UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
AMADOU LAMINE TOP

VERS UNE REPRÉSENTATION MATRICIELLE DES MÉTRIQUES DE COUPLAGE LOGICIEL

JUIN 2022

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Fadel Toure, directeur de recherche
Département de mathématiques et d'informatique, UQTR

M. François Meunier, membre du jury
Département de mathématiques et d'informatique, UQTR

M. Mhamed Mesfioui, membre du jury
Département de mathématiques et d'informatique, UQTR

REMERCIEMENTS

J’exprime mes sincères remerciements à mon directeur de recherche M. Fadel TOURE et à tous les professeurs de mon département pour leurs soutiens, conseils et encouragements dans la mise en œuvre de ce travail de recherche. Merci d’avoir partagé votre expérience et ma reconnaissance envers vous sera pour toujours.

Pareillement, je remercie mon grand frère pour le soutien financier de mes études et ma famille pour leur assistance et prières.

RÉSUMÉ

Les logiciels informatiques sont devenus indispensables dans beaucoup de domaines d'activité. Leur assurance qualité reste encore un défi du fait de la croissance continue de leur taille et de leur complexité. Après l'étape de mise en production, des défaillances peuvent survenir et faire l'objet de révisions ou de modifications de code source. Ces défaillances peuvent être dues à des fautes dans le code source et peuvent entrainer des corrections majeures dépendamment de la sévérité des fautes. Le maintien d'un taux de fautes et sévérité minimal est la principale préoccupation des activités d'assurance qualité des logiciels. Pour cela, des recherches sont menées (entre autres) sur l'analyse du couplage entres les modules des systèmes logiciels orientés objet. Cette analyse se base sur les métriques orientées objet pour prévenir/détecter les fautes assez tôt dans le cycle de vie ou encore orienter les tests vers des composants critiques.

Dans les domaines de l'ingénierie logicielle, la métrologie est un moyen objectif et fiable pour l'audit de la qualité d'un logiciel. Les métriques orientées objet ont été utilisées dans beaucoup de travaux dans la littérature. Leur nombre important dans les systèmes orientés objet engendre une masse d'informations qui est difficile à interpréter. Ces métriques sont utilisées pour capturer certains attributs internes et externes de la qualité logicielle comme le couplage que nous analyserons particulièrement dans ce mémoire.

L'objectif est de développer une nouvelle représentation intégrée des métriques logicielles qui capture la structure des relations entre les classes. Pour se faire, les métriques clés de code source liées à la conception orientée objet sont représentées sous forme matricielle. Cette matrice d'information est analysée grâce aux techniques d'apprentissage profond et nous permet dans le cadre d'application de ce mémoire, de détecter des patterns logiciels.

Les résultats de cette première application du modèle matricielle ont prouvé que cette représentation permet d'identifier certains patterns de conception orientés objet dans les codes sources logiciels.

ABSTRACT

Computer software has become indispensable in many fields of activity. Their quality assurance remains a challenge due to the continuous growth of their size and complexity. After the release stage, failures may occur and software may be subject to revisions or changes to the source code. These failures may be due to errors in the source code and may result in major corrections depending on the severity of the errors. Maintaining a minimum fault rate and severity is the main concern of software quality assurance activities. To this end, research is carried out (among other things) on the analysis of the coupling between the modules of object-oriented software systems.

To this end, research is carried out (among other things) on the analysis of the coupling between the modules of object-oriented software systems. This analysis is based on object-oriented metrics to prevent/detect faults early in the life cycle or to direct tests to critical components. In the fields of software engineering, metrology is an objective and reliable way to audit the quality of software. Object-oriented metrics have been used in much work in the literature. Their large number in object-oriented systems generates a mass of information that is difficult to interpret. These metrics are used to capture some internal and external attributes of software quality such as coupling that we will analyze particularly in this memory.

The goal is to develop a new integrated representation of software metrics that captures the structure of relationships between classes. To do this, key source code metrics related to object-oriented design are represented in matrix form.

This information matrix is analyzed using deep learning techniques and allows us to detect software patterns in the context of the application of this memory. The results of this first application of the raster model proved that this representation makes it possible to identify certain patterns of object-oriented designs in software source codes.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1. INTRODUCTION

Les logiciels développés sont de plus en plus complexes et traitent de grandes masses d'informations. Les exigences de qualité qui en découlent sont devenues élevées.

Pour gérer efficacement la qualité, il faut pouvoir la mesurer. Dans les systèmes logiciels de grandes tailles, la mesure devient une activité complexe et difficile. Dans le paradigme orientée objet, le développement des logiciels de qualité a connu des évolutions menant à plusieurs techniques de standardisation de la programmation. Plusieurs métriques ont été proposées sur ce paradigme pour mesurer (entre autres) les multiples dépendances présentées dans les composants. Les métriques les plus connues ont été introduites par Chidamber et Kemerer [1]. Les auteurs affirment que ces mesures peuvent aider les développeurs à comprendre la complexité de la conception orientée objet. Ces métriques sont de plus en plus utilisées par l'industrie du logiciel. Elles évaluent des attributs de qualité internes des logiciels comme l'héritage, le couplage, la taille, la complexité ainsi que d'autres attributs. Les résultats des investigations menées ont montré la capacité de mesure de certains attributs externes dont la maintenabilité (effort de maintenance) et la réutilisabilité des logiciels. Les résultats d'autres métriques restent discutables.

1.1 - Problématique

En génie logiciel, beaucoup de chercheurs et de développeurs se sont appliqués à définir des techniques pour améliorer le processus développement du logiciel dans le but d'obtenir des produits de meilleure qualité à un coût raisonnable. Des méthodologies, des techniques et des outils ont été proposés pour faciliter cette production. La programmation orientée objet fait partie d'une de ces techniques et renferme plusieurs concepts qui ont permis de mieux contrôler la conception des logiciels complexes. L'attribut de couplage entre objets issu de ce nouveau paradigme, mesure le degré d'interdépendance entres les composantes (classes) du logiciel OO. Il fait partie des attributs très importants pour juger

la qualité d'une conception [2]. Sachant que le logiciel évolue constamment, il s'avère très important de garantir un couplage faible pour faciliter ses phases de maintenance et de test [3].

Pour cela, plusieurs principes consistant à fournir des lignes directrices aux développeurs ont été dégagés. Beaucoup de ces principes sont appliqués dans les patterns de conception. Ces derniers permettent entre autres, de diminuer les interactions entre les classes logicielles et de protéger le logiciel des variations. Ils permettent aussi aux concepteurs logiciels de mettre en place de bonnes solutions éprouvées par la pratique et de les appliquer à large échelle.

Malgré les investigations menées par plusieurs équipes dans le monde, le développement logiciel reste encore une activité difficile principalement à cause de la complexité croissante des systèmes logiciels. L'évolution des logiciels ainsi que les langages et les systèmes d'exploitation ne facilitent pas les tâches de maintenance. Ces changements sont difficiles à prévoir et leur impact encore plus. Le respect scrupuleux des principes de conception (comme l'utilisation des patrons de conception) permet de se protéger de l'impact de ces changements.

Nous nous sommes donnés comme objectif de définir un nouveau modèle de métrique multidimensionnelle intégrant plusieurs attributs internes et capturant la structure du couplage et d'héritage entre classes dans le but d'aider à la conception et maintenance des systèmes orientés objet.

1.2 - Approche

L'approche consiste à construire une nouvelle métrique composite utilisant la représentation matricielle. Cette représentation est basée sur les liens entre composants avec en colonnes les classes sources des liens et en ligne, les classes destinataires. Les classes sont triées de la plus petite à la plus grande selon leur nombre de lignes de code. Chaque composante (case) de la matrice est alors un vecteur contenant des informations intrinsèques et extrinsèques qui lient la classe de ligne correspondante à la classe de la

colonne correspondante. L'analyse et l'exploitation de cette représentation matricielle se feront grâce à l'apprentissage profond des réseaux de neurones de convolution dans le but de détecter des patterns, de prédire les fautes, d'évaluer des caractéristiques de haut niveau ou d'identifier certaines architectures de code et à terme de détecter les erreurs architecturales.

Cependant, une problématique de représentation des données du format brute au format matriciel se pose avec cette approche et nous a obligé à utiliser une technologie de conversion des données brutes en (format) images. Cette technologie est présentée dans le chapitre sur l'analyse de la métrique matricielle.

1.3 - Organisation du travail de recherche

Le travail de recherche effectué dans ce mémoire est organisé comme suit:

Dans le second chapitre, l'état de l'art est abordé pour présenter quelques travaux en relation avec le domaine de recherche. Nous avons aussi abordé les méthodes d'analyse des sources de données utilisées dans l'expérimentation de nos travaux.

Le troisième chapitre porte sur la construction de la représentation matricielle du couplage. Cette représentation forme un jeu de données enregistré dans un fichier CSV et analysé suivant une approche qui sera présentée dans le quatrième chapitre. Cette partie explique les méthodologies, techniques et l'intérêt du choix de cette approche comparée à d'autres en termes de prétraitement des données, d'apprentissage des caractéristiques ou des fonctionnalités, etc.

Dans le cinquième chapitre, une étude expérimentale est présentée avec quelques concepts théoriques sur la conversion des données. Cette conversion des données brutes en images est ensuite fournie aux réseaux de neurones de convolution pour l'extraction des caractéristiques et la classification de patterns de conception. Nous présenterons les résultats de cette classification ou de détection de patterns et les comparerons à ceux obtenus à l'aide d'un outil de détection de patrons de conception. Les patrons détectés par cet outil issu de la littérature vont servir de base de comparaison.

Dans le sixième chapitre, les limitations ainsi que les menaces pour la validité de ces résultats sont aussi présentées.

Pour terminer, dans le septième chapitre, nous allons faire une conclusion générale de ce mémoire en rappelant la problématique, et des contributions apportées. Nous discuterons aussi des recherches futures sur cette technique de représentation et d'analyse des métriques orientées objet du couplage.

CHAPITRE 2. ÉTAT DE L'ART

Une étude du couplage dans les systèmes orientées objets a donné naissance à une série de travaux d'Hakim Lounis, Houari et Walcelio [3] qui peut être divisée en deux catégories. La première étudie le couplage au niveau de la conception et la deuxième le place au niveau du code source. Plusieurs métriques ont été proposées pour évaluer les formes d'interactions entre les modules. Ces propositions faites ont permis de comprendre le lien entre le couplage des classes et les attributs de qualité dans les systèmes orientés objet. Les données de validation des métriques utilisées dans ces travaux correspondent à un environnement de développement de systèmes multi-agents appelé LALO [3]. Ce dernier est un système développé au Centre de Recherche Informatique de Montréal (CRIM). Ces données sont utilisées pour faire des expériences avec des méthodes d'analyse dont l'objectif est de comprendre les relations entre les métriques (formes de couplage) et la prédisposition des classes à contenir des erreurs.

Par la suite de cette étude, d'autres travaux connexes sur l'analyse du couplage ont été aussi présentés de façon brève.

2.1 - Source des données

Dans cette étude [3], les données relatives au code source des classes, sur le couplage de différentes classes et celles en rapport avec les erreurs relevées dans les classes ont été utilisées. Elles sont extraites du code source du système de manière statique. Celles des erreurs relevées dans les classes correspondent à la présence concrète d'erreurs trouvées par les 50 bêta-testeurs des versions 1.1 et 1.1a de LALO. Cependant, seules les classes développées par l'équipe LALO ont été prises en compte. Les classes ayant fait l'objet de réutilisation à partir des librairies ou d'une génération automatique d'outils n'ont pas été considérées [4].

2.2 - Les méthodes d'analyse

Dans la littérature, des modèles d'analyse ont été utilisés pour évaluer les métriques orientées objet afin de voir leur impact sur la tendance des classes logicielles à générer des fautes. Ces métriques sont des variables indépendantes. Les variables dépendantes peuvent être la fréquence des fautes issues des classes. L'utilisation de certaines techniques d'apprentissage machine permet de mettre un lien entre les valeurs des métriques et la variable « propension aux fautes ».

- **La régression logistique :** fait partie des modèles d'apprentissage machine les plus utilisés selon Chauhan [5] dans les travaux d'évaluation des métriques orientées objet. Elle peut être univariée (avec une variable explicative pour mesurer son effet sur la propension aux erreurs des classes) ou multivariée (avec plusieurs variables explicatives).

- **L'algorithme C4.5 :** est un modèle de génération d'arbre de décision et prend en compte des ensembles d'apprentissage contenant des valeurs inconnues et des attributs de types connus.
- **Analyse syntaxique :** est un programme informatique permettant de mettre en évidence la structure d'un texte ou d'un module écrit dans un langage naturel. Elle est utilisée pour la structure d'un module donné en extrayant les définitions et utilisations de toutes les variables, les points d'appel, les paramètres et leurs types.
- **Analyseur de couplage :** un outil qui utilise l'information fournie par l'analyseur syntaxique pour déterminer la direction du couplage entre deux modules.
- **Technique de Visualisation du couplage :** permet, par l'intermédiaire d'une interface utilisateur, de visualiser le couplage d'un couple de modules.

2.3 - Métriques de couplage et de qualité logicielle

Chidamber et Kemerer [1] ont proposé un ensemble de métriques dénommé MOOSE (Metrics for Oriented Object Software Engineering). Parmi ces métriques, CBO (Coupling Between Object) mesure le nombre de classes couplées à une classe donnée. Cette mesure considère qu'il y'a couplage lorsque l'une des méthodes d'une classe fait référence à une autre méthode ou attribut d'une autre classe. Une autre métrique de

couplage RFC (Response For Class) indique le nombre de méthodes d'une classe augmenté du nombre de méthodes des autres classes auxquelles elles font référence. Basili et al [4] ont montré dans une étude que cinq des six mesures proposées par Chidamber et Kemerer étaient liées à la propension des classes à contenir des fautes. Brito e Abreu et Carapu [6] ont défini la suite de mesures MOOD (Metrics for Oriented Object Design) contenant une mesure de couplage appelée facteur de couplage. Cette mesure considère qu'une classe A est couplée à une classe B si elle lui envoie un message. Le modèle de ces mesures reste relativement simple et de haut niveau.

D'autres métriques orientées objet sur le couplage ont été définies par Briand et al [7] au niveau conception. Les auteurs prennent en compte trois mécanismes propres au langage C++ à savoir une classe amie, une spécialisation et une agrégation. Pour cela trois modalités sont considérées :

- Le type de couplage (classe-attribut, classe-méthode ou méthode-méthode),
- La relation entre classes couplées (classe amie ou autre),
- La localisation de l'impact (un couplage qui est importé par une classe ancêtre dont ladite classe est amie ou exporte vers une classe descendante ou amie).

Une validation de vingt-quatre (24) métriques de couplage a été faite en analysant la relation de cause à effet avec une probabilité de détection de fautes dans les classes logicielles.

Une autre approche développée par Price et Demurjian [8] formule qu'il n'est plus souhaitable de considérer le couplage entre les hiérarchies de classes pour une raison de réutilisation. Les auteurs considèrent que chaque classe du système est qualifiée de générale si elle peut être réutilisée ou spécifique si elle est spécifique à l'application en cours.

Dans les travaux de Page-Jones [9], huit différentes mesures de couplage ont été identifiées. Pour chacune de ces mesures, les données partagées (paramètres, variables globales, etc.) sont classées suivant leur type d'utilisation. Des travaux de J. Offutt et al [10] donnent une extension de ces huit (8) niveaux de couplage en douze (12). Cette

extension propose des mesures de couplage du code plus précises. Ces mesures sont définies entre paires d'unités P et Q où une unité correspond à une procédure ou fonction. Pour chacun des niveaux de couplage, les paramètres d'appel ou de retour sont classifiés selon l'utilisation qui en est faite :

- Usage-C : quand une donnée est utilisée comme partie droite d'une affectation ou dans une instruction de sortie,
- Usage-P : une donnée est utilisée dans un prédicat, de la même façon qu'Usage-C,
- Usage-I : une combinaison d'Usage-C et d'Usage-P,
- Usage-M : quand une donnée (ou paramètre) est modifiée (dans le cas d'une transmission de paramètres par pointeur ou par référence).

Dans la suite des trois premiers critères énumérés ci-dessus, différentes formes de couplage y sont dérivées dans les sections suivantes.

2.3.1 - Couplage dans le cas des appels de méthodes

Le couplage par appel de méthodes existe quand une méthode m1 d'une classe C_1 appelle une méthode m2 d'une classe C_2. Dans les cas qui suivent, les acronymes correspondent à l'expression en anglais de la forme de couplage. Un cas qui doit être traité à part est celui où la méthode appelée n'a pas de paramètre. Cette forme de couplage est appelée **NPI**. Le deuxième cas est celui de la transmission des paramètres par valeurs ou le retour d'appel par valeurs. Dans ce cas, plusieurs scénarios sont possibles en fonction de la variation des deux critères : complexité des paramètres transmis ou de la valeur de retour et l'usage qui en est fait. Il était clair que les auteurs n'avaient pas une combinaison de tous les cas. Par exemple, le mode Usage-M n'est pas possible. De plus, ils ont décidé de ne pas considérer le cas d'un retour par valeur de type complexe. Il reste donc les cas suivants :

- **SDI** : Cas où un paramètre transmis est de type simple et utilisé dans la méthode appelée en mode Usage-C.
- **StDI** : Cas où un paramètre transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-C.

- **RDI** : Cas où la valeur de retour d'un appel est utilisée dans la méthode appelante en mode Usage-C.

- **SCI** : Cas où un paramètre transmis est de type simple et utilisé dans la méthode appelée en mode Usage-P.

- **StCI** : Cas où un paramètre transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-P.

- **RCI** : Cas où la valeur de retour d'un appel est utilisée dans la méthode appelante en mode Usage-P.

- **SDCI** : Cas où un paramètre transmis est de type simple et utilisé dans la méthode appelée en mode Usage-I.

- **StDCI** : Cas où un paramètre transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-I.

- **RDCI** : Cas où la valeur de retour d'un appel est utilisée dans la méthode appelante en mode Usage-I.

- **TI** : Cas où un paramètre est transmis dans la méthode appelée à une troisième méthode avant tout autre usage. Le type de ce paramètre peut être simple ou complexe.

- **SRDI** : Cas où un paramètre-adresse transmis est de type simple et utilisé dans la méthode appelée en mode Usage-C.

- **StRDI** : Cas où un paramètre-adresse transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-C.

- **SRCI** : Cas où un paramètre-adresse transmis est de type simple et utilisé dans la méthode appelée en mode Usage-P.

- **StRCI** : Cas où un paramètre-adresse transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-P.

- **SRDCI** : Cas où un paramètre-adresse transmis est de type simple et utilisé dans la méthode appelée en mode Usage-I.

- **StRDCI** : Cas où un paramètre-adresse transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-I.

- **SRMI** : Cas où un paramètre-adresse transmis est de type simple et utilisé dans la méthode appelée en mode Usage-M.

- **StRMI** : Cas où un paramètre-adresse transmis est de type complexe et utilisé dans la méthode appelée en mode Usage-M.

2.3.2 - Couplage dans le cas des données ou types partages

On parle d'une donnée partagée quand elle est définie dans une classe et utilisée dans une autre classe. C'est par exemple le cas d'une donnée membre statique dans C++ ou Smaltalk. Un type peut aussi être partagé, en ce sens qu'il peut être défini dans une classe et utilisé dans une autre classe. Un exemple est celui d'un attribut d'une classe C_1 qui a pour type une autre classe C_2. En considérant le mode d'usage, les cinq cas suivants leur sont parus les plus significatifs :

- **GDI** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-C.

- **GCI** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-P.

- **GDCI** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-I.

- **GMI** : Cas où une donnée définie dans une classe est utilisée dans une autre classe en mode Usage-M.

- **TyI** : Cas où un type utilisateur est défini dans une classe et utilisé dans une autre classe.

2.4 - Autres travaux connexes

Dans cette partie, un résumé a été fait sur d'autres travaux en relation avec l'analyse du couplage pour démontrer certaines formes de couplage qui peuvent être une source importante de fautes ou de leur propagation dans le développement logiciel.

En 2009, Aggarwal et al [11] ont fait une étude analogue à celle conduite par Briand et al [8]. Ils ont étudié et validé des métriques capturant différents aspects de la conception OO dont le couplage, l'héritage, la cohésion, l'encapsulation, et le polymorphisme. Ces métriques ont été calculées à l'aide d'une application JAVA et la régression logistique a été utilisée comme méthode de prédiction. D'après leurs résultats, le couplage semble être un bon prédicteur de la prédisposition aux fautes, plus spécifiquement le couplage à travers l'invocation de méthodes (couplage sortant).

En 2005, Kaung et al [12] ont publié un article dont l'objectif était de trouver de nouvelles métriques de couplage et de créer un système qui mesure ces formes de couplage dans un programme donnée. Ils ont regroupé les formes de couplage en deux catégories. Le couplage d'importation qui compte les messages envoyés à partir du module et le couplage d'exportation qui compte les messages reçus par le module. Ce dernier définit qu'un module est considéré comme une simple fonction ou procédure et que le couplage est la mesure d'interdépendance de ces modules. Le système qui est développé par les auteurs détermine le couplage d'un programme C parmi les 8 types de couplage définis ci-dessous. Ce système comprend trois composants : analyseur syntaxique, analyseur de couplage et visualiseur de couplage. En appliquant ces composants, les auteurs ont visualisé les directions de couplage qui existent entre les procédures dans un ensemble de quatre (4) programmes comportant **45** à **75** procédures et **1360** à **2500** lignes de code.

- **Couplage d'importation d'appel** : un module appelle un autre module mais pas de paramètres ou variables de références communes.
- **Couplage d'importation scalaire** : une variable scalaire dans un autre module est passée comme paramètre effectif au module appelé A.
- **Couplage d'importation de timbre** : un enregistrement dans un autre module.
- **Couplage d'importation de « tramp » :** paramètre formel dans un autre module est passé comme paramètre effectif au module B. Ce module passe ensuite le paramètre formel au module A sans que B ait accédé ou modifié la variable.
- **Couplage d'exportation scalaire** : une variable scalaire dans le module A est passée comme paramètre effectif au module B.

- **Couplage d'exportation d'appel** : un module appelle un autre mais sans aucun paramètre ou variable ou variables de références communes.
- **Couplage d'exportation de timbre** : un enregistrement dans le module A est passe comme paramètre effectif à un autre module.
- **Couplage d'exportation de « tramp »** : un paramètre formel dans le module A est passé comme paramètre effectif au module B. Ce module B passe ensuite le paramètre formel correspondant à un autre module sans qu'il ait accédé ou modifié la variable.

En 2005, Yang et Berrigan [13] ont démontré qu'il existe des formes de fermetures transitives de couplage que l'on ne peut pas réduire à des couplages directs. Ces formes de couplage sont une source importante d'erreurs dans le développement logiciel Widad [14]. Ils ont décrit trente (30) métriques dont seulement deux (2) mesurent certaines formes de couplage indirect. Les auteurs de cet article ont utilisé un outil de mesure de couplage indirect (Indirect Coupling Detector) qui est un plugiciel d'Eclipse [15]. Il mesure une forme particulière du « couplage indirect » en implémentant l'algorithme de détection << **use-def>>**.

En 2012, Rathore et al [16] ont évalué la capacité des attributs de conception OO reliés au couplage, à la cohésion, à la complexité, etc., à prédire la propension aux fautes des classes logicielles. Cette évaluation est faite sur une base indépendante (de chacune des métriques respectives) et combinée (sur l'ensemble des métriques respectives). Ils ont utilisé les données (du projet PROP) du répertoire public de données PROMISE [17] pour investiguer dix-huit métriques de différentes suites (CK, Tang, QMOOD, HENDERSON-SELLERS, McCabe) avec les méthodes classiques de régression logistique et des méthodes d'apprentissage automatique (IBK, RF et NB). Dans leurs résultats, le couplage était un des attributs de conception qui pouvait influencer de façon significative la prédisposition aux fautes. D'après ces auteurs, les modèles de prédictions construits avec les métriques de couplage et de complexité avaient une meilleure précision que ceux construits avec le reste des métriques.

2.5 - Conclusion

Dans la littérature, l'utilisation des métriques orientées objet liées au couplage a fait l'objet d'investigations qui montrent leur pertinence notamment pour la détection de la prédisposition des classes aux fautes. Pour donner suite à la validation de ces métriques, d'autres métriques de couplage ont aussi été validées après des considérations sur les relations entre les classes, sur les types de couplage et sur la localisation de son impact. Au niveau du code source, différentes formes d'interactions ont été identifiées sur les appels de méthodes et les types partagés à travers des cas d'importations et d'exportations. Les métriques qui en sont issues ont fait l'objet d'expérimentations dans les cadres d'étude par l'utilisation des méthodes de validation statistique.

L'analyse du couplage avec les métriques orientées objet a permis de démontrer sa capacité de prédiction de fautes dans les classes logicielles. Elle a permis aussi d'identifier un ensemble de formes de couplage qui peuvent compléter la suite des métriques de code source OO proposées par Chidamber et Kemerer [1].

Sur les approches utilisées dans les travaux présentés dans ce chapitre, une remarque est faite sur l'utilisation des algorithmes d'apprentissage automatique, des méthodes d'analyse statistique. Ces modèles ont donné de belles performances. Dans une approche plus axée sur les relations entre classes logicielles, nous avons opté pour l'utilisation d'un algorithme d'apprentissage profond dans notre étude pour évaluer le couplage à travers une forme de représentation des données. Cette représentation capture plusieurs aspects du couplage, leur type, leur nombre, ainsi que la structure des liens entre classes logicielles.

CHAPITRE 3. REPRÉSENTATION MATRICIELLE DU COUPLAGE

3.1 - Métriques utilisées

Dans cette partie, nous décrivons les métriques utilisées dans la forme représentation matricielle du couplage ainsi que les techniques utilisées pour leur extraction.

Nous nous intéressons aux métriques de code source de la programmation orientée objet (POO). Ce paradigme d'analyse et de conception de logiciels est le plus utilisé dans l'industrie du génie logiciel actuellement. La POO vient avec des artefacts tels que l'héritage, le polymorphisme, l'abstraction, etc. D'autres mesures en relation avec cette dépendance entre classes comme le nombre d'appels des méthodes, les classes appelantes, appelées et leurs interfaces ont été extraites. Une métrique de complexité (de McCabe) est aussi calculée. Appelée aussi complexité cyclomatique [18], cette métrique est définie par le nombre de chemins linéairement indépendants qu'il est possible d'emprunter dans une méthode. Cette métrique capture (entre autres) la difficulté à la tester. Notre nouvelle représentation inclut certaines interactions et types de composants référant aux CK Metrics [1] comme CBO, RFC, DIT que nous allons présenter ici.

- **CBO** (Coupling Beetwen Objects) est le nombre de classes auxquelles une classe est couplée. Autrement dit il y'a couplage entre deux classes si une méthode définie dans l'une utilise les méthodes ou les variables d'instances de l'autre classe. Les références sont comptées dans les deux sens de sorte que le CBO d'une classe donnée soit la taille de l'ensemble des classes auxquelles elle fait référence. Dans la construction de la représentation matricielle, nous avons extrait le nombre d'appels de méthodes entre les classes appelantes et appelées. Nous avons aussi compté les liaisons d'agrégation qui existaient entre ces classes pour inclure cette mesure du couplage dans la matrice.

- **RFC** (Response For Class) est le nombre de méthodes et de constructeurs différents appelés par une classe. Il mesure le nombre de méthodes différentes qui peuvent être exécutées lorsqu'un objet de cette classe reçoit un message. Nous avons inclus cette mesure aussi en comptant les appels de méthodes différentes sur une classe donnée.
- **DIT** (Depth Inheritance Tree) est une métrique de code source qui mesure la longueur maximale entre un nœud et le nœud racine dans une hiérarchie d'héritage de classes. Nous avons utilisé la relation d'héritage entre les classes mais aussi son niveau d'abstraction. Cette dernière a été incluse dans la matrice pour indiquer le haut niveau de complexité de certaines classes par rapport à d'autres.

3.1.1 - Les classes appelantes et appelées

Pour construire notre nouvelle représentation matricielle du couplage des systèmes OO, nous avons construit une base de données comportant plusieurs tables. Deux tables permettant d'identifier les classes et les liens entre elles (Appels de méthodes, Héritages, Implémentations, Références...)

Ces deux tables sont ensuite fusionnées dans une table d'association sur laquelle les identifiants des classes sont exportés comme des clés étrangères.

ID	Nom de la classe	Interface	Abstraction	Nb Impl	Complexité Cyclomatique
0	TestEDate	0	0	1	1
1	UserEditAndPersistListing	0	0	1	1
2	Row	1	0	1	1
3	WaferMapDataset	0	0	1	1
4	RefEvalBase	0	1	1	1
5	TestLbsDataSubRecord	0	0	1	3
6	AbsExpression	0	0	1	1
7	TestXSSFRichTextString	0	0	1	1
8	TestXSSFChartSheet	0	0	1	1

Tableau 1 : Extrait de la table classe appelante du projet Apache POI.

- Le nom de la classe appelante avec son ID qui va migrer dans la table d'association ou matrice de représentation.
- Les champs abstraction et interface indiquent respectivement si la classe est abstraite ou pas, est une interface ou pas. Ces champs vont migrer dans la table d'association pour indiquer la profondeur de l'héritage (DIT) de la classe.
- Le nombre d'implémentations de la classe et de sa complexité cyclomatique sont décrits dans la section 3.1.1.3.

ID	Nom de la classe	Interface	Abstraction	Méthode	Nb Impl	Complexité Cyclomatique
0	AbortableHSSFListener	0	1	abordableProcessRecord	1	1
1	AbstractionAnnotation	0	1	addChangeListener	0	1
2	AbstractBlock	0	0	setFrame	7	1
3	AbstractButton	0	1	setSelected	3	1
4	AbstractCellEditor	0	1	fireEditingStopped	0	1
5	AbstractDataset	0	1	notifyListeners	0	1
6	AbstractDialLayer	0	1	setVisible	1	1
7	AbstractExcelUtilis	0	0	loadXls	1	1
8	AbstractFileHandler	0	1	handleExtracting	0	1

Tableau 2 : Extrait de la table classe appelée du projet Apache POI.

- Le champ Méthode récupère la méthode appelée dans la classe.
- La complexité cyclomatique de la classe qui est décrite dans la section 3.1.1.3

IS	ID	Nb_Appel	TIS	TID	CCS	CCD	Héritage	Agrégation
0	3301	1260	0	0	1	1	1	0
1	2525	122	0	0	9	1	0	1
1	3733	2	0	0	9	3	0	0
2	1151	6	0	0	1	1	0	0
3	247	6	1	0	1	1	0	0
3	330	6	1	1	1	1	0	1
3	361	6	1	1	1	1	0	1
3	362	6	1	0	1	1	0	0
3	1056	6	1	0	1	1	0	0

Tableau 3 : Extrait de la table d'association des classes appelantes et appelées POI.

- **IS** (Identifiant Source ou de la classe appelante) est importé depuis la table classe appelante pour l'identifier dans la matrice de représentation.

- **ID** (Identifiant Destination ou de la classe appelée) a migré comme clé étrangère dans la table d'association pour identifier la classe appelée.
- **Nb_Appel** ou nombre d'appels de méthodes entre la classe appelante et appelée.
- **TIS** et **TID** : indiquent respectivement si les classes (appelante) et (appelée) sont des interfaces (1) ou pas (0).
- **CCS** : Complexité Cyclomatique Source ou de la classe appelante.
- **CCD** : Complexité Cyclomatique Destination ou de la classe appelée.
- **Héritage** : indique s'il s'agit d'une relation d'héritage (1) ou pas (0).
- **Agrégation** : Il en est de même pour l'agrégation.

3.1.2 - Classe appelante (source) est de type interface

Les interfaces sont des composants abstraits définissant un contrat de communication entre deux entités. La conception d'interface participe à la réduction du couplage. Autrement dit le couplage faible ou lâche se produit lorsque la classe dépendante (classe source) contient un pointeur uniquement vers une interface (cible). Cette dernière peut ensuite être implémentée par une ou plusieurs classes concrètes. Un couplage sera dit fort ou serré lorsqu'une classe source contient directement un pointeur vers une classe (cible) concrète, qui fournit le comportement requis. Nous avons consigné l'information associée au type de composant (Interface ou non) dans la table d'identification des classes des systèmes logiciels sous forme booléenne 1 si la classe est une interface ou 0 dans le cas contraire.

3.1.3 - Complexités cyclomatiques des classes sources et cibles

La complexité cyclomatique est une métrique intrinsèque à une méthode. Elle mesure quantitativement le nombre de chemins linéairement indépendants contenus dans une section de code source. Nous l'avons intégrée à notre représentation du point de vue de la classe. Cette métrique pour une méthode est au moins égale à 1 (au moins un chemin linéairement indépendant) et la sommation de la complexité cyclomatique de l'ensemble des méthodes d'une classe donne la complexité cyclomatique de la classe. Une méthode

avec une haute complexité cyclomatique est plus difficile à comprendre et à maintenir McCabe [19].

3.1.4 - Le nombre d'invocations de la méthode appelée

Le couplage entre deux composants par l'intermédiaire d'appel de méthode peut se voir, selon le sens de flux de contrôle, comme entrant ou sortant.

- Le **couplage entrant** (Fan-in) : définit pour une classe donnée, le nombre de classes la référençant. Cela montre le niveau d'utilisation d'une classe par les autres classes du système logiciel. Les classes présentant un couplage entrant fort sont critiques vu le nombre de composants qui dépendent d'eux.
- Le **couplage sortant** (Fan-out) : compte le nombre de classes auxquelles fait référence une classe donnée. Il montre le degré de dépendance d'une classe par rapport au reste du système logiciel. Les classes qui forment un couplage sortant fort sont difficilement réutilisables à cause de leurs dépendances externes élevées.

3.1.5 - L'héritage entre les classes

Ce mécanisme de conception de la POO permet aux classes d'hériter des attributs et méthodes d'une autre classe (mère). Cela crée ainsi une forme de couplage entre la classe fille et la classe mère. L'approche orientée objet permet de :

- Décider si une considération doit être faite sur un simple lien de parenté entre deux classes,
- Gérer le polymorphisme lors de l'énumération des liens d'appel de méthodes,
- Décider de la manière d'assigner les attributs et méthodes en cas de présence de liens d'héritage entre classes concernées.

Dans les trois options, les mesures du couplage par héritage ont été définies par Briand et al [16]. Elles sont listées dans le tableau suivant :

Option	Description
1	Compter seulement le couplage basé sur l'héritage
2	Compter seulement le couplage non basé sur l'héritage
3	Compter séparément le couplage basé sur l'héritage et celui non basé sur l'héritage
4	Compter à la fois le couplage basé sur l'héritage et celui non basé sur l'héritage

Tableau 4: Option de comptage du couplage par héritage.

Dans notre étude, nous nous sommes limités au lien d'héritage entre deux classes en testant s'il y'a héritage ou pas. Après l'extraction de cette information par analyse de code source, elle est sauvegardée comme variable booléenne avec la valeur 1 s'il y'en a et 0 dans le cas contraire dans la table d'association.

La figure suivante montre une relation d'héritage entre deux classes du système Apache POI.

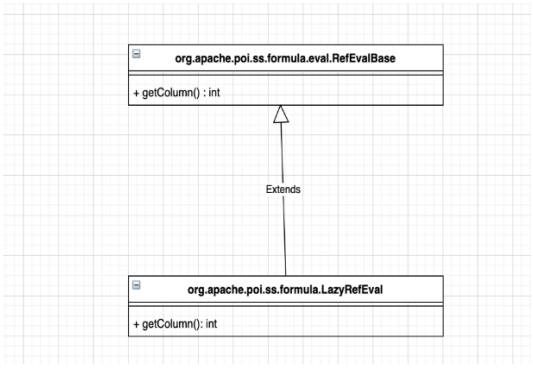

Figure 1 : Exemple d'héritage entre les classes RefEvalBase et LazyRefEval du projet Apache POI.

3.1.6 - L'agrégation entre les classes

Ce concept définissant l'appartenance de l'entité d'une classe à plusieurs classes différentes est choisi dans ce travail pour montrer le couplage par utilisation de classe comme attribut d'instance d'une autre classe. Dans la littérature, la métrique MOA fait partie des mesures QMOOD Bansiya et Davis [20]. Elle compte pour une classe donnée, le nombre d'attributs dont les types sont d'autres classes du système. Cette mesure permet

de mettre en évidence la relation de type “ un a plusieurs “ à savoir l’agrégation, une partie d’un tout.

Dans cette étude, nous avons extrait la relation d’agrégation entre deux classes logicielles que nous avons sauvegardée dans une variable booléenne avec 1 s’il y’en a ou 0 dans le cas contraire.

3.2 - Collecte des données

La première étape de ce processus est l’extraction de ces mesures dans les codes sources. Grace à un plugin (Eclipse JDT) développé sur l’environnement Eclipse, l’extraction des mesures est faite en compilant les classes des projets Java. Cette compilation génère un modèle de projet sous forme d’AST (Abstract Syntax Tree).
Dans les sections qui suivent, nous allons présenter les différents outils et techniques utilisés pour cette extraction.

3.2.1 - Abstract Syntax Tree (AST)

C’est un modèle de représentation de code Java se basant sur un graphe d’arborescence comme décrit dans ce tableau ci-dessous. Cette représentation permet, avec les composantes du pattern visiteur, d’extraire les différents types de liaison entre les classes par analyse statique du code.

Le tableau suivant décrit un modèle de projet Java.

Élément de projet	Élément de modèle Java	Description
Projet Java	IJavaProject	Le projet Java contenant tous les autres projets.
src/bin ou bibliothèque externe	IPackageFragmentRoot	Conserver les fichiers sources ou binaires
Chaque paquet	IPackageFragment	Chaque package est sous IPackageFragmentRoot
Fichier source Java	ICompilationUnit	Le fichier source est toujours sous le nœud du package
Types, champs et méthodes	IType / IField / IMethod	Types, Champs et méthodes

Tableau 5: Description d'un modèle de projet Java.

3.2.2 - Eclipse JDT (Java Developement Tools)

C'est un plugin d'Éclipse qui supporte le développement de systèmes Java avec des assistants (éditeurs, pré-compilateur, refactoring, etc.).

Ce module est divisé en composants comme suit :

- Infrastructure de traitement des annotations JDT API Java 5.0,
- Infrastructure JDT Core Java IDE,
- Gestion du débogage JDT pour Java,
- Gestion de l'édition Java du texte JDT,
- Interface Utilisateur JDT UI Java IDE.

Parmi ces modules, deux ont été utilisés pour l'extraction des données du code source à savoir les composants JDT Core et JDT UI. Le composant JDT Core fournit l'API permettant de naviguer dans l'arborescence des éléments du modèle de projet Java. Celui du JDT UI (User Interface) offre plusieurs contributions de l'espace de travail pour la gestion du code Java.

3.2.3 - Extraction des designs patterns

Dans nos démarches empiriques, nous avons utilisé la nouvelle technique de représentation matricielle que nous avons proposée dans ce travail. Ainsi, nous avons

utilisé une approche proposée dans la littérature pour extraire les patrons de conception à partir de l'analyse de code source. Proposé par Tsantalis et al [21], l'algorithme est un programme Java recevant en entrée, des fichiers compilés d'un projet (fichiers d'extension .class) et détecte en sortie la liste des patterns implémentés par les développeurs dans ce projet. Il utilise une méthodologie de détection de patterns de conception appliquée dans un système et décrite comme suite :

- La **rétro-ingénierie** (reverse engineering) où les relations de généralisation, d'invocations de méthodes, d'association entre les classes logicielles sont représentées par une matrice d'adjacence [N, N] (N étant le nombre de classes logicielles du système à l'étude).
- La **détection des hiérarchies d'héritage** où les associations de généralisation entre les classes sont présentées hiérarchiquement sous forme d'arbres avec des nœuds. Cette représentation permet d'identifier les hiérarchies auxquelles les classes peuvent appartenir à la fois mais aussi à celles qui ne participent à aucune hiérarchie.
- La **construction des matrices des sous-systèmes** en se basant sur les composants du système qui forment une ou plusieurs hiérarchies de classes. Suivant le nombre d'hiérarchies dans le pattern à détecter, deux approches sont prises en compte : celle où chaque hiérarchie dans le système forme un sous-système isolé et une autre dans laquelle le pattern contenant plusieurs hiérarchies forme des sous-systèmes en combinant toutes les hiérarchies du système.
- L'**application de l'algorithme de similarité** entre les matrices de sous-systèmes et de modèle qui correspond à la recherche de pattern dans chaque sous-système de manière indépendante.
- L'**extraction de motifs** dans chaque sous système où fréquemment une instance de chaque pattern est présente.

Dans la partie concernant l'étude expérimentale de ce travail de recherche, nous présenterons les résultats de l'extraction des patrons de conception appliquée sur le système Apache POI à travers cet outil de détection.

3.2.4 - Étiquetage du jeu de données

La maitrise de cette représentation matricielle doit nous permettre dans le cadre de notre application, de détecter des patterns de conception à travers une métrique synthétique en tenant compte du couplage mais aussi de sa structure et de ses différentes formes dans les systèmes orientés objet. Pour cela, cette représentation a fait l'objet d'une étude empirique se basant sur l'apprentissage profond plus précisément les réseaux de neurones de convolution. Ces derniers analysent cette représentation matricielle qui peut être vue comme une image à plusieurs dimensions où à la place des RGB (chaque cellule), nous avons défini un ensemble de métriques intégrées et formées (métrique synthétique) par les liaisons d'appels de méthodes, d'héritage, d'agrégation, etc. entre les classes appelantes et appelées. En effet cette représentation matricielle capture plusieurs éléments d'architecture du logiciel comme le couplage, l'héritage, etc. Pour chaque cellule ou liaison entre classes de la matrice, nous allons l'associer à un ensemble de patterns de conception à détecter avec notre modèle de réseaux de neurones de convolution.

Après avoir construit une partie de la représentation matricielle des données, autrement dit après avoir créé la table d'association entre les classes appelantes et appelées, il nous restait à étiqueter le jeu de données. Comme nous voulons analyser cette matrice image avec un modèle d'apprentissage supervisé (réseau de neurones de convolution), nous avons pensé à extraire les patterns formés par les liaisons entre les classes à l'aide de l'outil de détection présenté dans la section 3.2.3. Cette extraction nous a permis d'associer ces modèles de patterns aux groupes de classes qui les forment.

Dans la détermination des liaisons qui forment les types de patterns, nous avons sauvegardé l'extraction des patterns dans un fichier XML. Cette sauvegarde est une fonctionnalité intégrée dans l'outil de détection. Dans ce fichier, chaque élément de pattern a été parcouru en comparant les noms des classes impliquées dans le pattern avec ceux obtenus dans les tables classes appelantes et appelées. Ainsi, pour un type de pattern donné, nous avons associer dans la matrice de représentation les identifiants de ces classes au type de pattern trouvé dans le fichier XML. Chaque type de pattern est une variable booléenne 0 ou 1 (présence ou pas du type pattern).

La génération d'une partie du fichier XML ci-dessous par l'outil de détection du projet Java Apache POI montre l'exemple du pattern singleton de la classe **SectionIDMap.**

```
<pattern name="Singleton">
        <instance>
                <role name="Singleton" element="org.apache.poi.hpsf.wellknown.SectionIDMap" />
                <role name="uniqueInstance" element="org.apache.poi.hpsf.wellknown.SectionIDMap::defaultMap:org.apache.poi.hpsf.wellknown.SectionIDMap" />
        </instance>
</pattern>
```

Figure 2 : Exemple d'exportation au format XML d'une instance du pattern singleton avec l'outil de détection.

A la suite de ce modèle de document XML généré par l'outil de détection, nous avons pu écrire un programme Java avec la librairie **Dom4J** pour l'analyse et le parsing de ce document. Ce programme a permis aussi de faire la correspondance entre les classes extraites avec le plugin JDT utilisé et les patterns associés.

3.2.5 - Présentation complète de la matrice de représentation de couplage

Pour l'application de cette nouvelle forme de représentation du couplage, nous avons utilisé la détection de patterns architecturaux comme cas pratique d'analyse de la métrique synthétique. Pour cela, l'outil de détection présenté dans la section 3.2.3 nous a permis d'étiqueter les données et d'avoir des bases de comparaison sur les patterns détectés avec notre approche de représentation matricielle.

Comme présenté dans la section précédente sur l'étiquetage du jeu de données, chaque type de pattern a été associé aux groupes de liaisons formées par un ensemble de classes logicielles.

Le tableau suivant représente une portion de la représentation matricielle associée avec quelques patterns extraits du système Java Apache POI [22].

IS	ID	NA	TIS	TID	CCS	CCD	Héritage	Agrégation	S	A	ST	O	F	C	D
0	78	38	0	0	1	1	1	0	0	0	0	0	0	0	0
1	8	97	0	0	9	1	0	1	0	0	0	0	0	0	0
3	90	110	0	0	9	3	0	0	0	0	0	0	0	0	0
4	18	23	0	0	1	1	0	0	0	0	0	0	0	0	0
6	90	17	1	0	1	1	0	0	0	0	0	0	0	0	0
6	17	17	1	1	1	1	0	1	0	1	0	0	0	0	0
9	38	17	1	1	1	1	0	1	0	0	0	0	0	0	0
10	115	8	1	0	1	1	0	0	0	0	0	0	0	0	0
10	55	35	1	0	1	1	0	0	0	0	0	0	0	0	0
10	120	144	1	0	2	1	1	1	0	0	0	0	0	1	0
10	43	10	0	0	1	1	0	1	0	0	0	0	0	0	0
10	65	3	0	0	1	1	0	0	0	0	0	0	0	0	0

Tableau 6 : Extrait d'une partie de la matrice de représentation du couplage.

- **S** : Présence du pattern Singleton 0 ou 1 dans la liaison.
- **A** : Pattern Adapter.
- **ST** : Pattern Stratégie.
- **O** : Pattern Observer.
- **F** : Pattern Factory.
- **C** : Pattern Composite.
- **D** : Pattern Decorator.

3.3 - Conclusion

Dans cette partie, les métriques de code source orientées objet extraites pour l'analyse de cette représentation matricielle du couplage ont été présentées. Elles font références aux métriques CK dont quelques-unes ont été définies. Nous avons aussi présenté la manière dont elles sont extraites dans les codes sources en nous basant sur les classes appelantes et appelées, sur leurs relations d'héritage, d'agrégation et de leurs types d'interface et d'abstraction. Les complexités cyclomatiques de chacune de ces classes ont été aussi calculées et intégrées à la construction de la métrique synthétique.

Pour la collecte de ces données, nous avons présenté le plugin Eclipse JDT basé sur le modèle de représentation de code source Java (AST), qui nous a facilité l'extraction des données.

Et pour terminer, après la construction de la représentation matricielle, l'extraction d'un nombre de designs patterns architecturaux a été faite pour associer les liaisons entre les classes logicielles au type de pattern de conception qu'elles forment. Ces patterns extraits nous ont servi d'étiquettes pour l'apprentissage supervisé avec les réseaux de neurones de convolution dans le cadre applicatif de détection de patterns. Ces derniers vont analyser et évaluer la représentation matricielle pour détecter ces patterns. La possibilité d'utilisation de cette approche de représentation pour la détection de patterns a fait l'objet d'une étude expérimentale que nous allons présenter. Dans le chapitre suivant, nous allons d'abord présenter l'approche utilisée pour analyser la métrique synthétique construite dans ce chapitre en se basant sur la problématique d'interprétation de la donnée matricielle.

CHAPITRE 4. ANALYSE DE LA MÉTRIQUE MATRICIELLE

4.1 - Problématique d'interprétation de la donnée matricielle

Dans cette section, nous abordons la problématique de l'interprétation de la donnée matricielle que nous voulons analyser en utilisant les réseaux de neurones de convolution. En effet, la matrice de représentation de couplage obtenue est de nature multidimensionnelle et ne permet donc pas une interprétation ou une évaluation directe. Chaque cellule multidimensionnelle de la matrice représente les types de lien existants entre deux (2) classes du logiciel. En termes d'applications, nous avons tenté de voir à quel point notre représentation pourrait aider à détecter les patterns logiciels.

La nécessité des données d'entrainement, induite par la nature de l'apprentissage supervisé des réseaux de neurones de convolution soulève le problème de l'étiquetage des données, que nous avons présenté au terme du chapitre précèdent.

S'inspirant de Sharma et al [23], nous avons effectué une transformation afin de convertir les données brutes de la matrice en pixels images. Les pixels obtenus pourront être analysées par un réseau de neurones de convolution afin de détecter (dans notre cas d'application) les patterns de conception logiciels.

4.2 - Description de l'approche utilisée

DeepInsight est une méthode d'extraction et de classification de caractéristiques à travers l'utilisation d'un CNN (Convolutional Neural Networks). Elle prend en entrée des données au format non-image (vecteurs caractéristiques) et applique une étape de transformation en pixels images afin de procéder à la phase de classification de ces images.

Le principe est d'abord de convertir des échantillons non-image en une forme d'image organisée, puis de normaliser les caractéristiques après cette conversion pour pouvoir les

donner en entrée à l'algorithme d'apprentissage profond à des fins de classification/de reconnaissance.

4.2.1 - Conversion des vecteurs caractéristiques en pixels images

Ce processus prend en entrée des vecteurs de caractéristiques qui sont fournis. Sur ces vecteurs, un algorithme de mesure de similarité ou une technique de réduction de dimensionnalité non linéaire est appliquée pour faire un regroupement entre eux. Ce regroupement permet d'identifier les vecteurs corrélés permettant de former des coordonnées pixels qui se ressemblent. Les techniques de réduction de la dimensionnalité étudiées ici sont le PCA (Principal Component Analisys) ou le KPCA (Kernel Principal Component Analisys) ou encore t-SNE (t-distributed Stochastic Neighbor Embedding). Cette dernière, utilisée dans cette approche, est un algorithme d'apprentissage machine pour la visualisation basée sur l'incorporation. Cette technique projette un ensemble de points de plusieurs dimensions sur un plan à deux dimensions. Cette projection déformante regroupe les points proches tout en séparant ceux qui sont distants.

Une fois le regroupement fait, l'algorithme coque convexe de D. Kirkpatrick [24] est utilisé pour déterminer le plus petit rectangle contenant tous ces points proches. Comme l'image doit être encadrée sous une forme horizontale ou verticale pour le réseau de neurones de convolution, une rotation est faite par cet algorithme. Les coordonnées cartésiennes sont converties en pixels. Cette conversion est faite en déterminant la moyenne de certaines caractéristiques vues que les tailles des images ont des limitations en pixels.

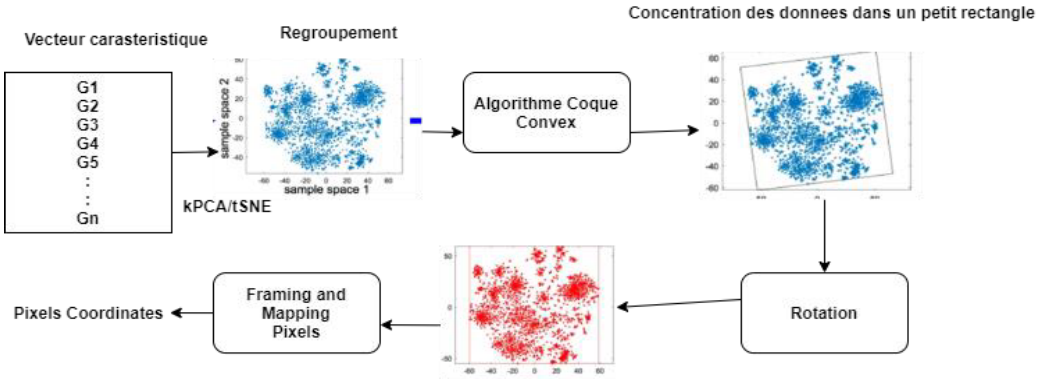

Figure 3: Transformation d'un vecteur de caractéristiques en pixels

4.2.2 - Normalisation des fonctionnalités

Dans cette étape, les valeurs des caractéristiques sont normalisées entre [0, 1] avant l'application de la transformation en pixels images. Pour cela deux types de normalisation ont été utilisés :

- Normalisation (1) : chaque entité ou caractéristique est supposée indépendante et donc normalisée par son minimum et maximum,
- Normalisation (2) : la structure ou forme des caractéristiques est conservée jusqu'à un certain point en la normalisant avec la valeur maximum de l'ensemble de la formation.

Les performances sur les données de validation sont évaluées sur les deux types de normalisation.

4.2.3 - Architecture du réseau de neurones de convolution

La deuxième partie de la méthode DeepInsight est la mise en place d'une architecture de réseau de neurones de convolution qui fait l'extraction et la classification des caractéristiques après la transformation en pixels images.

Les auteurs [23] de cet article ont développé une architecture CNN en parallèle pour pouvoir faire passer différentes tailles de filtres pendant l'entrainement du modèle. Elle est composée de deux blocs parallèles dont chacun contient des couches convolutives qui se composent comme suit :

- Une couche de convolution 2D,
- Une couche de normalisation par lots pour éviter le surajustement du modèle,
- Une couche de rectification linéaire (ReLU),
- Et une couche de max-pooling pour sous échantillonner la taille des images dans chaque couche.

Les sorties de la quatrième couche de convolution sont fusionnées et envoyées à une couche dense ou entièrement connectée pour la classification.

Une couche Softmax donne la sortie avec les étiquettes des classes.

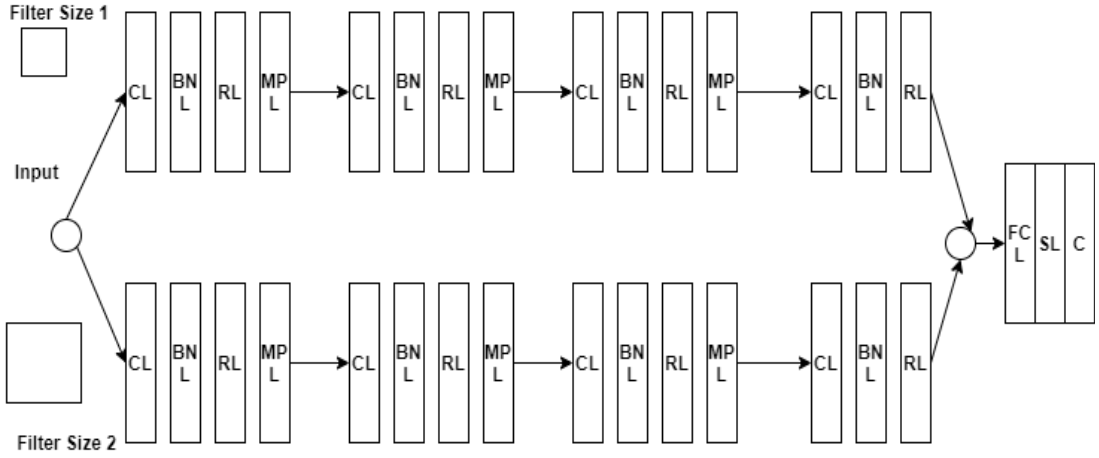

Figure 4 : Architecture CNN de la méthode DeepInsight.

- **CL** : Convolution Layer
- **BNL** : Batch Normalization Layer
- **RL** : ReeLu Layer
- **MPL** : Max-Pooling Layer
- **SL** : Softmax Layer
- **C** : Classification

Les hyperparamètres du réseau tels que les couches de convolution, le taux d'apprentissage, la taille des filtres, etc. ont été réglés à l'aide d'une technique d'optimisation bayésienne. Cette technique d'optimisation a démontré son intérêt de déterminer les valeurs des hyperparamètres pour lesquelles leur modèle de CNN a donné de bons résultats. Cela s'explique par la rapidité de convergence de cet algorithme d'optimisation tout en gardant le même caractère exploratoire permettant de ne pas tomber sur un minimum local.

Après une formation du modèle à l'aide de ces hyperparamètres optimaux, chaque nouvel échantillon peut être identifié dans l'une des catégories ou classes.

4.3 - Implémentation de l'approche

Dans cette partie, l'approche utilisée pour convertir les données et entrainer le modèle, a été implémentée avec les techniques auxquelles les auteurs [23] ont eu recours dans leur travail de recherche.

4.3.1 - Présentation de la technique de d'implémentation

Matlab [25] est une plateforme de programmation et de calcul numérique utilisée par des millions d'ingénieurs et de scientifiques pour analyser des données, développer des algorithmes et créer des modèles.

Depuis quelques années, Matlab intègre un composant nommé « DeepLearning Toolbox » pour l'apprentissage profond. Il s'agit d'une librairie permettant de créer et d'interconnecter les couches d'un réseau de neurones. Les auteurs de cette approche DeepInsight ont utilisé cet outil pour créer et entrainer le modèle. Nous avons utilisé ce modèle pré-entrainé et l'avons adapté à notre jeu de données.

4.3.2 - Implémentation

La programmation de l'approche est organisée en fonctions ou fichiers Matlab suivant les différentes étapes du processus d'analyse (prétraitement et normalisation des données, entrainement du modèle). Elle contient par ailleurs une fonction principale **Main.m** qui exécute le modèle avec tous les autres fonctions appelées.

Le jeu de données est divisé en deux parties. Les données de test qui représentent **10%** et les **90%** qui restent, sont consacrées aux données d'entrainement.

4.3.2.1 - Conversion du jeu de données CSV en fichier Matlab (.mat)

Dans cette étape le fichier CSV généré après la phase d'extraction, est transformé en fichiers (.mat). Ces derniers sont des fichiers binaires facilitant le stockage des variables dans l'espace de travail Matlab.

Cette transformation nous a permis de créer une structure de données avec un ensemble de paramètres stockés dans une variable « **dset** ». Cette structure de données faisant référence à l'ensemble de données et à ses paramètres, est définie comme suit :

- **Set** : un paramètre qui fixe le nom du jeu de données,
- **Class** : le nombre de classes à prédire qui est 1 (présence de pattern) ou (pas) 0,
- **Xtrain** : un paramètre référençant les données d'entrainement sur lesquelles, le modèle est formé avec aussi le jeu de validation pour sélectionner ses meilleurs paramètres,
- **Xtest** : l'ensemble de données utilisées pour l'étape de la prédiction représentant 10% des données d'entrainement,
- **Dim** : le nombre de caractéristiques du jeu de données référençant le nombre de colonnes de la matrice de représentation.
- **Num_tr** : il est considéré comme le paramètre des données d'entrainement qui décrit le nombre d'observations sur lesquelles, il y'a présence de pattern sur la colonne cible (1) ou pas (0).
- **Num_tst** : même chose du côté des données de test.

Ces deux derniers paramètres sont des tableaux parcourant la colonne cible du jeu de données en codant les instances à 2 (présence de pattern) et 1 dans le cas contraire.

4.3.2.2 - Conversion de vecteurs de caractéristiques en pixels images

La transformation du jeu de données en pixels images a été l'un des principaux problèmes rencontrés dans la mise en œuvre de ce travail de recherche. En effet l'analyse et l'exploitation de la matrice de données exigeaient l'utilisation d'un réseau de neurones de convolution qui par défaut, prend des images en entrée. Les auteurs [23] ont fait cette transformation en implémentant les algorithmes de réduction de dimensionnalité (t-SNE) et de coque convexe avec Matlab. Dans notre cinquième chapitre qui porte sur l'étude expérimentale, nous présenterons les résultats de cette implémentation faite avec le langage python.

4.3.3 - Entrainement du modèle

Après avoir effectué la transformation du jeu de données brutes en pixels images, l'étape suivante était la préparation des hyperparamètres ainsi que l'entrainement du réseau de neurones de convolution. Elle s'est déroulée comme suit :

- Une fixation des paramètres d'entrainement du modèle comme la méthode de réduction de la dimensionnalité ici t-SNE, la taille maximale des pixels (120 X 120) d'entrée dans le réseau neurones de convolution, la portion des données de validation qui est de 10% et de l'initialisation aléatoire des données.
- La division de l'ensemble de données en données d'entrainement et de test.
- L'utilisation des deux types de normalisation (1) et (2) présentés dans la section 4.2.2.
- L'initialisation des variables d'optimisation du modèle.

Le modèle de réseau de neurones de convolution est optimisé en ajustant le niveau de régularisation (L2), les intervalles de taille des filtres et l'utilisation de l'optimiseur du momentum. Ce dernier est une descente de gradient stochastique permettant d'accélérer les vecteurs de gradient dans les bonnes directions (sens vers lesquels les valeurs des paramètres du modèle seront modifiées pour minimiser la fonction d'erreur). Cette partie d'optimisation initialise aussi le taux d'apprentissage sur un intervalle bien défini. Les instructions de code Matlab suivantes montrent l'implémentation de l'initialisation des paramètres d'optimisation.

% change parameters as desired

optimVars = [

- optimizableVariable('filterSize',[2 10],'Type','integer')
- optimizableVariable('filterSize2',[4 30],'Type','integer')
- optimizableVariable('initialNumFilters',[2 16],'Type','integer')
- optimizableVariable('InitialLearnRate',[1e-5 1e-1],'Transform','log')
- optimizableVariable('Momentum',[0.8 0.95])
- optimizableVariable('L2Regularization',[1e-10 1e-2],'Transform','log')];

4.4 - Conclusion

Dans ce chapitre, nous avons introduit le problème d'évaluation directe causé par le nature des données à la suite de la construction de la représentation matricielle du couplage. Cela nous a poussé à orienter nos recherches vers la transformation des donnes brutes en pixels images qui seront analysées par le modèle de réseau de neurones de convolution. L'approche trouvée pour résoudre cette problématique exige une transformation des données brutes en pixels images basée une technique de réduction de dimensionnalité (t-SNE). Il s'en est suivi deux formes de normalisation (1) et (2) fondées respectivement sur l'indépendance et la forme des caractéristiques avant de fournir ces données aux réseaux de neurones de convolution pour l'analyse et la classification. Nous avons terminé ce chapitre par une implémentation de cette approche d'analyse.

Dans le début du chapitre suivant, une étude expérimentale est faite sur l'utilisation de cette représentation du couplage pour la détection de patterns de conception à travers le modèle d'analyse présenté dans ce chapitre.

CHAPITRE 5. ÉTUDE EXPÉRIMENTALE

Dans cette partie, nous allons faire une présentation des concepts théoriques utilisés avec l'approche d'analyse DeepInsight présentée dans le chapitre précèdent. Ces concepts sur lesquels nous nous sommes basés pour mener nos expérimentations recourent à des techniques statistiques et mathématiques que nous présenterons brièvement. A la suite de la présentation de ces concepts théoriques, les résultats de l'application de la technique de réduction de dimensionnalité (t-SNE) sur les données brutes sont présentés ainsi que ceux de l'entrainement du modèle de CNN. Les résultats de l'entrainement de ce modèle sont interprétés et comparés à ceux obtenus avec l'outil de détection.

5.1 - Concepts théoriques de la méthodologie DeepInsight

5.1.1 - La technique de réduction de dimensionnalité

Elle consiste à réduire le nombre de fonctionnalités ou caractéristiques tout en conservant un maximum d'informations. L'idée de cette technique est de faciliter la compression des données en éliminant celles qui sont redondantes. Elle permet aussi de réduire la dimensionnalité des données en 2D ou 3D. Cette réduction de dimensionnalité était la problématique de l'interprétation de la donnée matricielle que nous avons construite dans la phase d'extraction des données. En effet nous avons voulu construire une composante de métrique (métrique synthétique) regroupant les relations d'héritage, d'agrégation, les appels de méthodes entre une classe appelante (C_1) et appelée (C_2) ainsi que leurs complexités cyclomatiques. Dans chaque coordonnée formée par ce couple de classes, nous avons une donnée ou une métrique multidimensionnelle que nous devons convertir en image en implémentant cette technique de réduction de dimensionnalité.

Différentes méthodes se basant sur cet algorithme sont définies dans cette approche DeepInsight pour gérer ce problème de transformation des données.

5.1.1.1 - Analyse en Composantes Principales (ACP)

C'est une technique standard d'analyse des données consistant à transformer des variables corrélées en variables décorrélées ou indépendantes les uns des autres. C'est une approche géométrique où les variables sont représentées dans un espace portant des axes principaux indépendants pouvant expliquer la variabilité des données. Elle est appliquée sur un ensemble de N variables aléatoires X_1, X_2, ……, X_N connues à partir d'un échantillon de K réalisations structuré dans une matrice à K lignes et N colonnes comme telle :

$$M = \begin{bmatrix} x_{11} & x_{1N} \\ x_{KN} & x_{kN} \end{bmatrix} \quad (1)$$

Cette technique d'analyse factorielle vise trois principaux objectifs [26] que sont :

- Comprendre la structure de l'ensemble des variables notamment celles qui sont associées.
- Concevoir et raffiner des outils de mesures comme les tests psychométriques et les questionnaires basés sur des échelles de type Likert [27] permettant de mesurer des construits latents (qu'il est impossible de mesurer directement)
- Condenser l'information à l'intérieur d'un grand nombre de variables (d'un test) en un ensemble limite de nouvelles dimensions composites tout en assurant une perte minimale d'informations.

Les tableaux suivants décrivent sa démarche d'analyse divisée en trois étapes :

- L'étape de la détermination de l'approche selon le type problème

Approche exploratoire	Approche confirmatoire
Exploration de la structure causée par le manque de théorie sur la structure sous-jacente des données en identifiant ce dernier ou en réduisant le nombre de variables en quelques facteurs.	Présence de théories et un désire de confirmer une structure factorielle documentée.

Tableau 7 : Analyse de l'ACP selon le type de problème.

- Préparation de l'analyse

Nombre de variables	Types de variables	Taille de l'échantillon
Pour un ACP, il faudra un nombre minimum de variables subséquentes.	Il est recommandé d'avoir des variables continues même si quelques variables de l'ensemble peuvent être dichotomiques.	Il est recommandé d'avoir aussi un échantillon grand pour assurer une puissance statistique minimale.

Tableau 8 : Étape de la préparation de l'analyse d'un ACP.

- Respect des postulats

Corrélation en Items	Mesure de l'adéquation de l'échantillonnage	Test de sphéricité de Bartlett
S'assurer qu'il existe une corrélation minimale entres les variables qui feront l'objet d'analyse	Elle fournit un aperçu global de la corrélation entre les items. Elle varie entre 0 et 1 et communique une information complémentaire à l'examen de la matrice de corrélation	Une mesure qui indique si la matrice de corrélation est une matrice identité à l'intérieur de laquelle toutes les corrélations sont égales à 0.

Tableau 9 : Étape du respect des postulats d'un ACP.

- Choix de la méthode d'extraction

Choix de l'analyse des facteurs communs	Choix de l'analyse en composantes principales
Un choix basé sur la variance commune partagée par les variables analysées et est appropriée lorsque le chercheur découvre la structure latente ou les construits sous-jacents des variables.	Un choix basé sur la variance spécifique des variables et permet d'extraire un minimum de facteurs qui expliquent la plus grande partie possible de cette variance spécifique.

Tableau 10 : Les choix de méthode d'extraction d'un ACP.

5.1.1.2 - Analyse en Composantes Principales du noyau (Kernel ACP)

C'est une extension de l'analyse en composantes principales qui utilisent les techniques des méthodes du noyau. Elle consiste à projeter les données par l'intermédiaire d'une application non linéaire dans un espace de dimension élevée dénommé espace des caractéristiques ou l'ACP linéaire est appliquée. L'astuce du noyau (Kernel) s'utilise dans un algorithme qui ne dépend que du produit scalaire entre deux vecteurs d'entrée **X** et **Y.** Après passage à un espace de redirection ou espace de grande dimension où la méthode linéaire peut être utilisée et obtenir de grandes performances, l'algorithme n'est plus dépendant que du produit scalaire. Cela s'explique du fait que ce produit scalaire est utilisé dans un espace de grande dimension pouvant mener à des calculs qui pourraient être irréalisables dans la pratique, d'où l'idée d'une fonction noyau qui remplace le produit scalaire et qui est de cette forme :

$$k(X, Y) = \phi(X_i) \cdot \phi(Y_i) \quad (2)$$

Dans la réalisation de cela, le théorème de Mercer [28] est utilisé montrant une fonction noyau K continue et symétrique. Cette fonction peut maintenant s'exprimer comme un produit scalaire dans un espace de grande dimension.

Les avantages de cette astuce du noyau permettent un remplacement du produit scalaire par une fonction noyau dans un espace de grande dimension facile à calculer d'une transformation linéaire en un classifieur non linéaire. Il n'est pas nécessaire d'expliciter la transformation $\emptyset$ car elle peut transformer l'espace d'entrée en un espace de re-description infini.

5.1.1.3 - Intégration de voisin stochastique distribué (t-SNE)

C'est une technique de visualisation des données de grande dimension en fournissant à chaque point un emplacement dans un espace à deux ou trois dimensions. Sa différence avec l'ACP est qu'il maintient les représentations de faible dimension de points de données qui sont similaires les uns des autres sur une variété non linéaire de faible dimension.

Dans la mise en œuvre de cette méthodologie de transformation des données (DeepInsight), les auteurs ont utilisé cette technique de réduction de dimensionnalité (de même que aussi KPCA présentée dans la section précédente) dans leur étape de transformation de données génomiques, ou des données différenciant les phénotypes ou les catégories.

L'algorithme comprend trois étapes :

Dans la première étape, il calcule les similarités des points dans l'espace initial en grande dimension. Pour chaque point $\mathbf{X_i}$, une distribution gaussienne est centrée sur ce point. Pour chaque point $\mathbf{X_j}$ (avec j diffèrent i), une mesure de la densité sous cette distribution est appliquée. Ces points sont normalisés d'où sont notées une liste de probabilités conditionnelles définie comme suit :

$$pj|i = \frac{exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\varepsilon_{k\neq i}\, exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \quad (3)$$

L'écart type des distributions gaussiennes définies pour chaque point $\mathbf{X_i}$ est estimé par une valeur fixée par l'utilisateur à l'avance.

Une deuxième étape qui consiste à construire un espace de plus petite dimension dans laquelle sont représentées les données, qui sont reparties aléatoirement sous formes de points. Comme à l'étape première, le calcul des similarités des points se fait dans un nouvel espace mais en utilisant une distribution **t-student** [29] et non pas gaussienne. De la même manière, la liste de probabilités notées est obtenue comme suit :

$$Qij = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\varepsilon_{k\neq m}(1 + \|y_k - y_m\|^2)^{-1}} \quad (4)$$

Dans la troisième étape, la mesure Kullback Leibler (KL) [30] est utilisée pour comparer les similarités obtenues dans les deux espaces (faible et grande dimension). Dans l'espace de petite dimension, la descente de gradient est minimisée pour obtenir les meilleurs $\mathbf{Y_i}$ possibles.

5.1.2 - L'algorithme couverture convexe de Kirkpatrick-Seidel

C'est un algorithme qui calcule le coque convexe (une intersection de tous les ensembles convexes contenant un sous ensemble donné d'un espace euclidien) d'un ensemble de points dans un plan Ø (n log h) avec :

- n le nombre de points d'entrée,
- Et h le nombre de points maximaux ou dominés dans la coque.

Il divise l'entrée en trouvant la médiane des cordonnées **X** des points d'entrée et inverse l'ordre des étapes suivantes :

- La recherche de bord de la coque convexe qui croise la ligne verticale définie par cette coordonnée **X** médiane,
- Les points sur les côtés gauche et droit de la ligne de séparation qui ne peuvent pas contribuer à la coque éventuelle sont écartés.
- Sa façon récursive sur les points restants.

5.2 - Présentation des résultats et discussion

5.2.1 - Description des données

5.2.1.1 - Représentation multidimensionnelle des données

Dans cette partie, une représentation multidimensionnelle des données a été réalisée en se basant sur la technique de réduction de dimensionnalité t-SNE utilisée dans l'approche DeepInsight. Nous avons eu à faire le choix d'implémentation de cette technique non linéaire par rapport aux techniques linéaires comme l'ACP. Ce dernier se concentre sur le placement des points de données dissemblables éloignées dans une représentation de dimension inferieure. Or la similarité des points de données est importante dans notre approche de représentation matricielle du couplage pour trouver la structure des liens entre les classes logicielles. Ce qui n'est pas le cas pour les algorithmes de réduction de dimensionnalité linéaire. Notre choix a donc porté sur la technique du t-SNE que nous avons implémenté avec le langage python en lui fournissant les données

brutes de la représentation matricielle. Comme définie dans la partie théorique, cet algorithme nous a permis de faire un regroupement des points ou coordonnées similaires pour trouver les corrélations entre ces différentes coordonnées de la matrice multidimensionnelle. Ainsi à la suite de cette étape, notre modèle de réseaux de neurones de convolution pourra analyser ces données converties en images. L'idée de cette analyse est de capturer la structure des liens entre les classes et procéder à la classification pour le cas pratique de détection de design patterns que nous voulons appliquer dans ce mémoire.

La figure suivante montre l'application de la technique de réduction de dimensionnalité t-SNE avec les données brutes du système Apache POI. Ces représentations ont été réduites dans un espace à deux dimensions.

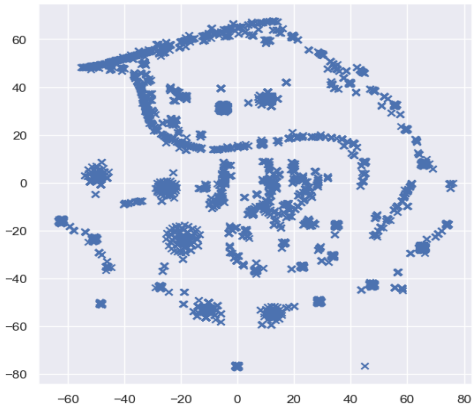

Figure 5 : Application de la technique de réduction de la dimensionnalité du t-SNE sur une partie des données brutes du système Apache POI.

Dans ce système écrit en Java (Apache POI) de plus de 8235 classes, nous avons extrait un nombre de liaisons (5344) entre ces classes que nous avons enregistrées dans une base de données MySQL avec les tables classe appelante, classe appelée et la table d'association qui, définie la matrice de données. Les données de cette table d'association sont étiquetées (voir la section 3.2.4) et exportées dans un fichier CSV sur lequel, la technique de réduction de dimensionnalité t-SNE est appliquée.

5.2.1.2 - Conversion des coordonnées en pixels images

A la suite de ce regroupement des cordonnées similaires, s'en est suivie une conversion de ces coordonnées en pixels images qui est présentée dans la figure suivante.

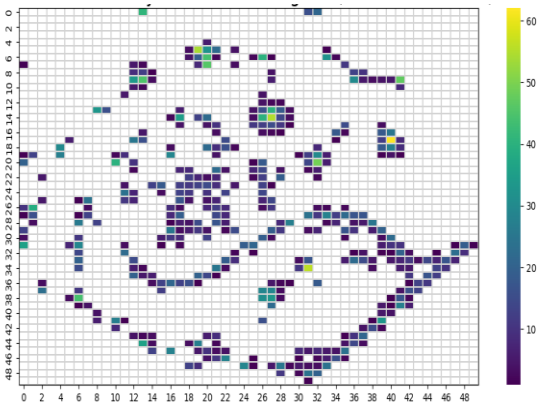

Figure 6 : Transformation des coordonnées en pixels images sur une partie des données brutes du système Apache POI.

A travers cette représentation, nous pouvons voir la disposition des pixels du format image avec une résolution de 50 * 50 pixels. Cette représentation est tirée de l'ensemble d'entrainement sur lequel, un système d'encodage et de normalisation des fonctionnalités sont appliqués.

5.2.2 - Entrainement du modèle

Après avoir fait la représentation des données du format brute en pixels images, nous avons pu résoudre la problématique de conversion des données abordée dans l'introduction de ce travail de recherche. Et cela grâce aux techniques de réduction de dimensionnalité qui sont utilisés dans l'approche DeepInsight.

La seconde partie concerne l'analyse de ces images issues de la représentation matricielle par un réseau de neurones de convolution. Cette analyse est un processus d'extraction des pixels permettant de voir les liens entre les classes logicielles mais aussi leur structure en eux. A travers cette analyse, notre modèle de réseaux de convolution devrait classifier quelques patterns de conception qui sont extraites avec l'outil de détection de pattern (section 3.2.3). Cette classification de patterns nous permettra dans notre cas

pratique de détection de patterns de montrer si la représentation matricielle des métriques de couplage permet de voir certains patrons de conception contenus dans les codes sources logicielles.

Pour cela, nous avons associé la matrice de représentation multidimensionnelle à chaque type de pattern dans l'apprentissage supervisée (section 3.2.4) et la fournir en entrée au réseau de neurones de convolution pour la classification. Cette classification est binaire (1 présence du pattern) ou 0 (dans le cas contraire).

5.2.2.1 - Résultats de l'entrainement du modèle

Dans cette phase d'entrainement des données issues de la matrice image obtenue par le modèle de réseau de neurones de convolution (CNN), nous avons utilisé l'architecture CNN présentée avec cette approche DeepInsight. Ces données sont générées en exportant la représentation matricielle obtenue à partir d'une table de la base de données MySQL dans un fichier CSV après étiquetage (section 3.2.4). Chaque ligne ou observation du fichier CSV correspond à une entrée sur le CNN pour détecter la présence ou pas du type de pattern associé. Comme présenté dans la section 4.2.3, dans chaque bloc de couches convolutives du CNN, les images ont été filtrées avec des filtres ayant des tailles variantes entre 2 et 30. Le nombre d'itérations a été fixé à **50** et le jeu de donnes a été divisé en données d'entrainement et de test avec les pourcentages respectifs de **90%** et de **10%**. Un ensemble de données de validation de **10%** a été aussi prélevé sur les données d'entrainement pour ajuster les paramètres du modèle. Nous avons utilisé le système Java Apache POI qui contenait beaucoup de classes et liaisons et qui nous a servi d'ensemble de données dans les étapes de conversion des données et d'entrainement du modèle CNN. Cet entrainement s'est fait sur les deux formes de normalisation 1 et 2 présentées dans la section 4.2.2. Le choix des hyperparamètres reste le même avec ceux utilisés avec l'approche DeepInsight.

Dans les sections suivantes, nous allons présenter les résultats obtenus sur chaque type pattern avec les deux formes de normalisation et nous allons comparer les patterns détectés suivant la classification avec ceux de l'outil de détection (section 3.2.3). Cette

comparaison permettra d'évaluer la représentation matricielle des métriques de couplage qui a été construite dans la mise en œuvre de ce travail de recherche à détecter les patterns (dans notre cas pratique) comme le fait l'outil de détection.

5.2.2.1.1 Pattern Stratégie

- Première normalisation

Avec ce modèle de conception, de meilleures étapes d'entrainement sont obtenues à la première (1er), troisième (3eme) et vingt-sixième (26eme) itération. La fonction d'erreur observée à la deuxième meilleure itération est de **0,017** tandis que celle de la première et troisième sont de **0,005** et dans l'ensemble des itérations la valeur de l'estimation de cette fonction est de **0,001**.

Pour la valeur estimée, la fonction d'erreur à la deuxième meilleure itération est de **0,001** tandis que celle de la première et troisième sont de **0,005**.

Le tableau suivant donne une aperçue des statistiques d'entrainement du modèle CNN avec ce pattern.

Itération	Résultat	Erreur observée	Erreur estimée	Filtre Size1	Filtre Size2	InitialLearning Rate	Momentum	L2 Régularisation
1ere	Best	0.005	0.005	5	6	0.0002	0.94	3.76
3eme	Best	0.017	0.001	9	27	0.03	0.93	0.0005
26eme	Best	0.005	0.005	6	6	0.02	0.86	9.76
Valeur estimée de la fonction d'erreur sur l'ensemble des itérations : **0.001** Erreur de test **: 0.105** Accuracy : **0.89**								

Tableau 11 : 1er normalisation Pattern stratégie : Apache POI.

- Deuxième normalisation

Itération	Résultat	Erreur observée	Erreur estimée	Filtre Size1	Filtre Size2	InitialLearning Rate	Momentum	L2 Régularisation
1ere	Best	0.013	0.013	5	6	0.0002	0.94	3.7669e-09
3eme	Best	0.012	0.012	9	5	0.03	0.93	0.0005
6eme	Best	0.008	0.011	3	6	0.044	0.94	9.7684e-07
Valeur estimée de la fonction d'erreur : **0.0104** Erreur de test **: 0.11** Accuracy : **0.88**								

Tableau 12 : 2eme normalisation Pattern stratégie : Apache POI.

5.2.2.1.2 Pattern Adaptateur

- Première normalisation

Itération	Résultat	Erreur observée	Erreur estimée	Filtre Size1	Filtre Size2	InitialLearning Rate	Momentum	L2 Régularisation
1ere	Best	0.005	0.005	9	23	0.0003	0.86	5.0147e-06
3eme	Best	0.003	0.005	3	20	0.09	0.81	1.1681e-05
20eme	Best	0.001	0.001	8	10	0.02	0.85	1.4807e-10
Valeur estimée de la fonction d'erreur : **0.001** Erreur de test **: 0.15** Accuracy : **0.85**								

Tableau 13 : 1ere normalisation Pattern Adaptateur : Apache POI.

- Deuxième normalisation

Itération	Résultat	Erreur observée	Erreur estimée	Filtre Size1	Filtre Size2	InitialLearning Rate	Momentum	L2 Régularisation
1ere	Best	0.015	0.015	9	23	3.9196e-05	0.86	3 5.0147e-06
2eme	Best	0.006	0.007	3	20	0.003	0.81	1.1681e-05
27eme	Best	0.003	0.003	7	5	0.04	0.91	1.0035e-09
Valeur estimée de la fonction d'erreur : **0.010** Erreur de test **: 0.17** Accuracy : **0.83**								

Tableau 14 : 2eme normalisation Pattern Adaptateur : Apache POI.

5.2.2.1.3 Pattern Singleton

- Première normalisation

Itération	Résultat	Erreur observée	Erreur estimée	Filtre Size1	Filtre Size2	InitialLearning Rate	Momentum	L2 Régularisation
1ere	Best	0.003	0.003	3	25	0.0005	0.94	1.999e-05
2eme	Best	0.001	0.001	4	9	0.002	0.93	0.008
27eme	Best	0	0.025	2	4	0.009	0.93	7.4253e-09
Valeur estimée de la fonction d'erreur : **0.002** Erreur de test **: 0.05** Accuracy : **0.95**								

Tableau 15 : 1ere normalisation Pattern singleton : Apache POI.

- Deuxième normalisation

Itération	Résultat	Erreur observée	Erreur estimée	Filtre Size1	Filtre Size2	InitialLearning Rate	Momentum	L2 Régularisation
1ere	Best	0.003	0.003	3	25	0.0005	0.94	1.999e-05
2eme	Best	0.001	0.002	10	27	0.006	0.92	6.4394e-08
21eme	Best	0.003	0.001	7	9	0.008	0.9	1.0035e-09
Valeur estimée de la fonction d'erreur : **0.010** Erreur de test **: 0.066** Accuracy : **0.93**								

Tableau 16 : 2eme normalisation Pattern singleton : Apache POI.

5.2.2.2 - Interprétation des résultats

Au vu de la présentation des résultats obtenus sur l'entrainement du modèle, l'analyse de notre approche de représentation matricielle des métriques de couplage montre les statistiques d'entrainement sur la classification des différents patterns de conception. Sur l'évaluation de l'ensemble de test qui contenait **645** liaisons entre classes, **19** instances du pattern Adaptateur ont été détectées. Il en est suivi des modèles de conception Observateur et Stratégie respectivement avec **12** et **7** instances et de Singleton avec **2** instances.

Le tableau suivant affiche les instances de patterns détectées avec la représentation matricielle comparées à celles avec l'outil de détection sur lequel nous nous sommes basés pour extraire ces patterns. La figure 8 montre aussi l'affichage des instances de patrons de conception extraites dans les codes sources par l'outil de détection.

Patterns	**Détection avec l'outil**		**Détection avec la représentation matricielle**	
	Instances détectées	**Nombre d'observations**	**Instances détectées**	**Nombre d'observations**
Singleton	21	5806	2	645
Adaptateur	109	5806	19	645
Stratégie	55	5806	7	645
Observateur	41	5806	12	645
Pont	4	5806	1	645
Total	230	5806	41	645

Tableau 17 : Comparaison des instances de patterns détectées avec la représentation matricielle et l'outil de détection.

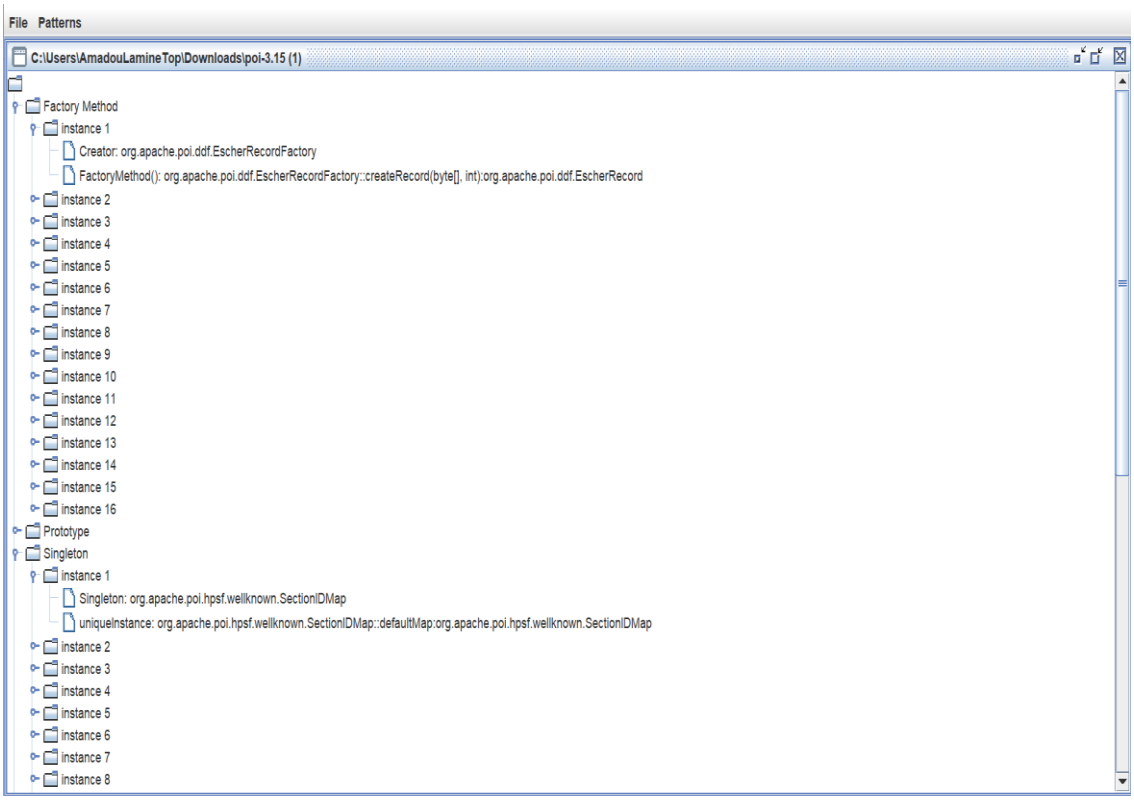

Figure 7: Interface principale de l'outil de détection de patterns de conception appliqué sur le Système Apache POI.

5.3 - Conclusion sur les résultats obtenus

Après une interprétation des résultats obtenus, nous avons remarqué que l'évaluation des métriques orientées objet relatives au couplage a permis de détecter un nombre de patterns de conception extraits dans les codes sources des programmes par l'outil de détection. Cependant le nombre de patterns détectés est faible par rapport à celui obtenu avec cet outil. En se basant sur l'objectif de ce travail de recherche qui était de mettre en œuvre un tel modèle de représentation des métriques de couplage (format image de convolution) pour capturer la structure du couplage et d'héritage, l'analyse et l'évaluation des métriques présentent un faible pourcentage de détection. Cela nous a permis de conclure que cette forme de représentation matricielle ne fournit pas intégralement les traces des patterns de conception trouvées dans cet outil de détection. Ce qui pourrait limiter son utilisation sur d'autres cas pratiques d'application comme la

détection de fautes dans les classes logicielles ou l'identification de certaines erreurs architecturales de conception. Par ailleurs, cette forme de représentation des métriques de code source a permis de capturer les liaisons de couplage, d'héritage. Elle peut être comparée à d'autres formes de représentation des métriques qui prend aussi en compte de la structure des liens entre les classes. En effet certains modèles d'analyse et d'évaluation des métriques de code source de couplage ne permettent pas de capturer cette structure. C'est la raison pour laquelle nous avons opté pour un modèle de réseau de neurones de convolution avec la représentation multidimensionnelle en construisant cette matrice dont chaque liaison a la forme de coordonnées (x, y, r, g, b,).

Avec cette forme de représentation matricielle des métriques, nous avons pu voir que le couplage peut être capturé en tenant en compte de l'aspect structurel des liaisons entre les classes logicielles. Cette matrice de données a été exploitée et analysée grâce aux techniques de conversion des données (coordonnées en pixels images) mais aussi aux réseaux de neurones de convolution. Ces derniers ont permis une analyse en profondeur de la représentation matricielle pouvant détecter des patterns de conception contenus dans les codes sources logicielles.

CHAPITRE 6. MENACE A LA VALIDITÉ

Les résultats obtenus dans ce travail de recherche sont à titre démonstratifs sur cette approche de représentation matricielle des métriques du couplage. Cependant certaines facteurs ou considérations sont à prendre en compte et peuvent invalider ces résultats.

6.1 - La validité interne

Dans notre nouvelle démarche, nous avons opté pour la représentation matricielle du couplage. Cette représentation est ensuite convertie sous forme d'une image à l'aide des regroupements des coordonnées de la matrice. Cette transformation pourrait causer une perte de données qui modifie la structure des liens entre les classes logicielles. Or cette structure constitue un concept clé dans l'analyse de cette forme de représentation car elle permet de détecter un type de pattern formé par un ensemble de liaisons entre ces classes.

La problématique de conversion des données brutes en images a une influence sur l'architecture des liaisons entre les composants. Pour cela nous devons encore réorienter nos recherches pour trouver des techniques de transformation de données qui pourraient remédier à ce problème afin de garantir que notre modèle d'apprentissage profond évaluera la composante de métrique matricielle du couplage en tenant compte de cet aspect structurel.

6.2 - La validité externe

Dans l'expérimentation de ce travail de recherche, nous avons utilisé le système Apache POI pour construire cette forme de représentation matricielle. Nous aurions pu aussi appliquer cette dernière sur d'autres langages orientés objet comme Python pour voir si la structure du couplage reste favorable à la détection de patterns.

Une généralisation des systèmes sur lesquels sont basés l'entrainement et l'évaluation du modèle pour la détection de patterns de conception pourrait être étendue sur d'autres systèmes logiciels orientés objet autres que Java.

6.3 - La validité de construction

Dans cette nouvelle forme de représentation du couplage, nous avons utilisé les métriques de code source notamment les métriques CK (CBO, RFC, DIT). Nous nous sommes limités donc à ses métriques pour construire cette nouvelle forme de représentation. Nous avons utilisé un outil de détection de pattern de conception à partir des classes Java compilées et nous l'avons comparé aux patterns détectés par le modèle d'apprentissage à travers la représentation matricielle. Cependant, il y'a d'autres mesures comme les métriques QMOOD qui sont des métriques de conception orientées objet et qui pourraient être utilisées avec cette nouvelle approche matricielle.

CHAPITRE 7. CONCLUSION GÉNÉRALE

7.1 - Rappel de la problématique

Dans ce mémoire, nous avons construit une représentation matricielle et multidimensionnelle du couplage. Pour se faire, un outil d'extraction des interactions des classes logicielles pour les projets en Java a été développé. Chaque cellule de la matrice représente plusieurs types de liaisons entre la composante appelante et appelée. Elle est composée d'un ensemble de mesures tirées du code source. Nous avons utilisé cette nouvelle représentation dans la détection de patrons de conception dans un système orienté objet (Apache POI). Cette matrice devrait tenir compte de la structure des liens entres les classes.

Un outil de détection de patterns tiré de la littérature nous a permis d'associer chaque type de pattern aux classes qui les forment. Cet outil nous a donc permis d'étiqueter l'ensemble de données.

Cet ensemble de données (données brutes) est converti en images. Cette transformation a été réalisée et a servi d'entrée à un modèle d'apprentissage profond (réseaux de neurones de convolution). Le choix de cette approche est dû à la problématique de conversion des données brutes en images pour les réseaux de convolution.

Nous avons pu comparer les patterns détectés par notre modèle d'apprentissage à ceux extraits par l'outil de détection de pattern trouvé dans la littérature.

Par la suite, à travers les résultats d'analyse et d'évaluation des métriques de couplage, nous avons pu conclure que notre approche de représentation matricielle ne détectait pas toutes les empreintes des patterns fournis dans cette outil de détection.

7.2 - Contributions

Les résultats obtenus avec le système POI ont montré un faible taux de détection de patterns même si la matrice de données prend en compte la structure des liens entre les classes logicielles.

Ce travail de recherche nous a donné la possibilité de construire une nouvelle forme de représentation des métriques de couplage orientées objet, d'adapter un modèle de réseaux de neurones de convolution (modèle pré-entraine) à des images de la représentation matricielle des métriques des systèmes orientés objet Java, de détecter des patterns de conception et d'identifier différentes formes de couplage.

7.3 - Recherches futures

Les résultats obtenus dans ce travail de recherche ouvrent des possibilités d'appliquer cette nouvelle forme de représentation matricielle à d'autres métriques relatives à certains autres concepts comme la cohésion, la taille, la complexité, etc. Ce sera notre prochain axe de recherche.

CHAPITRE 8. RÉFÉRENCES BIBLIOGRAPHIQUES

[1] Chidamber and C.F. Kemerer, A Metrics Suite for Object-Oriented Design, IEEE on Transactions Software Engineering, pp. 476-493, 1994.

[2] Edward Yourdon and Larry L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and System Design,* Prentice-Hall, 1979.

[3] Hakim Lounis, Houari et Walcelio, A Concept Formation Based Approach to Object Identification in Procedural Code*, Automated Software Engineering 6, pp. 387-410, 1997.

[4] V.R. Basili, L.C. Briand and W.L. Melo, A validation of object- oriented design metrics as quality indicators, IEEE Transactions on Software Engineering, 22, pp. 751–761, 1996.

[5] A.S. Chauhan and S.K. Dubey, Analytical Review of Fault- proneness for Object Oriented Systems, International Journal of Scientific and Engineering Research, Volume 3, Issue 12, December 2012.

[6] Brito e Abreu et Carapu, Formalizing Object-Relational Structural Metrics, 5eme conférence de l'association portugaise des systèmes d'information (CAPSI'2004), 1994.

[7] L. Briand, P. Devanbu and W. Melo, An investigation into coupling measures for C++, in: International Conference on Software Engineering, Association for Computing Machinery, pp. 412–421, 1997.

[8] Price, M. and Demurjian, S., Analysing and measuring reusability in object-oriented design, in proceedings OOPSLA'97, Atlanta, USA, 1997.

[9] Edward Yourdon, The Practical Guide to Structured Systems Design, Page-Jones, Yourdon Press, New York 1980.

[10] J. Offutt, M. J. Harrold and P. Kolte, A Software Metric System for Module Coupling, The Journal of Systems and Software, March 1993.

[11] K.K. Aggarwal, Y. Singh, A. Kaur and R. Malhotra, Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study, Software Process Improvement and Practice, 2009.

[12] H.M. Kaung, Nan Si Kham and Ni Lar Thein, To Visualize the Coupling among modules, journal 6th Asia-pacific Symposium on information and Telecommunication Technologies, 2005.

[13] H. Y. Yang, E. Tempero et R. Berrigan, Detecting Indirect Coupling, Brisbane QLD Australia, IEEE, April 2005.

[14] Widad Chami, Analyse de l'évolution des concepts de cohésion et de couplage, pp. 112-118, Mémoire présenté à l'Université du Québec à Montréal comme exigence partielle de la maitrise en Informatique, 2008.

[15] Eclipse Foundation, www.eclipse.org, 2004.

[16] S.S. Rathore and A. Gupta, Investigating object-oriented design metrics to predict fault-proneness of software modules, Sixth International Conference on Software Engineering (CONSEG), CSI, 1-10, 2012.

[17] PROMISE, (http://promise.site.uottawa.ca/SERepository), School of Information Technology and Engineering, University of Ottawa, 2005.

[18] THOMAS, J. McCABE, A Complexity Measure, IEEE Transactions of Software Engineering, Volume SE-2, December 1976.

[19] J. Bansiya, C.G. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering, 28, pp. 4–17, 2008.

[20] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, Design Pattern Detection Using Similarity Scoring, IEEE Transactions on Software Engineering, 2006.

[21] The Apache Software Foundation, Apache POI - the Java API for Microsoft Documents , 2022.

[22] A. Sharma, E. Vans, D. Shigemizu, T. Tsunoda, DeepInsight: A methodology to transform a non-image data to an image data for convolution neural network architecture, Scientific Reports 9 (article 11399), 2019.

[23] T. Kochi et T. Saito, «User-defined template for identifying document type and extracting information from documents,» *in ICDAR, Sep 1999, pp. 127–130.*

[24] D. Kirkpatrick et R. Seidel, The ultimate planar convex hull algorithm, Publications Library SIAM Journal on computing Vol 15, 1986.

[25] The MathWorks Inc, MATLAB - MathWorks - MATLAB & Simulink, 2022.

[26] Yergeau, E. et Poirier, Analyse en composantes principales | (usherbrooke.ca) , SPSS à l'UdeS, 2021.

[27] Juliette Caprais, Comment utiliser l'échelle de Likert dans une analyse statistique, ToutComment Rédactrice, 2017.

[28] Wikipédia article, Théorème de Mercer (wiko.wiki), The Creative Commons Attribution Share Alike 3.0, 2014.

[29] Acervo Lima, Distribution t de Student en statistiques, Engineering Mathematics, September 2021.

[30] Wikipédia article, Divergence Kullback-Leibler, The Creative Commons Attribution Share Alike 3.0, 2014.