce sous graphe est noté G_W. les liens de G_W sont les liens ayant leurs extrémités dans W qui appartiennent à E. Si G_W est connexe alors le réseau est opérationnel, et W est un état opérationnel. La fiabilité de la connectivité des nœuds est : $R(G, p) = p$(réseau opérationnel).

Où $p = (p_1, p_2, ..., p_n)$ est un vecteur des probabilités de succès de chaque nœud du réseau. Si pour tous les nœuds du réseau, $p_i = p$, alors $R(G, p) = R$. En outre si tous les nœuds opèrent indépendamment donc

$$R(G, p) = \sum_{i=1}^{n} S_i p^i (1 - p)^{n-i} \tag{2.4}$$

Où S_i est le nombre de sous-graphes de G ayant exactement i nœuds.

De la même façon la fiabilité R d'un réseau est calculée dans le cas où les liens sont sujets de défaillance en utilisant l'équation 2.4. En terme de sous-graphes connexes ayant i liens.

Les coefficients des fonctions de fiabilité des nœuds et liens peuvent aussi être définis en fonction des coupes de nœuds ou des coupes de liens. Il est aussi démontré que le calcul de R en fonctions de la défaillance des liens est un problème NP-difficile. Portant, il existe des algorithmes efficaces qui permettent de calculer R pour certaines classes de graphes [61].

Un réseau uniformément optimal est un réseau ayant une fonction de fiabilité maximale pour toutes les valeurs de p pour tous les réseaux avec le même nombre de nœuds et de liens. Mais que se soit pour les nœuds ou les liens les réseaux uniformément optimaux n'existent toujours pas [62, 63]. En outre il est possible qu'un graphe soit uniformément optimale pour p plus grande ne l'est pas pour p plus petite [64]

Une telle analogie entre le modèle de fiabilité de lien et le modèle de connectivité résiduelle de nœuds, n'est pas complète. Du fait que le modèle de nœud a quelques propriétés non partagées par le modèle de lien.

Le modèle définissant R suppose que chaque graphe résiduel connexe est acceptable quelque soit sa taille. La figure 2.4 montre un exemple de graphe inhabituel.

La fonction de fiabilité n'est pas monotone. Le fait de rendre un nœud plus fiable peut rendre le réseau moins fiable. Le comportement non-monotone n'est présente dans le modèle de fiabilité des liens. On considère un système constitué d'un ensemble E d'éléments et une collection de sous-ensemble de E appelé les états de fonctionnement. Si chaque sur-ensemble d'un état opérationel est un aussi un état opérationnel, donc le système est dit cohérent. Par définition tout système cohérent a une fonction de fiabilité monotone. On considère le graphe G de la figure 2.4, le sous-graphe $G - u - v$ est un état opérationnel. On considère le nœud v opérationnel, qui est précédemment défaillant. Le nouveau sous-graphe, $G - u$, est non connexe puisque, v est un nœud isolé. En outre $G - u - v$ est un état opérationnel mais, $G - u$ ne l'est pas.

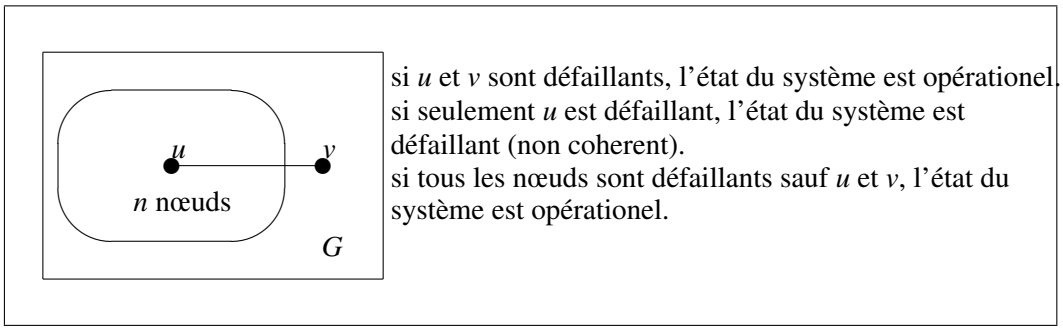


FIGURE 2.4 – Modèle de fiabilité de connectivité des nœuds résiduels

L'approche ci-dessous est une approche traditionnelle dans le sens où elle modélise la non opérabilité du réseau en fonction de la défaillance des nœuds, causée par les coupes de nœuds. Elle est de même pour les coupes de liens qui utilisent la même approche. D'autres

modèles probabilistes qui étudient la vulnérabilité d'un réseau due à la défaillance des nœuds ont été proposé. Amin et al.[65, 66] ont introduit une approche qui utilise le concept du nombre attendue de paires de nœuds qui sont reliés par un chemin dans la mesure de la vulnérabilité. Cette approche permet d'étudier la dégradation catastrophique d'un réseau multiprocesseur. Cependant cette mesure n'est pas une probabilité et elle n'est donc pas une fiabilité, il est difficile de comprendre comment les résultats obtenus par cette approche peuvent être utilisés dans la théorie de fiabilité.

Fotoh et Colbourn [67, 68] ont introduit une approche cohérente et qui ne souffre pas des faiblesses de l'approche traditionnelle introduite ci-dessous. Toutefaois, le modèle proposé par Fotoh et Colbourn suppose qu'un ensemble de k nœuds (k-terminal) sont parfaitement fiables que se soient des hôtes ou destinations qui communiquent via des nœuds commutateurs avec des probabilités de fonctionnement définies. Cette approche, peut être appliquée sur des réseaux de diffusion à base de fréquence radio, mais, ne peut pas être appliquée au cas des réseaux multiprocesseur, car, dans ce cas tous les nœuds sont sujets de défaillance.

2.3.3.2 Le modèle cohérent

Pour contourner les deux problèmes du modèle précédent. On pourrait dans un premier temps envisager qu'un modèle approprié pourrait être obtenue par une révision du dit modèle, dans lequel seulement les sous-graphes connexes avec au moins k nœuds sont dans des états opérationnels. Une telle révision consistera à contourner le problème que les sous-graphes connexes de petite taille sont considérés comme étant opérationnels. Cependant, il y a deux objections évidentes à l'adoption de cette révision particulière :
1) Il est encore en général non cohérent.
2) Plus important encore, du point de vue des réseaux multiprocesseurs, il n'est pas nécessaire d'exiger que chaque collection de plus de k nœuds soit un sous-graphe connexe. L'exigence raisonnable consiste à insister pour que le sous-graphe induit par les nœuds survivants contiennent un composant ayant au moins k nœuds.

Boesch et al[10] ont proposé un nouveau modèle cohérent résolvant le problème de construction de modèles appropriés pour la fiabilité du réseau lorsque les nœuds plutôt que les liens sont sujets de défaillance. Pour appliquer la théorie de fiabilité aux réseaux multiprocesseurs, un état opérationnel est défini comment étant toute collection de nœuds opérationnels qui induit un sous-graphe connexe qui contient au moins 1 composant ayant $l \leq k$ nœuds. Les propriétés de ce modèle sont traitées sous l'hypothèse supplémentaire que les nœuds sont indépendants les uns des autres, tous avec une probabilité p. La fiabilité d'un composant de $k-$nœuds opérationnel peut être noté par $R_{co}^{(k)}(G, p)$, $R_{co}^{(k)}(G)$, ou bien $R_{co}^{(k)}$.

D'où les propriétés du modèle peuvent être observées comme suit :

$$R_{co}^{(1)}(G,p) = 1 - (1-p)^n \tag{2.5}$$

pour $k \geq 2$

$$R_{co}^{(k)}(G,p) = \sum_{i=1}^{n} A_i^{(k)} p^i (1-p)^n - i \tag{2.6}$$

$A_j^{(k)}(G) \equiv$ le nœud j induisant les sous-graphes de G qui contiennent un composant avec au moins k nœuds.
$A_j^{(k)}(G) = 0$ pour $j < k$. Et pour $j \geq \max(k, n - k(G) + 1)$:

$$A_j^{(k)}(G) = \begin{pmatrix} n \\ j \end{pmatrix}$$

$$A_k^{(k)}(G) = S_k(G) \tag{2.7}$$

$A_j^{(k)}(G) \geq S_j(G)$, pour $k + 1 \leq j \leq n$

L'équation 2.7 montre que le calcul de la fiabilité du composant opérationnel associé aux k nœuds est NP-difficile. En effet, si l'on envisage des algorithmes polynomiaux pour calculer $R_{oc}(G)$ pour chaque $1 \leq k \leq n$ et $0 \leq p \leq 1$, donc chaque $A_k^{(k)}(G)$ peut être calculé dans un temps polynomial. Toutefois, cela signifie que chaque $S_k(G)$ et donc $R_n(G)$ peuvent être calculés avec une complexité d'exécution polynomiale. Mais le calcul de $R_n(G)$ est NP-difficile, par conséquent, le calcul du $R_{oc}^{(}k)(G)$ est NP-difficile.

2.3.4 Etudes de fiabilité d'un réseau avec la défaillance des nœuds et des liens

Dans les réseaux informatiques et télécommunication, chaque composant du réseau est sujet de défaillance. Quelques approches ont été proposées dans la littérature pour évaluer et analyser la fiabilité d'un réseau en fonction de la défaillance des nœuds [69, 70, 71, 72, 73, 74, 75, 76, 77].

Les méthodes d'évaluation de la fiabilité de ce type de réseaux peuvent être classifiées en deux catégories ; explicite ou implicite. La méthode explicite comporte deux étapes : tout d'abord une expression de fiabilité symbolique en supposant les nœuds parfaits est déterminée, puis on applique une méthode spéciale comme AGM [72] ou NPR/T [75] explicitement à l'expression qui en résulte pour compenser les nœuds non fiables. Avec la méthode explicite, il n'est pas nécessaire d'appliquer des méthodes spéciales pour tenir compte de la défaillance des nœuds ; la procédure pour le calcul de l'effet des nœuds non fiables est directement intégrée dans l'algorithme, donc l'expression de fiabilité avec les nœuds non fiables est calculée directement. Les méthodes, ENR/KW[77], TPR/NF[76] et KHR[73] sont des

méthodes implicites pour l'obtention de la fiabilité des logiciels avec les nœuds non fiables.

2.3.4.1 La méthode AGM

Cette méthode est la première méthode la plus utilisée qui a été proposée par Aggrawal, Gupta, Misra (AGM). Cette approche est prouvée comme un corollaire du théorème général de décomposition d'un système complexe. Il existe d'autres algorithmes plus efficaces qui en découlent. Cependant le temps de calcul de cette méthode augmente d'une façon exponentielle en fonction du nombre de liens.

La méthode AGM considère que chaque lien dans le réseau (avec la probabilité de défaillance du lien et du nœud donnée) comme une combinaison en série d'un nœud parfait et du lien avec une fiabilité modifiée, comme le montre la figure 2.5.

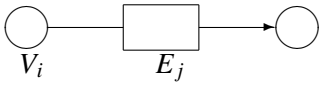


FIGURE 2.5 – Fiabilité modifiée d'un réseau orienté.

Pour le réseau orienté de la figure 2.5, la fiabilité du nœud i est α_i, la fiabilité du lien j est β_j, la fiabilité modifiée du lien j est $\beta'_j = \alpha_i\beta_j$.

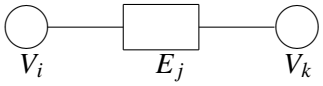


FIGURE 2.6 – Fiabilité modifiée d'un réseau non orienté.

Dans le réseau d'interconnexion, un lien peut être parcouru dans les deux sens. La fiabilité du nœud i est α_i, la fiabilité du nœud k est α_k, la fiabilité du lien j est β_j, la fiabilité modifiée du lien j est $\beta'_j = \alpha_i\alpha_k\beta_j$.

En substituant, α_i peut apparaître dans un terme de produit plus d'une fois. Donc on peut écrire $[\prod_i \alpha_i^{c_i}]^* = [\prod_i \alpha_i]$ Avec c_i est la multiplicité de α_i. Après avoir parcouru tous les nœuds, ils peuvent être considérés comme étant parfaitement fiables, n'importe quel algorithme d'évaluation de fiabilité d'un réseau avec les nœuds fiables peut être utilisé pour calculer la fiabilité du réseau.

La complexité de cette méthode étant exponentielle en fonction du nombre de liens. En outre l'utilisation des calculs symbolique au lieu des calculs directes augmente la complexité en espace mémoire.

2.3.4.2 La méthode NPR/T

La méthode NPR/T (Node Pair Reliability/Torrieri) est une méthode qui permet de calculer la fiabilité des réseaux larges avec des nœuds non fiables, elle est proposée par Torrieri

[75]. La méthode NPR/T est simple et directe, elle est dérivée de la méthode AGM, et peut faire le même travail que cette méthode. Avec la méthode NPR/T, un ensemble de formules concises définies sont utilisées pour déterminer les relations entre un nœud et les liens orientés associés. Par conséquent le coût de cette méthodes augmente linéairement avec le nombre de liens.

Pour les réseaux non orientés, la méthode NPR/T doit transformer ce réseau en un réseau orienté dans lequel chaque lien non orienté est remplacé par deux liens parallèles avec deux sens opposés ; cependant, cette transformation génère des événements dépendants dans la formule de calcul de fiabilité et donc, peut produire des résultats incorrects pour certains réseaux non orientés.

2.3.4.3 La méthode ENR/KW

En se basant sur le partitionnement d'un réseau, Ke et Wang[77] ont proposé la méthode ENR/KW (Evaluating Network Reliability / Ke and Wang). L'idée maîtresse de la méthode ENR/KW est le partitionnement du réseau en un ensemble de sous-réseaux disjoints de petites tailles, en prenant en considération seulement les liens si tous les nœuds sont considérés comme étant parfaits. Chaque sous-réseau est généré d'une façon à maintenir une structure du graphe orienté pour prendre en considération l'effet des nœuds imparfaits. Par conséquent, l'expression de fiabilité pour les nœuds imparfaits peut être directement obtenue du sous-réseau et le graphe orienté associé.

2.3.5 Les réseaux probabilistes et déterministes

A un instant donné les éléments (les liens) du réseau seront dans deux états possible, opérationnel ou défaillant. Dans le cas des réseaux déterministes, on considère qu'un attaquant peut perturber l'ensemble du réseau en attaquant l'élément opérationnel, ce qui provoquera une défaillance du réseau. La défaillance d'un lien signifie qu'il est supprimé du réseau, tandis que, la défaillance d'un nœuds signifie que les liens incidents à ce nœuds sont supprimés du réseau.

Les modèles pour les réseaux déterministes, s'intéressent principalement au cas de la performance du réseau dans le pire des cas, dans lequel un adversaire choisi d'une façon intelligente les éléments à rendre inactif, apportera le maximum de dommage au réseau. Ce type de réseau permet d'évaluer la conservation de performance du réseau, il peut être partiellement utilisé dans la phase de conception du système.

Dans le cas des réseaux probabilistes, à un instant donné, on admet que, les éléments du réseau tombent en panne aléatoirement et indépendamment, avec une probabilité de défaillance connue. Plus précisément, à chaque nœud i est associé une valeur de fiabilité p_i qui

indique la probabilité qu'il soit opérationnel, et à chaque lien l est attribué une fiabilité p_l qui indique la probabilité que l soit opérationnel. En outre à un instant donné les éléments du réseau tombent en panne indépendamment avec les probabilités $q_i = 1 - p_i$ et $q_l = 1 - p_l$.

Dans ces circonstances, on serait peut être intéressé à la performance moyenne du réseau, sous l'hypothèse des défaillances aléatoires des liens. Il est également possible de prendre en considération des modes de défaillance dépendants, dans lesquels des données supplémentaires seront prises en considération et éventuellement augmentera les opérations de calcul. Par exemple, les liens incidents à un nœud peuvent subir des influences communes (conditions météorologiques, interférences, ou blocage), ces liens ont tendance donc à tomber en panne ensemble, plutôt que de façon indépendante, ou encore la défaillance d'un lien peut influencer le fonctionnement d'autres liens, ce qui les rend plus susceptibles de tomber en panne.

La théorie des graphes joue un rôle primordiale dans l'analyse et la conception des réseaux fiables ou des réseaux invulnérables. Selon Boesch [10], on peut utiliser un modèle déterministe, au lieu d'un modèle probabiliste usuel de fiabilité du réseau. Plusieurs critères de vulnérabilité et les résultats de synthèse connexes ont été examinés. Ces problèmes de synthèse sont toutes des questions hors de la théorie des graphes. Certaines d'entre elles peuvent être converties en question de vulnérabilité. Boesch distingue entre deux types de modèles, le premier récapitule les notions relatives à la théorie des graphes et le seconde résume les principaux résultats correspondants à chaque modèle.

2.3.6 Les opérations du réseau

La fiabilité d'un réseau concerne ça capacité à mener à bien une opération réseau désirée. Par conséquent, une étape importante consiste à identifier les opérations réseaux nécessaires.

La première opération la plus importante d'un réseau est le maintien de connexions ou des liaisons entre un nœud sousrce s et un nœud terminal t. La fiabilité (Two-terminal reliability) dans ce cas la, est définie comme étant la probabilité qu'il existe un chemin reliant s à t dans le graphe probabiliste G. Dans le cas des réseaux orientés elle est appelée $s-t$ connectivité.

La seconde opération dans les réseaux est la diffusion (broadcasting). La fiabilité de tous les terminaux (all-terminal reliability) est la probabilité que pour toute paire de nœuds, il existe au moins un chemin qui la relie. Ceci est équivalent à la probabilité qu'il y ait au moins un arbre sous-tendu (spanning-tree) dans le graphe. Dans le cas d'un réseau orienté, l'accessibilité est la probabilité qu'il existe des chemins depuis le nœud source vers tous les autres nœuds du graphe.

La troisième et dernière opération, fait intervenir un ensemble de k nœuds qui peuvent communiquer par paire $2 \leq k \leq n$. La fiabilité k-terminaux (k-terminal reliability) est la probabilité que, pour k nœuds, le graphe contient des chemins entre chaque paire des nœuds k.

2.3.7 Approches pour le calcul de la fiabilité des réseaux probabilistes

Il existe différentes approches générales pour le calcul de la fiabilité des réseaux probabilistes. Supposons que $G = (V, E)$ est un réseau orienté, ayant un nœud source s et un nœud destination t différents. Les nœuds de G sont supposés comme étant parfaits, tandis que les liens $l \in E$ sont supposés tomber en panne de façon indépendante avec des probabilités connues $q_l = 1 - p_l$. Nous allons illustrer quelques approches générales de calcul de la fiabilité $R_{st}(G)$ (two-terminal) d'un réseau qui est la probabilité qu'il existe un chemin de liens opérationnels de s vers t dans G.

2.3.7.1 State-space enumeration

Cette méthode a été proposée par Moore et Shannon [78] est l'une des méthodes fondamentales de calcul de $R_{st}(G)$. Il s'agit d'une stratégie simple qui énumère tous les états possibles (tous les sous-graphes possibles) et détermine ensuite ceux qui sont des chemins (pathsets), enfin fait la somme des probabilités d'occurrence de chaque chemin (pathset). Pour déterminer si un état est un chemin, on emploie des algorithmes de reconnaissance qui utilisent des méthodes de recherche de chemins ou des méthodes qui utilisent les arbres sous-tendus (spanning tree).

Puisque chacun des $m = |E|$ liens de G peut être soit opérationnel ou en panne, donc, l'état du réseau peut être représenté sous forme d'un vecteur $\delta = (\delta_1, \delta_2, ..., \delta_m)$ avec les valeurs 1 ou 0. Le composant l de δ est égale à 1 si le lien l est opérationnel et 0 si il est en panne. En supposant que les liens tombent en panne indépendamment, la probabilité de l'état δ est : $p(\delta) = \prod_{l=1}^{m} p_l^{\delta_l}(1 - p_l)^{1-\delta_l}$ et la fiabilité du réseau est :

$$R_{st}(G) = \sum_{\delta \in D} I_{st}(\delta) P(\delta) \tag{2.8}$$

Avec $I_{st}(\delta)$ une variable qui prend la valeur 1 quand le sous-réseau de l liens opérationnels (ayant $\delta_l = 1$) contient un chemin reliant s à t. D est l'ensemble de tous les états du réseau.

Même si de point de vue conceptuel la méthode state-space est simple, elle est non pratique car $|D| = 2^m$ et le temps de calcul augmente exponentiellement avec la taille du réseau.

Nous illustrons maintenant l'utilisation de cette approche avec un simple réseau avec quatre nœuds et cinq liens (voir la figure 2.7). Ce réseau contiendra un chemin reliant s à t.

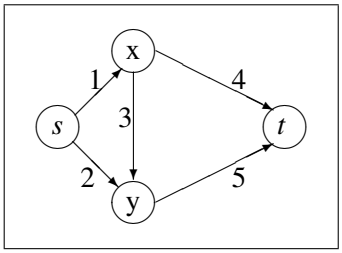


FIGURE 2.7 – Exemple de réseau avec un pont

Même si n'importe quel lien est en panne, ou bien deux autres liens excepte $\{1, 2\}, \{1, 5\}, \{4, 5\}$ sont en panne. En revanche, si trois liens ou plus sont en panne alors, le réseau tombera en panne, sauf si les liens en panne sont $\{1, 3, 4\}$ ou $\{2, 3, 5\}$. Donc la fiabilité R_{st} est :
$R_{st}(G) = p_1p_2p_3p_4p_5+q_1p_2p_3p_4p_5+p_1q_2p_3p_4p_5+p_1p_2q_3p_4p_5+p_1p_2p_3q_4q_5+p_1p_2p_3p_4q_5+$
$q_1p_2q_3p_4p_5 + q_1p_2p_3q_4p5 + p_1q_2q_3p_4p_5 + p_1q_2p_3q_4p_5 + p_1q_2p_3p_4q_5 + p_1p_2q_3q_4p_5 +$
$p_1p_2q_3p_4q_5 + q_1p_2q_3q_4p_5 + p_1q_2q_3p_4q_5$
En substituant $q_l = 1 - p_l$ dans l'équation ci-dessus et en simplifiant, nous obtenons,

$$R_{st}(G) = p_1p_2p_3p_4p_5 - p_1p_2p_3p_5 - p_1p_3p_4p_5 - p_1p_3p_4p_5 + p_1p_3p_5 + p_1p_4 + p_2p_5 \quad (2.9)$$

Bien que 55 termes pourraient avoir résulté de l'exécution de ces substitutions, une bonne partie a été annulée dans l'expression ci-dessus. Puisque seuls les états δ avec $I_{st}(\delta) = 1$ contribuent à l'équation 2.8, il n'est donc pas nécessaire d'examiner tous les états de D, à l'exception de ceux contenus dans l'équation 2.9. Il convient donc de se concentrer directement sur les chemins reliant s à t $\{P_1, P_2, ..., P_k\}$ dans G.

Supposons que E_i est un événement que tous les liens dans le chemin P_i sont opérationnels. Donc la fiabilité R_{st} est la probabilité qu'au moins un événement se produit :

$$R_{st}(G) = P(E_1 \cup E_2 \cup ... \cup E_k) \quad (2.10)$$

La fiabilité (two-terminal) d'un réseau peut être alternativement calculée en utilisant l'ensemble minimale des liens reliant s à t, ou les coupes minimales de G. Un ensemble déconnectant s et t est minimal s'il ne contient pas un ensemble d'autres liens qui déconnectent s et t. Pour cela supposons que les ensembles de coupes $s - t$ sont $\{C_1, C_2, ..., C_r\}$ et soit F_j l'événement que tous les liens dans la coupe C_j sont en panne. Donc la non fiabilité (two-terminal unreliability) $U_{st}(G)$ est :

$$U_{st}(G) = 1 - R_{st}(G) = P(F_1 \cup F_2 \cup ... \cup F_r) \quad (2.11)$$

Les événements E_i et F_j dans les équations 2.10 et 2.11 ne sont pas disjoint. Pourtant, il

existe des méthodes standards pour le calcul de la probabilité de l'union des événements.

2.3.7.2 Le principe d'inclusion exclusion

En utilisant le principe d'inclusion-exclusion, l'équation 2.10 peut être réécrite de la façon suivante :

$$R_{st}(G) = \sum_i P(E_i) - \sum_{1\leq i<j\leq k} P(E_iE_j) + \sum_{1\leq i<j<l\leq k} P(E_iE_jE_l) - ... + (-1)^{k+1}P(E_1E_2...E_k)$$

L'intersection des événements A et B est indiquée par AB. Chaque terme dans l'expression est facile à calculer en se basant sur les hypothèses d'indépendance. Cependant, il existe $2^k - 1$ termes dans l'expression, d'où le temps de calcul augmente exponentiellement avec le nombre de liens.

Pour le réseau de la figure 2.7, il existe trois chemins minimaux reliant s et t : $P_1 = \{1, 4\}$ $P_2 = \{2, 5\}$ $P_3 = \{1, 3, 5\}$

Par conséquent $P(E_1) = p_1p_4$, $P(E_2) = p_2p_5$, $P(E_3) = p_1p_3p_5$, $P(E_1E_2) = p_1p_2p_4p_5$, $P(E_1E_3) = p_1p_3p_4p_5$, $P(E_2E_3) = p_1p_2p_3p_5$, $P(E_1E_2E_3) = p_1p_2p_3p_4p_5$.

L'application du principe d'inclusion-exclusion produit le résultat suivant, $R_{st}(G) = P(E_1)+ P(E_2) + P(E_3) - P(E_1E_2) - P(E_1E_3) - P(E_2E_3) + P(E_1E_2E_3) = p_1p_4 + p_2p_5 + p_1p_3p_5 - p_1p_2p_4p_5 - p_1p_3p_4p_5 - p_1p_2p_3p_5 + p_1p_2p_3p_4p_5$

La formule topologique de Satyanarayana et Prabhakar [79] est la méthode la plus efficace qui se base sur le principe d'inclusion-exclusion, mais le nombre de termes dans l'expression réduite peut augmenter rapidement avec la taille du problème.

2.3.7.3 La méthode du produit disjoint

Une autre technique pour le calcul de la probabilité l'union des événements de l'équation 2.10 ou 2.11, consiste à décomposer $E_1 \cup E_2 \cup ... \cup E_k$ en une union des événements disjoints. Plus précisément on peut écrire
$R_{st}(G) = P(E_1 \cup E_2 \cup ... \cup E_k) = P(E_1 \cup \bar{E}_1E_2 \cup \bar{E}_1\bar{E}_2E_3 \cup ... \cup \bar{E}_1\bar{E}_2\bar{E}_3\bar{E}_{k-1}E_k)$
avec $\bar{E}_i$ dénote le complément de l'événement E_i. Puisque les événements qui composent l'expression ci-dessus sont deux à deux disjoints donc :

$$R_{st} = P(E_1) + P(\bar{E}_1E_2) + P(\bar{E}_1\bar{E}_2E_3) + ... + P(\bar{E}_1\bar{E}_2\bar{E}_3\bar{E}_{k-1}E_k) \tag{2.12}$$

La méthode du produit disjoint implique la sommation de seulement k probabilités. Toutefois, le calcul de la probabilité de chaque composant est impliquée. Il est également à signaler que l'efficacité de cette méthode dépend de l'ordre spécifique des événements E_i.

Un certain nombre de méthodes [80, 81] ont été proposées pour réaliser des produits disjoints, ces variantes de méthodes suivent la même stratégie. En général le nombre de

termes générés augmente rapidement en fonction du nombre k des chemins dans le réseau. En particulier la méthode des produits disjoints peut être utilisée efficacement, pour le calcul de la fiabilité de tous les terminaux (all-terminal) dans les réseaux orientés. Pour les réseaux deux-terminaux aucune méthode efficace n'est connue pour le calcul de la fiabilité.

CHAPITRE 3

Les Réseaux de Neurones

Sommaire

3.1 Introduction

Ce chapitre décrit les réseaux de neurones et l'algorithme de rétropropagation pour leur apprentissage. Dans la première section, nous définissons les réseaux de neurones (RNA). Dans la deuxième section nous abordons le neurone formel qui est l'unité élémentaire des réseaux de neurones. La section trois est consacrée à la discussion des fonctions de transfert. Alors que la quatrième section introduit l'architecture d'un réseau neurones. Une discussion de l'apprentissage des réseaux de neurones est abordée dans la section cinq et six et la dernière discute comment ces réseaux peuvent être utilisés comme des estimateurs de la fiabilité des logiciels.

3.2 Le réseau de neurones

Les réseaux de neurones sont des systèmes de calcul basés sur la simulation mathématique du système biologique nerveux humain [18]. Certes, les modèles de réseaux de neurones ne prennent pas en considération toutes les connaissances scientifiques à propos des systèmes nerveux biologiques, mais ils modélisent seulement quelques caractéristiques rudimentaires qui peuvent être exprimées en équations mathématiques simples. En général, un réseau neuronal peut être caractérisé en termes de trois entités :

1. Les modèles de neurones : les caractéristiques de l'unité de traitement (neurone).
2. Les modèles de la structure d'interconnexion : la topologie de l'architecture et les poids des connexions qui encode la connaissance.
3. L'algorithme d'apprentissage : les étapes à suivre pour ajuster les poids des connexions.

La communauté des chercheurs dans les réseaux de neurones a introduit plusieurs modèles de RN. Par conséquent, il existe une variété des réseaux de neurones et des algorithmes d'apprentissage. Nous présentons dans les sections suivantes quelques concepts relatifs à ces réseaux.

3.3 Le neurone formel

Un "neurone formel" (ou simplement "neurone") est une fonction algébrique non linéaire et bornée [82], dont la valeur dépend de paramètres appelés coefficients ou poids. Les variables de cette fonction sont habituellement appelées "entrées" du neurone et la valeur de la fonction est appelée sa "sortie". Un neurone est donc avant tout un opérateur mathématique dont on peut calculer la valeur numérique par quelques lignes de code. On a pris l'habitude de représenter graphiquement un neurone comme indiqué sur la figure 3.1.

Un neurone est essentiellement constitué d'un intégrateur qui effectue la somme pondérée de ses entrées. Le résultat n de cette somme est ensuite transformé par une fonction de

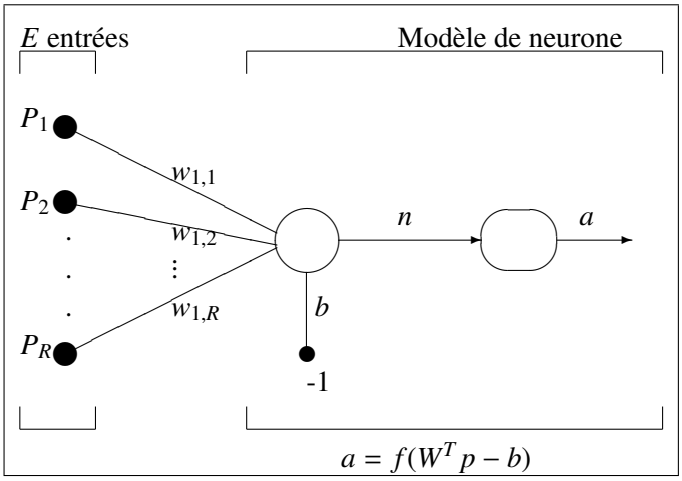


FIGURE 3.1 – Neurone formel

transfert f qui produit la sortie a du neurone. Les R entrées du neurone correspondent au vecteur $P = [p_1, p_2, ..., p_R]$, alors que $W = [w_{1,1}, w_{1,2}, ..., w_{1,R}]$ représente le vecteur des poids du neurone. La sortie n de l'intégrateur est donnée par l'équation suivante :

$$n = \sum_{i=1}^{R} w_{1,j} p_j - b = w_{1,1} p_1 + w_{1,2} p_2 + ... + w_{1,R} p_R - b \tag{3.1}$$

que l'on peut écrire sous forme matricielle : $n = W^T P - b$. Cette sortie correspond à une somme pondérée des poids et des entrées moins ce qu'on nomme le biais b du neurone. Le résultat n de la somme pondérée s'appelle le niveau d'activation du neurone. Lorsque le niveau d'activation atteint ou dépasse le seuil b, alors l'argument de f devient positif (ou nul). Sinon, il est négatif.

3.4 Fonctions de transfert

Plusieurs fonctions de transfert pouvant être utilisées comme fonctions d'activation du neurone. Elles sont énumérées dans le tableau 3.1. Les plus utilisées sont les fonctions "seuil", "linéaire" et "sigmoïde".

Comme son nom l'indique, la fonction seuil applique un seuil sur son entrée. Plus précisément, une entrée négative ne passe pas le seuil, la fonction retourne alors la valeur 0 (on peut interpréter ce 0 comme signifiant faux), alors qu'une entrée positive ou nulle dépasse le seuil, et la fonction retourne 1 (vrai). Utilisée dans le contexte d'un neurone, cette fonction est illustrée dans la figure 3.2a. On remarque ainsi que le biais b dans l'expression de $a = seuil(W^T P - b)$ détermine l'emplacement du seuil sur l'axe $W^T P$, où la fonction passe de 0 à 1. Cette fonction permet de prendre des décisions binaires.

Nom de la fonction	Relation d'entrée/sortie
seuil	$a = 0$ si $n < 0$ $a = 1$ si $n \geq 0$
seuil symétrique	$a = -1$ si $n < 0$ $a = 1$ si $n \geq 0$
linéaire	$a = n$
linéaire saturée	$a = 0$ si $n < 0$ $a = n$ si $0 \leq n \leq 1$ $a = 1$ si $n > 1$
linéaire saturée symétrique	$a = -1$ si $n < -1$ $a = n$ si $-1 \leq n \leq 1$ $a = 1$ si $n > 1$
linéaire positive	$a = 0$ si $n < 0$ $a = n$ si $n \geq 0$
simgmoïde	$a = \frac{1}{1+\exp^{-n}}$
tangente hyperbolique	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$
compétitive	$a = 1$ si n maximum $a = 0$ autrement

TABLE 3.1 – Fonctions de transfert $a = f(n)$.

La fonction linéaire est très simple, elle affecte directement son entrée à sa sortie : $a = n$. Appliquée dans le contexte d'un neurone, cette fonction est illustrée à la figure 3.2b. Dans ce cas la sortie du neurone correspond à son niveau d'activation dont le passage à zéro se produit lorsque $W^T P = b$.

La fonction de transfert sigmoïde quant à elle, est illustrée dans la figure 3.2c. Elle ressemble soit à la fonction seuil soit à la fonction linéaire, selon que l'on est loin ou près de b, respectivement. La fonction seuil est fortement non linéaire, car il y a une discontinuité lorsque $W^T P = b$. La sigmoïde est un compromis intéressant entre les deux précédentes. Notons finalement que la fonction "tangente hyperbolique" est une version symétrique de la sigmoïde.

3.5 Architecture du réseau de neurones

L'architecture d'un réseau de neurones peut être caractérisée en termes de deux attributs : (1) le nombre de couches dans le réseau et (2) la topologie utilisée [18].

1. **Le nombre de couches :** les neurones qui agissent en tant qu'interface entre les données externes et le réseau constituent la couche d'entrée. Généralement, la couche d'entrée n'est impliquée dans aucun calcul utile. Toutefois, chaque neurone dans la couche d'entrée agit comme un point de distribution pour les entrées externes. Les neurones qui interfacent l'environnement extérieure avec le réseau constituent la

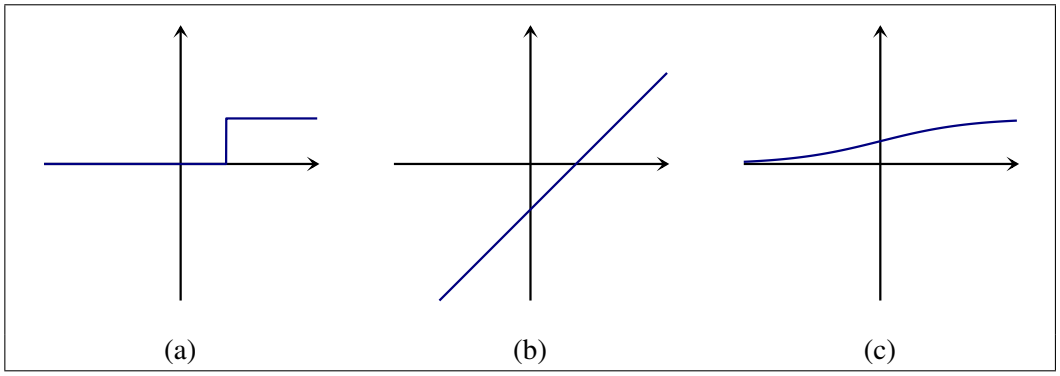


FIGURE 3.2 – Fonction de transfert : (a) du neurone "seuil" ; (b) du neurone "linéaire", et (c) du neurone "sigmoïde".

couche de sortie. Les couches n'ayant pas de connexions directes avec l'environnement externe s'appellent les couches cachées. Le nombre de couche d'un réseau peut varier entre un minimum de deux (une couche d'entrée et une couche de sortie) jusqu'à un nombre entier positif. Dans la littérature des réseaux neuronaux, certains auteurs ne considèrent pas la couche d'entrée comme une couche [18]. Le réseau ayant plusieurs couches cachées (appelé aussi réseau multicouches) permet de développer sa représentation interne du problème.

2. **Type de la connectivité :** basé sur la connectivité et la direction dans laquelle les liens propagent les valeurs d'activation, on peut classer les réseaux multicouches en deux classes : les réseaux feed-forward (l'information circulent des entrées vers les sorties sans retour en arrière) comme illustré dans la figure 3.3a, et les réseaux récurrents (figure 3.3b). Les réseaux multicouches sont parmi les réseaux les plus utilisés pour des problèmes de classification, d'approximation et de prédiction. Le nombre de couches et de neurones dans chaque couche dépend du problème à résoudre.

En se basant sur la façon avec laquelle les connexions sont établies, les réseaux récurrents peuvent être classés en trois catégories : les réseaux récurrents simples proposés par Elman [83], les réseaux semi récurrents proposés par Jordan [84] et les réseaux totalement récurrents [85], ces types de réseaux sont discutés dans [18]. L'expérience a montré que les réseaux feedforward (figure 3.3a) et le réseau de Jordan modifié (figure 3.3b) sont utiles dans les applications de l'ingénierie de la fiabilité des logiciels.

Un réseau de neurones est un maillage de plusieurs neurones, généralement organisé en couches. Pour construire une couche de S neurones, il s'agit simplement de les assembler comme à la figure 3.4. Si les S neurones d'une même couche sont tous branchés aux entrées R, alors on dit que la couche est totalement connectée. Un poids $w_{i,j}$ (i désigne le numéro du neurone dans la couche, j : le numéro de l'entrée) est associé à chacune des connexions.

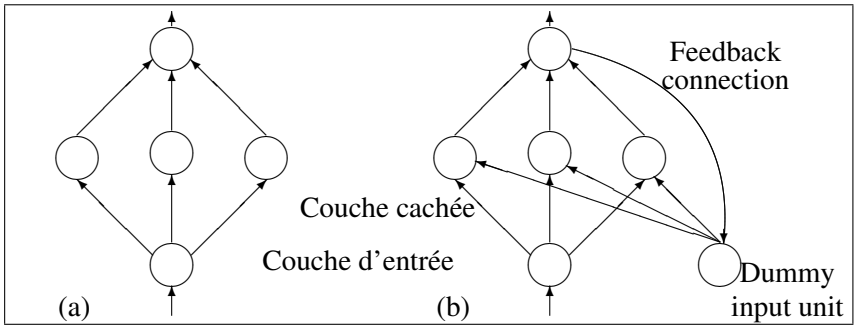


FIGURE 3.3 – (a) réseau feedforward standard et (b) réseau de Jordan modifié.

L'ensemble des poids d'une couche forme donc une matrice W de dimension $S\,xR$:

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & \cdots & \cdots & w_{2,R} \\ \vdots & \cdots & \cdots & \vdots \\ w_{S,1} & \cdots & \cdots & w_{S,R} \end{pmatrix}$$

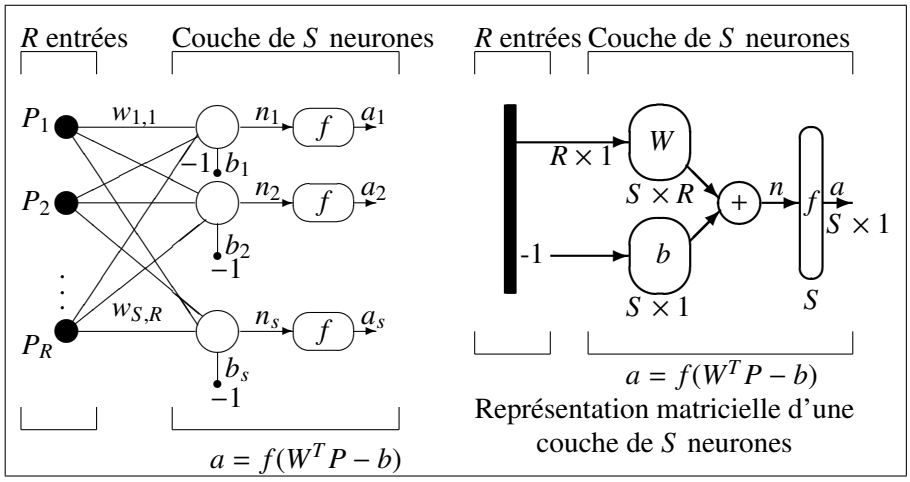


FIGURE 3.4 – Couche de S neurones.

Pour construire un réseau, il suffit de combiner des couches comme dans la figure 3.5.

Le réseau de la figure 3.5, comporte R entrées et trois couches (deux couches cachées et une couche de sortie). Il importe de remarquer dans cet exemple que les couches qui suivent la première ont comme entrée la sortie de la couche précédente. Ainsi, on peut enfiler autant de couche que l'on veut (au moins en théorie). La fonction de transfert peut changer d'une couche à l'autre. En effet, dans le cas général, $f^1 \neq f^2 \neq f^3$.

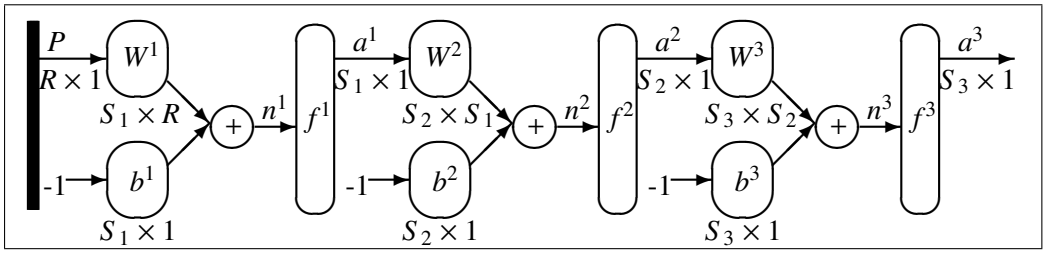


FIGURE 3.5 – Représentation matricielle d'un réseau de trois couches.

3.6 Apprentissage des réseaux de neurones

Pour résoudre un problème à l'aide d'un réseau de neurone, ce dernier doit être entraîné par un ensemble des paires de données d'entrées sorties typiques. Ce processus est connu sous le nom d'apprentissage. La procédure par laquelle le réseau apprend s'appelle un algorithme d'apprentissage. Durant la phase d'apprentissage du réseau, les poids d'une connexion sont ajustés pour réduire l'erreur résiduelle résultante à partir des données d'apprentissage. Il existe une multitude des algorithmes d'apprentissage des réseaux multicouches. Parmi eux, l'algorithme de rétro propagation qui est largement utilisé. L'algorithme de rétro propagation a été proposé indépendamment par plusieurs chercheurs [86]. Au milieu des années quatre vingt, Rumelhart, Hinton et Williams [87] publient l'algorithme de rétro propagation par l'entraînement des réseaux multicouches pour résoudre des problèmes intéressants.

3.7 L'algorithme de rétropropagation

L'algorithme de rétropropagation est un algorithme de la classe d'apprentissage supervisé dans lequel les poids des connexions du réseau sont itérativement adaptés en utilisant la rétropropagation des erreurs de la couche de sortie. Afin d'illustrer le fonctionnement de l'algorithme de rétro propagation, on considère un réseau feedforward à trois couches construites en utilisant des neurones avec comme fonction d'activation sigmoïde.

Algorithme : (1) Initialiser les poids de différents neurones avec des valeurs aléatoires. (2) Ajuster les poids en incrémentant pendant certaines itérations. Dans chaque itération, ajuster les poids dans la direction de descente du gradient de la surface de l'erreur. Continuer cet ajustement itératif jusqu'à l'obtention d'une valeur d'erreur minimale, ou bien jusqu'à un nombre d'itération maximum. Durant chaque itération, la somme quadratique des erreurs est calculée comme suite :

$$E = \frac{1}{2} \sum_{r=1}^{R} \sum_{j=1}^{M} (y_j^r - a_j^r)^2 \tag{3.2}$$

Où y_r est la sortie désirée associée au vecteur d'entrée P, a_r est la sortie actuelle du réseau, M est le nombre de neurones dans la couche de sortie et R est le nombre d'échantillons des données d'apprentissage. Cette erreur est rétro propagée dans le réseau pour ajuster les poids en utilisant la descente du gradient. La procédure de descente du gradient change les poids d'une quantité proportionnelle à la dérivée partielle de l'erreur pour chaque poids $w_{i,j}$. Ainsi, le changement du poids du lien à partir du neurone i au neurone j à la $t^{\text{ème}}$ itération est donné par :

$$\triangle w_{i,j}(t) = \mu\delta_j a_i + \alpha\Big(w_{i,j}(t) - w_{i,j}(t-1)\Big) \tag{3.3}$$

donc

$$w_{i,j}(t+1) = w_{i,j}(t) + \mu\delta_j a_i + \alpha\Big(w_{i,j}(t) - w_{i,j}(t-1)\Big) \tag{3.4}$$

Avec μ est le taux d'apprentissage, α le gain, $\triangle w_{i,j}(t)$ la valeur de changement du poids durant l'itération précédente, a_i la sortie calculée par le neurone i et δ_j la dérivée partielle de l'erreur du neurone j. La dérivée partielle de l'erreur pour le $j^{\text{ème}}$ neurone dans la couche de sortie est donnée par :

$$\delta_j = a_j(1 - a_j)(y_j - a_j) \tag{3.5}$$

et pour le $k^{\text{ème}}$ neurone dans la couche cachée :

$$\delta_k = h_k(1 - h_k)\sum_{j=1}^{M}(\delta_j w_{i,j}) \tag{3.6}$$

Où h_k est sa valeur de sortie.

Cet algorithme nécessite une fonction continue, non-linéaire et différentiable comme fonction objectif à optimiser.

Bien que l'algorithme de rétropropagation puisse être utilisé pour entraîner des réseaux multicouches, il nécessite un savoir-faire pour une utilisation efficace. Puisque l'algorithme de rétropropagation est une procédure d'optimisation de descente de gradient, il est vulnérable au problème de la convergence prématurée. La convergence prématurée se produit chaque fois que l'algorithme se coince dans un minimum local et la valeur de E (voir 3.2) est encore supérieure à la limite de tolérance permise.

En outre, de multiples variables sont à ajuster précisément en fonction du problème traité. Parmi ces variables à fixer, citons par exemple : les paramètres apparaissant dans les différentes équations (le taux d'apprentissage μ, le gain de la procédure de gradient α, ...), la sélection des exemples pour l'apprentissage et le test, l'ordre de présentation et les distributions relatives des exemples dans la base d'apprentissage, le choix du codage des informations en entrée et en sortie, la structure du réseau (présence éventuelle de connexions directes de la couche d'entrée sur la couche de sortie pour traiter à ce niveau la partie li-

néaire du problème, limitation pratique du nombre de couches, taille de la couche cachée), la configuration initiale des poids, le nombre d'itérations d'apprentissage, ...

3.8 Conclusion

Les réseaux de neurones peuvent être regardés comme des modèles non linéaires paramètriques [1] possédant des propriétés d'approximation universelle. D'où l'interêt de leur utilisation dans la construction de modèles de prédiction en général et la fiabilité des logiciels en particulier. Mais le problème confronté dans l'application des RNA réside dans leur paramètrage dont le nombre de couches, le nombre de neurones par couche, la connectivité, et leur apprentissage. Ces problèmes sont des problèmes d'optimisation qui peuvent être résolus par des méthodes appropriées. Nous nous intéressons ici aux algorithmes d'évolution ou évolutionnistes et le récuit simulé qui seront l'objet du chapitre suivant dans lequel nous dérivons ces algorithme et leur utilisation.

1. Contrairement aux modèles paramètriques clqassiques, les RNA ne permettent pas en général de mettre en évidence la paramétrisation qui correspond à une classe de fonctions à approximer

CHAPITRE 4

Les algorithmes evolutionnistes et le récuit simulé

Sommaire

4.1 Introduction

Les algorithmes évolutionnistes et le recuit simulé appartiennent à la famille des algorithmes métaheuristiques, qui regroupe un ensemble d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace. Les métaheuristiques sont généralement des algorithmes stochastiques itératifs, qui progressent vers un optimum global, c'est-à-dire l'extremum global d'une fonction, par échantillonnage d'une fonction objectif. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution (d'une manière proche des algorithmes d'approximation). La section 4.2 du présent chapitre est consacrée aux algorithmes évolutionnistes et la section 4.3 introduit l'algorithme du recuit simulé.

4.2 Les algorithmes évolutionnistes

Le succès des méthodes basées sur les algorithmes génétiques (AG) n'a cessé de grandir depuis leur apparition, notamment dans le domaine de la recherche opérationnelle et l'intelligence artificielle. Cependant, leurs applications à certains problèmes comme les problèmes dont opèrent d'autres approches, telles que les réseaux de neurones sont limités. Par conséquent, des extensions ont été développées [88] ce qui a donné naissance aux algorithmes évolutionnistes.

4.2.1 Les algorithmes génétiques

4.2.1.1 Définition et origine

Les algorithmes génétiques s'inspirent de l'évolution darwinienne des populations biologiques pour définir leurs mécanismes. Le principe de cette évolution est que, dans une population des individus ce sont les plus forts, c'est-à-dire les mieux adaptés au milieu, qui survivront et pourront donner une descendance ou une bonne progéniture.

C'est en 1950 que les algorithmes génétiques ont vu le jour par des biologistes utilisant des ordinateurs pour simuler l'évolution des organismes.

Vers 1960, John Holland et son équipe adaptaient ces algorithmes pour la recherche de solutions à des problèmes d'optimisation [89], en développant une analogie entre un individu dans une population et une solution d'un problème dans un ensemble de solutions.

En effet, un individu dans une population est caractérisé par son empreinte génétique autrement dit, un ensemble de chromosomes issus de la recombinaison des empreintes de ses deux parents, obtenu par croisement (en anglais : "Crossover") ou modifié par mutation. Le

croisement correspond à la reproduction sexuée des individus dans une population en respectant les phénomènes d'hérédité. Ainsi, lorsque deux individus considérés comme assez forts s'accouplent, ils vont créer un nouvel individu, membre de la génération suivante, qui aura lui-même de bonnes chances d'être assez fort et de résister à la sélection naturelle, ce qui n'est pas le cas pour les individus plus faibles. La mutation représente les modifications de l'empreinte génétique qui peuvent se produire sur les individus d'une génération à l'autre et qui évitent la dégénérescence de la population.

Par analogie, dans un AG, un individu ou une solution est caractérisé par une structure de données qui représente son empreinte génétique. La force d'un individu, encore appelée son "fitness", peut être mesurée par la valeur de la fonction d'évaluation correspondante. Les opérateurs génétiques de croisement et de mutation agissent sur les structures de données associées aux individus et permettent de parcourir l'espace des solutions du problème. Le renouvellement de la population, c'est-à-dire de l'ensemble de solutions courantes, autrement dit la création d'une nouvelle génération, est obtenue par itération de l'AG qui va créer de nouveaux individus et détruire d'autres, c'est équivalent au mécanisme de sélection naturelle.

L'exécution d'un tel algorithme doit continuer, à partir d'une population initiale, après de nombreuses générations, aboutir à une population finale où les individus sont tous très forts, en d'autres termes, à un ensemble de meilleures solutions au problème considéré.

Avant d'expliquer le fonctionnement d'un algorithme génétique, nous devons définir quelques termes souvent utilisés pour décrire un algorithme génétique :

- **Individu :** c'est une solution réalisable, on l'appelle aussi Génotype ou chromosome. Il correspond au codage sous forme de gènes d'une solution potentielle à un problème d'optimisation.
- **Gène :** un chromosome est composé de gènes. Dans le codage binaire, un gène vaut soit 0 soit 1.
- **Phénotype :** chaque génotype (individu) représente une solution potentielle à un problème d'optimisation. La valeur de cette solution potentielle est appelée le phénotype.
- **Population :** une population est un ensemble de chromosomes (individus).

En résumé, pour mettre en œuvre un AG, il est nécessaire de disposer :

- D'une représentation génétique du problème, c'est-à-dire d'un codage approprié des solutions sous forme de chromosomes.
- D'un moyen de créer la population initiale.
- D'une fonction d'évaluation ou fonction objective pour mesurer la force de chaque chromosome.
- D'un mode de sélection des chromosomes à reproduire.
- D'opérateurs génétiques adaptés au problème.
- De valeurs pour les paramètres qu'utilisent l'algorithme : la taille de la population, la

probabilité d'appliquer tel ou tel opérateur.

4.2.1.2 Corps d'un algorithme génétique

Les étapes de base d'un algorithme génétique sont comme suit :

Etape0 : Définir un codage du problème.
Etape1 : Créer une population initiale P_0 de N individus$X_1, X_2, ..., X_N$
i := 0 ;
Etape2 : Evaluation des individus ; Soit F la fonction d'évaluation. Calculer $F(X_i)$ pour chaque individu X_i de P_i
Etape3 : Sélection ; Sélectionner les meilleurs individus (au sens de F) et les grouper par paire.
Etape4 : Application des opérateurs génétiques.

1. Croisement : Appliquer l'opération de croisement aux paires sélectionnées.
2. Mutation : Appliquer la mutation aux individus issus du croisement.
3. Ranger les nouveaux individus obtenus (de 1 et 2) dans une nouvelle génération P_{i+1}.

Etape5 : Répéter les étapes 2, 3 et 4 jusqu'à l'obtention du niveau de performance souhaité.

Nous détaillons dans les paragraphes suivants les cinq étapes de l'AG.

4.2.1.3 Codage

Le premier pas dans l'implémentation des algorithmes génétiques est de créer une population initiale d'individus. Par analogie avec la biologie, chaque individu de la population est codé par un chromosome ou un génotype [89].

Une population est donc un ensemble de chromosomes. Chaque chromosome code un point dans l'espace de recherche. L'efficacité de l'algorithme génétique va donc dépendre du choix du codage d'un chromosome. Il faut retenir que non seulement le codage dépend de la nature de la solution, mais aussi de la fonction à optimiser.

Dans le cas des AG, c'est le codage binaire qui est le plus utilisé. Le chromosome est alors un vecteur dont les éléments ou les gènes appartiennent à {0,1}. Nous verrons plus loin, dans la section 4.2.2, que dans l'extension des AG, d'autres types de codage ont été proposés et utilisés.

4.2.1.4 Construction de la population initiale

La population initiale est générée d'une manière aléatoire. Cependant, il est très recommandé de disposer d'un mécanisme permettant de générer une population non homogène qui servira de base pour les générations futures. Ce choix conditionne la rapidité de la convergence vers l'optimum. Cela est dit, dans le cas où l'on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit répartie sur l'espace de recherche.

4.2.1.5 Evaluation des individus

Evaluer un individu ou un chromosome consiste à calculer sa force. Cette évaluation permettra aux individus les plus forts d'être retenus lors de la sélection. Le but de l'AG est de maximiser la force des individus de la population. Soit $C(x)$ la valeur du critère à optimiser pour l'individu x. Si ce critère doit être maximisé, il peut servir directement de mesure de la force. Dans le cas où l'on désire minimiser ce critère, il est nécessaire de le complémenter pour se ramener à un cas de maximisation. Goldberg [90] propose donc de calculer la force $F(x)$ d'un individu x de la façon suivante :

$$F(x) = \begin{cases} C_{max} - C(x) & si \;\; C(x) > 0 \\ 0 & autrement \end{cases} \tag{4.1}$$

Où C_{max} peut être un coefficient fixé ou la plus grande valeur observée de $C(x)$ soit dans une population, soit depuis le début de l'algorithme. Il y a d'autres alternatives possibles comme :

$$F(x) = \frac{1}{((1 + C(x)) - C_{min})} \tag{4.2}$$

Pour les problèmes de minimisation, C_{min} est la valeur minimale estimée de la fonction objective. Pour les problèmes de maximisation, la fonction d'évaluation devient :

$$F(x) = \frac{1}{((1 + C_{max}) - C(x))} \tag{4.3}$$

Aucune condition de régularité n'est requise pour la fonction objective. Il suffit simplement que cette fonction retourne des valeurs numériques comparables. La performance de l'AG peut être sensible au choix de cette fonction.

4.2.1.6 Sélection

La sélection consiste à choisir les individus à partir desquels on va créer la génération suivante. Comme dans la sélection naturelle, un caractère stochastique est introduit dans la probabilité de sélection qui est souvent basée sur la fonction d'évaluation. Les individus sélectionnés sont placés dans un bassin de reproduction dans lequel auront lieu des opérations

de croisement et de mutation. Plusieurs procédures de sélections existent. On peut citer la sélection proportionnelle basée sur le principe de la roulette [90]. Par rang de classement dans la population [91], par tournoi et par élitisme.

L'opérateur de sélection est caractérisé par un paramètre connu sous le nom de la "pression sélective", il est défini comme la rapidité avec laquelle le meilleur individu devrait couvrir toute la population sous l'effet d'applications répétées [90, 91]. Si le temps de sélection est grand (c'est-à-dire que le meilleur individu n'a pu couvrir la population qu'après un grand nombre d'itération), alors le temps de la pression sélective de l'opérateur est petit, et vice versa. Ainsi, la pression sélective est un paramètre important pour la réussite des algorithmes génétiques.

Cependant, si un opérateur de sélection a une grande pression sélective, la population perd sa diversité rapidement. Pour éviter la convergence prématurée vers un optimal local, nous avons le choix entre l'utilisation d'une population très large pour augmenter le degré de la diversité, ou bien l'utilisation d'opérateurs qui provoquent de grandes perturbations. D'un autre côté, un opérateur de sélection avec une pression sélective réduite mène vers une lente convergence qui permet aux opérateurs de recombinaison et de mutation un grand nombre d'itérations, dans le but d'améliorer l'exploration de leur propre espace de recherche.

A. Sélection de la roulette (Roulette Wheel Selection) : La forme la plus populaire de la sélection est la sélection par roulette. Cette approche a été proposée et analysée par Holland [89] et a été largement utilisée dans l'implémentation des algorithmes génétiques.

La roulette donne à chaque individu une tranche de la roulette (voir la figure 4.1) dont la surface est équivalente à la fitness de l'individu. La roulette tourne et la balle se pose dans l'une des parties sous forme de coin, et qui correspond à l'individu sélectionné. Donc, pour chaque tour de la roulette, un seul individu sera sélectionné.

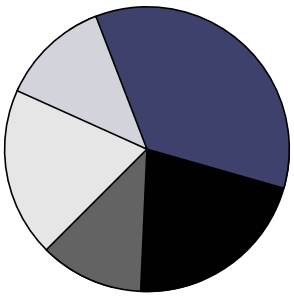


FIGURE 4.1 – Position des chromosomes dans la roulette

Nous décrivons son processus ci-dessous :

On affecte à chaque individu X_i une probabilité d'apparition $p(X_i)$, appelée encore force

relative :

$$p(X_i) = \frac{F(X_i)}{\sum_{k=1}^{N} F(X_k)} \tag{4.4}$$

La sélection d'un individu se fait de la manière suivante :

$$soit \qquad q_i = \sum_{k=1}^{N} p(X_k) \tag{4.5}$$

La probabilité d'apparition cumulée d'un individu X_i est : soit r un nombre aléatoire compris entre 0 et 1. L'individu retenu est :

$$\begin{cases} X_1 \quad si \quad q_i < r \\ X_i \quad si \quad q_{i-1} < r \leq q_i \quad pour\ 2 \leq i \leq N \end{cases} \tag{4.6}$$

N étant le nombre d'individu dans la population.

Ce processus est répété N fois. Avec ce principe, un individu fort peut être sélectionné plusieurs fois. Ce qui peut conduire à une convergence prématurée. En effet, la roulette ne permet pas de contrôler la pression de sélection. Si quelques super individus se dégagent, ils seront toujours sélectionnés, par contre, les individus ayant une faible fitness auront moins de chance d'être sélectionnés, ce qui nuira à l'algorithme évolutionnaire.

B. Sélection par rang de classement : Dans ce schéma de sélection, les individus d'une population sont classés dans une liste selon l'ordre croissant de leur évaluation, ensuite la sélection est proportionnelle à leur rang dans la liste de la population [91]. Cette méthode est utilisée pour une fonction d'évaluation dont les valeurs sont très proches.

Le classement permet de calculer la nouvelle fonction d'évaluation basée sur le rang :

$$F = max - \frac{(rang - 1)(max - min)}{N - 1} \tag{4.7}$$

N est la taille de la population, $1 < max \leq 2$, $min = 2 - max$ et $rang = \{1, 2, ..., N\}$.

Avec cette nouvelle fonction d'évaluation, les meilleurs ont toujours plus de chance d'être choisis, mais moins souvent que la roulette et les moins bons individus auront plus de chance de participer au bassin de reproduction.

C. Sélection par tournoi : Dans la sélection par tournoi, un groupe de (n) individus est choisi aléatoirement (avec ou sans remise) à partir d'une population. Le meilleur individu est sélectionné de façon déterministe (tournoi déterministe) ou probabiliste (tournoi stochastique) [92].

- Tournoi déterministe : le paramètre de ce type est la taille du tournoi (le nombre d'individus (n) qui participent à ce tournoi). Ce nombre varie de 2 jusqu'au nombre total des individus de la population. Dans sa forme la plus simple (appelée sélection binaire par tournoi), deux individus sont choisis aléatoirement (avec ou sans remise), le meilleur

de ces deux individus est sélectionné. L'avantage de la sélection déterministe est que la pression sélective peut être contrôlée. En effet, plus la taille du tournoi augmente plus la pression sélective augmente. Cependant, ce type de sélection ne retourne jamais le plus "mauvais" des individus.

- Tournoi stochastique : ce type se base sur une probabilité pour choisir l'individu gagnant au tournoi. Une des méthodes utilisée dans ce type de tournoi est de calculer une probabilité pour chaque candidat au tournoi. Cette probabilité (p_i) dépend des fitness f_i et f_j des deux individus en compétition :

$$p_i = \frac{f_i}{f_i + f_j} \tag{4.8}$$

D. Sélection par Elitisme : L'élitisme [93] est une stratégie qui permet de garder les meilleurs chromosomes au-delà d'une seule génération. En effet, les individus sont triés selon leurs valeurs de fitness. Seule le nombre d'individus, fixé dans l'algorithme et correspond aux meilleurs individus, est sélectionné. Cette méthode induit une convergence prématurée de l'algorithme : la pression sélective est trop forte. Pour remédier à ce problème, il est nécessaire de maintenir une diversité génétique suffisante dans la population (par exemple on génère le reste de la population selon l'algorithme de reproduction usuel).

4.2.1.7 Opérateurs génétiques

Les parents sélectionnés sont introduits dans le bassin de reproduction où ils seront de nouveaux choisis aléatoirement, pour voir leurs chromosomes subir des transformations par les opérateurs génétiques. Le croisement réalise une opération binaire ou sexuée, et nécessite deux parents. La mutation est une opération unaire ou asexuée, utilisée pour introduire une faible variation dans la solution ou changer la direction de la recherche.

A. Opérateur de croisement : Le croisement le plus simple (1-point figure 4.2) fonctionne de la manière suivante. Soient deux chromosomes X et Y devant subir le croisement, un nombre aléatoire entier l tiré entre 1 et $m-1$, m étant le nombre de gènes d'un chromosome. Le nombre l indique la position de coupure dans les deux chromosomes :

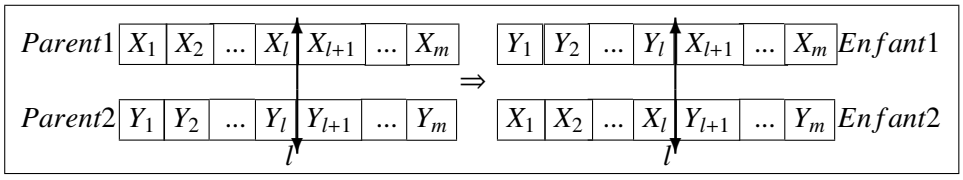


FIGURE 4.2 – Croisement simple (1-point)

Ce processus est appliqué à chaque paire de chromosomes sélectionnés avec une certaine

probabilité P_s. Les paires de chromosomes sont recopiées sans modification dans la génération suivante avec la probabilité $1 - P_{cross}$. Ce croisement est appelé croisement '1-point' car un seul point de coupe est effectué dans le chromosome. Cette notion est généralisée au croisement 'k-points' où k points de coupe sont effectués dans le chromosome, générant $k+1$ génomes qui sont re-combinés pour créer deux chromosomes fils. Le premier chromosome fils est alors la concaténation des génomes du numéro paire du premier parent et des génomes du numéro impaire du deuxième parent. Le deuxième fils est obtenu de manière symétrique.

Reprenons les deux chromosomes X et Y auxquels on applique un croisement '2-points' (figure 4.3) :

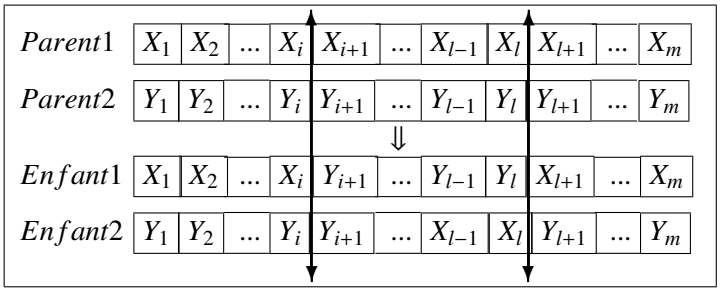


FIGURE 4.3 – Croisement en deux point (2-points)

B. Opérateur de mutation : La mutation a pour rôle de diversifier un matériel génétique, elle constitue un mécanisme important pour produire de nouvelles informations génétiques. Holland [89] et Goldberg [90] ont souligné que la mutation sert comme un "opérateur d'arrière plan" et qui supporte l'opérateur de croisement, elle assure la diversité du "pool" de gènes de la population déjà recombiné par le croisement. C'est pour cette raison que des algorithmes génétiques utilisent la mutation combinée avec le croisement.

L'opérateur de mutation est appliqué avec une certaine probabilité P_{mut}, aux individus issus du croisement. Des valeurs très petites de P_{mut} $[0.001, 0.01]$ sont recommandées pour les algorithmes génétiques canoniques [94, 95]. Cependant, des résultats empiriques établis par Forgarty [96] ont favorisé un taux de mutation initialement grand et qui décroît de façon exponentielle au cours du temps.

La mutation la plus classique consiste à sélectionner aléatoirement un gène du chromosome d'un individu et à modifier sa valeur.

Exemple :

$$000011\boldsymbol{0}11011 \Rightarrow 000011\boldsymbol{1}11011$$

4.2.1.8 Critère d'arrêt :

Le critère d'arrêt peut être arbitraire, par exemple le nombre maximal de générations, ou basé sur le critère de convergence décrit ci-dessous.

4.2.1.9 Convergence :

Holland a défini des propriétés concernant la convergence des AG en introduisant la notion de schéma [97].

Un schéma est un masque qui permet d'explorer les similitudes entre chromosomes. La mise en œuvre de schémas nécessite d'ajouter à l'alphabet des gènes un symbole supplémentaire * qui représente indifféremment 0 ou 1. Ainsi, un schéma 1*10 représente les chromosomes 1010 et 1110. D'une manière générale, un schéma représentera 2^r chromosomes, r étant le nombre de symboles * présents dans le schéma. De même, un chromosome de longueur m est représenté par 2^m schémas. Les schémas sont caractérisés par leur ordre et leur longueur de définition. L'ordre d'un schéma est le nombre de 0 et 1 contenus dans ce schéma. C'est aussi la différence entre la longueur du chromosome et le nombre de symboles *. La longueur de définition d'un schéma est la distance entre la première et la dernière position fixe du chromosome ; une position fixe est un gène dont la valeur est différente de *. Par exemple, pour le schéma ***001*110, la longueur de définition est 10-4=6. Cette notion permet de calculer la probabilité de survie d'un chromosome après un croisement.

A. Théorème des schémas : Dans un algorithme génétique, si le croisement est du type "un-point" et si la sélection repose sur le principe de la roulette, alors, le nombre de chromosomes correspondant à des schémas de longueur de définition courte, d'ordre faible et de force supérieure à la moyenne, augmente au cours des itérations.

En résumé, les travaux de Holland [97] montrent que, si les chances de reproduction sont proportionnelles à la force (évaluation), les schémas de force au-dessus de la moyenne auront tendances à apparaître plus fréquemment dans la prochaine génération. Au contraire, les schémas de force en dessous de la moyenne deviendront de plus en plus rares. Par ailleurs, la longueur de définition d'un schéma est liée à la position des gènes dans le chromosome. La convergence d'un algorithme génétique dépend donc des opérateurs génétiques mis en œuvre. De leur côté K. De Jong [94], et Greffenstette [95] ont proposé des mesures statistiques permettant aussi de juger la convergence. Nous en citons deux :

B. Performance en-ligne : La performance en cours de la recherche est mesurée par la performance moyenne de l'algorithme basée sur la fonction d'évaluation :

$$PerformanceEnLigne(T) = \frac{1}{T}\sum_{t=1}^{t=T} F(t) \tag{4.9}$$

t est le nombre total des itérations jusqu'à l'itération T, $F(t)$ est la valeur moyenne de la fonction d'évaluation à la $i^{ème}$ itération. Lors de l'exécution de l'AG, PerformanceEnLigne(T) converge de plus en plus vers une valeur stable (état stationnaire), et comme PerformamnceEnLigne(T) converge les solutions trouvées par l'algorithme deviennent de plus en plus stables.

C. Performance hors-ligne : La mesure de performance hors-ligne est similaire à celle en ligne sachant que la performance hors-ligne accorde plus d'importance aux meilleures performances obtenues :

$$PerformamnceHoreLigne(T) = \frac{1}{T}\sum_{t=1}^{t=T} F_{max}(t) \quad avec \quad F_{max}(t) = Sup(F(t)) \;\; et \;\; 1 \leq t \leq T \tag{4.10}$$

Dans l'équation 4.10, ce qui change par rapport à la performance en ligne est la fonction $F_{max}(t)$ qui enregistre la meilleure fitness jusqu'à présent et ignore toute autre évaluation.

Ces mesures de performance peuvent être utilisées comme critère d'arrêt. Puisque si PerformanceHorsLigne(T) converge vers une valeur stable durant l'évolution, la chance d'obtenir une solution encore meilleure diminue fortement et l'on peut déjà arrêter la simulation, car si "meilleur" signifie seulement une légère amélioration, continuer l'algorithme serait une perte de temps.

Enfin, on a montré que la convergence peut-être observée directement en mesurant comment les individus changent à chaque position de gène [98]. Un gène est dit avoir convergé lorsque 95% de la population partage la même valeur et la population est qualifiée d'avoir convergé si tous les gènes ont convergé.

4.2.2 Naissance des algorithmes évolutionnistes

Les AG ainsi introduits, ont retenu l'attention de nombreux chercheurs et praticiens de l'optimisation, en raison de leurs qualités particulières. Cependant, le codage binaire des solutions constitue un frein majeur à l'utilisation des AG. En effet, ce type de codage s'est révélé insuffisamment adapté à des problèmes d'optimisation avec des variables réelles, à des problèmes nécessitant la représentation de permutations ou d'autres problèmes complexes. Des extensions des AG utilisant d'autres types de codages et d'autres opérateurs génétiques, tout en conservant l'esprit des AG ont été proposées. Toutefois, il est difficile de trouver des frontières réelles entre les deux types d'algorithmes si ce n'est que l'on peut considérer les AG comme une classe particulière des algorithmes évolutionnistes (AE).

Parmi les principales extensions des AG nous en citons :

1. Les algorithmes génétiques parallèles.
2. Extension vers les problèmes d'optimisation.

4.2.3 Les algorithmes génétiques parallèles

Dans cette classe d'algorithmes, on cherche à améliorer parallèlement chaque individu par la méthode du Hill-Climbing, puis chaque individu s'accouple avec un autre individu de son voisinage. Il n'y a donc plus de sélection comme dans les algorithmes génétiques classiques. Les individus évoluent en parallèle, sans contrôle central. Un exemple d'AG parallèle est décrit par Mühlenbein H. [99].

4.2.4 Extension vers les problèmes d'optimisation

Dans un algorithme évolutionniste, quand on est devant un problème à n variables réelles, on pourra le représenter par un chromosome codé sous la forme d'un vecteur composé de n réels. En fait, la représentation binaire a quelques inconvénients lorsqu'elle est appliquée à des problèmes de plusieurs variables avec une grande précision numérique.

A titre d'exemple, pour un problème à 100 variables comprises dans un intervalle réel [-500,500] et pour une précision requise de six chiffres après la virgule, la taille du chromosome est 3000 [88]. Un tel chromosome engendre un espace de taille 10^{1000}. Janikow et Michalewicz ont confirmé par des expérimentations que le codage réel augmente la rapidité de l'algorithme et apporte la précision dans des résultats performants. Ainsi, ils ont montré que les résultats sont uniformes d'un essai à l'autre [100]. L'utilisation du codage réel donne la possibilité d'utiliser des domaines larges (aussi des domaines dont les bornes sont inconnues) [101] pour les variables, ce qui est difficile dans le codage binaire et ce, lorsque les domaines des différentes variables est inconnue et ceci en supposant que la taille du chromosome est fixe. Un autre avantage en utilisant le codage réel des paramètres est sa capacité d'exploiter progressivement la fonction objective avec des variables continues, où le concept de progressivité se réfère au fait que de légers changements dans les variables correspondent à de légers changements dans la fonction.

Des opérateurs de croisement et de mutation adaptés à ce genre de codage ont été proposés tels que : le croisement arithmétique [88] défini comme combinaison linéaire de deux vecteurs ou chromosomes, la mutation uniforme [99] et non uniforme [102]. Nous citons par la suite quelques opérateurs génétiques adaptés à ce type de codage.

4.2.4.1 Les opérateurs de croisement

On considère deux chromosomes $C_1 = (c_1^1, c_2^1, ..., c_n^1)$ et $C_2 = (c_1^2, c_2^2, ..., c_n^2)$, sélectionnés pour se croiser entre eux. Nous expliquons dans ce qui suit quelques opérateurs de croisement pour le codage réel des algorithmes génétiques (RCGAs Real Coded Genetic Algorithms).

A. Le croisement Plat (Flat Crossover) : On génère un chromosome $H = (h_1, ..., h_i, ..., h_n)$ tel que h_i est choisi aléatoirement dans l'intervalle $[c_1^i, c_2^i]$.

B. Le croisement simple (Simple Crossover) : On choisi aléatoirement une position $i = 1, 2, ..., n-1$ puis on construit deux chromosomes de la façon suivante :

$$H_1 = (c_1^1, c_2^1, ..., c_i^1, c_{i+1}^2, ..., c_n^2)$$
$$H_2 = (c_1^2, c_2^2, ..., c_i^2, c_{i+1}^1, ..., c_n^1)$$

C. Croisement arithmétique (Arithmitical Crossover) : Deux chromosomes, $H_k = (h_1^k, ..., h_i^k, ..., h_n^k)$ avec $k = 1, 2$ sont générés tel que $h_i^1 = \lambda c_i^1 + (1-\lambda)c_i^2$ et $h_i^2 = \lambda c_i^2 + (1-\lambda)c_i^1$ avec λ est une constante (croisement arithmétique uniforme) ou bien varie en fonction des itérations (croisement arithmétique non uniforme).

D. Croisement BLX-α (BLX-α Crossover) : Un chromosome $H = (h_1, ..., h_i, ..., h_n)$, est généré où h_i est choisi unifomement d'une façon aléatoire dans l'intervalle $[C_{min} - I\alpha, C_{max} + I\alpha]$ avec $C_{max} = max(c_i^1, c_i^2)$, $C_{min} = min(c_i^1, c_i^2)$ et $I = C_{max} - C_{min}$. Si $\alpha = 0$ le croisement $BLX - \alpha$ est identique au croisement plat.

4.2.4.2 Opérateur de mutation

On considère un chromosome $C = (c_1, ..., c_i, ..., c_n)$ avec $c_i \in [a_i, b_i]$ et c_i le gène sélectionné pour subir la mutation et c_i' le gène résultant après l'application d'un opérateur de mutation.

A. Mutation aléatoire ou uniforme (Random Mutation) : c_i' est choisi uniformément d'une façon aléatoire dans l'intervalle $[a_i, b_i]$.

B. Mutation non-uniforme (Non-uniform Mutation) : Si cet opérateur est appliqué dans une génération t et G_{max} le nombre maximum de générations alors :

$$c_i' \begin{cases} c_i + \triangle(t, b_i - c_i) & si \quad \tau = 0 \\ c_i - \triangle(t, c_i - a_i) & si \quad \tau = 1 \end{cases} \tag{4.11}$$

Avec τ prend aléatoirement la valeur 1 ou 0, et

$$\triangle(t, y) = y(1 - r^{(1 - \frac{t}{G_{max}})^b}) \tag{4.12}$$

Où r un nombre aléatoire qui appartient à l'intervalle $[0, 1]$ et b un paramètre qui peut être choisi par l'utilisateur, qui détermine le degré de la dépendance au nombre des itérations. Cette fonction donne une valeur dans l'intervalle $[0, y]$ avec une probabilité de s'approcher de 0 qui augmente avec l'avancement de l'algorithme. La taille de cet intervalle diminue au fil des générations. Cette propriété pousse cet opérateur à faire une recherche uniforme dans l'espace initial, quand t est faible et très localement dans les dernières itérations en favorisant la recherche locale. Pour en savoir plus sur d'autres opérateurs génétiques, consultez [101].

4.2.5 Recommandations

Pour finir, nous citons quelques recommandations concernant le choix des valeurs des paramètres. En effet, d'après Grefensette [95] les valeurs des paramètres doivent être choisies avec les considérations suivantes :

1. **La taille de la population :** elle doit être choisie en fonction de la taille du problème et du code.
   - Trop faible : l'AE n'a pas assez d'échantillons de l'espace de recherche.
   - Elevée : l'AE est plus uniforme. Une taille élevée prévient contre la convergence prématurée.
   - Trop élevée : le nombre élevé d'évaluations de la fonction objective par génération ralentit la convergence.
2. **Taux de croisement :** plus la probabilité de croisement P_{cross} est élevée, plus il y aura de nouvelles structures qui apparaissent dans la population.
   - Trop élevé : les bons individus risquent de ne pas être cassés trop vite par rapport à l'amélioration que peut apporter la sélection.
   - Trop faible : la recherche risque de stagner.
   - Le taux habituel est choisi entre 60% et 100%.
3. **Taux de mutation :** la mutation est l'opérateur secondaire pour introduire la diversité dans la population. Son taux d'application P_{mut} est choisi entre 0.1% et 5%.
   - Trop élevé : le taux de mutation rend la recherche trop aléatoire.
   - Trop faible : la recherche risque de stagner.

4.3 Le recuit simulé

Le recuit simulé (RS, Simulated Annealing) est une méthode d'optimisation métaheuristique. Elle a été mise au point par trois chercheurs de la société IBM, S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983 [103], et indépendamment par V. Černy en 1985 [104]. Cette méthode est issue d'une analogie entre le phénomène physique de refroidissement lent d'un corps en fusion, qui le conduit à un état solide, de basse énergie. A l'origine le RS est une adaptation des expériences réalisées par Metropolis et al.[105] dans les années 50 pour simuler l'évolution de ce processus de recuit physique.

4.3.1 Du recuit réel au recuit simulé

Pour modifier l'état d'un matériau, le physicien dispose d'un paramètre de commande : la température [106]. Le recuit constitue ainsi une stratégie de contrôle de la température en vue de trouver un état optimum. La croissance d'un monocristal est pris comme exemple pour expliquer cette stratégie. La technique de recuit consiste à chauffer préalablement le

matériau pour lui conférer une énergie élevée. Puis, le matériau est refroidi lentement, en marquant des paliers de température de durée suffisante ; si la descente en température est trop rapide, il apparaît des défauts qui peuvent être éliminés par réchauffement local. Cette stratégie de baisse contrôlée de la température conduit à un état solide cristallisé stable, correspondant à un minimum absolu de l'énergie. La technique opposée est celle de la trempe, qui consiste à abaisser très rapidement la température du matériau ; dans ce cas, une structure amorphe relative à un état métastable correspondant à un minimum local de l'énergie est obtenue. Ainsi, avec la technique du recuit, le refroidissement du matériau a provoqué une transformation désordre-ordre, tandis que la technique de la trempe a abouti à figer un état désordonné [106].

Cette technique a donné naissance à la méthode du recuit simulé. Le processus du recuit simulé répète une procédure itérative qui cherche des configurations de coût plus faible tout en acceptant de manière contrôlée des configurations qui dégradent la fonction de coût. Cette méthode consiste à introduire un paramètre de contrôle qui joue le rôle de la température. La température du système à optimiser doit avoir le même effet que la température du système physique, c'est-à-dire conduire vers l'état optimal si elle est abaissée de façon lente et bien contrôlée (technique de recuit) et vers un minimum local si elle est abaissée brutalement (technique de trempe).

4.3.2 Algorithme de recuit simulé

L'algorithme se base sur trois résultats de la physique statique.

1. Pour une température donnée T, l'équilibre thermodynamique d'un système est atteint. La probabilité, pour ce système, de posséder une énergie donnée E, est proportionnelle au facteur de Boltzmann :exp($\frac{-E}{K_B T}$), où K_B désignant la constante de Boltzmann.

2. Le deuxième résultat s'appuie sur l'algorithme de Metropolis pour simuler l'évolution d'un système physique vers son équilibre thermodynamique à une température T donnée. Le principe de cet algorithme est le suivant : partant d'une configuration donnée, le système subit une modification élémentaire ; si cette transformation a pour effet de diminuer la fonction objectif f (ou énergie) du système, elle est acceptée, si elle provoque au contraire une augmentation $\triangle f$ de la fonction objectif, elle peut être acceptée tout de même, avec la probabilité exp($\frac{-\triangle f}{T}$).

3. La notion de l'équilibre thermique dépend de la chaine de Markov qui est l'ensemble des configurations explorées à température constante. L'équilibre thermique est caractérisé par la convergence de la distribution des énergies calculées sur les différentes configurations de la chaîne de Markov vers une loi normale. En pratique, ce critère d'équilibre thermique, trop sévère, est remplacé par la notion de quasi-équilibre. Ce

dernier est atteint lorsque la longueur de la chaîne de Markov, à savoir le nombre de configurations explorées, est suffisamment grande.

Kirkpatrick a fait une analogie entre l'optimisation et le phénomène physique de refroidissement, en faisant une correspondance entre arrangements des atomes et paramètres de conception, énergie et fonction objectif à minimiser, minimum de l'énergie et minimum global, chaîne de Markov et nombre de configurations explorées à température constante. Toutefois, le concept de température d'un système physique n'a pas d'équivalent direct avec le problème à optimiser. Ainsi, le paramètre température T est un paramètre de contrôle, indiquant le contexte dans lequel se trouve le système à savoir le stade de la recherche. Le critère de Metropolis détermine si une nouvelle configuration générée présente une variation de la fonction objectif acceptable et permet aussi de sortir des minima locaux quand la température est élevée.

La figure 4.4 présente les étapes de l'algorithme du recuit simulé. Celui-ci débute par une température initiale élevée et une configuration x initiale prise au hasard. A chaque itération, x varie ; si cette variation diminue la fonction f, elle est acceptée sinon la règle de Metropolis est appliquée. Si l'équilibre thermodynamique est atteint, c'est-à-dire lorsque les variations élémentaires de x ne font plus varier l'énergie du système, à savoir la fonction f, la température est diminuée lentement et les modifications de x sont reprises jusqu'à atteindre un seuil préalablement choisi.

4.3.3 Paramètres du recuit simulé

La convergence du recuit simulé dépend du contrôle de la "température" du système pour atteindre, le plus vite possible, une solution. Le programme de recuit doit préciser les valeurs des paramètres, de contrôle de la température, suivants :

- La température initiale.
- La longueur de chaîne de Markov, ou le critère de changement de palier de température.
- La loi de décroissance de la température.
- Le critère d'arrêt du programme ou seuil.

La température T a un rôle très important au cours du processus de recuit simulé. En effet, une forte décroissance de température risque de piéger l'algorithme dans un minimum local, alors qu'une faible décroissance au début du processus entraîne une convergence très lente de l'algorithme. La loi, selon laquelle la température décroît, est importante pour l'efficacité de l'algorithme puisqu'elle doit laisser le temps au système de tester le maximum de configurations pour être sûr d'obtenir le minimum global. Aussi, la température initiale doit aussi être suffisamment élevée pour que la descente en température soit aussi lente que possible.

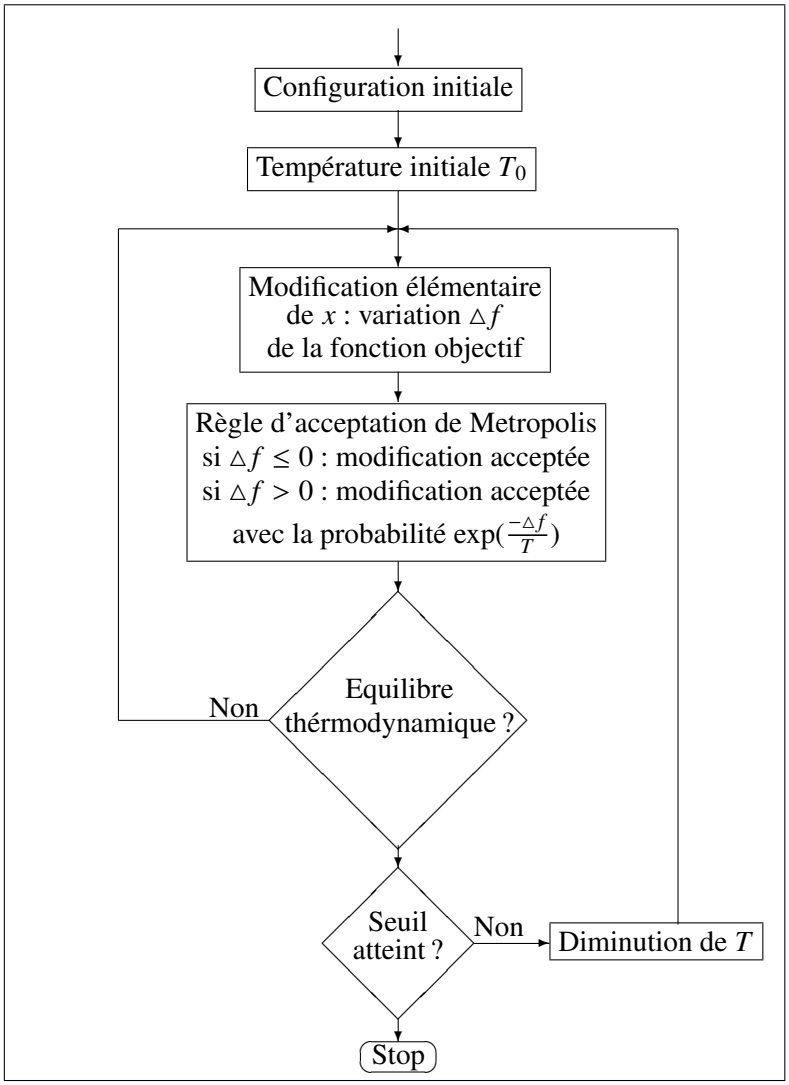


FIGURE 4.4 – Algorithme du recuit simulé

Le changement de température de T_k vers T_{k+1} est effectué au moment où l'équilibre thermique (ou l'état de quasi-équilibre) est détecté. La recherche de cet équilibre s'effectue en générant une succession de chaînes de Markov. La variation de température se fait donc par paliers suivant la fonction de décroissance utilisée. Le tableau 4.1 présente les fonctions les plus couramment rencontrées dans la littérature : les fonctions linéaires, discrètes ou exponentielles, pour en savoir plus sur d'autres fonctions consulter [107].

Types	Fonctions	Paramètres
Linéaires	$T_{K+1} = \alpha T_K$	$0 \prec \alpha \prec 1$
Discrètes	$T_{K+1} = T_K - \triangle T$	$\triangle T \succ 0$
Exponentielles	$T_{K+1} = T_K \exp(\frac{-\lambda T_K}{\sigma_K})$ avec $0 \leq \frac{T_{K+1}}{T_K} \leq 1$	σ_K est l'écart type des fonctions objectifs des configurations acceptées à la température T_K ; λ est un paramètre de réglage fixé par l'utilisateur, tel quel $\lambda > 0$.

TABLE 4.1 – Lois de décroissance de la température.

4.4 Conclusion du chapitre

Dans ce chapitre nous avons présenté des méthodes d'optimisation métaheuristiques qui sont les AG et le RS. Ces méthodes savèrent interessantes dans la recherche d'optimum global d'une fonction objectif. L'avantage de telle méthode est de n'imposer aucune contrainte sur la fonction objectif. Les méthodes exposées ainsi seront adoptées pour la construction des modèles hybrides de prédiction de la fiabilité des logiciels qui seront l'objet du chapitre suivant.

Deuxième partie

Contributions

CHAPITRE 5

Méthodes hybrides pour la prédiction des défaillances cumulées d'un logiciel

Sommaire

5.1 Introduction

Les logiciels sont les noyaux de plusieurs applications critiques et non critiques, et pratiquement toutes les industries dépendent des ordinateurs pour leurs fonctions de base. Le logiciel a envahis notre société moderne, et continuera le progrès dans le futur [108]. Le nombre des utilisateurs qui s'intéressent à la fiabilité a augmenté au fil du temps, spécialement, avec les avantages des systèmes de contrôle en temps réel tels que le contrôle des satellites et des navettes et dans les systèmes de téléphonie et des banques etc. Il est important que ces systèmes soient testés avant d'être utilisés. Quoiqu'on ne puisse pas empêcher les défaillances de se produire, il est souhaitable de prévoir la fiabilité. Les modèles de la fiabilité des logiciels permettent l'estimation ou la prévision de la fiabilité actuelle ou futur d'un système, ainsi que la planification d'autres activités telles que les tests et la maintenance. Au cours des 40 dernières années, plus de 100 modèles de fiabilité des logiciels ont été développés [109]. La plupart de ces modèles dépendent à priori d'un ensemble d'hypothèses sur la nature des défaillances et le comportement du logiciel [13, 110, 111]. En conséquence différents modèles ont des performances de prédiction différents à travers diverses phases de tests sur des projets différents. Un modèle universel unique qui peut fournir des prédictions très précises en toutes circonstances, sans aucune hypothèse est le plus souhaitable [56, 112].

Les réseaux de neurones sont des approximateurs universels pour n'importe quelle fonction continue non linéaire [111, 113, 114, 115]. Par conséquent ils sont devenus une alternative pour la modélisation et la prédiction de la fiabilité des logiciels. Plusieurs auteurs ont proposés des modèles qui se basent sur les réseaux de neurones (voir section 2.2.8.1 du chapitre 2) et ils ont obtenus des résultats très performants par rapport aux modèles paramétriques.

L'algorithme de rétropropagation des erreurs est la méthode la plus utilisée pour entraîner les réseaux de neurones, mais cet algorithme soufre, en plus des problèmes cités dans le section 3.7 du chapitre 3 du temps de calcul qui peut s'étaler sur plusieurs journées en fonction du réseau et du problème.

Les méthodes de régression sont les méthodes les plus utilisés pour calibrer la quasi-totalité des modèles classiques. Le modèle d'auto-regression est un modèle alternatif proposé pour prédire le nombre de défaillances du logiciel comme il est démontré par Aljahdali et al.[116]. Les estimateurs des moindres carrées est la méthode utilisée pour estimer les paramètres des modèles linéaires comme le modèle d'auto-regression.

Dans ce chapitre nous proposons une alternative pour estimer les paramètres des deux modèles des réseaux de neurones et d'auto-regression. Cette alternative utilise un algorithme évolutionniste et le recuit simulé qui optimisent l'erreur produite par les deux modèles [12, 13, 14]. L'optimisation de l'ensemble des paramètres est effectuée en utilisant l'inverse de

l'erreur comme fonction objectif.

5.2 Le réseau de neurones

L'architecture du réseau utilisé pour la modélisation de la prédiction des défaillances du logiciel, est un réseau multicouches feedforward. Il comporte une couche d'entrée, une couche cachée et une couche de sortie. La couche d'entrée contient quatre neurones, égaux au nombre de mesures précédentes. Elles sont $y(i-1)$, $y(i-2)$, $y(i-3)$ et $y(i-4)$ avec $y(i-k)/k = \{1,2,3,4\}$ est le nombre des défaillances cumulées observées par jour avant le jour i, pour lequel nous voulons savoir le nombre de défaillances cumulées. La couche cachée comporte deux neurones avec comme fonction d'activation la sigmoïde et deux neurones linéaires. La couche de sortie comporte un seul neurone avec comme fonction d'activation une linéaire, cette couche a comme sortie $\hat{y}(i)$ qui est le nombre de défaillances dans le jour i. Les neurones de la couche cachée sont totalement connectés aux neurones de la couche d'entrée et celui de la couche de sortie, comme illustré dans la figure 5.1.

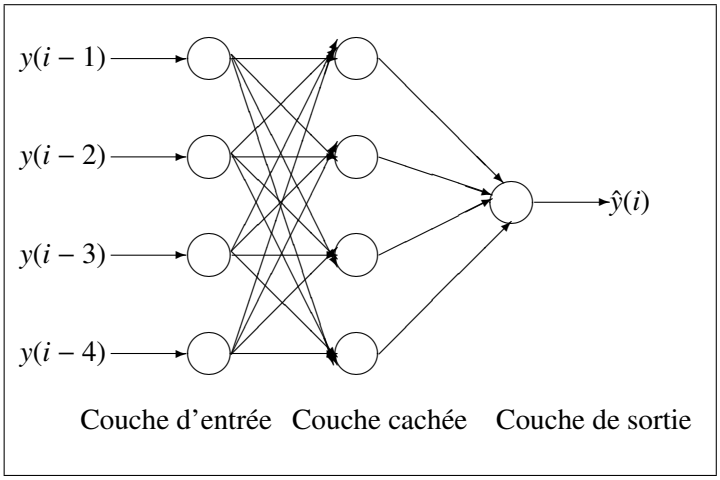


FIGURE 5.1 – Structure du réseau de neurones

5.3 Le modèle d'auto-regression

Nous avons repris le modèle d'auto-regression d'ordre 4 développé par S.Aljahdali [116] pour lequel nous avons amélioré la phase d'apprentissage à l'aide d'un algorithme évolutionniste. La structure du modèle est donnée par l'équation 5.1. Afin de déterminer l'ordre

optimal du modèle nous avons utilisé deux autres modèles d'ordre 7 et 10 [13, 14].

$$\hat{Y}(i) = a_0 + \sum_{k=1}^{n} a_k Y(i-k) \tag{5.1}$$

avec $n = \{4, 7, 10\}$ l'ordre du modèle, $Y(i-k)$ le nombre de défaillances cumulées pendant $(i-k)$ jours avant le jour i, $\hat{Y}(i)$ le nombre de défaillances cumulées qui sera prédite par le modèle et les a_k sont les paramètres du modèle.

5.4 Apprentissage des modèles

5.4.1 Critères d'évaluation

Nous avons utilisé comme critère d'évaluation des différents modèles, la moyenne quadratique des erreurs (mean square error MSE) définie dans l'équation 5.2 et la moyenne normale de la racine carrée de l'erreur (Normalized Root Mean of the Square Error NRMSE) définie dans l'équation 5.3.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y(i) - \hat{y}(i))^2 \tag{5.2}$$

$$NRMSE = \frac{1}{N} \sqrt{\frac{\sum_{i=1}^{N} (y(i) - \hat{y}(i))^2}{\sum_{i=1}^{N} (y(i))^2}} \tag{5.3}$$

5.4.2 Algorithme évolutionniste pour l'apprentissage du réseau de neurones

5.4.2.1 Le codage des individus

Le premier pas dans l'implémentation d'un AE pour entraîner un RNA est le codage des individus qui constitueront la population initiale. Nous adoptons la représentation réelle des gènes puisque notre problème est un problème d'optimisation à paramètres réels. Reprenons l'architecture de la figure 5.1, soit $w_{i,j}$ le poids d'une connexion du neurone i vers le neurone j et b_j son biais. Un chromosome est constitué de tous les poids et biais du réseau concaténés les uns après les autres, les valeurs des poids appartiennent à $\mathbb{R}$. La figure 5.2 donne une représentation du chromosome associé à l'architecture adoptée. Tous les poids sont initialisés avec des valeurs aléatoires dans l'intervalle $[0, 1]$ et les biais sont initialisés à 1.

$$\underbrace{w_{1,1}w_{1,2}w_{1,3}w_{1,4}b_1 w_{2,1}w_{2,2}w_{2,3}w_{2,4}b_2 w_{3,1}w_{3,2}w_{3,3}w_{3,4}b_3 w_{4,1}w_{4,2}w_{4,3}w_{4,4}b_4}_{\textit{Couche cachée}} \underbrace{w_{5,1}w_{5,2}w_{5,3}w_{5,4}b_5}_{\textit{Couche de sortie}}$$

FIGURE 5.2 – La représentation chromosomique du réseau de neurones

5.4.2.2 La fonction objectif

La fonction objectif (fitness) utilisée pour calculer la force des individus est :

$$fitness = \frac{1}{1 + MSE} \tag{5.4}$$

Ici le terme MSE est la moyenne de la somme quadratiques des erreurs (cf. équation 5.2). Le terme MSE peut être remplacé par le terme NRMSE (équation 5.3), qui est largement utilisé pour mesurer l'exactitude d'une méthode de prévision.

Pour calculer la force d'un individu X_i, on affecte les poids de l'individu au réseau de neurones, puis on utilise les défaillances d'un projet comme des entrées à partir desquelles le réseau calcule ses sorties. La force de l'individu concerné sera calculée suivant l'équation 5.4. La probabilité d'apparition de cet individu est calculée selon l'équation 5.5 suivante :

$$p(X_i) = \frac{fitness(X_i)}{\sum_{k=1}^{N} fitness(X_k)} \tag{5.5}$$

avec N le nombre totale des individus.

5.4.2.3 La sélection des individus

La sélection consiste à choisir les individus candidats à former la population suivante. Le mécanisme de sélection choisi est la sélection par roulette (section 4.2.1.6 du chapitre 4). Pour se faire, on calcule la probabilité d'apparition cumulée de chaque individu X_k selon l'équation 5.6 et les individus sont sélectionnés selon le principe de l'équation 5.7.

$$soit \quad q_i = \sum_{k=1}^{N} p(X_k) \tag{5.6}$$

avec N le nombre totale des individus.

$$\begin{cases} X_1 \quad si \quad q_i < r \\ X_i \quad si \quad q_{i-1} < r \leq q_i \quad pour\ 2 \leq i \leq N \end{cases} \tag{5.7}$$

avec r un nombre choisi aléatoirement dans l'intervalle [0,1].

5.4.2.4 Les paramètres des opérateurs de croisement et de mutation

Un jeu de paramètres a été effectué, dans le tableau 5.1 nous reprenons les valeurs des paramètres que nous avons jugé après plusieurs tests comme une base de décision.

				MSE		
Npop	Gmax	Croisment	Mutation	Projet1	Projet40	ProjetSS1C
20	50	BLX-α = 0.1 P_c = 0.2	Non-uniform b=2 P_m = 0.02	6.5257	3.9718	14.5309
40	100	BLX-α = 0.2 P_c = 0.3	Non-uniform b=3 P_m = 0.03	5.1546	3.0704	4.6134
60	200	BLX-α = 0.3 P_c = 0.4	Non-uniform b=4 P_m = 0.04	4.1030	3.6056	4.4123
60	500	BLX-α = 0.4 P_c = 0.4	Non-uniform b=5 P_m = 0.05	3.0618	2.8732	3.5309
100	700	BLX-α = 0.5 P_c = 0.5	Non-uniform b=5 P_m = 0.06	4.1134	3.6056	5.5876
150	1000	BLX-α = 0.5 P_c = 0.6	Non-uniform b=5 P_m = 0.06	3.1649	2.0422	4.3402
200	1000	BLX-α = 0.5 P_c = 0.6	Non-uniform b=5 P_m = 0.06	3.3814	2.7746	3.3402
200	2000	BLX-α = 0.5 P_c = 0.6	Non-uniform b=5 P_m = 0.06	2.8865	1.8591	3.9742
200	2000	BLX-α = 0.5 P_c = 0.7	Non-uniform b=5 P_m = 0.06	2.9278	2.0563	2.2525
200	3000	BLX-α = 0.5 P_c = 0.7	Non-uniform b=5 P_m = 0.06	2.6597	2.2676	2.3969
200	10000	BLX-α = 0.5 P_c = 0.7	Non-uniform b=5 P_m = 0.06	3.1340	2.1971	3.2319

TABLE 5.1 – Résulats de MSE obtenus pour différentes valeurs des opérateurs génétiques

D'après les résultats obtenus sur les projets considérés nous remarquons que la MSE s'améliore considérablement pour les déférents projets pour une population de 200 individus, BLX-α = 0.5, P_c = 0.7 et la mutation non uniforme ($b = 5, P_m = 0.06$) à partir d'un maximum de génération égale à 1000.

Les individus sélectionnés sont groupés aléatoirement par paires pour subir le croisement BLX-α (section 4.2.4 D du chapitre 4) avec le paramètre $\alpha = 0.5$ et la probabilité de croisement $P_c = 0.7$. Les individus issus du croisement vont subir la mutation non-uniforme (section 4.2.4 B du chapitre 4) avec une probabilité de mutation $P_m = 0.06$ et le paramètre $b = 5$.

5.4.2.5 Le corps de l'algorithme évolutionniste

Le corps de notre algorithme évolutionniste est illustré dans la figure 5.3 :

Pour chaque projet faire :
Etape 1 : Créer une population initiale P_0 de N individus$\{X_1, X_2, ..., X_N\}$
$i := 0$;
Etape 2 : Evaluation des individus : entraîner le RN et calcul de la force des individus ;
Etape 3 : Sélectionner les individus selon le mécanisme de la roulette ;
Etape 4 : Application des opérateurs génétiques ;
1-Croisement : Appliquer le croisement BLX-α (Crossover) avec la probabilité P_c ;
2-Mutation : Appliquer la mutation Non-uniforme avec la probabilité P_m ;
Ranger les nouveaux individus obtenus (de 1 et 2) dans une nouvelle génération P_{i+1} ;
Répéter les étapes 2, 3 et 4 jusqu'à l'obtention du niveau de performance souhaité ;

FIGURE 5.3 – Corps de l'algorithme évolutionniste

5.4.3 Algorithme évolutionniste pour l'apprentissage du modèle d'auto-regression

5.4.3.1 Le codage des individus

Un chromosome est constitué de l'ensemble des paramètres a_i du modèle. Dans notre cas nous avons 5, 8 et 11 paramètres à ajuster pour les modèles d'ordre 4, 7 et 10 respectivement. La longueur des trois chromosomes est $l = 5$, $l = 8$ et $l = 11$ respectivement. Les paramètres prennent des valeurs dans $\mathbb{R}$, ils sont initialisés aléatoirement par des valeurs dans l'intervalle [0,1]. Les paramètres sont placés dans le chromosome comme illustré dans la figure 5.4.

$$\underbrace{a_0a_1a_2a_3a_4}_{Modèle\ d'odre4}, \underbrace{a_0a_1a_2a_3a_4a_5a_6a_7}_{Modèle\ d'odre7}, \underbrace{a_0a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}}_{Modèle\ d'odre10}$$

FIGURE 5.4 – La représentation chromosomique des modèles d'auto-regression

5.4.3.2 Paramètrage de l'algorithme

En effectuant les mêmes tests que précédemment, nous avons décidé de conserver les mêmes paramètres vus dans le cas du modèle de réseaux de neurones, à savoir la fonction objectif les opérateurs de séléction de croisement et de mutation ainsi que les valeurs des paramètres associés.

5.4.4 Le recuit simulé adapté pour l'apprentissage du réseau de neurones

Une solution W_{opt} est une matrice qui contient tous les poids du réseau de neurones. Chaque élément de W_{opt} prend une valeur dans $\mathbb{R}$. Dans notre cas nous avons 4x4 poids, et 4x1 biais pour la couche cachée et 4x1 poids et 1x1 biais pour la couche de sortie. La taille l de W_{opt} est donc l =4x4 + 4x1+1x1 = 25. Les poids et biais du réseau de neurones sont

placés dans la matrice W_{opt} comme suit.

$$W_{opt} = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & b_1 \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & b_2 \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & b_3 \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & b_4 \\ w_{5,1} & w_{5,2} & w_{5,3} & w_{5,4} & b_5 \end{pmatrix}$$

Nous avons choisi $F = \frac{1}{1+MSE}$ comme fonction objectif à maximiser (minimiser l'erreur MSE entre la valeur prédite et la valeur observée du nombre de défaillances cumulé dans le logiciel), avec MSE est décrite par l'équation 5.2

L'algorithme de recuit simulé que nous avons adopté est illustré dans la figure 5.5.

Initialiser aléatoirement les valeurs de la matrice W_{opt} des poids du réseau de neurones.
$F_{opt} = F(W_{opt})$
$T = T_{max}$
$T_{min}, iter = 0, maxIter$
$tantQue(T > T_{min})\{$
  $tantQue(iter < maxIter)\{$
    $W_{neighbor} = neighborOf(W_{opt})$
    $si(\triangle f = F(W_{neighbor}) - F(W_{opt}) > 0)\ alors\{$
      $W_{opt} = W_{neighbor}$
      $actualiser\ F_{opt}$
    $\}sinon$
    $si(random < \exp(\frac{-\triangle f}{T}))alors$
      $W_{opt} = W_{neighbor}$
    $iter++$
    $\}T = T * (1 - \varepsilon)$
}

FIGURE 5.5 – Algorithme du recuit simulé adapté pour l'apprentissage du réseau de neurones

Pour régler les différents paramètres de l'algorithme, un jeu de tests a été effectué sur plusieurs projets [58] entre autres les trois projets cités auparavant. Les meilleurs résultats ont été obtenus pour les valeurs suivantes des paramètres :

- Le nombre maximale des itérations $iterMax = 1000$.
- $T_{max} = 100$, $T_{min} = 5$, $\beta = 0.25$ et $\varepsilon = 0.02$.

Pour la génération d'une solution $W_{neighbor}$ voisine nous avons développé et implémenté deux méthodes inspirées de l'opérateur de mutation d'un algorithme évolutionniste utilisé dans [12, 13, 14]. La première méthode appelée "onePoint" consiste à sélectionner aléatoirement une valeur de W_{opt} et la modifiée aléatoirement avec un paramètre β suivant l'équa-

tion 5.8. La deuxième méthode appelée "multiPoint" consiste à choisir aléatoirement un ensemble d'éléments de W_{opt} et les modifier d'une façon aléatoire en utilisant l'équation 5.8.

$$W_{i,j} = W_{i,j} + randomValueIn(\frac{-\beta}{2}, \frac{\beta}{2}) \tag{5.8}$$

Pour perfectionner l'espace de recherche nous pouvons utiliser une autre méthode de génération de solutions voisines telle qu'elle est décrite dans l'équation 5.9.

$$W_{i,j} = W_{i,j} + randomValueIn(\frac{-2 * \beta}{3}, \frac{4 * \beta}{3}) \tag{5.9}$$

avec $randomValueIn(a, b)$ est une fonction qui renvoie une valeur aléatoire dans l'intervalle [a,b].

Les résultats expérimentaux ont montrés que la stratégie de génération "multiPoint" est très efficace et converge rapidement que la stratégie "onePoint" comme le montre la figure 5.6.

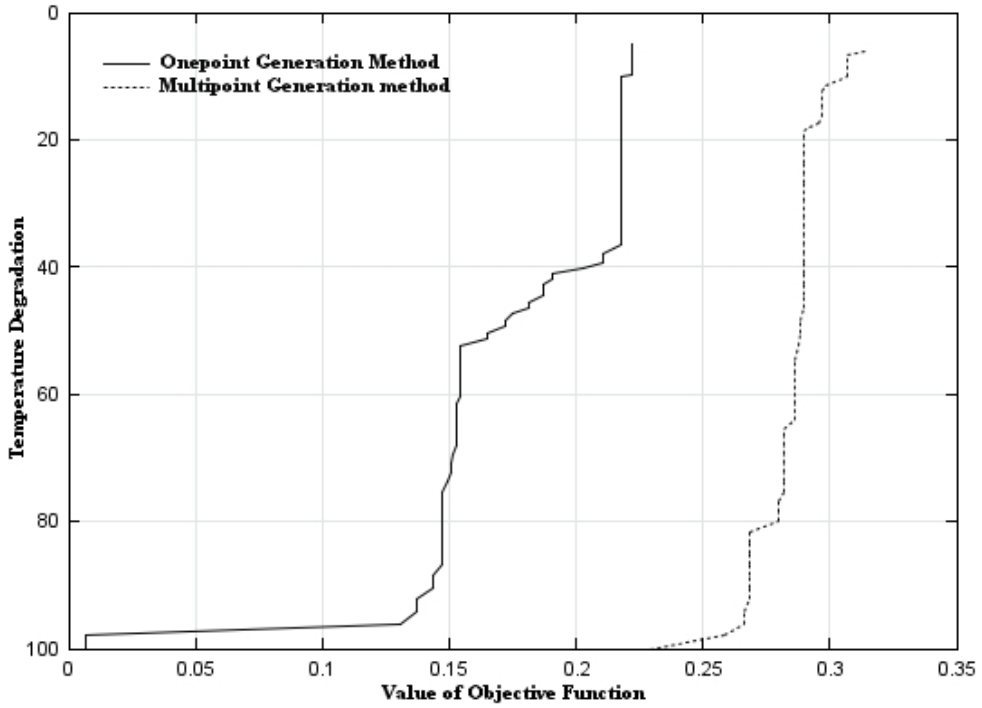


FIGURE 5.6 – La méthode "onePoint" versus "multiPoint" de génération des voisins.

5.5 Résultats et interprétation

5.5.1 Les données de fiabilité

Afin de valider et de tester la performance des modèles que nous avons proposé, nous avons sélectionner trois projets ; le projet 1 (Real Time & Control), le projet 40 (Military) et le projet SS1C (Operating System Application). Les données de défaillances des trois projets sont présentées dans les tableaux .1, .2 et .3 de l'annexe 7.7. Les défaillances sont ensuite regroupées par jour d'occurrence. Enfin ils sont présentés aux modèles par groupes de 4 pour les modèles de réseau de neurones et celui de régression d'ordre 4 et par groupes de 7 et 10 pour les modèles de régression d'ordre 7 et 10. Les données de défaillances des différents projets ont été divisées en deux parties, une partie composée de 70% des défaillances pour la phase d'apprentissage et 100% des données est utilisée pour tester et valider les modèles proposés.

5.5.2 Implémentation

Pour pouvoir expérimenter nos approches, nous avons opté pour le choix du langage Java pour implémenter les différents modèles (réseau de neurones, auto-regression, l'algorithme évolutionniste et le recuit simulé). Le choix du langage java est justifié par la prise en considération de l'ouverture de l'application, son évolution, sa modularité et ses fonctionnalités.

5.5.3 Résultats et interprétation

5.5.3.1 Le RNA entraîné par l'algorithme de rétro-propagation et le modèle d'AR entraîné par l'estimateur des moindres carrés

Le tableau 5.2 présente les résultats de $NRMSE$ obtenus pour les trois projets par le RNA entraîner par l'algorithme de rétropropagation et le modèle d'auto-regression optimisé par l'estimateur des moindres carrées durant la phase de test [1].

Project Name	Military#40	Real Time & Control #1	Operating System Application #SS1C
Number of Faults	101	136	277
Training Data	71	96	194
Testing Data	101	136	277
Regression Model $NRMSE$	3.1434	1.7086	1.0659
Neural Networks $NRMSE$	1.0755	0.5644	0.7714

TABLE 5.2 – Résulats $NRMSE$ obtenus par le RNA et le modèle d'auto-regression 4, entraînés par l'algorithme de rétro-propagation et les esimateurs des moindres carrées durant la phase de test [1].

Le modèle d'auto-regression entraîné par l'estimateur des moindres carrées donne de pauvres résultats par rapport au modèle de réseau de neurones entraîné par l'algorithme de rétropropagation, qui diminue le taux d'erreur observé pour les différents projets.

5.5.3.2 Le RNA entraîné par l'AE

Le tableau 5.3, présente les résultats des MSE et $NRMSE$ obtenus par le RNA entraîné par notre algorithme évolutionniste, dans les deux phases d'apprentissage et de test. Les figures de 5.7 à 5.15 présentent les courbes des résultats obtenus pour chaque projet durant la phase d'apprentissage, de test et les différences de l'erreur entre les défaillances observées et prédites.

Project Name	Military#40	Real Time & Control #1	Operating System Application #SS1C
Number of Faults	101	136	277
Training Data	71	96	194
MSE	2.859155	2.0515463	2.0515463
$NRMSE$	2.0635e-4	5.3898e-4	3.4186e-5
Testing Data	101	136	277
(MSE)	4.2277226	2.6617646	3.0758123
($NRMSE$)	4.7373e-5	3.1216e-4	1.5705e-5

TABLE 5.3 – Résulats MSE et $NRMSE$ obtenus par le RNA entraîné par l'AE

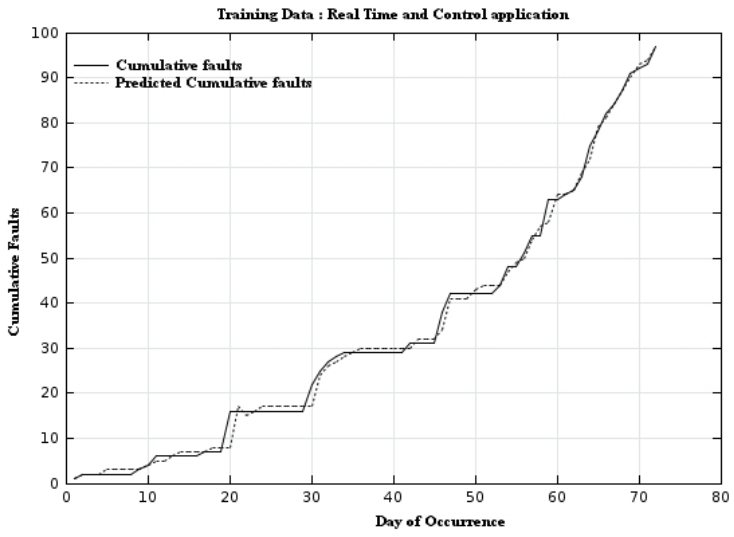


FIGURE 5.7 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#1

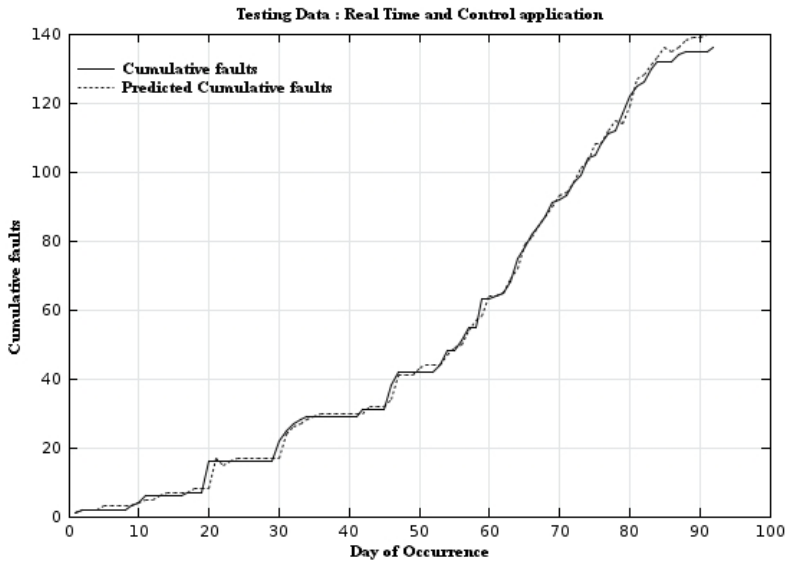


FIGURE 5.8 – Défaillances observées et prédites pendant la phase de test du projet#1

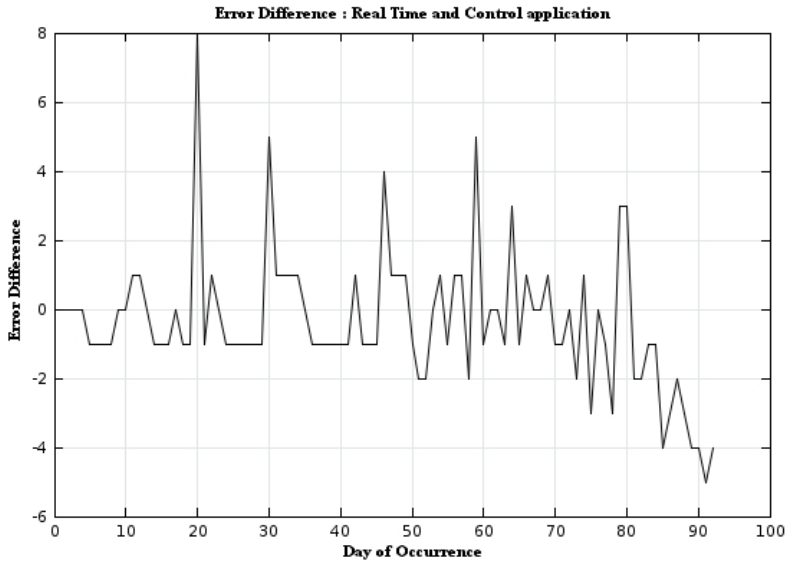


FIGURE 5.9 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#1

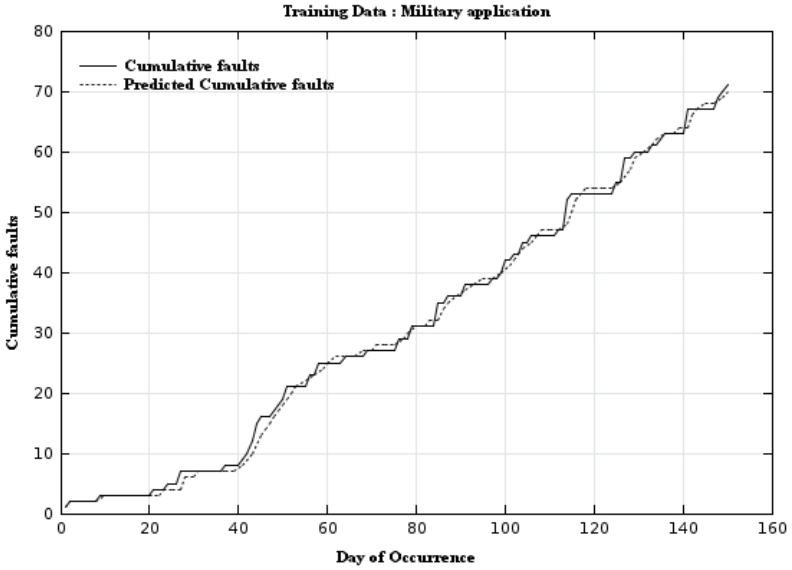


FIGURE 5.10 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#40

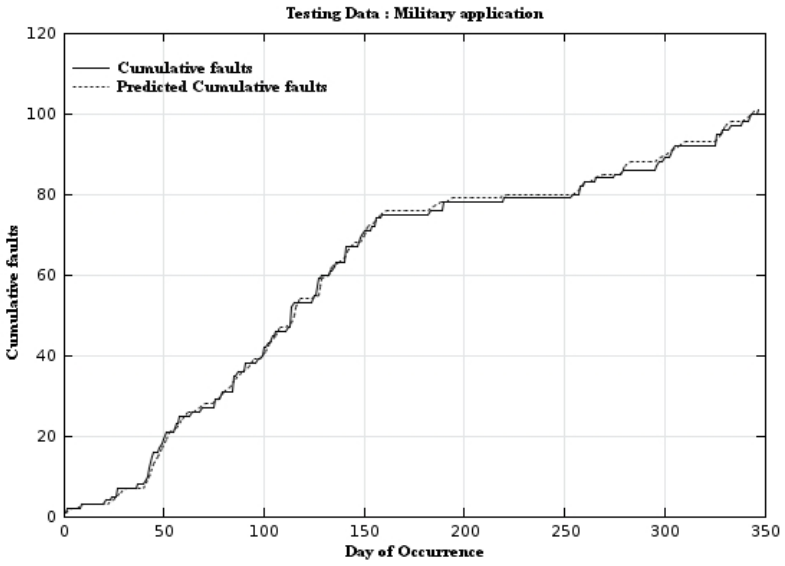


FIGURE 5.11 – Défaillances observées et prédites pendant la phase de test du projet#40

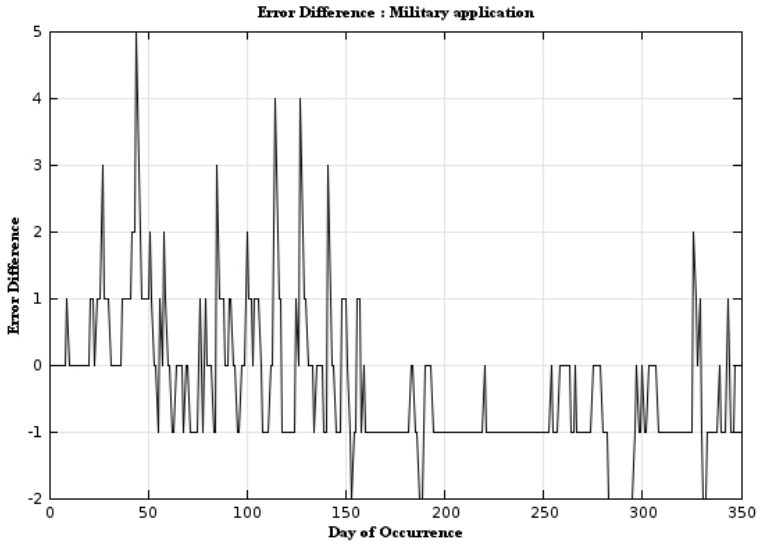


FIGURE 5.12 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#40

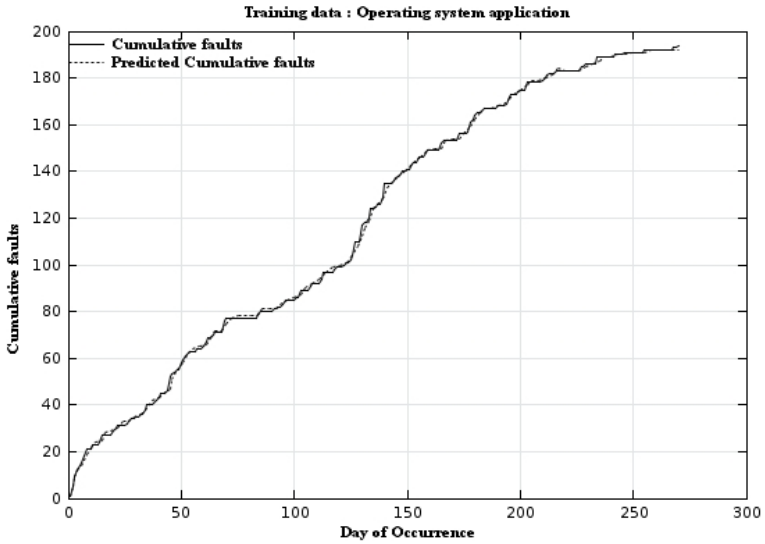


FIGURE 5.13 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#SS1C

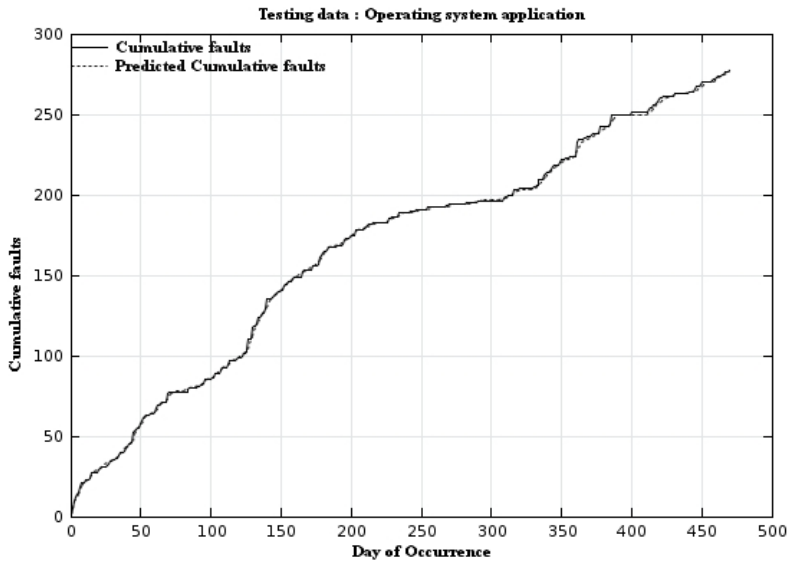


FIGURE 5.14 – Défaillances observées et prédites pendant la phase de test du projet#SS1C

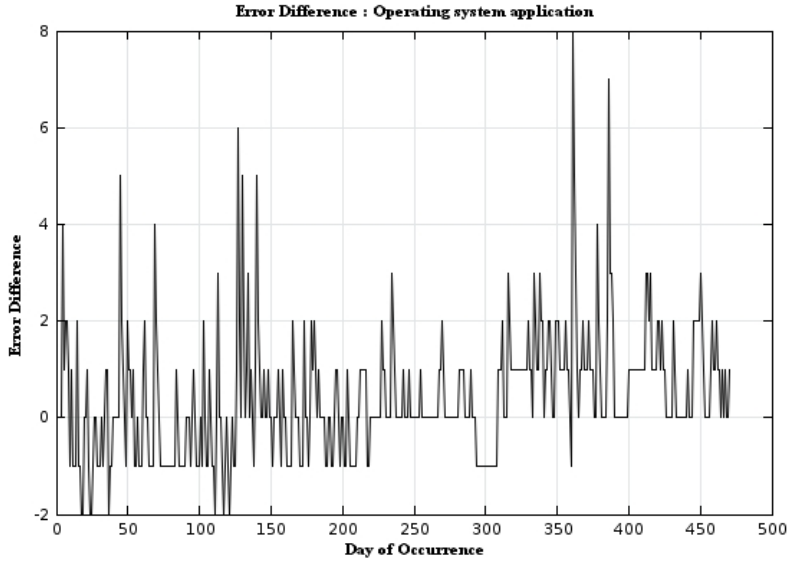


FIGURE 5.15 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#SS1C

D'après les résultats du tableau 5.3 et des figures de 5.7 à 5.15 nous remarquons que le taux d'erreur s'est diminué considérablement pour les trois projets, en utilisant le RNA entraîné par l'AE. Ainsi la $NRMSE$ a passée de 1.0755 à 4.7373x10^{-5} pour le projet#40 (Military), de 0.5644 à 3.1216x10^{-4} pour le projet#1 (Real Time & Control) et de 0.7714 à 1.5705x10^{-5} pour le projet#SS1C (Operating System). En outre l'allure des défaillances prédites suit celle des défaillances observées pour les différents projets. De même la prédiction à l'aide du RNA entraîné par l'AE ne dépasse pas 5 comme différence entre les valeurs observées et prédites pour le projet#40, 8 pour le projet#1 et le projet#SS1C.

5.5.3.3 Le RNA entraîné par le RS

Le tableau 5.4 présente les résultats des MSE et $NRMSE$ obtenus par le RNA entraîné par le recuit simulé adapté. Les figures de 5.16 à 5.24 présentent graphiquement ces résultats obtenus dans les phases d'apprentissage et de test ainsi que la différence de l'erreur observée pour chaque projet.

Project Name	Military#40	Real Time & Control #1	Operating System Application #SS1C
Number of Faults	101	136	277
Training Data	71	96	194
MSE	1.7323943	2.4329896	1.9329897
$NRMSE$	1.60628E-4	5.869615E-4	3.318441E-5
Testing Data	101	136	277
MSE	2.8415842	2.4044118	2.4187725
$NRMSE$	3.88384E-5	2.966878E-4	1.392739E-5

TABLE 5.4 – Résulats MSE et $NRMSE$ obtenus par le RNA entraîné par le RS

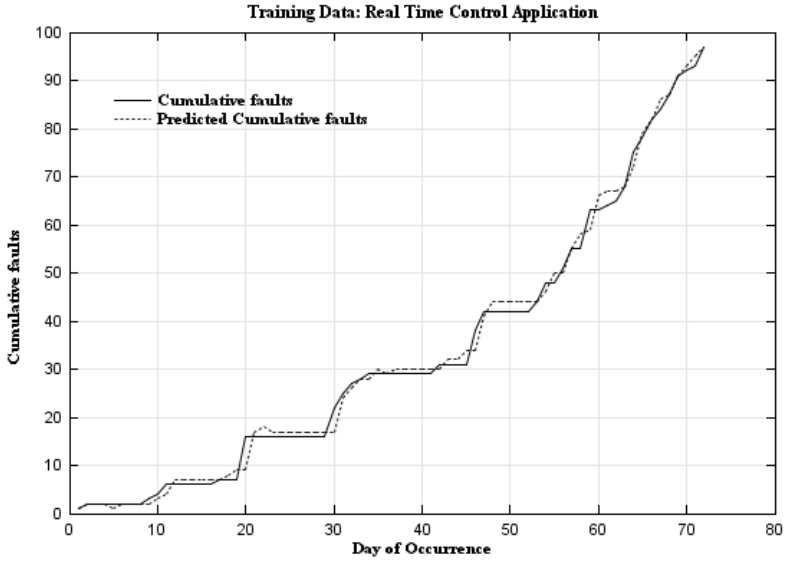


FIGURE 5.16 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#1

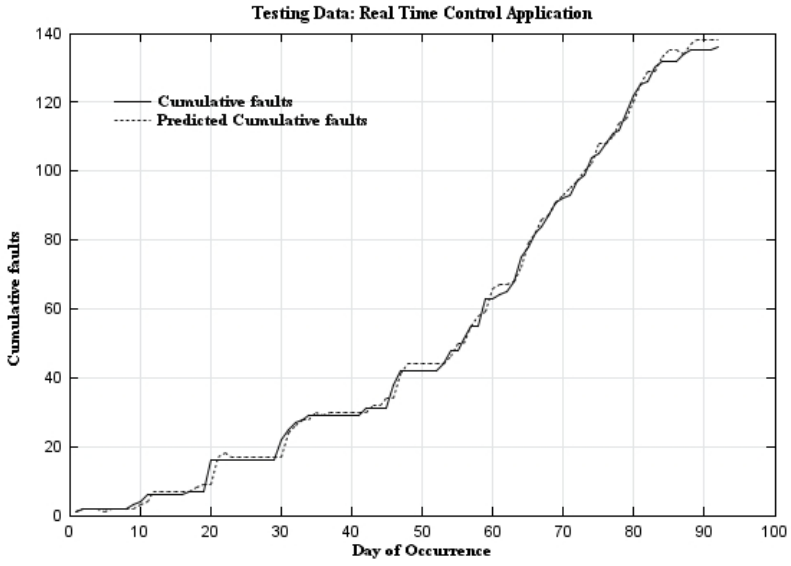


FIGURE 5.17 – Défaillances observées et prédites pendant la phase de test du projet#1

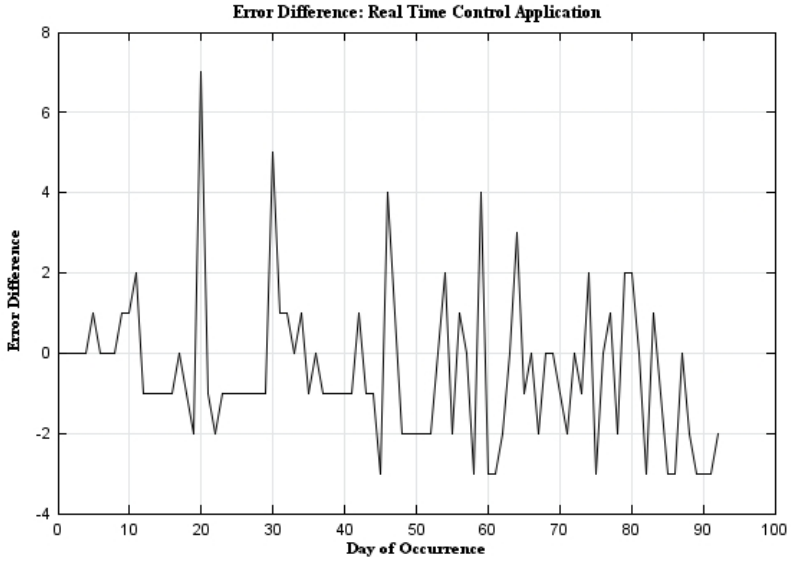


FIGURE 5.18 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#1

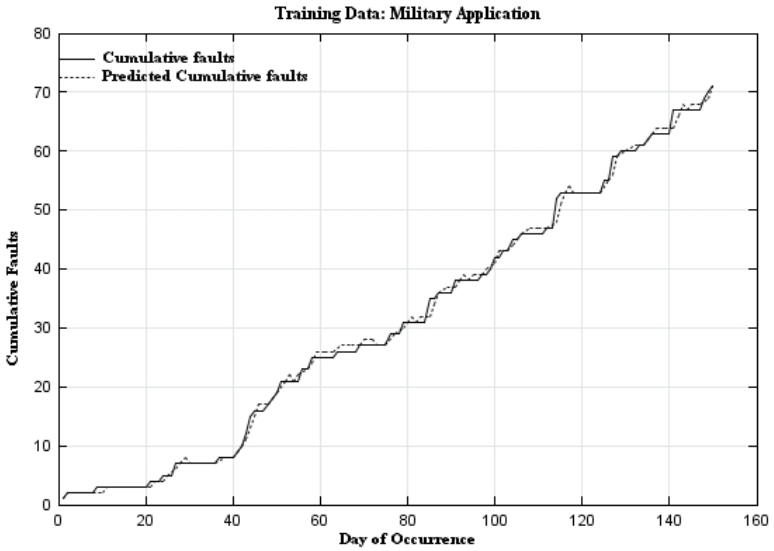


FIGURE 5.19 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#40

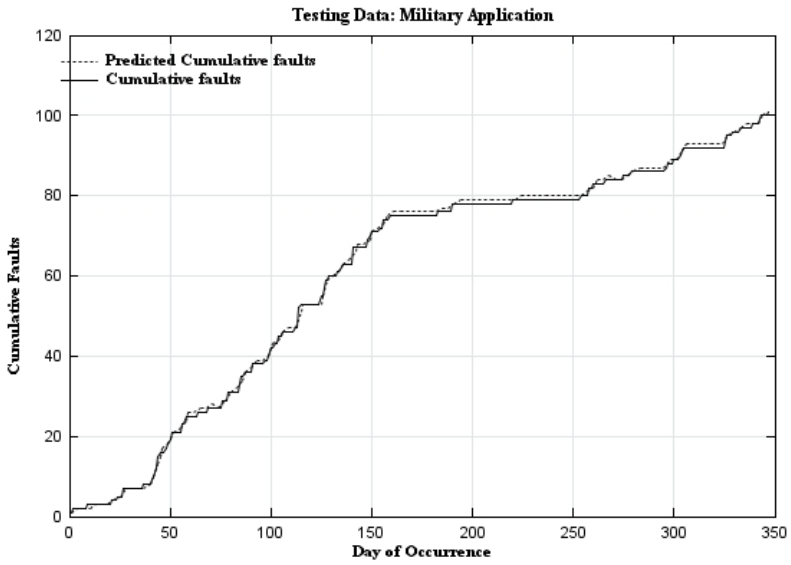


FIGURE 5.20 – Défaillances observées et prédites pendant la phase de test du projet#40

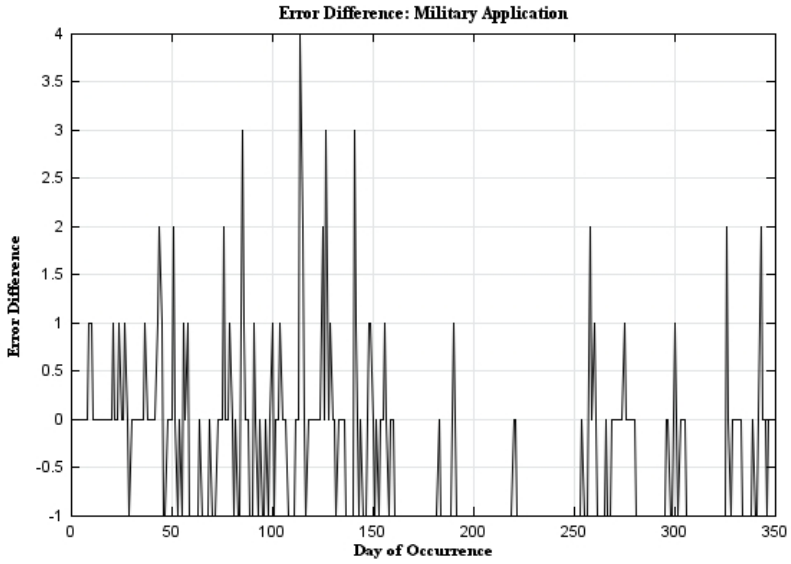


FIGURE 5.21 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#40

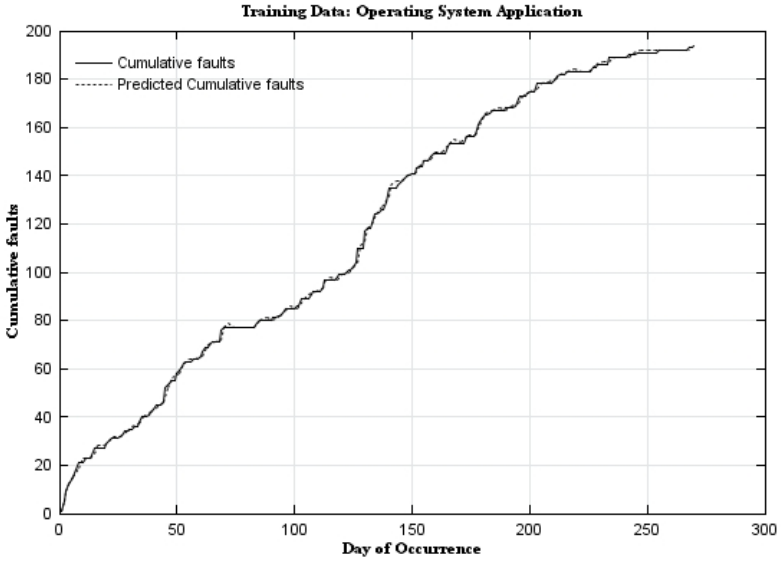


FIGURE 5.22 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#SS1C

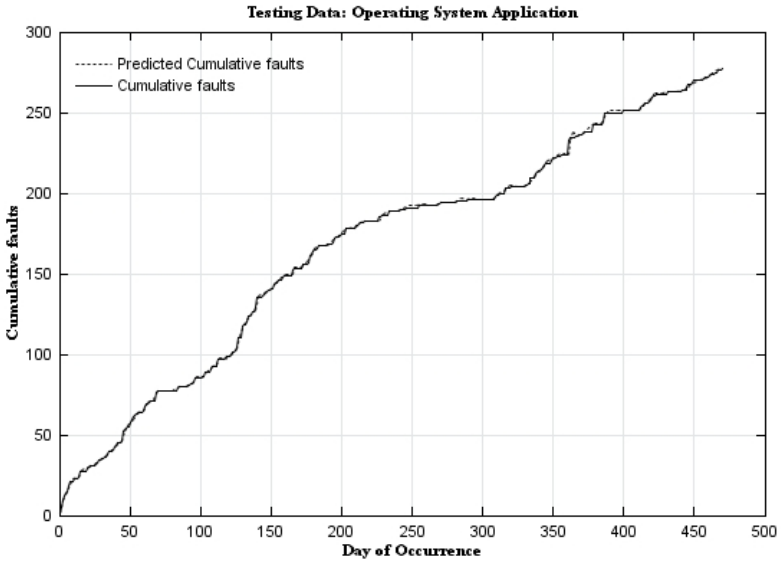


FIGURE 5.23 – Défaillances observées et prédites pendant la phase de test du projet#SS1C

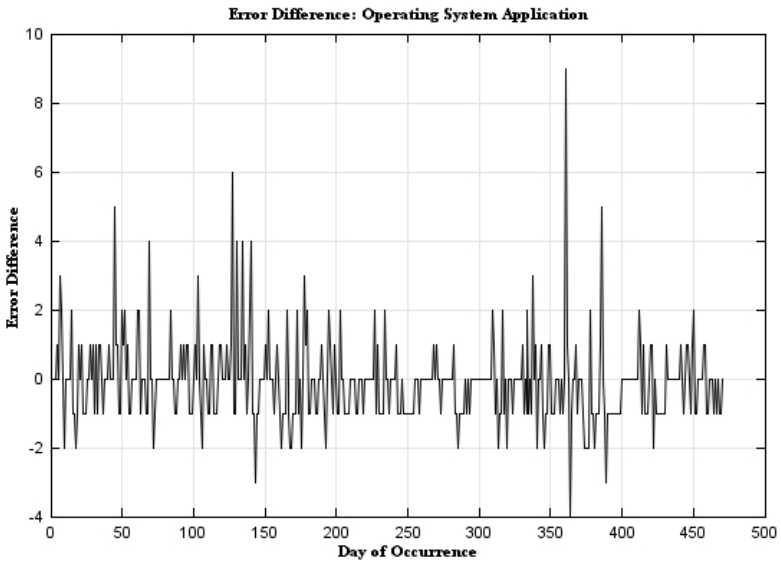


FIGURE 5.24 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#SS1C

Selon les résultats du tableau 5.4 et les figures de 5.16 à 5.24, nous remarquons que le RNA entraîné par le RS fonctionne très bien, à travers toutes les expériences. En outre, il a une meilleure capacité prédictive pour tous les projets. Les résultats de la $NRMSE$ produits par le RNA entraîné par le RS sont 3.88384x10^{-5}, 2.966878x10^{-4} et 1.392739x10^{-5} contre 1.07755, 0.5644 et 0.7714 obtenus par le RNA entraîné par l'algorithme de rétro-propagation, respectivement pour les projets "Real Time & Control#1", "Military#40" et "Operating System Application#SS1C" dans la phase de test. Nous remarquons aussi les différences entre l'AE et le RS de 1.07545 (AE) et 1,07546 (RS), 0.5640 (AE) et 0,5641 (RS) et 0.7713 (AE) et 0,7714 (RS) respectivement pour les trois projets. En conséquence, bien que les réseaux de neurones sont des meilleurs prédicateurs dans la plupart des cas, leur apprentissage en dépend.

En conclusion les modèles de prédiction basé sur les réseaux de neurones entraîné par l'AE et le RS donne des résultats très satisfaisants comparés avec le même modèle entraîné par l'algorithme de rétro-propagation. Une légère différence est remarquée entre l'apprentissage à l'aide des AE et le RS qui donne une meilleure optimisation des paramètres. L'utilisation de l'algorithme du RS peut diminuer le temps de calcul, car, dans ce cas nous essayons d'optimiser une seule solution tandis que dans l'AE nous cherchons à améliorer l'ensemble d'une population de solutions.

5.5.3.4 Les modèles d'AR entraînés par l'AE

Le tableau 5.5 illustre les résultats des MSE et $NRMSE$ obtenus par les trois modèles d'auto-regression entraînés par notre algorithme évolutionniste pendant les deux phases d'apprentissage et de test pour les trois projets. Dans les figures de 5.25 à 5.33 sont représentés les résultats de prédiction et la marge de l'erreur observée.

Project Name	Military#40	Real Time & Control #1	Operating System Application #SS1C
Number of Faults	101	136	277
Training Data	71	96	194
Modèle d'auto-regression d'odre 4			
MSE	1.7323943	2.6185567	1.7474227
$NRMSE$	1.6062E-4	6.0893E-4	3.1551E-5
Modèle d'auto-regression d'odre 7			
MSE	2.6197183	3.8659794	2.3556702
$NRMSE$	1.9752E-4	7.3989E-4	3.6633E-5
Modèle d'auto-regression d'odre 10			
MSE	2.915493	1.8762887	3.9948454
$NRMSE$	2.0837E-4	5.1545E-4	4.7705E-5
Testing Data	101	136	277
Modèle d'auto-regression d'odre 4			
MSE	2.6930692	2.7867646	2.1227436
$NRMSE$	3.7809E-5	3.1940E-4	1.3047E-5
Modèle d'auto-regression d'odre 7			
MSE	4.4059405	8.536765	2.9530685
$NRMSE$	4.8361E-5	5.5903E-4	1.5388E-5
Modèle d'auto-regression d'odre 10			
MSE	7.643564	6.529412	5.6498194
$NRMSE$	6.3698E-5	4.8891E-4	2.1285E-5

TABLE 5.5 – Résulats MSE et $NRMSE$ obtenus par les modèles d'AR d'ordre 4, 7 et 10 entraînés par l'AE

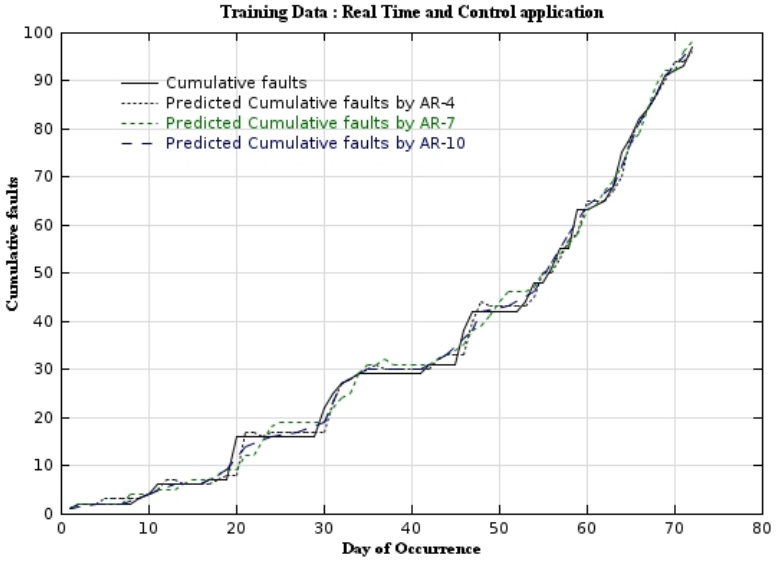


FIGURE 5.25 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#1

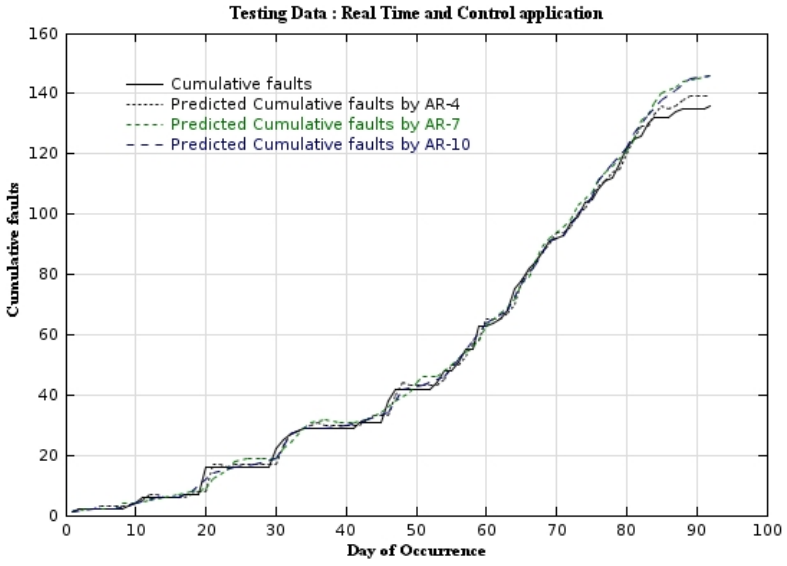


FIGURE 5.26 – Défaillances observées et prédites pendant la phase de test du projet#1

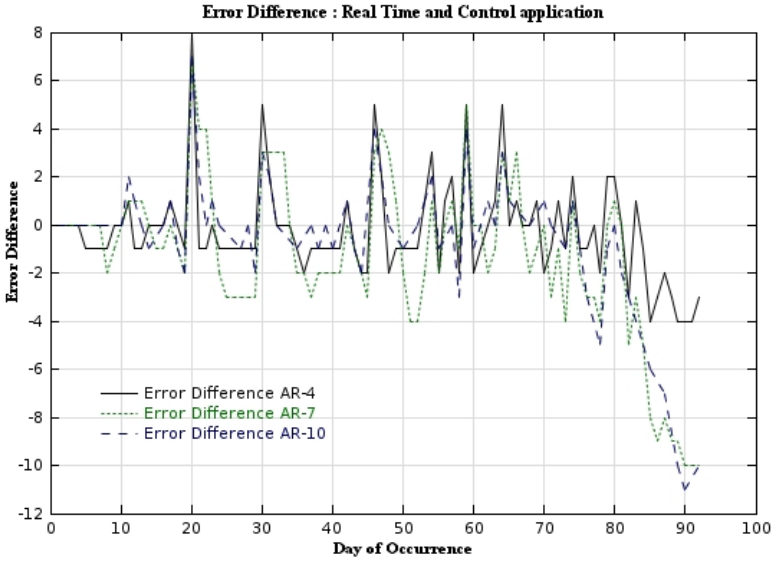


FIGURE 5.27 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#1

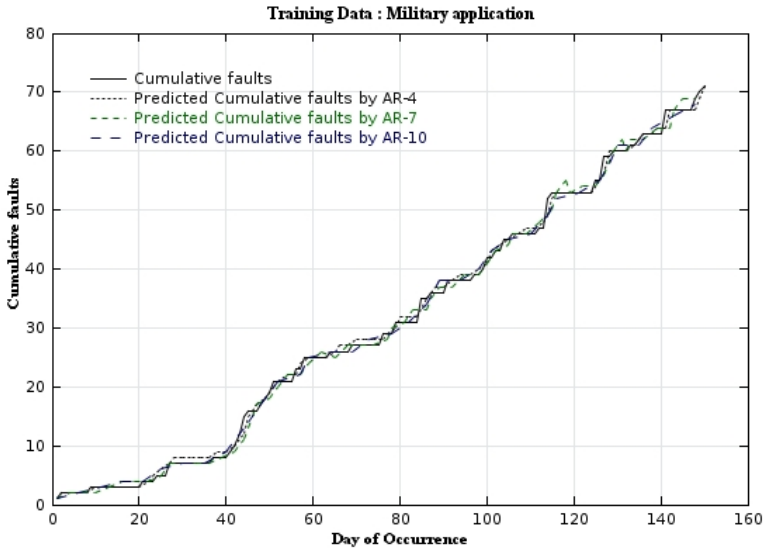


FIGURE 5.28 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#40

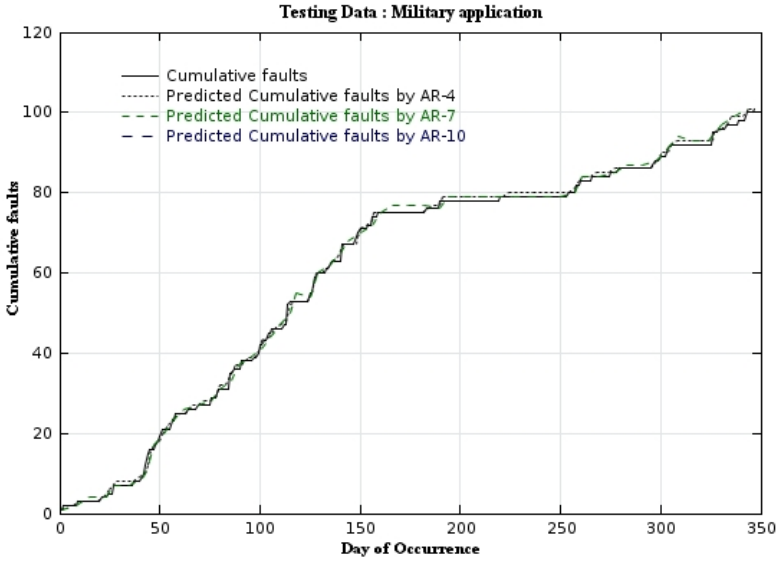


FIGURE 5.29 – Défaillances observées et prédites pendant la phase de test du projet#40

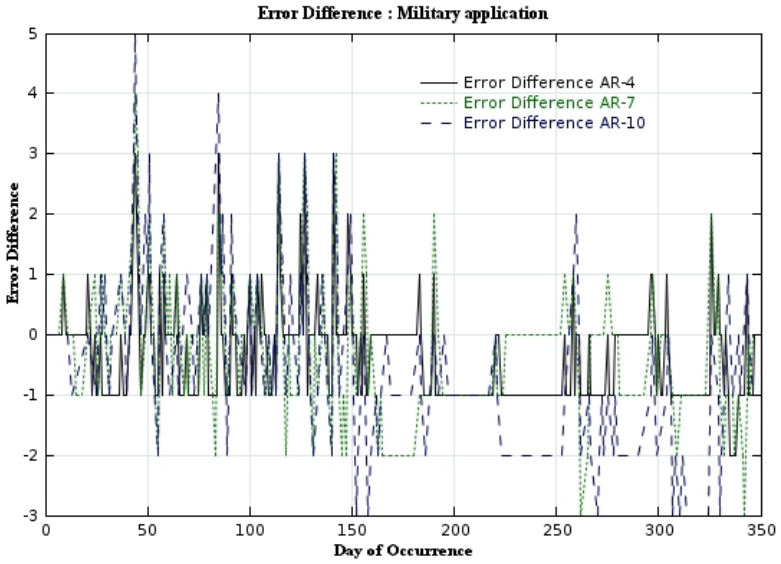


FIGURE 5.30 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#40

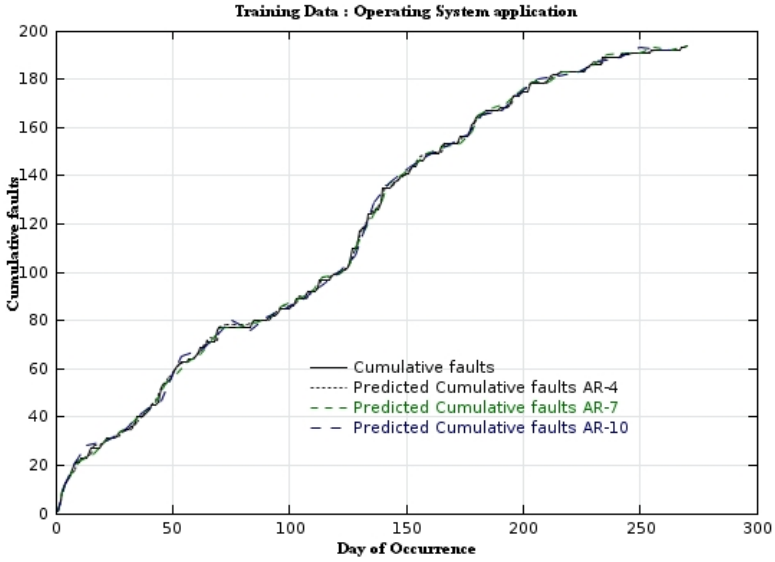


FIGURE 5.31 – Défaillances observées et prédites pendant la phase d'apprentissage du projet#SS1C

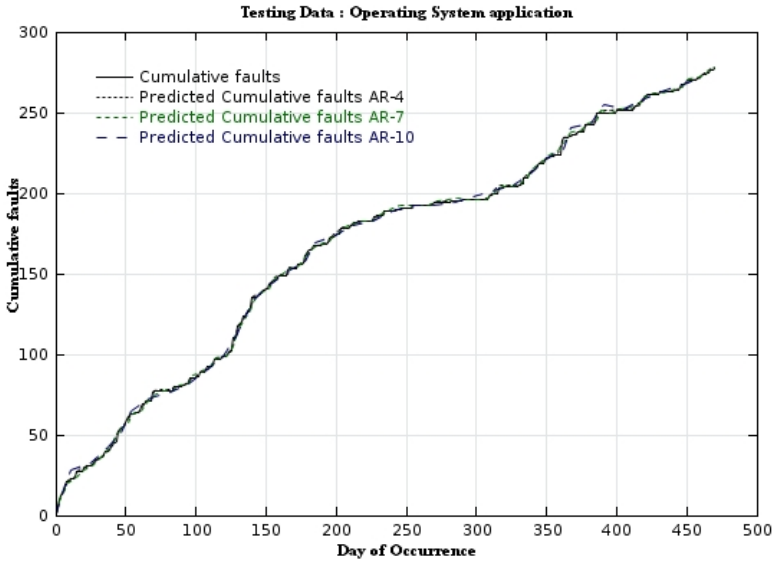


FIGURE 5.32 – Défaillances observées et prédites pendant la phase de test du projet#SS1C

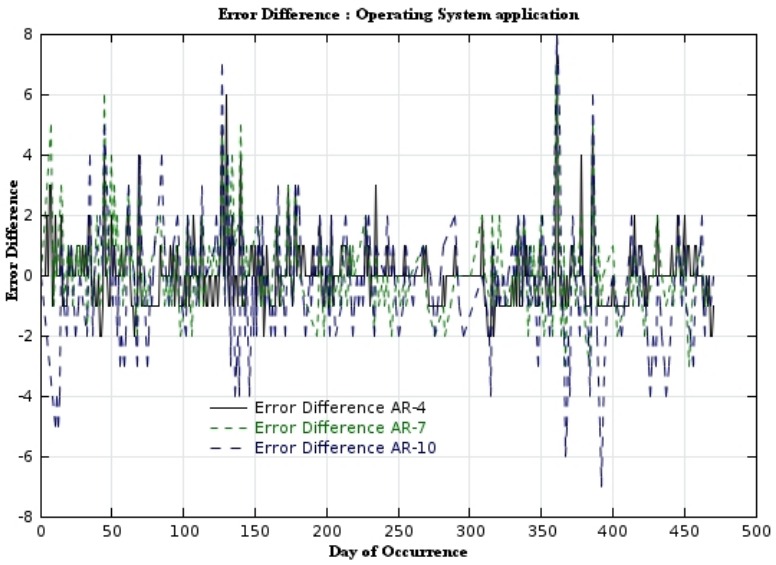


FIGURE 5.33 – Différence de l'erreur entre les défaillances observées et les défaillances prédites du projet#SS1C

Comme le montre les résultats du tableau 5.5 et les figures de 5.26 à 5.33 la prédiction à l'aide du modèle d'AR d'ordre 4 entraîné par l'AE est très performante par rapport au même modèle entraîné par l'estimateur des moindres carrés. Par exemple on remarque pour le projet "Military#40" un taux de la $NRMSE$ de 3.7809x10^{-5} contre 3.1434 soit une différence de 3,1433, pour le projet "Real Time & Control#1" la $NRMSE$ est de 3.1940x10^{-4} contre 1.70806 soit une différence de 1,7077 et pour le projet "Operating System Application#SS1C" la $NRMSE$ est de l'ordre de 1.3047x10^{-5} contre 1.0659 soit une différence de 1,0658. L'augmentation de l'ordre du modèle de 4 à 7 puis à 10 perturbe légèrement la prédictibilité du modèle à travers les trois projets étudiés. En effet la différence entre les défaillances observées et prédites est de 8 enregistré une fois par l'AR-4 pour le projet#1 contre 11 pour les modèles d'ordre 7 et 10, 3 défaillances et la différence obtenu par l'AR-4 contre 5 par l'AR-7 et l'AR-10 pour le projet#40, 8 défaillances est la différence maximale obtenue par les trois modèles pour le projet#SS1C. Par conséquent l'AR d'ordre 4 donne des résultats intéressants par rapport à ceux d'ordre 7 et 10 et c'est ce qui est souhaité, un modèle qui permet de prédire les futures défaillances avec un nombre minimal d'observations.

5.6 Conclusion

Dans ce chapitre nous avons présenté deux modèles hybrides non paramétriques en se basant sur les réseaux de neurones et le modèle d'auto-regression comme prédicateurs, pour lesquels nous avons intervenu dans la phase d'apprentissage par les AE et le RS. Les résultats présentés montrent que les réseaux de neurones et le modèle d'auto-regression peuvent être utilisés pour construire des modèles de prédiction de fiabilité des logiciels à condition que leur apprentissage soit optimisé par un algorithme évolutionniste et du recuit simulé. La différence de l'erreur obtenu en intervenant dans la phase d'apprentissage est très petite par rapport à celle obsevée en utilisant les méthodes d'apprentissage classiques. Cela donne une très bonne indication de la capacité de prévision des modèles développés.

CHAPITRE 6

Ennumération des chemins minimaux dans un réseau

Sommaire

6.1 Introduction

Certains réseaux tels que les réseaux électriques, les réseaux de distribution d'eau et de gaz ainsi que certains cas des réseaux de communication peuvent être représentés sous forme d'un graphe orienté. Dans ces réseaux les liens sont souvent sujets de défaillance, donc la fiabilité dans ce cas peut être exprimée en fonction des liens.

L'évaluation de la fiabilité d'un réseau est un problème NP-complet [11]. Les approches d'évaluation de fiabilité exploitent une série d'outils pour la modélisation du système et pour le calcul de sa fiabilité. Parmi eux celles qui utilisent les chemins et/ou coupes minimales entre une pair de nœuds d'un graphe donné [117, 118].

De nombreux algorithmes ont été conçus pour énumérer l'ensemble des chemins et des coupes minimales. Arunkumar et Lee [119] ont présenté une méthode implicite qui permet d'énumérer toutes les coupes minimales dans des graphes orientés. Biegel [120] a utilisé la transposée et la somme des matrices pour les graphes acyclique orientés. Rai et Aggarwal [121] ont fait appel à l'algèbre de Boole, pour déterminer les coupes minimales à partir des chemins minimaux et vice-versa dans un système cohérent. Jamson [122, 123] a proposé une méthode qui permet de réduire le nombre de chemins minimaux et des coupes minimales en des chemins minimaux et des coupes minimales de base, respectivement.

Dans ce chapitre nous présentons un nouveau algorithme évolutionniste [16] pour énumérer tous les chemins minimaux dans un réseau orienté. Cet algorithme exploite le principe des algorithmes évolutionnistes à savoir une population initiale et les opérateurs de mutation et de reproduction. L'algorithme ainsi développé tourne avec une complexité polynomiale.

6.2 Notations et nomenclature

Un réseau est un graphe orienté $G(V, E)$ avec V l'ensemble des nœuds et E l'ensemble des liens dans le graphe G. M la matrice des successeurs des différents nœuds de G, $M \in R^{nxm}$ avec n le nombre de nœuds dans V et $m = \max_{i=1}^{n} d_s(i)$, la première colonne de M est formée des numéros des nœuds et chaque ligne contient les successeurs d'un nœud donné à partir de la deuxième colonne. Une ligne L_i de la matrice M est complétée par des zéros si le nombre de successeurs du nœud $i < m - 1$.

- $d_s(i)$: Degré sortant du nœud i qui représente le nombre de nœuds successeurs de celui-ci.
- $e =< u, v >$: Lien dans le réseau dont la source est u et v sa destination.
- MP : Chemin minimale qui est sous forme de $e_1 e_2 ... e_i ... e_n$ avec $e_1 =< s, i >$ et $e_n =< i, t >$, avec i un nœud successeur ou prédécesseur selon le cas. Un chemin minimal est un chemin simple (un chemin orienté sans répétition des nœuds) d'un nœud source s vers un nœud destination t.

- p_j : Porbabilité que le lien j soit opérationnel.

La figure ci-dessous présente un exemple d'un réseau orienté.

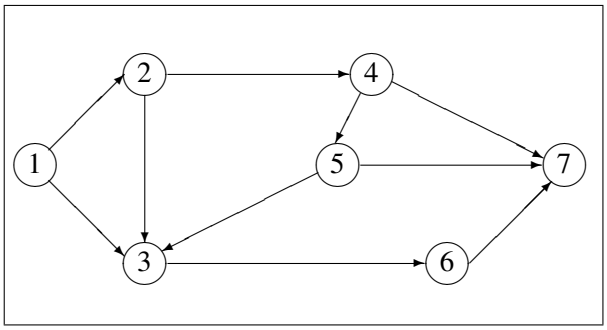


FIGURE 6.1 – Réseau test

La matrice M ci-dessous est la matrice des successeurs correspondante au réseau de la figure 6.1.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 6 & 0 \\ 4 & 5 & 7 \\ 5 & 3 & 7 \\ 6 & 7 & 0 \\ 7 & 0 & 0 \end{pmatrix}$$

Dans cette représentation nous remarquons que nous avons seulement besoin de sauvegarder les successeurs des nœuds. Cette représentation a plusieurs avantages dont :

- Le nombre de colonnes est égal au maximum des degrés sortants des nœuds, qui est généralement inferieur au nombre total des nœuds (n) dans le réseau.
- Le nœud de destination peut facilement être identifié comme n'ayant aucun successeur.
- Cette représentation du réseau est très flexible car elle permet de sauvegarder les liens bidirectionnels et les graphes planaires.
- Aucune restriction sur la limite de nombre des nœuds sources et les nœuds destinations.

6.3 Algorithme de parcours évolutionniste

Dans notre travail nous adoptons une stratégie d'évolution qui consiste à muter un chromosome en remplaçant un composant de celui-ci par un autre (son successeur dans le réseau) et une reproduction qui consiste à dupliquer une partie valide du chromosome en

nombre de successeurs à partir d'une composante valide. Dans cette section, nous décrivons un nouvel algorithme de parcours évolutionniste permettant de retrouver tous les chemins minimaux liant le nœud source s au nœud terminal t. Cet algorithme est basé sur un concept exploitant la démarche d'un algorithme évolutionniste à savoir une génération d'une famille de chromosomes (chemins) qui vont subir par la suite des phases de mutations et de reproduction et qui s'articule autour des étapes présentées dans ce qui suit.

Définitions :

- A chaque itération i de l'algorithme ci-dessous, notons par C_i, un chromosome de la population.
- Une composante d'ordre k de C_i est dite valide si elle est dans la liste des successeurs de la composante d'ordre $k-1$.
- Le degré de validité d'un chromosome C_i, noté par $dv(C_i)$, est l'indice de la dernière composante valide C_i.
- Un chromosome C_i est dit valide si son degré de validité correspond au nœud destination t.

Règles :

- Un chromosome déclaré valide doit quitter la liste des chromosomes vers la liste des chemins minimaux après son décodage.
- Tout chemin trouvé ne doit pas être dupliqué dans la liste des chemins.

La population initiale des chromosomes :

On génère une population de chromosomes de taille égale au nombre de successeurs du nœud source $\{s, s_i, t, 0, ..., 0\}$ avec s_i un successeur de s.

Evolution de la liste des chromosomes :

Dans ce paragraphe, nous décrivons un processus itératif permettant de faire évoluer la population des chromosomes vers la liste des chemins.

1. **Etape 1 : Evolution :** Soit C_i un chromosome, q son degré de validité (à l'itération en cours) et $\ell = C_i(q)$ qui correspond au nœud valide dans C_i. On calcule m le degré sortant du nœud ℓ puis à partir de C_i on génère m chromosomes ayant les q premières composantes identiques.
2. **Etape 2 : Mutation ou multiplication d'un chromosome :** Soit C_i un chromosome après régénération de sa $(q+1)^{eme}$ composante notée f.
   - si $f = t$, le chromosome est déclaré totalement valide. Et dans ce cas C_i doit rejoindre la liste des chemins. sinon nous avons alors l'un des deux cas suivants :
   - $ds(f) = 1$, dans ce cas, nous faisons subir à C_i une mutation simple qui consiste à augmenter sa validité de 1 et remplacer f par son successeur.

- $ds(f) > 1$, le chromosome C_i va se multiplier pour donner naissance à $ds(f)$ chromosomes à partir des successeurs de f. Le degré de validité de chacun des nouveaux chromosomes doit être augmenté de 1.

3. **Etape 3 :** Tant que la liste des chromosomes est non vide reprendre depuis l'étape 1

A chaque niveau du processus nous devons exécuter des tests relatifs aux règles présentées dans l'organigramme de la figure 5.2

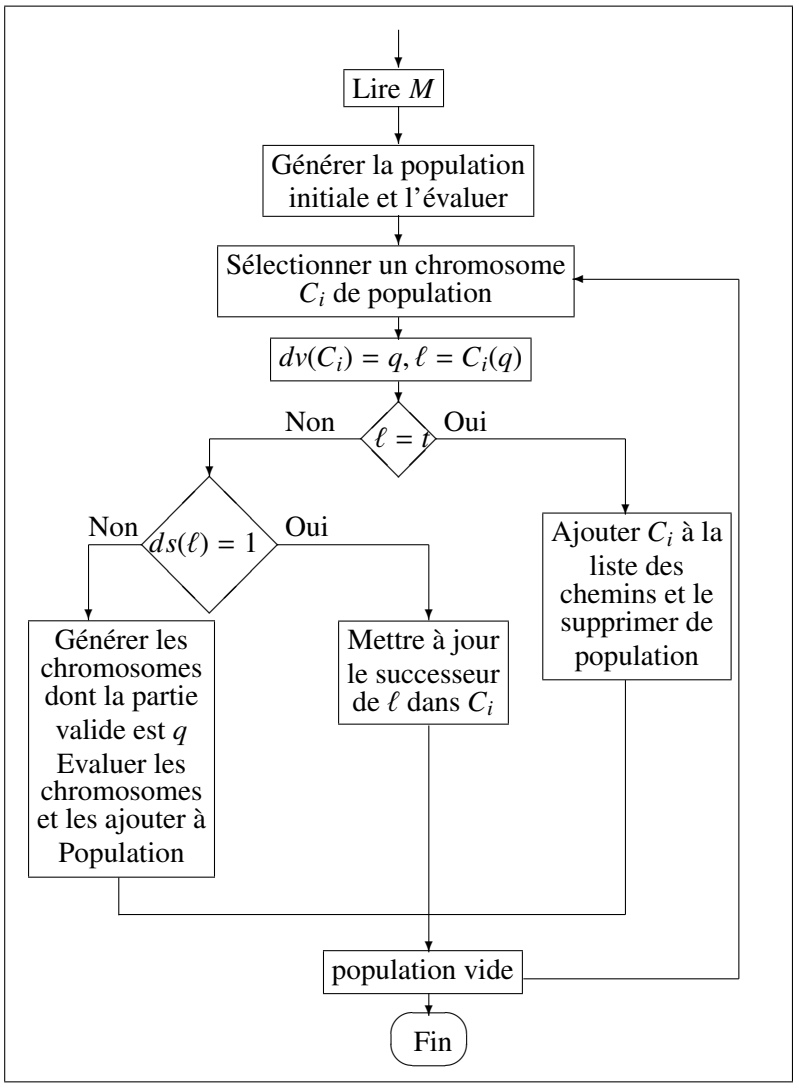


FIGURE 6.2 – Organigramme de l'algorithme

6.4 Résultats et interprétations

L'algorithme a été implanté en utilisant le langage de programmation Java. Notre algorithme a une complexité $O(n^2)$, en effet, dans un graphe orienté le processus de recherche va atteindre le nœud terminal dans le pire des cas dans n étapes (n étant le nombre de nœuds du réseau). En outre l'ordre du processus de recherche dans le pire des cas est de $O(n)$ et chaque nœud aura $n-1$ successeurs dans le pire des cas, donc, la complexité de l'algorithme est de $O(n(n-1)) = O(n^2)$. Dans le cas d'un réseau avec plusieurs sources, pour chaque nœud on opère de la même façon que précédemment, avec le nombre de nœuds sources limité donc la complexité est toujours de $O(n^2)$.

Illustration :
Soit le réseau de la figure 6.1, avec le nœud source $s = 1$ et le nœud terminal $t = 7$. La population des chromosomes $P_{ch} = \{C_1, C_2\}$ contient les deux chromosomes suivants : $C_1 = \{1, 2, 7, 0, 0, 0, 0\}$, $C_2 = \{1, 3, 7, 0, 0, 0, 0\}$ et la liste des chemins $L_{MP} = \{\emptyset\}$. L'évaluation des chromosomes donne : $d_v(C_1) = 2$ et $d_v(C_2) = 2$.
Soit le chromosome C_1 de degré de validité $d_v(C_1) = 2$, $C_1[2] = 2$ et $d_s(2) = 2$, donc C_1 va donner naissance à deux chromosomes, qui sont : $C_3 = \{1, 2, 3, 0, 0, 0, 0\}$ de $d_v(C_3) = 3$ et $C_4 = \{1, 2, 4, 0, 0, 0, 0\}$ de $d_v(C_4) = 3$, donc $P_{ch} = \{C_2, C_3, C_4\}$
Soit $C_2 = \{1, 3, 7, 0, 0, 0, 0\}$ de $d_v(C_2) = 2$, $C_2[d_v(C_2)] = 3$ et $d_s(3) = 1$ donc C_2 va subir une mutation au niveau de sa $3^{\text{ème}}$ composante qui est remplacée par le successeur de 3 d'où $C_2 = \{1, 2, 3, 6, 0, 0, 0\}$, $d_v(C_2) = 4$ et $C_2[d_v(C_2)] = 6$, $d_s(6) = 7$ qui est la destination donc C_2 est déclaré valide et doit quitter la population des chromosomes vers la liste des chemins d'où $P_{ch} = \{C_3, C_4\}$ et $L_{MP} = \{C_2\}$.
Soit $C_3 = \{1, 2, 3, 0, 0, 0, 0\}$, $d_v(C_3) = 3$, $C_3[3] = 3$ et $d_s(3) = 1$ donc on mute C_3 au niveau de sa $4^{\text{ème}}$ par le successeur de $C_3[d_v(C_3)] = 3$, C_3 devient alors $C_3 = \{1, 2, 3, 6, 0, 0, 0\}$, $d_v(C_3) = 4$, $d_s(C_2[4]) = 1$ et le successeur de $C_2[4] = 7$ donc C_3 devient $C_3 = \{1, 2, 3, 6, 7, 0, 0\}$ de $d_v(C_3) = 5$ avec $C_3[5] = 7$ qui est le nœud destination, d'où C_3 est déclaré totalement valide et il doit quitter P_{ch} vers $L_{MP} = \{C_3, C_2\}$ et $P_{ch} = \{C_4\}$.
Soit $C_4 = \{1, 2, 4, 0, 0, 0, 0\}$ de $d_v(C_4) = 3$, $d_s(3) = 2$, alors C_4 va donner naissance à deux chromosomes $C_5 = \{1, 2, 4, 5, 0, 0\}$ de $d_v(C_5) = 4$ et $C_7 = \{1, 2, 4, 7, 0, 0, 0\}$ de $d_v(C_7) = 4$, alors $P_{ch} = \{C_5, C_7\}$.
Soit $C_7 = \{1, 2, 4, 7, 0, 0\}$ $d_v(C_7) = 5$, $d_s(d_v(C_7)) = 1$ et $C_7[5] = 7$ qui est la destination donc C_7 est totalement valide et il doit rejoindre la liste des chemins, qui devient $L_{MP} = \{C_3, C_2, C_7\}$ et $P_{ch} = \{C_5\}$.
Soit $C_5 = \{1, 2, 4, 5, 0, 0\}$, $d_v(C_5) = 5$, $d_s(d_v(C_5)) = 2$ d'où C_5 va donner naissance à deux chromosomes en gardant sa partie valide et en mutant la $5^{\text{ème}}$ composante par les deux successeurs de $C_5[d_v(C_5)]$, donc $C_8 = \{1, 2, 4, 5, 7, 0, 0\}$ avec $d_v(C_8) = 5$ et $C_9 = \{1, 2, 4, 5, 3, 0, 0\}$ avec $d_v(C_9) = 5$, alors $P_{ch} = \{C_8, C_9\}$.

Soit $C_8 = \{1, 2, 4, 5, 7, 0, 0\}$, $d_v(C_8) = 5$ et $C_8[5] = 7$ qui est la destination donc C_8 va quitter la population des chromosomes vers la liste des chemins ; $L_{MP} = \{C_3, C_2, C_7, C_8\}$ et $P_{ch} = \{C_8\}$.
Soit $C_9 = \{1, 2, 4, 5, 3, 0, 0\}$, $d_v(C_9) = 5$, $d_s(C[5]) = 1$ donc C_9 va subir une mutation au niveau sa $6^{\text{ème}}$ et devient alors $C_9 = \{1, 2, 4, 5, 3, 6, 0\}$ ayant $d_v(C_9) = 6$ d'où C_9 va subir une mutation au niveau de sa $7^{\text{ème}}$ composante, C_9 devient alors $C_9 = \{1, 2, 4, 5, 3, 6, 7\}$ de $d_v(C_9) = 7$ et $C_9[d_v(C_9)] = 7$ donc C_9 va rejoindre $L_{MP} = \{C_2, C_3, C_7, C_8, C_9\}$ et $P_{ch} = \{\emptyset\}$
Après exécution on obtient les chemins suivants :
$MP_1 = \{< 1, 2 >, < 2, 4 >, < 4, 7 >\}$, $MP_2 = \{< 1, 2 >, < 2, 4 >, < 4, 5 >, < 5, 7 >\}$, $MP_3\{< 1, 2 >, < 2, 3 >, < 3, 6 >, < 6, 7 >\}$, $MP_4 = \{< 1, 3 >, < 3, 6 >, < 6, 7 >\}$, $MP_5 = \{< 1, 2 >, < 2, 4 >, < 4, 5 >, < 5, 3 >, < 3, 6 >, < 6, 7 >\}$
Dans cet exemple l'algorithme a démarré avec une population de deux solutions initiales. Ce test concerne un réseau sans cycle, mais l'algorithme est aussi opérationnel sur les réseaux avec cycle.

Dans le but de tester la robustesse de l'algorithme mis en œuvre, nous avons utilisé 9 réseaux orientés (figure 6.3) publiés dans la littérature [124, 125].

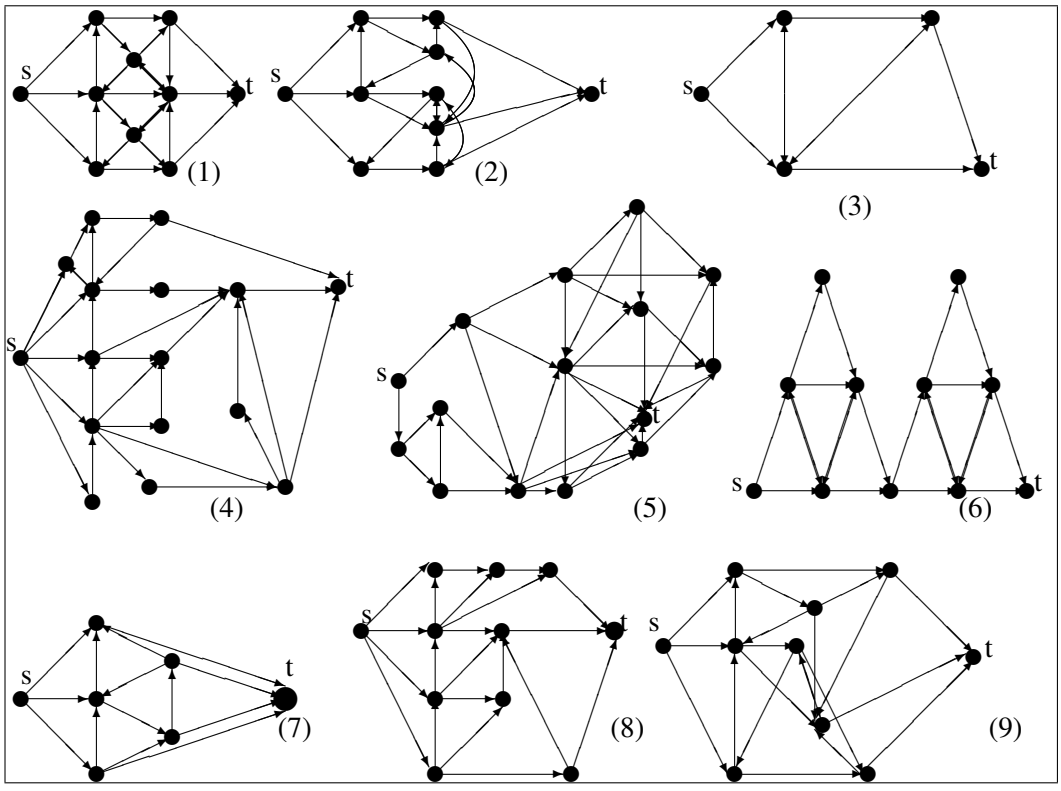


FIGURE 6.3 – Réseaux utilisés comme benchmark

6.5 Evaluation de la fiabilité

Après l'énumération des chemins minimaux du réseau, l'une des méthodes d'évaluation de la fiabilité peut être utilisée, à savoir le principe d'inclusion-exclusion (voir section 2.3.7.2 du chapitre 2) ou la méthode des produits disjoints (voir section 2.3.7.3 du chapitre 2). Nous utilisons ici le principe d'inclusion-exclusion pour calculer la fiabilité des réseaux du benchmark. Un réseau est opérationnel si au moins un de ses k chemins minimaux est opérationnel et est défaillant si tous les k chemins minimaux reliant s à t sont défaillants. Soient E_i l'événement que tous les liens dans le chemin MP_i sont opérationnels, et F_i l'événement qu'un chemin MP_i soit défaillant. Un chemin est opérationnel si toutes ses composantes (liens) sont opérationnelles et il est défaillant si au moins un de ses liens est défaillant. Donc la non fiabilité $1 - R_{st}$ est la probabilité que tous les événements F_i se produisent ; selon le principe d'inclusion-exclusion (section 2.3.7.2 du chapitre 2) nous avons :

$$1 - R_{st}(G) = \left(\sum_{i}^{k} P(F_i) - \sum_{1 \le i < j \le k} P(F_i F_j) + \sum_{1 \le i < j < l \le k} P(F_i F_j F_l) - ... + (-1)^k P(F_1 F_2 ... F_k) \right)$$

donc

$$R_{st}(G) = 1 - \left(\sum_{i}^{k} P(F_i) - \sum_{1 \le i < j \le k} P(F_i F_j) + \sum_{1 \le i < j < l \le k} P(F_i F_j F_l) - ... + (-1)^k P(F_1 F_2 ... F_k) \right) \tag{6.1}$$

Un chemin minimale peut être considéré comme un ensemble de composantes (liens) montées en série donc la fiabilité r_i et la non fiabilité q_i d'un chemin MP_i de longueur l sont :

$$\begin{aligned} r_i &= P(E_i) = \prod_{j=1}^{l} p_j \\ q_i &= 1 - r_i \end{aligned} \tag{6.2}$$

avec p_i est la probabilité de succés du lien i.

La probabilité de succés des liens des différents réseaux du benchmark est fixée à 0.9 et les liens sont indépendants. En utilisant les équations 6.1 et 6.2, le tableau 6.1 présente les résultats obtenus pour chaque réseau.

			Résultats dans [124]		Résultats dans [125]	
Réseau	Nombre de Chemins	Fiabilité	Nombre de Chemins	Fiabilité	Nombre de Chemins	Fiabilité
Réseau test	5	0.997821	-	-	-	-
(1)	64	0.996176	-	-	-	-
(2)	63	0.996172	64	0.997506	-	-
(3)	7	0.987918	-	-	-	-
(4)	36	0.997070	36	0.997186	36	0.99719
(5)	131	0.996727	-	-	-	-
(6)	100	0.999829	-	-	100	0.95962
(7)	14	0.977038	14	0.996665	14	0.99666
(8)	18	0.996345	18	0.994076	18	0.99408
(9)	64	0.996176	-	-	64	0.99751

TABLE 6.1 – Comparaison des résultats obtenus avec ceux de la bibliographie.

6.6 Conclusion

Dans ce chapitre nous avons présenté un algorithme évolutionniste qui permet d'énumérer tous les chemins minimaux dans un graphe orienté avec ou sans cycle. Les résultats obtenus sur des cas académiques [124, 125] ont montré que notre algorithme produit bien en quantité et en qualité les chemins minimaux. Les chemins issues de cet algorithme sont ensuite utilisés pour quantifier la fiabilité du réseau en question, en utilisant le principe d'inclusion-exclusion. En utilisant les mêmes données que dans [124, 125], nous avons produit des valeurs similaires de fiabilité des réseaux. Cet algorithme peut être utilisé pour énumérer les chemins minimaux de tout autre système qui peut être modélisé sous forme d'un graphe orienté.

CHAPITRE 7

Evaluation de la fiabilité d'un réseau en fonction des coupes minimales

Sommaire

7.1 Introduction

Ces dernières années, la théorie de fiabilité des réseaux a été appliquée dans de nombreux systèmes, à savoir les systèmes de communication, les réseaux électriques et les réseaux de distribution. La fiabilité des réseaux joue un rôle important dans notre société moderne [126]. de nombreuses études ont été réalisées pour évaluer la fiabilité d'un réseau [127, 128, 129, 130, 131], la plupart de ces méthodes évaluent la fiabilité du réseau en terme des coupes minimales ou des chemins minimaux. Toutefois, la recherche de toutes les coupes minimales ou de tout les chemins minimaux est un problème NP-difficile. Mais, l'énumération de toutes les coupes minimales peut être moins coûteux que celle des chemins minimaux, si le nombre de chemins est trop énorme pour énumérer. Un exemple de ce type est le réseau maillé 2x100 qui a 2^{99} chemins et 10000 coupes [128, 129], il n'est pas pratique d'énumérer tout ces chemins, mais si le nombre de coupes minimales est beaucoup moins, alors il peut être un moyen efficace pour évaluer la fiabilité en terme de ces coupes. Une autre approche consiste à obtenir les coupes minimales à partir des chemins minimaux par inversion [132], cependant, il n'est pas pratique de générer l'ensemble des chemins minimaux dans certains graphes non orientés. Tsukiyama [133] a présenté des algorithmes et les notions mathématiques nécessaires pour énumérer toutes les coupes minimales d'un graphe non orienté. Dans ce chapitre nous présentons un algorithme [17] efficace pour énumérer toutes les coupes minimales dans un graphe sans duplication. Dans la section 7.2 nous présentons quelques notions préliminaires et hypothèses sur lesquelles nous nous baserons dans le développement de l'algorithme. La section 7.3 est consacrée pour la présentation du dit algorithme. Tandis que la section 7.4 présente les résultats obtenus par notre algorithme.

7.2 Notations préliminaires et hypothèses

$G = (V, E)$: un réseau avec un ensemble de nœuds V et de liens E.

s : le nœud source dans le réseau G.

t : le nœud terminal dans le réseau G.

$e_{v,w} =< v, w >$: lien reliant le nœud v au nœud w.

S : un ensemble de nœuds contenant s.

T : un ensemble de nœuds qui contient t.

C_i : la coupe minimale numéro i.

C_v : la coupe minimale associée au nœud v.

MC : coupe minimale.

MCs : ensemble des coupes minimales.

Fiabilité : la probabilité qu'un signale peut être transmis d'un nœud source s vers un nœud terminal t.

7.2.1 Préliminaires

Tout les réseaux étudiés ici sont des réseaux avec deux nœuds spéciaux : le premier appelé le nœud source s et le deuxième appelé le nœud terminal t. Le succès ou la défaillance du réseau dépend du fait qu'il y est une connexion entre s et t. Comme il est signalé dans la section 2.3.2 du chapitre 2 la fiabilité du réseau peut être étudiée en prenant on considération ; soit la défaillance des nœuds, dans ce cas les MC sont formées de nœuds, soit la défaillance des liens, donc les MC sont formées de liens et le dernier cas prend en considération la défaillance des liens et des nœuds, par conséquent, les MC sont formées des deux éléments (liens et nœuds). Dans notre travail nous avons considéré des réseaux avec la défaillance des liens, dans lesquels chaque liens est représenté par une pair non ordonnée de nœuds $e =< v, w >$, v et w sont dits adjacents, et le lien e est incident avec v et w. Un chemin entre deux nœuds v_0 et v_n dans G est une séquence de liens distincts $< v_0, v_1 >, < v_1, v_2 > ... < v_{n-1}, v_n >$, n est la longueur du chemin. Un graphe est connexe s'il existe un chemin reliant chaque pair de nœuds du graphe. Désormais nous considérons que tous les graphes étudiés sont des graphes connexes et ne contiennent pas ni des liens parallèles ni des boucles. Une coupe de graphe est un ensemble de liens qui lorsqu'il est retiré du graphe, interrompt toutes les connexions s et t. Une coupe minimale est un ensemble minimal de liens dont la défaillance entraîne la défaillance du réseau. Toutes les défaillances du système peuvent être exprimées par la suppression de au moins une coupe minimale du graphe [130].

On considère, deux nœuds distincts s et t du graphe $G = (V, E)$, pour deux sous-ensembles disjoints X et $Y(X, Y \subset V)$, tels que $s \in X$ et $t \in Y$ on donne :

$$\omega(X, Y) = \{< v, w > : v \in X, w \in Y\} \tag{7.1}$$

Soit X et Y tels que $s \in X$, $t \notin X$ et $Y = \bar{X}$. Si $G(X)$ et $G(Y)$ sont deux sous-graphes connexes, alors la coupe séparant s à t est donnée par :

$$C(s, t) = \omega(X, Y) \tag{7.2}$$

avec $C(s, t)$ une coupe $s - t$ qui sépare les deux nœuds s et t. Si une coupe $s - t$ ne contient pas d'autres coupes, donc elle est appelée une coupe minimale.

Pour un nœud v et un sous ensemble $A(A \subset V)$ donnés nous définissons les ensembles suivants :

$$\Gamma(v) = \{w \in V \ :< v, w > \in E\} \tag{7.3}$$

$$\Gamma(A) = \cup_{v \in A} \Gamma(v) \tag{7.4}$$

$$\Gamma^{+}(v) = \Gamma(v) - \{v\} \tag{7.5}$$

$$\Gamma^{+}(A) = \Gamma(A) - A \tag{7.6}$$

Définition 7.2.1 *: Pour un graphe $G = (V, E)$, un sous-ensemble $C \subset E$ est une coupe minimale si et seulement si la suppression de tous les liens dans C de G, divise ce dernier en deux composants connexes. Lorsqu'un nœud est isolé, il est considéré comme un composant. Une coupe minimale devise alors les nœuds de G en deux sous-ensembles disjoints X et Y, chacun induit un sous graphe connexe.*

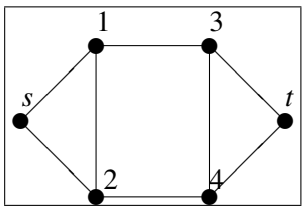
Définition 7.2.2 *: La coupe C_s est la coupe minimale qui sépare le graphe en deux sous-ensembles disjoints X et Y, avec $X = \{s\}$ et $Y = V \setminus X$.*

Observation 7.2.1 *: A partir des équations et des définitions ci-dessous, on remarque que chaque coupe d'un réseau peut être déduite d'une autre sauf, la coupe qui sépare le nœud source du reste du réseau. Une coupe C_v peut être obtenue en utilisant l'équation suivante :*

$$\begin{aligned} C_v = & C_{ancien} \cup \{< v, w >\in E : w \in \Gamma^{+}(S)\} \\ & \cup \{< v, w >\in E : w \in \Gamma^{+}(v)\} \\ & \setminus \{< w, v >\in E : w \in S\} \end{aligned} \tag{7.7}$$

avec $S \subset V$ l'ensemble initialisé par le nœud source s et v est un nœud choisi dans $\Gamma^{+}(S)$.

Illustration : soit le graphe ci-dessous qui contient 9 coupes minimales en totale
$C_s = \{< s, v >\in E : v \in \Gamma^{+}(s)\} = \{e_{s,1}, e_{s,2}\}$ est la première coupe à définir.



On considère l'ensemble $S \subset V = \{s\}$, donc, $\Gamma^{+}(S) = \{1, 2\}$.
Soit $v \in \Gamma^{+}(S) = \{1, 2\}$ Donc

$$\begin{aligned}
C_1 = C_{v=1} &= C_s \cup \{< 1, w >\in E : w \in \Gamma^+(\{s\})\} \cup \{< 1, w >\in E : w \in \Gamma^+(1)\} \\
&\quad \setminus \{< w, 1 >\in E : w \in \{s\}\} \\
&= \{e_{s,1}, e_{s,2}\} \cup \{e_{1,2}\} \cup \{e_{1,2}, e_{1,3}, e_{s,1}\} \setminus \{e_{s,1}\} \\
&= \{e_{s,2}, e_{1,2}, e_{1,3}\} \\
C_2 = C_{v=2} &= C_s \cup \{< 2, w >\in E : w \in \Gamma^+(\{s\})\} \cup \{< 2, w >\in E : w \in \Gamma^+(2)\} \\
&\quad \setminus \{< w, 2 >\in E : w \in \{s\}\} \\
&= \{e_{s,1}, e_{s,2}\} \cup \{e_{1,2}\} \cup \{e_{1,2}, e_{2,4}, e_{s,2}\} \setminus \{e_{s,2}\} \\
&= \{e_{s,1}, e_{1,2}, e_{2,4}\}
\end{aligned}$$

On mis à jour l'ensemble $S = S \cup \{2\}(S) = \{s, 2\}$ d'où :
Soit $v \in \Gamma^+(S) = \{1, 4\}$ Alors

$$\begin{aligned}
C_3 = C_{v=1} &= C_2 \cup \{< 1, w >\in E : w \in \Gamma^+(\{s, 2\})\} \cup \{< 1, w >\in E : w \in \Gamma^+(1)\} \\
&\quad \setminus \{< w, 1 >\in E : w \in \{s, 2\}\} \\
&= \{e_{s,1}, e_{1,2}, e_{2,4}\} \cup \{\varnothing\} \cup \{e_{1,2}, e_{1,3}\} \setminus \{e_{1,2}, e_{s,1}\} \\
&= \{e_{1,3}, e_{2,4}\} \\
C_4 = C_{v=4} &= C_2 \cup \{< 4, w >\in E : w \in \Gamma^+(\{s, 2\})\} \cup \{< 4, w >\in E : w \in \Gamma^+(4)\} \\
&\quad \setminus \{< w, 4 >\in E : w \in \{s, 2\}\} \\
&= \{e_{s,1}, e_{1,2}, e_{2,4}\} \cup \{\varnothing\} \cup \{e_{3,4}, e_{4,t}\} \setminus \{e_{2,4}\} \\
&= \{e_{s,1}, e_{1,2}, e_{3,4}, e_{4,t}\}
\end{aligned}$$

On mis à jour l'ensemble $S = S \cup \{1\}(S) = \{s, 1, 2\}$ d'où :
Soit $v \in \Gamma^+(S) = \{3, 4\}$ Alors

$$\begin{aligned}
C_5 = C_{v=3} &= C_3 \cup \{< 3, w >\in E : w \in \Gamma^+(\{s, 1, 2\})\} \cup \{< 3, w >\in E : w \in \Gamma^+(3)\} \\
&\quad \setminus \{< w, 3 >\in E : w \in \{s, 1, 2\}\} \\
&= \{e_{1,3}, e_{2,4}\} \cup \{e_{3,4}\} \cup \{e_{3,4}, e_{3,t}\} \setminus \{e_{1,3}\} \\
&= \{e_{2,4}, e_{3,4}, e_{3,t}\} \\
C_6 = C_{v=4} &= C_3 \cup \{< 4, w >\in E : w \in \Gamma^+(\{s, 1, 2\})\} \cup \{< 4, w >\in E : w \in \Gamma^+(4)\} \\
&\quad \setminus \{< w, 4 >\in E : w \in \{s, 1, 2\}\} \\
&= \{e_{1,3}, e_{2,4}\} \cup \{\varnothing\} \cup \{e_{3,4}, e_{4,t}\} \setminus \{e_{2,4}\} \\
&= \{e_{1,3}, e_{3,4}, e_{4,t}\}
\end{aligned}$$

On mis à jour l'ensemble $S = S \cup \{3\}(S) = \{s, 1, 2, 3\}$ d'où :
Soit $v \in \Gamma^+(S) = \{4, t\}$ Alors

$$C_7 = C_{v=4} = C_5 \cup \{< 4, w >\in E : w \in \Gamma^+(\{s,1,2,3\})\} \cup \{< 4, w >\in E : w \in \Gamma^+(4)\}$$
$$\setminus \{< w, 4 >\in E : w \in \{s,1,2,3\}\}$$
$$= \{e_{2,4}, e_{3,4}, e_{3,t}\} \cup \{e_{4,t}\} \cup \{e_{3,4}, e_{4,t}\} \setminus \{e_{3,4}, e_{2,4}\}$$
$$= \{e_{3,t}, e_{4,t}\}$$

Soit $S = \{s,1\}$ donc la coupe C_8 est obtenue à partir de C_1
Soit $v = 3 \in \Gamma^+(S) = \{3,2\}$

$$C_8 = C_{v=3} = C_1 \cup \{< 3, w >\in E : w \in \Gamma^+(\{s,1\})\} \cup \{< 3, w >\in E : w \in \Gamma^+(3)\}$$
$$\setminus \{< w, 3 >\in E : w \in \{s,1\}\}$$
$$= \{e_{s,2}, e_{1,2}, e_{1,3}\} \cup \{\varnothing\} \cup \{e_{3,4}\} \setminus \{e_{1,3}\}$$
$$= \{e_{s,2}, e_{1,2}, e_{3,4}, e_{3,t}\}$$

7.2.2 Hypothèses

Tous les réseau satisfont les hypothèses suivantes :

- Tout nœud est parfaitement fiable.
- Le graphe est connexe.
- Tout lien peut être soit défaillant ou opérationnel (deux états).
- Les états des liens sont statistiquement indépendants.

7.3 L'algorithme

En utilisant les hypothèses, les définitions et l'observation citées ci-dessous, nous avons développé [17] un algorithme récursif qui permet d'énumérer toutes les coupes minimales dans un réseau. Ainsi, il peut être facilement vérifié que l'algorithme "MCsAlgorithm" énumère l'ensemble des coupes $s-t$, sans duplication.

L'algorithme proposé a besoin d'un ensemble de paramètres ; le réseau (graphe G) avec un ensemble de nœuds V et un ensemble de liens E, le nœud source s et le nœud terminal t. La première étape est la détermination de la coupe qui sépare s au reste des nœuds du réseau, l'étape suivante de l'algorithme consiste à déterminer une autre coupe en utilisant la coupe C_s, ce processus est exécuté récursivement en prenant en paramètre la dernière coupe affichée et ses nœuds associés. L'algorithme termine par l'énumération de toutes les coupes du réseau. A chaque appel à l'algorithme "*McsAlgorithm*" les cinq équations de 7.1 à 7.6 sont utilisées pour déduire chaque coupe minimale. Le graphe $G = (\bar{S} \setminus \{v\}, E)$ est le sous-graphe avec $(\bar{S} \setminus \{v\})$ l'ensemble de ses nœuds et l'ensemble des liens définit par $< u, w >\in E$ et $u, w \in (V \setminus S) \setminus \{v\}$.

$Procedure\_all\_cuts(G, V, E, s, t)\{$
$S = \{s\}$
$C_s = \{< v, w >\in E : v \in S, w \in \Gamma^+(S)\}$
$MCsAlgorithm(S, C_s)\{$
$\forall v \in \Gamma^+(S) \setminus \{t\} \neq \varnothing$
$si(G = (\bar{S} \setminus \{v\}, E)\ est\ connexe\ \{$

$C_v = C_s \cup \{< v, w >\in E : w \in \Gamma^+(S)\} \cup \{< v, w >\in E : w \in \Gamma^+(v)\}$
$\setminus \{< w, v >\in E : w \in S\}$
$affiche r(C_v)$
$S = S \cup \{v\}$
$MCsAlgorithme(S, C_v)$

$\}//fin\_si$
$\}//fin\_MCsAlgorithm$
$\}//fin\_Procedure\_all\_cuts$

FIGURE 7.1 – Algorithme recursif pour énumérer toutes les coupes minimales d'un réseau

7.4 Résultats et interprétations

Après l'énumération de toutes les coupes minimales dans le réseau, la fiabilité du réseau peut être évaluée en fonction de ces coupes. L'évaluation de la fiabilité peut être effectuée en utilisant la méthode d'inclusion-exclusion détaillée dans la section 2.3.7.2 du chapitre 2. Supposons que l'ensemble des coupes qui séparent s de t sont $\{C_1, C_2, ..., C_k\}$ et soit F_i l'événement que tous les liens dans la coupe C_i soient en panne. Donc la fiabilité $R_{st}(G)$ en utilisant l'équation 2.11 du chapitre 2 est :

$$\begin{aligned} R_{st} &= 1 - P(F_1 \cup F_2 \cup ... \cup F_k) \\ &= 1 - \sum_{i=1}^{k} P(F_i) + \sum_{1\leq i<j} P(F_i F_j) - \sum_{i<j<l} P(F_i F_j F_l) \\ &\quad + ... - (-1)^{k+1} P(F_1 F_2 ... F_k) \end{aligned} \tag{7.8}$$

avec k le nombre de coupes dans le réseau.
La probabilité qu'un événement F_i se produit est équivalente à la probabilité que tous les liens de C_i soient en panne. En d'autre terme c'est la probabilité d'un système dont les éléments sont montés en parallèle donc la fiabilité d'une coupe en fonction des probabilités de succès des liens est :

$$r_i = 1 - \prod_{j=1}^{l} (1 - p_j) \tag{7.9}$$

avec p_j la probabilité qu'un lien j soit opérationnel et l la nombre de lien dans la coupe j.

L'algorithme "*McsAlgorithm*" est implémenté en langage Java sous Linux. Nous avons utilisé la bibliothèque JGraphT [1] qui est une bibliothèque [2] libre de classes écrites en langage Java.

Pour évaluer l'algorithme proposé nous avons utilisé 7 réseaux non orientés [17] comme illustré dans la figure 7.2. Ces réseaux sont choisi parmi 19 réseaux (orientés et non orientés ou mixtes) publiés dans la littérature et répertoriés par Soh et Rai dans [124] (repris aussi dans [125, 129, 134].

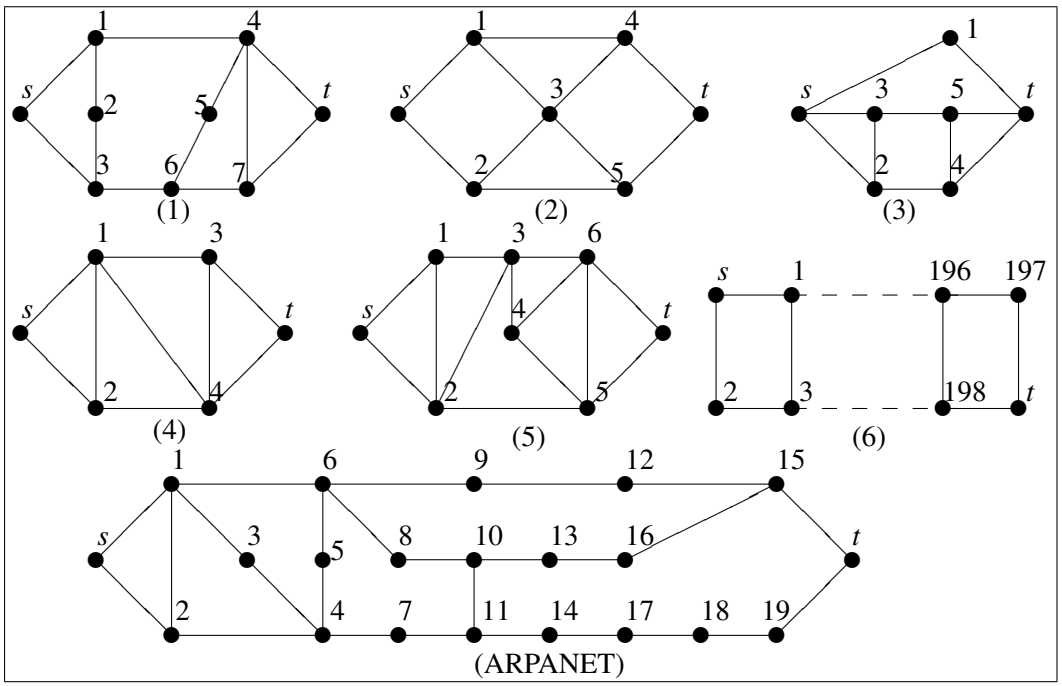


FIGURE 7.2 – Réseaux utilisés comme "benchmark"

Les probabilités de succès des liens des réseaux de (1) à (5) sont présentées ci-dessous, et pour les deux réseaux (6) et (ARPANET) elles sont fixées à 0.9.

1. http ://jgrapht.org/
2. JGraphT fournit un ensemble d'algorithmes et d'objets de la théorie des graphes orientés, non orientés ainsi que les graphes étiquetés.

		1	2	3	4	5	6	7	t
	s	0.97	–	0.9	–	–	–	–	–
	1		0.87	–	0.98	–	–	–	–
	2			0.86	–	–	–	–	–
(1)	3						0.67	–	–
	4					0.89	–	0.9	0.9
	5						0.78	–	–
	6							0.9	–
	7								0.57

		1	2	3	4	5	6	7	t
	s	0.78	0.98	–	–	–	–	–	–
	1			0.95	0.65	–	–	–	–
(2)	2			0.78	–	0.98	–	–	–
	3				0.69	0.61	–	–	–
	4							–	0.85
	5							–	0.9

		1	2	3	4	5	6	7	t
	s	0.9	0.87	0.56	–	–	–	–	–
	1					0.76	–	–	0.97
(3)	2			0.87	0.59	–	–	–	–
	3					0.98	–	–	–
	4					0.79	–	–	0.55
	5								0.69

		1	2	3	4	5	6	7	t
	s	0.6	0.7	–	–	–	–	–	–
	1		0.9	0.8	0.75	–	–	–	–
(4)	2				0.85	–	–	–	–
	3				0.95	–	–	–	0.7
	4								0.55

(5)

	1	2	3	4	5	6	7	t
s	0.87	0.9	–	–	–	–	–	–
1		0.88	0.68	–	–	–	–	–
2			0.9	–	0.94	–	–	–
3				0.87	–	0.98	–	–
4					0.59	0.64	–	–
5								0.88
6								0.9

En utilisant les équations 7.9 et 7.8 sous l'hypothèse que les liens tombent en panne indépendamment, on obtient les résultats du tableau 7.1 qui présente le nombre de coupes, la fiabilité, et le temps d'exécution comparé avec celui obtenu par l'algorithme *RM_BDD* [129] des différents réseaux du benchmark.

			Temps d'exécution/seconde	
Réseau	Nombre des MCs	Fiabilité R_{st}	MCsAlgorithm	*RM_BDD* [129]
1	26	0.943995	0	0
2	16	0.966967	0	0
3	19	0.983068	0	0
4	9	0.749283	0	0
5	19	0.965378	0	0
6	10000	0.304317	12.5	96.71
ARPANET	528	0.925807	0	-

TABLE 7.1 – Le nombre de coupes obtenu et la fiabilité des réseaux du benchmark

Le réseau (6) comporte 2^{99} [129] chemins minimaux et seulement 10000 coupes minimales, donc, il est préférable d'énumérer toutes les coupes minimales au lieu des chemins puisque leur nombre est assez large et prendra éventuellement énormément de temps.

Afin de vérifier que notre algorithme énumère toutes les coupes minimales sans duplication, nous avons implémenté l'algorithme *BAKTRACK#* [133], les deux produisent les mêmes résultats.

7.5 Conclusion

Dans ce chapitre nous avons présenté un nouveau algorithme efficace, qui permet d'énumérer toutes les coupes minimales dans un réseau non orienté. L'algorithme ainsi développé utilise l'idée de base qu'une coupe minimale peut être déduite d'une autre, sauf, celle qui sépare le nœud source s du reste du réseau. L'algorithme proposé est implémenté en langage Java en utilisant la bibliothèque JGraphT. Le principe d'inclusion-exclusion est utilisé ensuite pour calculer la fiabilité du réseau en fonction des coupes énumérées par l'algorithme

McsAlgorithm. L'algorithme est testé sur un ensemble de réseaux dans la littérature et a fait seulement 12.5 seconde pour énumérer 10000 coupes dans un réseau maillé de 200 nœuds.

Conclusion générale

7.6 Conclusion

Ce mémoire est consacré à la modélisation et la prédiction de la fiabilité des logiciels et des réseaux. La première partie du mémoire traite la fiabilité des logiciels en terme du nombre de défaillances résiduelles dans le logiciel en considérant ses défaillances observées. La deuxième partie est consacrée à la fiabilité des réseaux orientés ou non, en terme de l'ensemble des chemins minimaux et/ou des coupes minimales.

Concernant la fiabilité des logiciels, nous sommes arrivés à développer deux modèles de prédiction du nombre de défaillances dans le logiciel en utilisant les réseaux de neurones et un algorithme évolutionniste et/ou le recuit simulé. Le troisième modèle est basé sur le modèle d'auto-régression linéaire et les algorithme évolutionniste. Tous les modèles ont été testés avec des données de fiabilité de trois projets proposés dans la littérature parmi 16 projets disponibles. L'utilisation d'un algorithme évolutionniste (AE) et du recuit simulé dans l'apprentissage du réseau de neurones donnent des résultats avec des taux d'erreurs très minimales par rapport à ceux obtenus par l'algorithme de rétro-propagation. L'utilisation du recuit simulé (RS) dans l'apprentissage du réseau de neurones donne des résultats intéressants par rapport à l'apprentissage à l'aide de l'algorithme évolutionniste. En effet, pour le RS on essaie d'améliorer une seule solution à la fois contrairement à l'AE qui essaie d'améliorer une population de solutions qui peut être coûteux en temps de calcul et d'exécution.

Entraîné à l'aide d'un algorithme évolutionniste, le modèle linéaire d'auto-régression donne de très bons résultats, qui sont proches de ceux obtenus par le réseau de neurones entraîné par le même algorithme, avec une différence de $9.564\text{x}10^{-6}$ (projet#40), $7.24\text{x}10^{-6}$ (projet#40) et $2.658\text{x}10^{-6}$ (projet#SS1C). Ceci réaffirme la validité et la capacité de ce modèle contrairement à la conclusion avancée par S. Aljahdali [116]. La phase d'apprentissage du réseau de neurones est la clé de réussite du modèle. La méthode de rétro-propagation généralement utilisée pour apporter une réponse présente une faiblesse nette dés que la fonction objectif à optimiser est non régulière. L'utilisation d'un algorithme évolutionniste et/ou du recuit simulé, nous a permis d'une part de contourner les optimums locaux et d'autre part de pouvoir utiliser des fonctions objectifs non régulières.

La linéarité du modèle d'auto-régression peut être interprétée comme une caractéristique simpliste du modèle et non pas une contrainte négative du moment qu'il produit des résultats

satisfaisantes à condition que son entraînement soit réalisé par un algorithme évolutionniste. Un jeu de tests sur les projets précédents a été effectué en changeant l'ordre du modèle (4, 7, 10) qui s'avérait performant pour l'ordre 4 pour tous les projets.

Les approches ainsi développées peuvent être intégrer dans des boîtes à outils qui serviront les ingénieurs de fiabilité, les développeurs et les testeurs dans les processus du cycle de vie d'un logiciel. En indiquant le nombre de défaillances cumulées éventuellement résiduelles dans le logiciel, le nombre d'erreur à corriger et le nombre de journée de travail à envisager pour arriver à produire des logiciels fiables seront aisément déterminés.

En ce qui concerne la fiabilité des réseaux nous avons développé deux algorithmes efficaces qui permettent d'évaluer la fiabilité des réseaux que se soient orientés ou non avec la défaillance des liens :

- Un algorithme évolutionniste qui permet d'énumérer tous les chemins minimaux dans un réseau orienté, cet algorithme utilise le principe des algorithmes évolutionnistes dans la recherche des chemins minimaux. L'algorithme ainsi développé est testé avec des réseaux utilisés dans la littérature et donne les mêmes résultats que d'autres algorithmes. L'avantage de cet algorithme est qu'il ne pose aucune contrainte sur le réseau c-à-d qu'il peut être utilisé avec des réseaux orientés avec ou sans cycle.
- Un algorithme récursif qui permet d'énumérer toutes les coupes minimales dans un réseau non orienté, l'algorithme ainsi développé utilise l'idée que chaque coupe minimale peut être déduite d'une autre coupe déjà calculée sans parcourir de nouveau le réseau à la recherche de la nouvelle coupe. L'algorithme est testé avec succès sur des réseaux proposés dans la littérature.

Les deux algorithmes ont été combinés avec la méthode d'inclusion-exclusion pour évaluer la fiabilité (s-t deux terminaux) du réseau dont l'architecture est disponible. Ils peuvent être utilisés avec n'importe quel système modélisé sous forme de réseau.

7.7 Perspectives

Les modèles proposés pour la prédiction du nombre de défaillances résiduelles dans un logiciel peuvent être étendues à prédire les temps inter-défaillances pour calculer la fiabilité en fonction du temps moyen de bon fonctionnement (MTBF).

Puisque le réseau de neurones entraîné par l'algorithme évolutionniste ou le recuit simulé donnent des résultats très satisfaisants, ce modèle hybride peut être revue et étendu pour qu'il soit une approche boîte blanche pour évaluer la fiabilité d'un système logiciel en fonction de celle de ses composants.

Les algorithmes développés pour l'évaluation de la fiabilité des réseaux s'intéressent principalement aux réseaux à deux terminaux $s-t$. L'une des perspective est d'étendre ces algorithmes pour l'évaluation de la fiabilité des réseaux $k-terminaux$ et tous-terminaux. Dans ce travail nous nous sommes intéressé principalement à la fiabilité en fonction des liens, mais, dans un système qui peut être modélisé sous forme d'un réseau les nœuds tant les liens peuvent être sujets de défaillances. Le meilleur que l'on peut espérer c'est un modèle qui permet d'évaluer la fiabilité en prenant en considération tous les composants (liens et nœuds) du système.

Un système est en générale composé d'une partie logicielle et matérielle, la fiabilité du système intégré (logiciel et matériel) est influencée à la fois par la partie matérielle et la partie logicielle. Un modèle qui permettra d'évaluer la fiabilité d'un système en prenant en considération la fiabilité des composants (liens et nœuds) et la fiabilité du logiciel est mieux souhaitable qu'un modèle qui évalue la fiabilité de chaque partie indépendamment.

Données de fiabilité

.1 Les données de fiabilité du projet 1 (Real Time Command & Control)

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
1	3	1	34	8	46
2	30	2	35	227	46
3	113	9	36	65	46
4	81	10	37	176	46
5	115	11	38	58	46
6	9	11	39	457	47
7	2	17	40	300	47
8	91	20	41	97	47
9	112	20	42	263	47
10	15	20	43	452	53
11	138	20	44	255	53
12	50	20	45	197	54
13	77	20	46	193	54
14	24	20	47	6	54
15	108	20	48	79	54
16	88	20	49	816	56
17	670	30	50	1351	56
18	120	30	51	148	56
19	26	30	52	21	57
20	114	30	53	233	57
21	325	30	54	134	57
22	55	30	55	357	57
23	242	31	56	193	59
24	68	31	57	236	59
25	422	31	58	31	59
26	180	32	59	369	59
27	10	32	60	748	59
28	1146	33	61	0	59
29	600	34	62	232	59
30	15	42	63	330	59
31	36	42	64	365	61
32	4	46	65	1222	62
33	0	46	66	543	63
67	10	63	102	143	74
68	16	63	103	108	74

Suite page suivante . . .

suite de la page précédente

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
69	529	64	104	0	74
70	379	64	105	3110	75
71	44	64	106	1247	76
72	129	64	107	943	76
73	810	64	108	700	76
74	290	64	109	875	77
75	300	64	110	245	77
76	529	65	111	729	77
77	281	65	112	1897	78
78	160	65	113	447	79
79	828	66	114	386	79
80	1011	66	115	446	79
81	445	66	116	122	79
82	296	66	117	990	79
83	1755	67	118	948	80
84	1064	67	119	1082	80
85	1783	68	120	22	80
86	860	68	121	75	80
87	983	68	122	482	80
88	707	69	123	5509	81
89	33	69	124	100	81
90	868	69	125	10	81
91	724	69	126	1071	82
92	2323	70	127	371	83
93	2930	71	128	790	83
94	1461	72	129	6150	83
95	843	72	130	3321	83
96	12	72	131	1045	84
97	261	72	132	648	84
98	1800	73	133	5485	87
99	865	73	134	1160	87
100	1435	74	135	1864	88
101	30	74	136	4116	92

.2 Les données de fiabilité du projet 40 (Military)

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
1	320	1	8	113540	37
2	14390	2	9	112137	41
3	9000	9	10	660	42
4	2880	21	11	2700	43
5	5700	24	12	28793	43
6	21800	27	13	2173	44
7	26800	27	14	7263	44
15	10865	44	60	55794	129
16	4230	45	61	42632	133
17	8460	48	62	267600	135
18	14805	49	63	87074	136

Suite page suivante . . .

suite de la page précédente

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
19	11844	50	64	149606	141
20	5361	51	65	14400	141
21	6553	51	66	34560	141
22	6499	56	67	39600	141
23	3124	56	68	334395	148
24	51323	58	69	296015	148
25	17010	58	70	177399	149
26	1890	64	71	214622	150
27	5400	69	72	156400	154
28	62312	76	73	166800	156
29	24826	76	74	10800	156
30	26335	79	75	267000	159
31	363	79	76	2098833	183
32	13989	85	77	614080	190
33	15058	85	78	7680	190
34	32377	85	79	2629667	220
35	41632	85	80	2948700	254
36	4160	87	81	187200	258
37	82040	91	82	18000	258
38	13189	91	83	178200	260
39	3426	97	84	487800	266
40	5833	99	85	639200	275
41	640	100	86	334560	279
42	640	100	87	1468800	296
43	2880	102	88	86720	297
44	110	104	89	199200	300
45	22080	104	90	215200	303
46	60654	106	91	86400	304
47	52163	112	92	88640	305
48	12546	114	93	1814400	326
49	784	114	94	4160	326
50	10193	114	95	3200	326
51	7841	114	96	199200	329
52	31365	114	97	356160	333
53	24313	115	98	518400	339
54	298890	125	99	345600	343
55	1280	125	100	31360	343
56	22099	127	101	265600	347
57	19150	127			
58	2611	127			
59	39170	127			

.3 Les données de fiabilité du projet SS1C (Operating System)

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
1	5400	1	46	146640	44
2	81060	2	47	35760	45
3	2100	2	48	16080	45
4	13620	2	49	900	45
5	22800	2	50	10320	45
6	13380	3	51	2280	45
7	4620	3	52	28800	45
8	5100	3	53	75960	46
9	26640	3	54	52140	47
10	45960	4	55	28920	48
11	5100	4	56	52920	50
12	28800	4	57	240	50
13	58620	5	58	600	50
14	1920	5	59	13080	51
15	99540	6	60	58980	52
16	60000	7	61	1140	52
17	3000	7	62	62160	53
18	34500	7	63	34620	54
19	52500	8	64	118140	57
20	2280	8	65	262980	60
21	32100	8	66	47400	61
22	167940	11	67	1320	61
23	42840	11	68	41880	62
24	145920	14	69	39300	62
25	43380	15	70	115860	64
26	43380	15	71	66960	65
27	26820	15	72	178740	69
28	286500	20	73	2340	69
29	27000	20	74	6780	69
30	44400	21	75	780	69
31	110220	22	76	42840	69
32	269100	26	77	55500	70
33	60840	27	78	890340	84
34	85800	28	79	1620	84
35	98820	30	80	73560	85
36	61260	32	81	284407	91
37	156300	34	82	85140	93
38	52740	34	83	177360	95
39	54960	35	84	45360	96
40	32100	35	85	44400	96
41	129660	38	86	270472	101
42	116760	39	87	172140	103
43	75000	40	88	960	103
44	37260	41	89	23700	103
45	24180	41	90	190020	107

Suite page suivante . . .

suite de la page précédente

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
91	55260	107	139	86880	147
92	48660	108	140	130980	148
93	264900	112	141	172140	150
94	1080	112	142	144660	152
95	86700	113	143	12180	152
96	60	113	144	66960	153
97	1	113	145	155820	155
98	337920	118	146	5100	155
99	31320	119	147	242820	158
100	245760	122	148	33840	158
101	36480	123	149	34140	159
102	187260	125	150	479968	165
103	28980	126	151	31260	165
104	120	126	152	23520	165
105	119340	127	153	57720	166
106	1740	127	154	454200	173
107	840	127	155	36060	173
108	1500	127	156	3780	173
109	1200	127	157	226440	177
110	2820	127	158	73560	178
111	51960	130	159	180	178
112	8940	130	160	20760	178
113	1800	130	161	10340	178
114	1680	130	162	45660	179
115	7980	130	163	117720	180
116	420	130	164	10860	180
117	11460	130	165	71280	181
118	25440	131	166	83460	183
119	99360	133	167	79560	184
120	45360	133	168	319260	190
121	62040	134	169	219328	194
122	24360	134	170	38940	195
123	3420	134	171	19200	195
124	2220	134	172	6900	195
125	41460	136	173	101280	196
126	128040	137	174	174180	199
127	134340	139	175	80040	200
128	16800	139	176	188280	203
129	5760	139	177	2880	203
130	41760	140	178	43260	203
131	540	140	179	494640	210
132	540	140	180	38700	211
133	600	140	181	53520	212
134	420	140	182	100560	213
135	540	140	183	207840	216
136	305040	144	184	810840	227
137	102480	145	185	48720	227
138	60840	146	186	130920	229

Suite page suivante . . .

suite de la page précédente

Failure	Failure Interval Length	Day of Failure	Failure	Failure Interval Length	Day of Failure
187	280260	234	233	480	361
188	23160	234	234	87960	362
189	25020	234	235	228360	366
190	501060	242	236	100020	368
191	175476	246	237	163260	371
192	659392	255	238	66420	372
193	790440	268	239	255840	378
194	107400	270	240	13080	378
195	625140	282	241	4740	378
196	507300	290	242	1980	378
197	1016040	309	243	430800	385
198	1080	309	244	12660	385
199	30540	310	245	40020	386
200	56160	312	246	16680	386
201	224520	316	247	1080	386
202	12120	316	248	2100	386
203	13080	316	249	1020	386
204	142140	320	250	13080	386
205	510780	330	251	842220	400
206	181500	332	252	805176	412
207	131580	334	253	18780	412
208	6180	334	254	47100	413
209	120	334	255	110520	415
210	228900	338	256	840	415
211	4320	338	257	266760	418
212	9300	338	258	33480	419
213	75720	339	259	38520	420
214	13080	340	260	70800	421
215	118020	342	261	119100	423
216	33900	343	262	462600	431
217	29940	344	263	17400	431
218	86400	345	264	543060	441
219	118980	349	265	191040	445
220	8400	349	266	3720	445
221	33000	350	267	103080	446
222	43020	350	268	117360	449
223	118620	353	269	63960	450
224	137940	356	270	3960	450
225	287160	361	271	442860	457
226	720	361	272	22260	458
227	660	361	273	241500	461
228	2460	361	274	49920	462
229	2760	361	275	144240	465
230	8880	361	276	37260	467
231	420	361	277	122580	470
232	840	361			

Bibliographie

[1] S. H. Aljahdali, A. Sheta, and D. Rine. Prediction of software reliability : a comparison between regression and neural network non-parametric models. In *ACS/IEEE Int Computer Systems and Applications Conf.*, pages 470–473 (2001). (Cité en pages xix, 45, 46 et 104.)

[2] Hoang Pham. *System Software Reliability (Springer Series in Reliability Engineering)*. Springer (2 Nov 2006). (Cité en pages 21, 23, 36, 38, 42, 43, 44 et 45.)

[3] O. Gaudoin and J. Ledoux. *Modélisation aléatoire en fiabilité des logiciels*. Hermès Science Publications (2007). (Cité en pages 22, 36 et 41.)

[4] Richard Bromley Patrick D. T. O'Connor, David Newton. *Practical Reliability Engineering*. John Wiley & Sons (2002). (Cité en pages 24 et 35.)

[5] Richard Moss Ireson W.G., Clyde Coombs. *Handbook of Reliability Engineering and Management*. McGraw-Hill Professional (1995). (Cité en page 24.)

[6] Joe Palma, Jeff Tian, and Peng Lu. Collecting data for software reliability analysis and modeling. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research : software engineering - Volume 1*, CASCON '93, pages 483–494. IBM Press (1993). (Cité en page 25.)

[7] Okumoto K Musa JD, lannino A. *Software Reliability : Measurement, Prediction, and Application*. McGraw-Hill, NewYork (1987). (Cité en pages 25 et 45.)

[8] N.E. Rallis and Z.F. Lansdowne. *Reliability estimation for a software system with sequential independent reviews*. IEEE Transactions on Software Engineering **27**, 1057–1061 (2001). (Cité en page 25.)

[9] E.F. Moore and C.E. Shannon. *Reliable circuits using less reliable relays*. Journal of the Franklin Institute **262**(3), 191 – 208 (1956). (Cité en page 26.)

[10] F.T. Boesch. *Synthesis of reliable networks - a survey*. IEEE Trans. Reliability **35**, 240–246 (August 1986). (Cité en pages 26, 55 et 59.)

[11] C.J. Colbourn. *The combinatorics of network reliability*. International series of monographs on computer science. Oxford University Press (1987). (Cité en pages 26 et 124.)

[12] M. Benaddy, M. Wakrim, and S. Aljahdali. Evolutionary neural network prediction for cumulative failure modeling. In *Proc. IEEE/ACS Int. Conf. Computer Systems and Applications AICCSA 2009*, pages 179–184 (2009). (Cité en pages 27, 28, 45, 96 et 102.)

[13] M. Benaddy, S. Aljahdali, and M. Wakrim. Evolutionary prediction for cumulative failure modeling : A comparative study. In *Proc. Eighth Int Information Technology : New Generations (ITNG) Conf*, pages 41–47 (2011). (Cité en pages 27, 28, 45, 46, 96, 98 et 102.)

[14] M. Benaddy, M. Wakrim, and S. Aljahdali. Evolutionary regression prediction for software cumulative failure modeling : A comparative study. In *Proc. Int. Conf. Multimedia Computing and Systems ICMCS '09*, pages 286–292 (2009). (Cité en pages 27, 28, 46, 96, 98 et 102.)

[15] Mohamed Benaddy and Mohamed Wakrim. *Simulated annealing neural network for software failure prediction*. International Journal of Software Engineering and Its Applications **6. No. 4**, 35–46 (October 2012). (Cité en page 28.)

[16] Wakrim M. Benaddy, M. Enumération des chemins minimaux dans un réseau. In *Conférence Méditeranéenne de L'ingénierie sûre des systèmes complexes (MISC'11)* (2011). (Cité en pages 28 et 124.)

[17] Mohamed Benaddy and Mohamed Wakrim. *Cutset enumerating and network reliability computing by a new recursive algorithm and inclusion exclusion principle*. International Journal of Computer Applications **45**(16), 22–25 May (2012). Published by Foundation of Computer Science, New York, USA. (Cité en pages 28, 134, 138 et 140.)

[18] M.R. Lyu. *Handbook of Relaibility Engineering*. IEEE Computer Society Press and McGraw Hill (1996). (Cité en pages 34, 42, 45, 66, 68 et 69.)

[19] Olivier Gaudoin. *Modèles stochastiques et statistiques pour la fiabilité des systèmes*. Thèse de Doctorat, Institut national polytechnique de Grenoble (2002). (Cité en pages 35 et 36.)

[20] Frédérique VALLÉE. *Fiabilité des logiciels*. Techniques de l'ingénieur **Référence SE2520** (10 oct. 2004). (Cité en page 37.)

[21] Hoang Pham. *Software Reliability*. Springer (2000). (Cité en pages 37, 38, 40 et 42.)

[22] Printz J. *Productivité des programmeurs*. Hermès-Lavoisier (2001). (Cité en page 38.)

[23] Laprie J.C. ed. *Guide de la sûreté de fonctionnement 1996.* Cepaduès (1996). (Cité en page 38.)

[24] J.C. Laprie. *Dependability : basic concepts and terminology in English, French, German, Italian, and Japanese*. Dependable computing and fault-tolerant systems. Springer-Verlag (1992). (Cité en page 38.)

[25] Habrias H. *La mesure du logiciel*. Teknea (1994). (Cité en page 39.)

[26] VALLÉE F. ET VERNOS D. Comment utiliser la fiabilité du logiciel comme critère d'arrêt du test. In *13éme Colloque Européen de Sûreté de Fonctionnement* (ESREL 2002). (Cité en page 40.)

[27] MARTIN L. SHOOMAN. *Reliability of Computer Systems and Networks : Fault Tolerance, Analysis, and Design*. Wiley-Interscience, Jhon Wiley & Sons Inc. (2002). (Cité en page 41.)

[28] ALAN WOOD. Sftware relaibility growth models. Technical report TANDEM (1996). (Cité en page 41.)

[29] GOEL AL. A guidebook for software reliability assessment. Technical report Data and Analysis Centre for Software, Rome Air Development Centre (RADC), Rome, New York, Tech. Report RADCTR- 83-176 (1983). (Cité en page 42.)

[30] OKUMOTO K. MUSA J., IANNINO A. *Software Reliability : Measurement, Prediction, Application*. Mc Graw-Hill Book Company (1987). (Cité en page 42.)

[31] A.L. GOEL. *Software reliability models : Assumptions, limitations, and applicability*. Software Engineering, IEEE Transactions on **SE-11**(12), 1411 – 1423 dec. (1985). (Cité en page 42.)

[32] MILLS H.D. On the statistical validation of computer programs. Technical report IBM FSD (1970). (Cité en page 43.)

[33] CAI K-Y. *On estimating the number of defects remaining in software*. Journal of Systems and Software **40(l)** (1998). (Cité en page 43.)

[34] OHBA M JACOBY R TOHMA Y, YAMANO H. *The estimation of parameters of the hypergeometric distribution and its application to the software reliability growth model*. IEEE Trans. Software Engineering **SE-17(5)** (1991). (Cité en page 43.)

[35] MORANDA PB JELINSKI Z. *Software reliability research*. Statistical Computer Performance Evaluation, W. Freiberger (ed), Academic Press, New York (1972). (Cité en page 43.)

[36] WOLVERTON RW SCHICK GJ. *An analysis of competing software reliability models*. IEEE Trans. Software Engineering **SE- 4(2)** (1978). (Cité en page 43.)

[37] MORANDA PB. *An error detection model for application during software development*. IEEE Trans. Reliability **R-28(5)** (1979). (Cité en page 43.)

[38] LITTLEWOOD B. *Software reliability model for modular program structure*. IEEE Trans. Reliability **R-28(3)** (1979). (Cité en pages 43 et 44.)

[39] A.N. SUKERT. An investigation of software reliability models. In *Annual Reliability & Maintainability Symposium, IEEE Reliability Society, Piscataway* (1977). (Cité en page 43.)

[40] Okumoto K Goel AL. *Time-dependent error-detection rate model for software and other performance measures*. IEEE Transaction on Reliability **R-28**, 206–211 (1979). (Cité en pages 43 et 45.)

[41] Coutinho JS. Software reliability growth. In IEEE Computer Society Press, editor, *International Conference on Reliable Software*, Los Angeles (1973). IEEE Computer Society Press. (Cité en page 44.)

[42] Ferguson PA Wall JK. *Pragmatic software reliability prediction*. Proc. Annual Reliability & Maintainability Symposium, IEEE Reliability Society, Piscataway (1977). (Cité en page 44.)

[43] Okumoto K Goel AL. A markovian model for reliability and other performance measures of software systems. In *COMPCON, IEEE Computer Society Press, Los Angeles* (1979). (Cité en page 44.)

[44] Yamada S Tokuno K. *Markovian availability measurement and assessment for hardware-software systems*. Int. J Reliability, Quality and Safety Engineering **4(3)** (1997). (Cité en page 44.)

[45] Goel AL. Software reliability modeling and estimation techniques. Technical report RADCTR-82-263 (October 1982). (Cité en page 45.)

[46] Goel AL. Guidebook for software reliability assessment. Technical report RADC-TR-83-176 (August 1982). (Cité en page 45.)

[47] Musa DJ. *A theory of software reliability and its application*. IEEE Trans. Software Engineering **SE-1**, 312–327 (1971). (Cité en page 45.)

[48] Yamada S Ohba M. S-shaped software reliability growth models. In *4th Int.Conf. Reliability and Maintainability*, pages 430–436 (1984). (Cité en page 45.)

[49] Osaki S Yamada S, Ohba M. *S-shaped reliability growth modeling for software error detection*. IEEE Transactions on Reliability **12**, 475–484 (1983). (Cité en page 45.)

[50] Osaki S Yamada S, Ohba M. *S-shaped software reliability growth models and their applications vol*. IEEE Trans. Reliability **R-33** (October 1984). (Cité en page 45.)

[51] Osaki S Yamada S. *Software reliability growth modeling : models and applications*. IEEE Transactions on Software Engineering **11**, 1431–1437 (1985). (Cité en page 45.)

[52] Nachimuthu Karunanithi, Darrell Whitley, and Yashwant K. Malaiya. *Using neural networks in reliability prediction*. IEEE Softw. **9**(4), 53–59 July (1992). (Cité en page 45.)

[53] W.A. Adnan and M.H. Yaacob. An integrated neural-fuzzy system of software reliability prediction. In *Software Testing, Reliability and Quality Assurance, 1994.*

Conference Proceedings., First International Conference on, pages 154 –158 dec (1994). (Cité en page 45.)

[54] W.A. Adnan, M. Yaakob, R. Anas, and M.R. Tamjis. Artificial neural network for software reliability assessment. , **3**, pages 446 –451 vol.3 (2000). (Cité en page 45.)

[55] S.L. Ho, M. Xie, and T.N. Goh. *A study of the connectionist models for software reliability prediction*. Computers &amp ; Mathematics with Applications **46**(7), 1037 – 1045 (2003). <ce :title>Applied Stochastic System Modelling</ce :title>. (Cité en page 45.)

[56] Lee S.U. Park J.H. Park, J.Y. *Neural network modeling for software reliability prediction from failure time data*. Journal of Electical Engineering and information Science **4 No.4**, 431–562 (1999). (Cité en pages 45 et 96.)

[57] R. Sitte. *Comparison of software-reliability-growth predictions : neural networks vs parametric-recalibration*. IEEE Transactions on Reliability **48 :3**, 285–291 (Sep 1999). (Cité en page 45.)

[58] John D. Musa. https ://www.thedacs.com/databases/sled/swrel.php. Technical report Software Reliability Data, Data & Analysis Center for Software (1980). (Cité en pages 46 et 102.)

[59] H. Frank. Maximally survival node vulnerable networks. Technical report Memorandum for File, Div. Emergency preparedness of Office of the President, Washington D.C. (March 1969). (Cité en page 53.)

[60] H. Frank. Some new results in the design of survivable networks. In *Proceedings of 12th Annual Midwest Circuit Theory Symposium*, pages I3.1–I3.8 (Septembre 1969). (Cité en page 53.)

[61] A. Suffel C. Sutner K. Colbourn, C. Satyanarayana. *Computing the residual node connectedness reliability problem*. SIAM J. Computing **20**, 149–155 (1991). (Cité en page 54.)

[62] A. Suffel C. Sutner K. Colbourn, C. Satyanarayana. *On residual connectedness network reliability*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science **5**, 51–59 (1991). (Cité en page 54.)

[63] Cheung K. Page L. Perry J. Myrvold, W. *Uniformly most reliable graphs do not always exist*. Networks **21**, 417–419 (1991). (Cité en page 54.)

[64] F.S. Roberts, F. Hwang, C.L. Monma, DIMACS (Group), American Mathematical Society, and Association for Computing Machinery. *Reliability of Computer and Communication Networks : Proceedings of a Dimacs Workshop, December 2-4, 1989*. DIMACS series in discrete mathematics and theoretical computer science. American Mathematical Society (1991). (Cité en page 54.)

[65] Siegrist K. Slater P. Amin, A. *On the nonexistence if uniformly optimal graphs for pair-connected reliability*. Networks **21**, 359–368 (1991). (Cité en page 55.)

[66] K. Slater Amin, A. Siegrist. *On uniformly optimally reliable graphs for pair-connected reliability with vertex failures*. Networks **23**, 185–193 (1993). (Cité en page 55.)

[67] H.M. AboElFotoh and C.J. Colbourn. *Computing 2-terminal reliability for radio-broadcast networks*. Reliability, IEEE Transactions on **38**(5), 538–555 dec (1989). (Cité en page 55.)

[68] Colbourn C. J. AboElFotoh, H.M. *Efficient algorithms for computing the reliability of permutation and interval graphs*. Networks **20**(7), 883–898 (1990). (Cité en page 55.)

[69] K. B. Misra and T. S. M. Rao. *Reliability analysis of redundant networks using flow graphs*. Reliability, IEEE Transactions on **R-19**(1), 19 –24 feb. (1970). (Cité en page 56.)

[70] Young H. Kim, Kenneth E. Case, and P. M. Ghare. *A method for computing complex system reliability*. Reliability, IEEE Transactions on **R-21**(4), 215 –219 nov. (1972). (Cité en page 56.)

[71] Gupta J.S. Misra K.B. Aggarwal, K. *A new method for system reliability evaluation*. Microelectronics Reliability **12**(5), 435 – 440 (1973). (Cité en page 56.)

[72] Gupta J. Misra K. Aggarwal, K. *A simple method for reliability evaluation of a communication system*. Communications, IEEE Transactions on **23**(5), 563 – 566 may (1975). (Cité en page 56.)

[73] HARIRI S. RAGHAVENDRA C. S. KUMAR, P. V. K. *Distributed program reliability analysis*. IEEE Transactions on Software Engineering **SE-12**, 42–50 (Jan. 1986). (Cité en page 56.)

[74] Y.B. Yoo and N. Deo. *A comparison of algorithms for terminal-pair reliability*. Reliability, IEEE Transactions on **37**(2), 210 –215 jun (1988). (Cité en page 56.)

[75] D. Torrieri. *Calculation of node-pair reliability in large networks with unreliable nodes*. Reliability, IEEE Transactions on **43**(3), 375 –377, 382 sep (1994). (Cité en pages 56 et 58.)

[76] V.A. Netes and B.P. Filin. *Consideration of node failures in network-reliability calculation*. Reliability, IEEE Transactions on **45**(1), 127 –128 mar (1996). (Cité en page 56.)

[77] Wei-Jenn Ke and Sheng-De Wang. *Reliability evaluation for distributed computing networks with imperfect nodes*. Reliability, IEEE Transactions on **46**(3), 342 –349 sep (1997). (Cité en pages 56 et 58.)

[78] E.F. Moore and Shannon C.E. *Reliable circuits using less reliable relays*. Journal of the Franklin Institute **262**(3), 191 – 208 (1956). (Cité en page 60.)

[79] A. Satyanarayana and A. Prabhakar. *New topological formula and rapid algorithm for reliability analysis of complex networks*. Reliability, IEEE Transactions on **R-27**(2), 82 –100 june (1978). (Cité en page 62.)

[80] Mitchell O. Locks. *Recursive disjoint products : A review of three algorithms*. Reliability, IEEE Transactions on **R-31**(1), 33 –35 april (1982). (Cité en page 62.)

[81] Mitchell O. Locks. *A minimizing algorithm for sum of disjoint products*. Reliability, IEEE Transactions on **R-36**(4), 445 –453 oct. (1987). (Cité en page 62.)

[82] Martinez J.M. Samuelides M. Gordon M.B. Badran F. Thiria S. Hérault L. Dryfus, G. *Réseaux de Neurones Méthodologie et Applications*. Eyrolles, 2éme édition edition (2004). (Cité en page 66.)

[83] J.L. Elman. *Finding structure in time*. Cognitive Science **14**(2), 179–211 (1990). (Cité en page 69.)

[84] M. Jordan. Attractor dynamics and parallelism in connectionist sequential machine. In *18th annual Conference of the Cognitive Science*, pages 531–546 (1986). (Cité en page 69.)

[85] Zipser D. Williams, R. *A learning algorithm for continually running fully recurrent neural networks*. Neural Computation **1, no. 2**, 270–280 (1989). (Cité en page 69.)

[86] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. *Forecasting with artificial neural networks : The state of the art*. International Journal of Forecasting **14**(1), 35 – 62 (1998). (Cité en page 71.)

[87] Hinton G. Rumelhart, D. and R. Williams. *Learning internal representations by error propagation*. Parallel Distributed Processing **I, MIT Press**, 318–362 (1986). (Cité en page 71.)

[88] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag (1992). (Cité en pages 76 et 86.)

[89] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA (1992). (Cité en pages 76, 78, 80 et 83.)

[90] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition (1989). (Cité en pages 79, 80 et 83.)

[91] L. D. Davis. *Handbook Of Genetic Algorithms*. Van Nostrand Reinhold (1991). (Cité en pages 80 et 81.)

[92] D.B. Fogel, T. Bäck, and Z. Michalewicz. *Evolutionary Computation 1 : Basic Algorithms and Operators*. Evolutionary Computation. Institute of Physics Publishing (2000). (Cité en page 81.)

[93] K. A. De Jong. *Genetic algorithms are not function optimizers*. cs.gmu.edu (1993). (Cité en page 82.)

[94] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* Thèse de Doctorat, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, USA (1975). AAI7609381. (Cité en pages 83 et 84.)

[95] J.J. Grefenstette. *Optimization of control parameters for genetic algorithms*. Systems, Man and Cybernetics, IEEE Transactions on **16**(1), 122 –128 jan. (1986). (Cité en pages 83, 84 et 88.)

[96] T.C. Fogarty. An incremental genetic algorithm for real-time optimisation. In *Systems, Man and Cybernetics, 1989. Conference Proceedings., IEEE International Conference on*, pages 321 –326 vol.1 nov (1989). (Cité en page 83.)

[97] J H Holland. , **Ann Arbor**. University of Michigan Press (1975). (Cité en page 84.)

[98] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms : Part 1, fundamentals, (1993). (Cité en page 85.)

[99] Heinz Mühlenbein. Evolutionary algorithms : Theory and applications. In *Local Search in Combinatorial Optimization*. Wiley (1993). (Cité en page 86.)

[100] Cezary Z. Janikow and Zbigniew Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *ICGA*, pages 31–36 (1991). (Cité en page 86.)

[101] LOZANO M. VERDEGAY J.L. HERRERA, F. *Talking real-coded genetic algorithms : Operators and tools for behavioural analysis*. Artificial Intelligence Review **12**, 265–319 (1998). (Cité en pages 86 et 87.)

[102] Zbigniew Michalewicz. *Evolutionary computation techniques for nonlinear programming problems*. International Transactions in Operational Research **1**(2), 223–240 (1994). (Cité en page 86.)

[103] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. *Optimization by simulated annealing*. Science **220**(4598), pp. 671–680 (1983). (Cité en page 88.)

[104] V. Cerný. *Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm*. Journal of Optimization Theory and Applications **45**, 41–51 (1985). 10.1007/BF00940812. (Cité en page 88.)

[105] Marshall N. Rosenbluth Augusta H. Teller Nicholas Metropolis, Arianna W. Rosenbluth and Edward Teller. *Equation of state calculations by fast computing machines*. Journal of Chemical Physics **21**, 1087–1092 (1953). (Cité en page 88.)

[106] Pétrowski A. Siarry P. Taillard E. Dréo, J. *Métaheuristiques pour l'optimisation difficile*. Eyrolles (2005). (Cité en pages 88 et 89.)

[107] Draper D. Fouskakis, D. *Stochastic optimization : A review*. International Statistical Review / Revue Internationale de Statistiques **70**(3), 315–349 (Dec. 2002). (Cité en page 91.)

[108] Chen H. Zhang, Y. Predicting for mtbf failure data series of software reliability by genetic programming algorithm. In Bo Yang and Yuehui Chen, editors, *Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)*, pages 666–670, Jinan University, China 16-18 October (2006). IEEE Computer Society. (Cité en page 96.)

[109] M.R. Lyu. Software reliability engineering : A roadmap. In *Future of Software Engineering, 2007. FOSE '07*, pages 153 –170 may (2007). (Cité en page 96.)

[110] Kai-Yuan Cai, Chuan-Yuan Wen, and Ming-Lian Zhang. *A critical review on software reliability modeling*. Reliability Engineering &amp ; System Safety **32**(3), 357 – 371 (1991). (Cité en page 96.)

[111] Kai-Yuan Cai, Lin Cai, Wei-Dong Wang, Zhou-Yi Yu, and David Zhang. *On the neural network approach in software reliability modeling*. Journal of Systems and Software **58**(1), 47 – 62 (2001). (Cité en page 96.)

[112] N. Karunanithi, D. Whitley, and Y.K. Malaiya. *Prediction of software reliability using connectionist models*. Software Engineering, IEEE Transactions on **18**(7), 563 –574 jul (1992). (Cité en page 96.)

[113] H.K. Lam, S.H. Ling, F.H.F. Leung, and P.K.S. Tam. Tuning of the structure and parameters of neural network using an improved genetic algorithm. , **1**, pages 25 –30 vol.1 (2001). (Cité en page 96.)

[114] Liang Tian and Afzel Noore. *On-line prediction of software reliability using an evolutionary connectionist model*. Journal of Systems and Software **77**(2), 173 – 180 (2005). (Cité en page 96.)

[115] Liang Tian and Afzel Noore. *Evolutionary neural network modeling for software cumulative failure time prediction*. Reliability Engineering &amp ; System Safety **87**(1), 45 – 51 (2005). (Cité en page 96.)

[116] Sultan Hamadi Aljahdali. *Prediction of Software Reliability Using Neural Network and Fuzzy logic*. Thèse de Doctorat, George Mason University (2002). (Cité en pages 96, 97 et 145.)

[117] P. Camarda, F. Corsi, and A. Trentadue. *An efficient simple algorithm for fault tree automatic synthesis from the reliability graph*. Reliability, IEEE Transactions on **R-27**(3), 215 –221 aug. (1978). (Cité en page 124.)

[118] S.H. Ahmad. *Simple enumeration of minimal cutsets of acyclic directed graph*. Reliability, IEEE Transactions on **37**(5), 484 –487 dec (1988). (Cité en page 124.)

[119] S. Arunkumar and S.H. Lee. *Enumeration of all minimal cut-sets for a node pair in a graph*. Reliability, IEEE Transactions on **R-28**(1), 51 –55 april (1979). (Cité en page 124.)

[120] John E. Biegel. *Determination of tie sets and cut sets for a system without feedback*. Reliability, IEEE Transactions on **R-26**(1), 39 –42 april (1977). (Cité en page 124.)

[121] Suresh Rai and K.K. Aggarwal. *On complementation of pathsets and cutsets*. Reliability, IEEE Transactions on **R-29**(2), 139 –140 june (1980). (Cité en page 124.)

[122] G. B. Jasmon and O. S. Kai. *A new technique in minimal path and cutset evaluation*. Reliability, IEEE Transactions on **R-34**(2), 136–143 june (1985). (Cité en page 124.)

[123] G. B. Jasmon. *Cutset analysis of networks using basic minimal paths and network decomposition*. Reliability, IEEE Transactions on **R-34**(4), 303 –307 oct. (1985). (Cité en page 124.)

[124] S. Soh and S. Rai. Experimental results on preprocessing of path/cut terms in sum of disjoint products technique. In *INFOCOM '91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s., IEEE*, pages 533 –542 vol.2 apr (1991). (Cité en pages 129, 131 et 140.)

[125] Sy-Yen Kuo, Shyue-Kung Lu, and Fu-Min Yeh. *Determining terminal-pair reliability based on edge expansion diagrams using obdd*. Reliability, IEEE Transactions on **48**(3), 234 –246 sep (1999). (Cité en pages 129, 131 et 140.)

[126] Wei-Chang Yeh. *A simple algorithm for evaluating the k-out-of-n network reliability*. Reliability Engineering &amp ; System Safety **83**(1), 93 – 101 (2004). (Cité en page 134.)

[127] Nasser S. Fard and Tae-Han Lee. *Cutset enumeration of network systems with link and node failures*. Reliability Engineering &amp ; System Safety **65**(2), 141 – 146 (1999). (Cité en page 134.)

[128] Yung-Ruei Chang, Hung-Yau Lin, Ing-Yi Chen, and Sy-Yen Kuo. A cut-based algorithm for reliability analysis of terminal-pair network using obdd. In *Proceedings of the 27th Annual International Conference on Computer Software and Applications*, COMPSAC '03, pages 368–, Washington, DC, USA (2003). IEEE Computer Society. (Cité en page 134.)

[129] Hung-Yau Lin, Sy-Yen Kuo, and Fu-Min Yeh. *Minimal cutset enumeration and network reliability evaluation by recursive merge and bdd*. Computers and Communications, IEEE Symposium on **0**, 1341 (2003). (Cité en pages 134, 140 et 142.)

[130] Zhibin Tan. *Minimal cut sets of s-t networks with k-out-of-n nodes*. Reliability Engineering and System Safety **82**(1), 49 – 54 (2003). (Cité en pages 134 et 135.)

[131] Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. *Enumerating disjunctions and conjunctions of paths and cuts in reliability theory*. Discrete Appl. Math. **155**(2), 137–149 January (2007). (Cité en page 134.)

[132] D. R. Shier and D. E. Whited. *Algorithms for generating minimal cutsets by inversion*. Reliability, IEEE Transactions on **R-34**(4), 314 –319 oct. (1985). (Cité en page 134.)

[133] S. Tsukiyama, I. Shirakawa, H. Ozaki, and H. Ariyoshi. *An algorithm to enumerate all cutsets of a graph in linear time per cutset*. J. ACM **27**(4), 619–632 October (1980). (Cité en pages 134 et 142.)

[134] Wei-Chang Yeh. *A new algorithm for generating minimal cut sets in k-out-of-n networks*. Reliability Engineering & System Safety **91**(1), 36 – 43 (2006). (Cité en page 140.)