UNIVERSITÉ DU QUÉBEC

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

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE AVEC MÉMOIRE EN MATHÉMATIQUES ET
INFORMATIQUE APPLIQUÉES

PAR
ALEXANDRE OUELLET

PRIORISATION DES TESTS LOGICIELS : UNE APPROCHE BASÉE SUR LA MAXIMISATION DE LA COUVERTURE ET LA RÉDUCTION DU RISQUE

SEPTEMBRE 2021

Résumé

L'ordonnancement et la sélection des cas de tests sont des sujets qui ont été largement étudiés en raison de l'impératif de qualité des logiciels et du coût élevé du processus de test. La majorité des approches de sélection ou de priorisation des tests ont été développées dans le contexte des tests de régression. Dans ce mémoire, nous proposons une nouvelle approche de priorisation et de sélection qui peut être appliquée dans un contexte de développement (maintenance) plus global. Le but de cette approche est de trouver un ensemble pertinent de classes candidates au test, classes sur lesquelles les efforts de test devraient être concentrées en tenant compte des diverses contraintes (temps, ressources) auxquelles fait face le processus de test. Cette approche tient compte de la couverture atteinte et du risque que chaque classe a de contenir des fautes en utilisant des algorithmes d'apprentissage automatique. Nous avons comparé l'approche développée à sept autres approches de contrôle tirées de la littérature. Nous avons utilisé cinq critères d'évaluation pour comparer l'approche développée (score locale - SL) aux approches de contrôle sélectionnées, soient : la couverture, la détection des fautes, l'effort de test, l'efficience de l'ensemble de classes candidates au test proposé et la qualité de l'ordonnancement. L'approche développée présente de meilleures performances que les sept approches de contrôle auxquelles elle a été comparée.

Mots-clés : priorisation des tests — tests logiciels — étude empirique — couverture des tests – détection des fautes — orienté objet — apprentissage automatique — automatisation des tests.

Alexandre Ouellet

Mourad Badri

Abstract

Ordering and selecting test cases are important topics that have been widely studied in literature because of the importance they have in the quality assurance process and in reducing the high cost of software testing. Most of selection or prioritization approaches have been proposed for regression testing. This memoir aims at introducing a new approach that can be applied at any stage of the software development (and maintenance) rather then regression testing only. The new approach (local score LS) aims at identifying suitable candidate classes for testing (named candidate set), knowing that software testing is often conducted under severe pressure and constraints (time and resources limitations). The introduced approach is based on code coverage and fault-proneness of classes. The fault-proneness concern is addressed by using machine learning algorithms. We compared the new approach to seven control approaches from the literature. We used five quality criteria to perform the comparison. Those criteria are code coverage, fault detection, testing effort, candidate set efficiency and ordering quality. We performed an empirical study and found that the new approach outperforms the seven control approaches.

Keywords : tests prioritization—software testing—empirical study—testing coverage—fault detection—object-oriented—machine learning—test automation.

Avant-propos

Le travail présenté dans ce mémoire est la dernière partie d'un projet de recherche d'envergure étalé sur plusieurs années. Le projet a commencé durant la période de mes études de premier cycle avec des stages de recherche subventionnés par le Conseil de recherche en sciences naturelles et génie (CRSNG) et le Fonds de recherche du Québec en sciences et technologie (FRQNT). La première partie des travaux est présentée dans [Ouellet et Badri, 2019] et porte sur l'exploration de la prédiction de classes fautives (dans les systèmes orientés objet) basée sur une combinaison des métriques orientées objet et des mesures de centralité. Au cours de la maîtrise, une seconde partie du projet a porté sur l'amélioration de la méthodologie d'une part et l'élargissement du champ de recherche d'autre part. La prédiction des fautes a été explorée sous différentes perspectives. La validation s'est effectuée sur plusieurs systèmes d'envergure repris dans ce mémoire. Les travaux effectués dans cette deuxième partie ont fait l'objet d'un article soumis au journal « Journal of Systems and Software, Elsevier » [Ouellet et Badri, 2021]. Ce mémoire (avec des objectifs spécifiques, différents de ceux des articles, représentant le but ultime du projet de recherche global) s'appuie sur des résultats importants présentés dans les deux articles mentionnés. Des résumés des travaux antérieurs sont donnés dans le mémoire lorsque jugé opportun pour expliquer notre méthodologie, faciliter la compréhension des différents travaux importants réalisés et les lier avec le sujet de ce mémoire.

Remerciements

Mes premiers remerciements iront à mon directeur de recherche, le Pr Mourad Badri. Sans son soutien pendant mes études universitaires, je ne serais pas devenu l'enseignant en informatique que je suis aujourd'hui. J'espère pouvoir transmettre avec autant de passion mes connaissances en informatique aux prochaines générations d'étudiant.e.s en informatique.

Je tiens également à remercier les professeurs et chargés de cours du département de mathématiques et d'informatique pour la formation reçue. J'en profite également pour remercier les dépanneurs et assistants d'enseignement avec qui j'ai eu la chance d'échanger. J'aimerais de même remercier les amis que j'ai rencontrés durant mon parcours et qui ont marqué mon cheminement universitaire chacun à sa manière.

Je souhaite aussi remercier tous les étudiants et membres du personnel de l'UQTR qui ont croisé mon chemin lors de mes implications étudiantes, syndicales et des projets étudiants que j'ai menés, ainsi que mes collègues exécutants de l'AMI (2014-2021) et de l'ASTRE (2018-2021).

Je souhaite remercier le Conseil de recherche en sciences naturelles et génie du Canada (CRSNG), le Fonds de recherche du Québec en sciences et technologies (FRQNT) ainsi que Mme Corina Risher, par l'entremise de la Fondation de l'UQTR, pour le soutien financier accordé m'ayant permis de mener à terme ma maîtrise.

Un gros merci à mes parents, Nathalie et Eric, pour leur soutien et encouragement ainsi qu'à mes soeurs Mélissa et Marie-Pier.

Table des matières

Résumé III

Abstract V

Avant-propos VII

Remerciements IX

Table des matières XI

Liste des tableaux XVII

Table des figures XIX

Liste des symboles XXI

Liste des abréviations XXIII

1 Introduction 1
- 1.1 Objectif . . . 2
- 1.2 Questions de recherche . . . 2

2 Cadre théorique 5
- 2.1 Priorisation et sélection des tests . . . 5
- 2.2 Représentation d'un logiciel . . . 8
  - 2.2.1 Graphe de dépendance . . . 8
  - 2.2.2 Mesures de centralité . . . 10
    - Centralité de proximité . . . 10
    - Centralité KATZ . . . 10
    - Centralité semilocale . . . 11
  - 2.2.3 Métriques orientées objet . . . 11

Complexité pondérée des méthodes 12
Couplage entre les objets 12
Couplage sortant 12
Lignes de code 12
Nombre de réponses pour la classe 13
Taille de l'interface de la classe 13
2.3 Conclusion du chapitre 2 13

3 Cadre méthodologique **15**
3.1 Adaptation du problème 15
3.2 Description des procédures statistiques 16
3.2.1 Test sur les moyennes de deux échantillons appariés 17
3.2.2 Test de Pitman sur les variances de deux échantillons appariés . 18
3.2.3 Corrélation linéaire 18
3.2.4 Régression linéaire 19
3.3 Description des approches de contrôle 21
3.3.1 Séquence aléatoire 21
3.3.2 Algorithmes gloutons 22
3.3.3 Descente de gradient 23
3.3.4 Algorithmes génétiques 24
3.3.5 Algorithme de classification hiérarchique 26
3.3.6 Résumé des approches de contrôle 28
3.4 Approche de priorisation développée — Score local 28
3.4.1 Algorithmes d'apprentissage automatique 30
K plus proches voisins 32
Forêt aléatoire 33
Sélection des métriques et mesures utilisées 33
Sur-échantillonnage 34
Variantes des algorithmes 34
3.5 Métriques d'évaluation 36
3.5.1 Couverture 38
3.5.2 Détection des fautes 39
3.5.3 Efficience de la détection 40
3.5.4 Qualité de l'ordonnancement 41
3.6 Systèmes utilisés et collecte des données 42
3.7 Conclusion du chapitre 3 45

4 Expérimentations **47**

4.1 Caractéristiques des systèmes 47

4.1.1 Distribution des métriques 47

4.1.2 Couplage et taille 48

4.2 Approches de contrôle sans variante 49

4.2.1 Couverture 51

4.2.2 Détection des fautes 54

4.2.3 Efficience de la détection 57

4.2.4 Qualité de l'ordonnancement 61

4.3 Prédiction intra-système 63

4.3.1 Approches de priorisation 63

Couverture 63

Détection des fautes 68

Efficience de la détection 71

Qualité de l'ordonnancement 75

4.3.2 Comparaison des approches de priorisation 77

Comparaison des approches développées 77

Comparaison aux approches de contrôle 79

4.4 Prédiction inter-systèmes 85

4.4.1 Approches de priorisation 85

Couverture 85

Détection des fautes 89

Efficience de la détection 92

Qualité de l'ordonnancement 97

4.4.2 Comparaison des approches de priorisation 99

Comparaison des approches développées 99

Comparaison aux approches de contrôle 100

4.5 Caractéristiques influençant notre approche de priorisation 106

4.6 Résumé des comparaisons aux approches de contrôle 107

4.7 Limitations et validité des résultats 110

4.8 Conclusion du chapitre 4 111

5 Conclusion **113**

Bibliographie **115**

A Démonstration de la moyenne de l'APFD lorsque les fautes sont distribuées uniformément 121

B Graphiques 123

Liste des tableaux

3.1 Rangs des chromosomes lors du croisement 25
3.2 Approches de contrôle 29
3.3 Variantes de l'approche de priorisation développée et de l'algorithme glouton basé sur le risque 35
3.4 Liste des métriques d'évaluation selon l'axe évalué 42
3.5 Description des systèmes étudiés 44

4.1 Statistique descriptive des métriques OO et mesures de centralité . . . 49
4.2 Corrélations entre le couplage entrant moyen et la taille des systèmes en nombre de classes 50
4.3 Couverture totale (% LOC) des approches de contrôle sans variante . . 52
4.4 Couverture directe (% LOC) des approches de contrôle sans variante . 52
4.5 Couverture indirecte (% LOC) des approches de contrôle sans variante 53
4.6 Détection des fautes directes et indirectes des approches de contrôle sans variante 54
4.7 Détection directe des fautes des approches de contrôle sans variante . . 55
4.8 Détection indirecte des fautes des approches de contrôle sans variante . 56
4.9 Nombre de fautes détectées directement et indirectement par millier de lignes de code pour les approches sans variante 58
4.10 Nombre de fautes détectées directement par millier de lignes de code pour les approches sans variante 59
4.11 Nombre de fautes détectées indirectement par millier de lignes de code pour les approches sans variante 60
4.12 Ratio de l'efficience directe par rapport à l'efficience indirecte pour les approches sans variante 60
4.13 APFD pour les approches sans variante 62
4.14 Couverture totale (% LOC) des approches intra-système 64
4.15 Couverture directe (% LOC) des approches intra-système 65
4.16 Couverture indirect (% LOC) des approches intra-système 65

4.17 Taille des ensembles de classes candidates au test pour les approches intra-système 66
4.18 Corrélations entre la différence de taille de l'ensemble de classes candidates et la taille du système (nombre de classes) par système pour les approches intra-système 67
4.19 Détection des fautes directes et indirectes dans les approches intra-système 68
4.20 Détection des fautes directes dans les approches intra-système 69
4.21 Détection des fautes indirectes dans les approches intra-système 70
4.22 Nombre de fautes détectées directement et indirectement par millier de lignes de code pour les approches intra-système 71
4.23 Nombre de fautes détectées directement par millier de lignes de code pour les approches intra-système 72
4.24 Nombre de fautes détectées indirectement par millier de lignes de code pour les approches intra-système 73
4.25 Ratio de l'efficience directe par rapport à l'efficience indirecte pour les approches intra-système 74
4.26 APFD pour les approches intra-système 76
4.27 Comparaison des approches intra-SL par métrique d'évaluation 78
4.28 Comparaison de la couverture d'intra-SL (KNN+RF) aux approches de contrôle 79
4.29 Comparaison de la taille de l'ensemble de classes candidates au test de l'approche en contexte intra-système aux approches de contrôle 80
4.30 Comparaison de la détection des fautes d'intra-SL (KNN+RF) aux approches de contrôle 81
4.31 Comparaison de l'efficience de la détection des fautes d'intra-SL (KNN+RF) aux approches de contrôle 82
4.32 Comparaison de l'APFD d'intra-SL (KNN+RF) aux approches de contrôle 83
4.33 Couverture totale (% LOC) des approches de priorisations inter-systèmes 86
4.34 Couverture directe (% LOC) des approches inter-systèmes 87
4.35 Couverture indirecte (% LOC) des approches inter-systèmes 87
4.36 Taille des ensembles de classes candidates au test pour les approches inter-systèmes 88
4.37 Corrélations entre la différence de taille de l'ensemble de classes candidates et la taille du système (nombre de classes) par système pour les approches inter-systèmes 88

4.38 Détection des fautes directes et indirectes dans les approches inter-systèmes 90
4.39 Détection des fautes directes dans les approches inter-systèmes 91
4.40 Détection des fautes indirectes dans les approches inter-systèmes 91
4.41 Nombre de fautes détectées directement et indirectement par millier de lignes de code pour les approches inter-systèmes 93
4.42 Nombre de fautes détectées directement par millier de lignes de code pour les approches inter-systèmes 94
4.43 Nombre de fautes détectées indirectement par millier de lignes de code pour les approches inter-systèmes 95
4.44 Ratio de l'efficience directe par rapport à l'efficience indirecte pour les approches inter-systèmes 96
4.45 APFD pour les approches inter-systèmes 98
4.46 Comparaison des approches développées inter-systèmes par métrique d'évaluation 101
4.47 Comparaison de la couverture d'inter-SL (KNN) aux approches de contrôle 103
4.48 Comparaison de la taille de l'ensemble de classes candidates au test de l'approche en contexte inter-systèmes aux approches de contrôle 103
4.49 Comparaison de la détection des fautes d'inter-SL (KNN) aux approches de contrôle 104
4.50 Comparaison de l'efficience de la détection des fautes d'inter-SL (KNN) aux approches de contrôle 105
4.51 Comparaison de l'APFD d'inter-SL (KNN) aux approches de contrôle . 105
4.52 Corrélation entre la taille des systèmes en nombre de classes et leurs caractéristiques 106
4.53 Résumé de la comparaison d'intra-SL (KNN + RF) aux approches de contrôle 108
4.54 Résumé de la comparaison d'inter-SL (KNN) aux approches de contrôle 109

Table des figures

3.1 Liste des approches 35
3.2 Types de couverture 38

4.1 Couplage entrant moyen et la taille des systèmes en nombre de classes . 50

B.1 Métriques d'évaluation pour les approches sans variante 124
B.2 Métriques d'évaluation pour les approches intra-SL 125
B.3 Métriques d'évaluation pour les approches inter-SL 126

Liste des symboles

$\forall$	Quantificateur universel en logique
$\exists(\nexists)$	Quantificateur existentiel en logique (négation)
$\bigwedge$	Opérateur de conjonction logique
$\bigvee$	Opérateur de disjonction logique
$\bigcap$	Opérateur d'intersection d'ensembles
$\bigcup$	Opérateur de réunion d'ensembles
$\uparrow$	Notation itérée de Knuth pour les grands nombres
$\mid$	Opérateur de caractérisation d'ensemble
$\lvert X \rvert$	La cardinalité de l'ensemble X
$\{\ \}$	Définition d'ensemble
(a, b)	Intervalle ouvert entre $a \in \mathbb{R}$ et $b \in \mathbb{R}$
$X \sim L$	La variable aléatoire X suit une loi L
$x \in X (x \notin X)$	Opérateur d'appartenance de l'élément x à l'ensemble X (négation)
$\mathrm{cov}(x, y)$	Fonction de la covariance de X et Y
$\max_{x \in X} f(s)$	Le maximum de la fonction f appliquée sur les éléments de l'ensemble X
$\bar{x}$	Valeur de la moyenne d'un échantillon de la variable X
$\hat{\theta}$	Valeur de l'estimateur du caractère θ
F_{ν_1, ν_2}	Loi de Fisher avec ν_1 degrés de liberté au numérateur et ν_2 degrés de liberté au dénominateur
$\mathcal{P}(X)$	Ensemble des permutations des éléments de l'ensemble X
$\mathbb{P}(X)$	Ensemble des parties de l'ensemble X
T_ν	Loi de Student à ν degrés de liberté
X	Variable aléatoire
x_i	$\mathrm{i}^{\mathrm{ème}}$ observation d'une variable aléatoire

Liste des abréviations

ALEA-couv	Approche de priorisation aléatoire basée sur la couverture totale
CBGA-ES	Algorithme génétique de la classification avec sélection par élitisme
CBO	Couplage entre les objets
CE	Couplage efférent
CH	Classification hiérarchique
CIS	Taille de l'interface de la classe
CL	Centralité de proximité
DG	Algorithme de descente de gradient
GL	Algorithme glouton
KATZ	Centralité KATZ
KNN	Classification des K plus proches voisins
LOC	Lignes de code
OO	Orienté objet
QR	Question de recherche
RF	Classification de forêt aléatoire
RFC	Nombre de réponses pour la classe
SL	Approche de priorisation score local
SLC	Centralité semi-locale
WMC	Complexité pondérée des méthodes

Chapitre 1

Introduction

La priorisation et la sélection des tests logiciels sont deux sujets importants largement étudiés dans la littérature du génie logiciel. Dans plusieurs travaux, les approches de priorisation des tests se limitent uniquement au contexte des tests de régression. Les tests de régression sont effectués après des changements apportés à un système (extention, maintenance et évolution) pour s'assurer que les changements n'introduisent pas des fautes et que le reste du système continue de bien fonctionner. Les tests de régression portent sur des tests déjà écrits pour des versions antérieures du système qui sont réutilisés. Ces tests sont effectivement importants en termes d'assurance qualité, mais néanmoins très coûteux, car ils demandent de tester non pas seulement les parties ayant subies des changements ou nouvellement développées, mais toutes les parties du logiciel potentiellement affectées par ces changements. Lorsqu'une équipe de développement travaille sur plusieurs parties d'un logiciel simultanément, exécuter les tests de régression peut impliquer de tester l'ensemble du logiciel, ce qui bien souvent se traduit par un test demandant un effort ou un coût trop grands pour être réalisable. C'est là qu'interviennent les techniques de priorisation des tests permettant de concentrer l'effort de test sur les éléments modifiés ou potentiellement impactés par les changements apportés. Les approches de priorisation sélectionnent donc les tests les plus critiques ou qui permettent de diversifier les parties testées afin d'atteindre une meilleure couverture des tests. Les approches de sélection, pour leur part, sélectionnent des cas de tests qui peuvent ne pas être exécutés tout en préservant la qualité des tests. Cela a pour conséquence de réduire l'effort de test, mais peut aussi retiré des cas de test qui auraient permis d'identifier une erreur.

1.1 Objectif

L'objectif principal de ce mémoire est de développer une nouvelle approche de priorisation des tests. Cette approche devra comporter les caractéristiques particulières suivantes :

1. Maximiser la couverture des tests
2. Maximiser la détection des fautes
3. Contrôler l'effort de test résultant
4. Être applicable hors du contexte des tests de régression

Comme certains des objectifs entraînent un plus grand nombre de tests (maximisation de la couverture et de la détection des fautes) et d'autres un plus petit nombre de tests (contrôler l'effort de test), nous devrons trouver un niveau acceptable de réalisation de chacun de ces objectifs. Alors, contrairement à plusieurs approches de la littérature, plutôt que de chercher à atteindre une couverture complète par exemple, nous chercherons à avoir la meilleure couverture en fixant un effort de test réaliste ou en paramétrant les approches pour qu'elles contrôlent l'effort de test résultant.

Pour le quatrième objectif, plutôt que d'utiliser l'information des cas de test déjà existants, nous utiliserons l'information directement du code source, en considérant la granularité des classes (de facto cela restreint notre étude aux systèmes orientés objet). De ce fait, nous ne chercherons pas à obtenir un ordonnancement des cas de test, mais plutôt un ordonnancement de classes candidates au test. Afin de contrôler l'effort de test, les classes candidates aux tests représentent seulement un sous-ensemble de toutes les classes du système.

1.2 Questions de recherche

Afin de vérifier l'atteinte de l'objectif principal décrit ci-dessus, nous comparerons les résultats de l'approche développée avec sept approches de contrôle de la littérature

qui seront décrites plus loin. La question de recherche centrale du mémoire est la suivante.

QR : Est-ce que l'approche développée est meilleure que les approches de contrôle ?

La réponse à cette question de recherche principale comporte plusieurs facettes, car les approches de priorisation se distinguent et peuvent être évaluées de plusieurs façons. C'est pourquoi nous apporterons une réponse à cette question à travers cinq sous-questions. Chacune est en lien avec l'évaluation d'une caractéristique requise par l'approche identifiée à la section précédente.

QR1 : Est-ce que l'approche développée propose une meilleure couverture que les approches de contrôle ?
QR2 : Est-ce que l'approche développée permet de détecter plus de fautes que les approches de contrôle ?
QR3 : Est-ce que l'approche développée consomme moins de ressources (réduit l'effort de test) que les approches de contrôle ?
QR4 : Est-ce que l'approche développée est plus efficiente dans la recherche de fautes que les approches de contrôle ?
QR5 : Est-ce que l'approche développée ordonne mieux les classes que les approches de contrôle ?

Les trois premières questions de recherche sont en lien direct avec les trois premiers objectifs, soient la maximisation de la couverture, la maximisation de la détection des fautes et le contrôle de l'effort de test. La quatrième question de recherche est utile pour comparer la consommation de ressource (effort de test) par rapport à la qualité de la solution produite pour des systèmes ayant des caractéristiques différentes. La cinquième question de recherche permet d'évaluer non pas la capacité de l'approche à atteindre un seuil de performance, mais plutôt son caractère à ordonner les classes candidates (par utilité) à tester. Les précédentes questions de recherche considèrent le résultat global de l'ensemble candidat sans tenir compte de l'ordonnancement, et donc de la qualité de la priorisation. Cette question porte sur l'évaluation de la qualité de la priorisation. Les quatre premières questions portent sur l'aspect « sélection » des classes candidates de notre approche, la cinquième porte sur l'aspect « priorisaion » de notre approche.

Le présent mémoire est structuré comme suit ; le second chapitre présente le cadre théorique supportant les recherches menées, notament en détaillant les processus de priorisation et sélection des tests existants dans la littérature et les représentations des logiciels utilisées à des fins d'analyse. Le troisième chapitre détaille le cadre méthodologique employé en présentant les approches de contrôle et notre approche. J'y traiterai également des procédures statistiques utilisées et des méthodes d'évaluation des approches de priorisation. Le quatrième chapitre présente les résultats obtenu en testant notre approche et les approches de contrôle sur nos jeux de données. Le cinquième chapitre conclu le mémoire en revenant sur les résultats importants et en présentant les limites du présent travail.

Chapitre 2

Cadre théorique

2.1 Priorisation et sélection des tests

Le problème de la priorisation des tests consiste à trouver l'ordonnancement d'une suite de tests maximisant un critère donné [Elbaum *et al.*, 2002]. Soit f une fonction objective associant un ensemble ordonné de tests à un nombre réel, T une suite de tests et $\mathcal{P}(T)$ une permutation de la suite de tests. Alors, la priorisation des tests consiste à trouver $T^* \in \mathcal{P}(T)$ tel que

$$f(T^*) \geq f(T^{'}) \forall T^{'} \in \mathcal{P}(T) \tag{2.1}$$

La fonction f peut être adaptée pour répondre à de l'optimisation multi-objectif ($f : \mathcal{P}(T) \rightarrow \mathbb{R}^m$). La sélection des tests de son côté a pour but de sélectionner, parmi un ensemble de tests, un sous-ensemble qui satisfait un certain nombre de conditions. Soit T une suite de tests, et $\{c_i\}$ un ensemble de conditions, trouver $T^* \subseteq T$ tel que

$$|T^*| \leq |T^{'}| \forall\ T^{'} \subseteq T \text{ avec } T^{'} \text{satisfait tous les } c_i \tag{2.2}$$

Plusieurs travaux de la littérature ont traité la priorisation des tests, particulièrement dans un contexte de tests de régression. Rothermel, Elbaum et coll. [Elbaum *et al.*, 2002, Rothermel *et al.*, 1999, Rothermel *et al.*, 2001], en particulier, ont publié plusieurs articles introduisant diverses fonctions objectives univariées pour les algo-

rithmes gloutons. Ces fonctions permettent d'optimiser l'un des cinq objectifs pouvant être recherchés par la priorisation des tests [Rothermel *et al.*, 1999] :

1. Détecter les fautes plus rapidement ;
2. Détecter les fautes les plus sévères ;
3. Détecter les fautes liées à des changements spécifiques ;
4. Couvrir le code avec le moins de tests possible ;
5. Augmenter la confiance dans la validité du système plus rapidement.

Ils ont aussi présenté des fonctions objectives qui évaluent les cas de test en se basant sur deux stratégies. La stratégie totale stipule que le meilleur test est celui qui permet d'atteindre la plus grande valeur pour l'ensemble des tests sélectionnés. La stratégie additionnelle qui, pour sa part, sélectionne le test qui apporte le plus de nouvelles informations, par exemple, le plus de nouvelles lignes de code couvertes.

Ces approches ont aussi été étudiées par d'autres chercheurs qui les ont utilisées pour comparer leurs résultats. Notamment par Li et coll. [Li *et al.*, 2007] qui ont comparé ces approches aux algorithmes génétiques et à l'algorithme de descente de gradient. Ils ont trouvé que les algorithmes gloutons, avec une stratégie totale, sont ceux qui ont une moins bonne performance. De même, l'algorithme de la descente de gradient est celui qui admet la plus grande variabilité. Dans plusieurs cas, l'algorithme glouton avec une stratégie additionnelle et l'algorithme génétique admettent des performances similaires (sans différence significative).

Carlson et coll. [Carlson *et al.*, 2011] ont apporté l'idée de rassembler les cas de test similaires ensemble, car ils conjecturent que ces tests ont des capacités de détection des fautes similaires. Leur approche consiste à utiliser un algorithme de classification hiérarchique afin de regrouper les tests, puis de sélectionner des tests provenant de chaque groupe. Ils ont comparé leur approche avec les algorithmes gloutons de type total (couverture, complexité et historique des fautes). Ils ont trouvé que l'utilisation de la classification hiérarchique améliore la qualité de la priorisation. La seconde partie de leur étude porte sur les résultats en cas de contrainte de temps, soit les contextes où il est possible d'exécuter seulement un certain pourcentage des tests. Encore une fois, leur algorithme faisant intervenir la classification hiérarchique offre de meilleures performances au niveau du nombre de fautes détectées.

Les approches de classification ont été reprises dans d'autres contextes, notamment par Chaurasia et coll. [Chaurasia *et al.*, 2015] qui ont classé les tests selon trois aspects de leur couverture, avant d'ordonner les tests par clusters et d'ordonner les clusters. Zhao et coll. [Zhao *et al.*, 2015] ont également exploré la classification des tests en utilisant un algorithme de classification hiérarchique basé sur la couverture des cas de test. Ensuite, un algorithme d'apprentissage automatique basé sur les réseaux bayésiens, a été utilisé pour inférer à chaque cas de test une probabilité de détecter des erreurs, valeur utilisée pour ordonner les tests dans leur groupe respectif. Le réseau bayésien est entraîné sur des éléments des versions antérieures et considère trois éléments : des métriques de qualité, les informations sur le changement du code source et la couverture des classes par les tests. Les approches de classification ont été reprises plus récemment dans d'autres domaines connexes de la priorisation des tests tel que le test aléatoire [Chen *et al.*, 2018], ou encore la priorisation des tests issus des besoins (approches semi-supervisées) [Kandil *et al.*, 2017]. La classification des cas de test dans le but de prioriser les tests est une technique bien implémentée.

D'autres approches se sont intéressées aux algorithmes évolutifs comme la colonie de fourmis, les algorithmes génétiques et l'essaim particulaire. Masri et coll. [Masri et El-Ghali, 2009] ont introduit des algorithmes génétiques dans le but d'effectuer une priorisation et une sélection des tests (réduction de la suite de tests). Leur approche a été validée sur deux systèmes de petite taille. En raison d'une phase de leur algorithme qui explore toutes les combinaisons possibles d'éléments, il serait coûteux d'adapter cette approche pour des systèmes de plus grande taille. La sélection est toutefois très efficace permettant de réduire de 75% la taille de l'ensemble des tests tout en détectant toutes les fautes. Catal [Catal, 2012] propose une revue systématique de la littérature en lien avec les principales approches basées sur les algorithmes génétiques. Seuls sept articles ont été identifiés comme satisfaisant les critères de la revue systématique de littérature. La revue suggère que les algorithmes génétiques performent généralement mieux que l'approche aléatoire et la descente de gradient, mais pas toujours mieux que les algorithmes gloutons. Toutefois, ces résultats n'ont pas tous été validés sur des systèmes réels ; certains n'ont été validés que sur des données générées. Autre point à remarquer, les critères d'évaluation de la priorisation des tests ne sont pas constants entre les études, ce qui laisse croire que les algorithmes génétiques peuvent être plus performants que les approches gloutonnes sous certaines circonstances seulement. Nous n'étudierons, toutefois, pas cette hypothèse dans la suite du mémoire, car celle-ci se trouve à côté de nos questions de recherche.

Plus récemment, Wang et coll. [Wang *et al.*, 2016] ont mené une étude comparant plusieurs algorithmes évolutifs. Ils arrivent à la conclusion que le meilleur algorithme varie selon le critère utilisé pour mesurer la qualité d'une approche. Aucune approche ne semble dominer les autres. Ils ont utilisé des critères de couverture, de détection des fautes et de fréquence d'exécution de certains tests. Dans certains cas, les approches ont été évaluées avec un seul objectif, et dans d'autres cas les approches étaient multi-objectifs. Pradhan et coll. [Pradhan *et al.*, 2017] ont repris les algorithmes génétiques et ont intégré une phase de classification pendant l'exécution de l'algorithme. Leur approche est présentée dans le détail à la section 3.3.4.

2.2 Représentation d'un logiciel

La représentation d'un logiciel peut être faite de plusieurs façons. Généralement, le but des représentations est d'extraire une information précise, dans un format ou structure utilisable pour effectuer des analyses. Les métriques logicielles sont très utilisées et permettent de mesurer plusieurs aspects des logiciels [Chidamber et Kemerer, 1994, Martin, 1994]. Une autre approche consiste à représenter le système sous forme de structure mathématique, tel un graphe afin de représenter un type de relation entre les éléments du système. Divers types de représentation en graphe existent, comme les graphes de flux de contrôle, les graphes d'appels et les graphes de dépendance [Chen *et al.*, 1997, Horwitz et Reps, 1992, Larsen et Harrold, 1996]. Dans le cadre de ce mémoire, nous nous sommes intéressés aux graphes de dépendance entre classes.

2.2.1 Graphe de dépendance

Nous avons adapté les définitions de graphe de dépendance pour les rendre utilisables au calcul des mesures de centralité (définies plus loin). Nous avons retiré les sommets qui ne sont pas des classes, car la plupart des mesures de centralité ne peuvent pas considérer plusieurs types de sommets. Pour les mêmes raisons, les arêtes multiples sont simplifiées en arêtes simples non pondérées. Pour certaines mesures, nous avons aussi dû retirer la direction des arêtes. Pour d'autres, nous avons adapté

la définition de la mesure pour supporter le cas où les arêtes sont dirigées. Cela est indiqué le cas échéant.

Un graphe de dépendance est un graphe simple dirigé. Un graphe simple G est représenté par une paire d'ensembles $G = (V, E)$ où V est l'ensemble des sommets du graphe et E un ensemble de paires ordonnées de sommets représentant les arêtes du graphe. Une arête dirigée relie deux sommets appelés sommet d'origine et sommet d'arrivée. Dans un graphe dirigé, les arêtes peuvent seulement être parcourues du sommet d'origine vers le sommet d'arrivée. Dans un graphe non dirigé, l'arête peut être parcourue dans les deux sens.

Un chemin entre deux sommets dans un graphe est une suite ordonnée d'arêtes dont chaque arête a comme sommet d'arrivé le sommet d'origine de l'arrête suivante. Le sommet d'origine de la première arête est le début du chemin et le sommet d'arrivée de la dernière arête est la fin du chemin. En notation formelle, un chemin P du sommet v_i à v_m est

$$P_{i,m} = ((v_i, v_j), (v_j, v_k), \cdots, (v_l, v_m)) \tag{2.3}$$

La longueur du chemin est le nombre d'arrêtes qui le composent. La distance entre deux sommets dans un graphe est la longueur du plus court chemin entre les deux sommets. La distance D entre les sommets v_i et v_m d'un graphe G est

$$D_{i,m} = \begin{cases} \infty \text{ si } \nexists P_{i,m} \in G \\ |P^*_{i,m}| \text{ tel que } \forall P_{i,m} \in G, |P^*_{i,m}| \leq |P_{i,m}|, \text{ sinon} \end{cases} \tag{2.4}$$

La distance entre deux sommets pour lesquels il n'existe pas de chemin permettant de les relier est considérée comme infinie. Un sommet v est voisin d'un autre sommet u, s'il existe une arête du sommet u vers le sommet v.

On peut représenter un graphe par une matrice appelée matrice d'adjacence. Une telle matrice est carrée et possède une ligne et une colonne pour chaque sommet du graphe. Les éléments de la matrice d'adjacence A sont définis comme suit :

$$a_{i,j} = \begin{cases} 1 & \text{si } P_{j,i} \in G \\ 0 & \text{sinon} \end{cases} \tag{2.5}$$

2.2.2 Mesures de centralité

Les mesures de centralité sont des fonctions qui associent à un sommet d'un graphe une valeur réelle. Les mesures de centralité sont généralement croissantes, donc plus un sommet est central, plus sa mesure sera élevée. Les prochains paragraphes présentent les mesures de centralité utilisées [Chen *et al.*, 2016, Tosun *et al.*, 2009, Zimmermann *et al.*, 2009] dans notre étude.

Centralité de proximité

La centralité de proximité (CL — Closeness centrality) est la somme des distances à tous les autres sommets du graphe des dépendances. On utilise une échelle exponentielle telle que définie par Dangalchev dans [Dangalchev, 2006]. Cette formulation permet de traiter le cas où le graphe est dirigé. La formule de la centralité de proximité est pour un somme u du graphe G.

$$\mathrm{CL}(u) = \sum_{v \in V(G)} \frac{1}{2^{d(u,v)}} \tag{2.6}$$

où d est la distance pour passer du sommet u au sommet v par le plus court chemin. Dans le cas où aucun chemin n'existe entre u et v, la distance est infinie et le sommet v ne contribue pas à augmenter la centralité de u. Plus la mesure est importante, plus le sommet est central.

Centralité KATZ

La centralité KATZ (KATZ centrality) est une adaptation de la mesure de centralité des vecteurs propres (eigenvalue centrality). La centralité est calculée selon un processus itératif faisant intervenir la matrice d'adjacence du graphe des dépendances.

$$\mathbf{x}_i' = \alpha \sum_j A_{ij} x_j + \beta_i \tag{2.7}$$

Avec $\mathbf{x}$ un vecteur ayant la centralité de chaque sommet. La valeur initiale de $\mathbf{x}$ est 1 pour tous les sommets. Le facteur α détermine la prépondérance entre le terme itéré

et le facteur β qui est une valeur constante pour chaque sommet. Nous avons utilisé la valeur suivante :

$$\alpha = \frac{1}{2\lambda_1} \tag{2.8}$$

Où λ_1 est la plus grande valeur propre de la matrice d'adjacence A. La valeur doit nécessairement être plus petite que $1/\lambda_1$ afin que la mesure converge. La valeur utilisée est la même que celle dans [Ouellet et Badri, 2021] afin de différencier la mesure de la mesure du vecteur propre. Nous avons considéré le graphe non dirigé pour le calcul de cette mesure. Nous avons utilisé la complexité intrinsèque de chaque classe pour le terme β. La complexité intrinsèque est évaluée par la métrique WMC (voir section 2.2.3). La mesure KATZ indique une proximité aux sommets complexes du graphe.

Centralité semilocale

La centralité semi-locale (SLC — Semi-local centrality) est le nombre de voisins de chaque voisin atteignable par un chemin de longueur 2 dans le graphe [Chen *et al.*, 2012]. Soit la fonction $\Gamma(x)$ qui retourne l'ensemble des voisins du sommet x et $N(x)$ qui retourne le nombre de voisins de x, alors

$$\mathrm{SLC}(x) = \sum_{v \in \Gamma(x)} \sum_{u \in \Gamma(v)} N(u) \tag{2.9}$$

Cette mesure peut donner de grandes valeurs, car chaque sommet peut être compté plus d'une fois selon le nombre de chemins de longueur 2 permettant de s'y rendre. Notons, aussi, que nous avons utilisé un graphe dirigé. Donc, les chemins peuvent revenir sur eux-mêmes seulement si une relation de dépendance bidirectionnelle existe. La mesure permet d'avoir un aperçu de la densité locale du graphe de dépendance.

2.2.3 Métriques orientées objet

Dans cette section, les métriques orientées objet utilisées dans notre approche (et nos expérimentations) sont présentées. Une métrique logicielle est une mesure d'un caractère que peut adopter une unité logicielle ou la valeur de la mesure [Abran, 2010].

Dans le cadre des métriques orientées objet, les caractères doivent être observables sur des classes. On associe alors à ces caractères une valeur réelle.

Complexité pondérée des méthodes

La complexité pondérée des méthodes (WMC — weighted method complexity) évalue la complexité cyclomatique [McCabe, 1976] de chaque méthode, puis somme les valeurs pour obtenir la complexité de la classe [Chidamber et Kemerer, 1994].

Couplage entre les objets

Le couplage entre les objets (CBO — Coupling between objects) est une métrique de couplage qui compte le nombre d'objets qui interagissent avec l'objet pour lequel la métrique est calculée [Chidamber et Kemerer, 1994]. Une interaction est définie comme l'utilisation de méthodes ou d'attributs.

Couplage sortant

Le couplage sortant (CE — Efferent coupling) est le nombre de classes appelées par la classe [Martin, 1994]. Plusieurs travaux ont utilisé cette métrique pour la détection des classes fautives [Anwer *et al.*, 2017, Jureczko et Madeyski, 2010]. Il s'agit d'une métrique mesurant le couplage.

Lignes de code

Le nombre de lignes de code (LOC — Lines of code) est une métrique de taille très utilisée dans les études empiriques en génie logiciel [Basili *et al.*, 1996, He *et al.*, 2015, Kaur et Malhotra, 2008].

Nombre de réponses pour la classe

Le nombre de réponses de la classe (RFC — Response For Class) est le nombre de méthodes pouvant être appelées en réponse à un message reçu [Chidamber et Kemerer, 1994]. La métrique indique en partie la complexité d'une classe, car un grand ensemble de réponses indique un traitement possiblement complexe, et la taille du couplage.

Taille de l'interface de la classe

La taille de l'interface de la classe (CIS - Classe Interface Size) compte le nombre de méthodes publiques de chaque classe [Bansiya et Davis, 2002].

2.3 Conclusion du chapitre 2

Dans le chapitre, j'ai situé l'objectif de ma recherche dans la littérature et parcouru une partie de la littérature traitant du sujet. Par la suite, plusieurs bases théoriques ont été établies, principalement sur les différentes façons de représenter un logiciel. La prochaine section continuera à établir des bases pour les expérimentations en abordant les aspects plus pratiques de la méthodologie.

Chapitre 3

Cadre méthodologique

3.1 Adaptation du problème

Dans le cadre théorique (chapitre 2), la définition suivante du problème de la priorisation des tests a été présentée. Soit f une fonction objective associant un ensemble ordonné de tests à un nombre réel, T une suite de tests et $\mathcal{P}(T)$ une permutation de la suite de tests. Alors, la priorisation des tests consiste à trouver $T^* \in \mathcal{P}(T)$ tel que

$$f(T^*) \geq f(T^{'}) \; \forall T^{'} \in \mathcal{P}(T) \tag{3.1}$$

Cette définition présuppose l'existence d'une suite de tests T. Cela restreint donc le contexte d'utilisation d'approche de priorisation aux situations où des tests ont déjà été écrits. L'objectif de ce mémoire est de proposer des ordonnancements hors du contexte des tests de régression. Par conséquent, nous devons adapter la définition de priorisation pour lever cette condition d'existence de tests au moment de la priorisation. Nous ajoutons la sélection au problème afin de lever cette condition. La définition que nous utiliserons est la suivante :

Soit C l'ensemble des classes d'un programme. Notons $\mathbb{P}$ l'ensemble des parties, T un test unitaire et $T(c)$ le test de l'unité c (par extension de notation, $T(C)$ est la réunion de tous les tests pour les éléments de l'ensemble C). Étant donnée f une fonction objective de l'ensemble des parties de C vers les nombres réels. La priorisation

des candidats aux tests consiste à trouver $C^* \in \mathcal{P}(\mathbb{P}(C))$ tel que

$$f(C^*) \geq f(C') \; \forall C' \in \mathcal{P}(\mathbb{P}(C)) \tag{3.2}$$

Ici, l'ensemble $\mathcal{P}(\mathbb{P}(C))$ est de taille énorme, soit $O(2^n!)$. Pour la taille moyenne des systèmes étudiés dans ce mémoire, soit 689 classes (voir tableau 3.5) cela représente plus de $(10 \uparrow)^2$ 209.7 solutions possibles. La grande différence avec la priorisation des tests de régression est que, dans notre approche, la taille de l'ensemble solution n'est pas fixée. Pour la priorisation des tests, tous les tests sont retournés tandis que dans notre approche de priorisation des candidats, la sélection des classes candidates et non-candidates fait partie intégrante du processus.

Également, si la fonction objective f n'inclut que des éléments croissants avec la taille de la solution, telle que la couverture l'est (plus la taille de l'ensemble candidat est grande, plus la couverture est grande) alors nécessairement toutes les classes du système seront considérées comme candidates. Cette solution est irréalisable en termes d'effort de test nécessaire. Notons, ici, que les mesures d'ordonnancement ne peuvent être utilisées dans les fonctions objectives comme c'est le cas dans les approches de priorisation des tests en raison de la taille variable de l'ensemble candidat. Nous expliquons cette limitation en détail à la section 3.5.4 où nous décrivons ces mesures.

Pour les approches de contrôle inspirées d'approches normalement utilisées pour les tests de régression, nous les avons adaptées en fixant la taille de l'ensemble candidat. Nous avons fait ce choix afin de limiter les modifications apportées aux approches de contrôle afin de les rendre applicables au contexte de priorisation des candidats aux tests. Pour l'approche développée, nous avons introduit des paramètres et des étapes visant à limiter la taille de l'ensemble candidat.

3.2 Description des procédures statistiques

Tout au long de nos expérimentations, nous utilisons diverses procédures statistiques pour mesurer la dépendance de certaines variables ou vérifier si la différence entre deux échantillons est significative, tant par rapport à la tendance centrale que

pour la dispersion. Nous décrivons les procédures statistiques utilisées dans les prochains paragraphes.

3.2.1 Test sur les moyennes de deux échantillons appariés

Le test permet de mesurer le caractère significatif de la différence entre les moyennes de deux échantillons apparié (où chaque observation d'un échantillon est associé à une observation de l'autre échantillon par un caractère ou des données communes). C'est un test robuste même en cas de violation des hypothèses du test [Saporta, 2011]. Nous avons toujours utilisé le test dans un contexte bilatéral pour mesurer une différence significative et non pas nous limiter à l'amélioration ou la dégradation d'un échantillon par rapport à un autre. Le test vérifie les hypothèses suivantes :

H_0 : Les deux échantillons proviennent de population de moyennes égales.
H_1 : Les deux échantillons proviennent de population de moyennes inégales.

Soit deux échantillons appariés x et y de taille n, alors la statistique du test est

$$\frac{\bar{d}}{\sqrt{\frac{s_d^2}{n}}} \sim Z \tag{3.3}$$

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i) \tag{3.4}$$

$$s_d^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - y_i - \bar{d})^2 \tag{3.5}$$

Avec Z une loi normale centrée réduite. Si la statistique, en valeur absolue, est plus grande que la valeur de référence de la loi de normale, alors l'hypothèse nulle est rejetée.

3.2.2 Test de Pitman sur les variances de deux échantillons appariés

Le test de Pitman sur les variances permet de vérifier si la différence entre les variances de deux échantillons appariés est significative [Saporta, 2011]. Nous avons toujours utilisé le test dans un contexte bilatéral afin de mesurer les différences significatives et non pas nous limiter à l'amélioration ou la dégradation d'un échantillon par rapport à un autre. Les hypothèses du test de Pitman sont :

H_0 : Les deux échantillons proviennent de population de variances égales.
H_1 : Les deux échantillons proviennent de population de variances inégales.

Le test de Pitman repose sur le test de corrélation linéaire présenté à la section suivante (3.2.3). Pour tester la variance entre deux variables X et Y, il s'agit de tester la corrélation entre les deux variables suivantes :

$$U = X + Y \qquad V = X - Y \tag{3.6}$$

Le test de Pitman repose sur l'identité suivante :

$$\frac{\sigma_X^2}{\sigma_Y^2} = 1 \Leftrightarrow r_(U, V) = 0 \tag{3.7}$$

3.2.3 Corrélation linéaire

La corrélation linéaire permet d'évaluer la relation de dépendance linéaire entre deux échantillons appariés. Le modèle de dépendance doit être d'abord validé par une observation graphique afin de déterminer la forme de la dépendance. Afin de gagner en pertinence, les graphes des variables pour lesquelles nous étudions la corrélation ne sont pas présentés. La validation graphique du lien linéaire a tout de même été réalisée et a été observée à tous les endroits où nous avons conclu à une corrélation significative. La corrélation linéaire r entre deux échantillons x et y de moyennes

respective $\bar{x}$, $\bar{y}$ et de taille n est donnée par

$$r(x,y) = \frac{\sum_{i=1}^{n}[(x_i - \bar{x})(y_i - \bar{y})]}{\sqrt{\left(\sum_{i=1}^{n}(x_i - \bar{x})^2\right)\left(\sum_{i=1}^{n}(y_i - \bar{y})^2\right)}} \tag{3.8}$$

La corrélation linéaire est sensible aux données extrêmes [Saporta, 2011]. Comme tout coefficient de corrélation, la valeur de la corrélation linéaire est comprise dans l'intervalle $[-1, 1]$. Une corrélation positive de 1 indique une dépendance linéaire proportionnelle parfaite des deux échantillons, une corrélation négative de -1 indique une dépendance linéaire inversement proportionnelle parfaite des deux échantillons et une corrélation nulle indique généralement une absence de dépendance linéaire.

La significativité statistique de la corrélation peut être testée avec le test de corrélation. Les hypothèses du test sont :

H_0 : La corrélation entre les deux échantillons est nulle.
H_1 : La corrélation entre les deux échantillons est non nulle.

Soit un coefficient de corrélation linéaire estimé $\hat{r}$ obtenu pour des échantillons de taille n, alors

$$\frac{\hat{r}}{\sqrt{\frac{1-\hat{r}^2}{n-2}}} \sim T_{n-2} \tag{3.9}$$

Avec T_{n-2} une loi de Student avec $n-2$ degrés de liberté.

3.2.4 Régression linéaire

La régression linéaire vise à développer une équation, à partir de la méthode des moindres carrés, afin d'obtenir une fonction affine pour estimer un caractère [Saporta, 2011]. La forme générale d'une équation de régression linéaire est la suivante ou $\mathbf{x}$ est un vecteur colonne des données, $\mathbf{b}$ un vecteur colonne des coefficients de la régression

et β_0 un terme constant.

$$y = \mathbf{x}^T\mathbf{b} + b_0 \tag{3.10}$$

Les coefficients de $\mathbf{b}$ et b_0 sont estimés par la méthode des moindres carrés. Il est possible de tester le caractère significatif de la régression (test global) et de chaque coefficient (test marginal). La réalisation des deux tests est importante, car toutes les variables explicatives ne sont pas nécessairement liées à la variable expliquée.

Les hypothèses du test global de significativité d'une régression linéaire sont les suivantes :

H_0 : Les variables explicatives n'expliquent pas la variance de la variable expliquée.
H_1 : Les variables explicatives expliquent la variance de la variable expliquée.

La statistique du test est le ratio de la somme des carrés moyens de la régression (CMR) sur les carrés moyens résiduels (CM_{Res}). Les carrés moyens sont obtenus en décomposant la variance du modèle en deux composantes : la composante liée à la régression et la composante résiduelle (erreur de la régression). Le ratio suit une loi de Fisher à 1 degré de liberté au numérateur et $n-2$ degrés au dénominateur avec n le nombre d'observations. Avec un ratio suffisamment grand ($CMR > CM_{Res}$), l'hypothèse nulle peut être rejetée et on peut conclure à la significativité de la régression.

Pour le test marginal des variables explicatives, les hypothèses sont :

H_0 : Le paramètre b_j est nul (x_j ne contribue pas à la régression).
H_1 : Le paramètre b_j significativement différent de 0 (x_j contribue à la régression).

La statistique du test est la valeur du paramètre estimé divisée par la variance du paramètre. Le ratio obtenu suit une loi de Student avec $n-k-1$ degrés de liberté où n est le nombre d'observations et $s(b_j)$ l'écart-type du paramètre b_j.

$$\frac{b_j}{s(b_j)} \sim T_{n-k-1} \tag{3.11}$$

3.3 Description des approches de contrôle

Nous avons comparé l'approche introduite (approche SL décrite à la section 3.4) avec les autres approches de la littérature. Pour ce faire, nous avons implémenté certaines approches proposées par d'autres auteurs [Rothermel *et al.*, 1999, Do et Rothermel, 2006, Russell et Norvig, 2016, Elbaum *et al.*, 2002, Li *et al.*, 2007, Rothermel *et al.*, 1999, Rothermel *et al.*, 2001, Li *et al.*, 2007, Pradhan *et al.*, 2017, Carlson *et al.*, 2011]. Ces approches seront appelées « approches de contrôle ». Les descriptions complètes de ces approches suivent au cours des prochains paragraphes.

Afin de sélectionner les approches de contrôle, nous avons utilisé trois critères :

1. L'article présentant l'approche doit avoir comparé l'algorithme proposé avec au moins un autre algorithme sur au moins un système ;
2. L'article présentant l'approche doit fournir une description suffisamment précise de l'algorithme pour nous permettre de l'implémenter et de répliquer la méthodologie ;
3. L'approche doit pouvoir être modifiée, sans être dénaturée, pour être utilisable hors du contexte des tests de régression, dans le cas où l'article la présente dans un contexte de tests de régression.

Ces critères permettent d'assurer que les approches de contrôle ont été suffisamment étudiées pour servir de référence (critère 1), et que nous pouvons répliquer ces approches avec suffisamment de précision (critère 2). Le critère 3 sert à élargir le bassin dans lequel nous pouvions sélectionner les approches de contrôle, car la majorité des approches de priorisation des tests ont été développées dans le contexte des tests de régression. Également, afin d'obtenir une vue d'ensemble, nous avons diversifié les approches en sélectionnant des approches de différentes familles d'algorithmes de priorisation des tests.

3.3.1 Séquence aléatoire

L'algorithme de séquence aléatoire [Rothermel *et al.*, 1999, Do et Rothermel, 2006] consiste à ordonner aléatoirement les tests un certain nombre de fois puis de conserver le meilleur résultat de ces ordonnancements multiples. Cette approche de contrôle sert

souvent de seuil minimal de performance. Si une approche plus complexe performe moins bien que l'algorithme de séquence aléatoire, alors cette approche est somme toute équivalente à un tirage au sort. Il va sans dire que dans ce cas l'approche plus complexe perd en pertinence.

La version de séquence aléatoire que nous avons utilisée effectue 5 000 ordonnancements aléatoires et sélectionne la meilleure séquence en fonction de la couverture totale en lignes de code. Nous répétons la procédure un grand nombre de fois (boosting), car la génération aléatoire prend très peu de temps. Il serait, donc, envisageable dans un contexte réel d'utiliser un procédé de répétition pour sélectionner la meilleure séquence. Cela rend aussi très improbable la possibilité [1] que l'algorithme retourne un cas significativement inférieur à la médiane, et par conséquent, garanti que la séquence aléatoire fournit une séquence raisonnable, au moins dans le 50 % des meilleurs ordonnacements.

3.3.2 Algorithmes gloutons

L'algorithme glouton est un algorithme itératif qui, à chaque étape, ajoute un élément à la solution maximisant localement la fonction objective. Les algorithmes gloutons sont réputés pour être rapides [Russell et Norvig, 2016]. Cette rapidité d'exécution se fait généralement aux dépend de la qualité de la solution rencontrée tel qu'il a pu être identifié dans certains travaux [Li *et al.*, 2007]. Les algorithmes gloutons ont été utilisés dans de nombreux travaux de priorisation des tests logiciels [Elbaum *et al.*, 2002, Hemmati, 2019, Li *et al.*, 2007, Rothermel *et al.*, 1999, Rothermel *et al.*, 2001], d'où l'importance de les inclure comme approche de contrôle. Comme mentionné précédemment, les algorithmes utilisant des fonctions objectives de type « stratégie additionnelle » offrent de meilleurs résultats [Elbaum *et al.*, 2002, Li *et al.*, 2007, Rothermel *et al.*, 1999, Rothermel *et al.*, 2001]. Par conséquent, la stratégie additionnelle est celle utilisée par les approches de contrôle présentées dans ce mémoire.

Trois types d'algorithmes gloutons seront considérés à titre d'approche de contrôle.

1. Priorisation de la couverture du nombre de lignes de code.

1. Notons que la probabilité que tous les tirages renvoie un résultat pire que la médiane est d'environ 7.08×10^{-1506}. Nous pouvons, donc, assumer que la meilleure séquence aléatoire retourne un résultat d'au moins la médiane.

2. Priorisation des classes plus complexes à l'aide d'une métrique de complexité.

3. Priorisation des classes qui sont plus à risque de contenir des fautes (apprentissage automatique).

Ces trois approches ont déjà été considérées dans la littérature [Elbaum *et al.*, 2002, Li *et al.*, 2007, Rothermel *et al.*, 1999, Rothermel *et al.*, 2001]. Les fonctions objectives utilisées par l'algorithme glouton sont aussi reprises par d'autres approches dont la descente de gradient [Li *et al.*, 2007]. L'utilisation ici de chacune de ces approches permettra de constater leur efficacité dans un contexte univarié.

La couverture du nombre de lignes de code se fait à l'aide de la métrique d'évaluation de la couverture totale. La métrique de complexité est la même que celle utilisée dans l'approche hiérarchique (voir section 3.3.5). Pour les méthodes d'apprentissage automatique, les mêmes algorithmes (K plus proches voisins et forêt aléatoire) que ceux utilisés dans la définition de notre approche ont été utilisés. Comme nous avons effectué nos expérimentations avec trois algorithmes d'apprentissage (voir section 3.4.1), trois variantes de l'algorithme glouton sont considérées pour le critère 3, chacun traitant respectivement d'un algorithme d'apprentissage.

3.3.3 Descente de gradient

L'algorithme de descente de gradient est simple et relativement peu coûteux en temps d'exécution, car il est d'ordre $O(n^2)$. L'algorithme désigne une solution initiale au hasard. Ensuite, il évalue l'impact de chaque permutation de deux éléments sur la valeur de la fonction d'évaluation. La solution est par la suite mise à jour pour intégrer la permutation engendrant la plus grande amélioration. Les étapes d'évaluation et de permutation sont répétées tant qu'il existe une permutation qui améliore le résultat de la fonction d'évaluation [Li *et al.*, 2007]. La dernière solution est la solution retournée par l'algorithme. Cette approche tend à retourner des optima locaux plutôt que globaux ce qui représente son principal défaut. C'est un algorithme qui privilégie la rapidité d'exécution au détriment de l'optimalité de la solution.

3.3.4 Algorithmes génétiques

Cette approche issue des algorithmes génétiques [Pradhan *et al.*, 2017] permet d'ordonner les tests qui doivent être exécutés. Elle ne fournit pas d'ordre d'exécution de ces tests. Originalement proposée dans un contexte de tests de régression, cette approche pouvait être adaptée sans modification majeure pour être utilisable dans un contexte de sélection et de priorisation à partir du code source. D'abord, une population initiale d'individus est générée. Les individus représente une séquence de test, c'est-à-dire une solution potentielle. Pour les exécutions, nous avons généré des populations de 1000 individus. Chaque individu est composé d'un ensemble de chromosomes. Chaque chromosome correspond à une classe candidate au test. Chaque individu contient un nombre de chromosomes égal à 15 % du nombre total de classes. Nous avons choisi d'utiliser 15 %, car il s'agit du nombre de classes moyen sélectionné par notre approche (détaillée à la section 3.4) ce qui permet d'obtenir une taille d'ensemble de classes candidates au test comparable.

Ensuite, la procédure suivante est répétée sur chaque génération (itération de l'algorithme), en commençant par la génération de la population initiale, tant que la solution n'est pas stagnante ou que le seuil maximal d'itérations n'a pas été atteint. Une solution est stagnante lorsque l'amélioration entre deux générations est plus petite qu'une valeur donnée. Au début de chaque génération, les individus sont classés par groupe à l'aide de l'algorithme de Lloyds.

L'algorithme de Lloyds génère un certain nombre de groupes à partir d'individus sélectionnés aléatoirement. Chaque groupe contient au début un seul individu. Nous avons choisi de former 10 groupes. L'idée ici est d'avoir suffisamment de groupes pour diversifier les élites, mais aussi des groupes assez importants pour qu'ils représentent une partie significative du système. Nous avons trouvé, après avoir testé diverses valeurs, que la valeur 10 offrait un bon compromis. Ensuite, chaque individu restant est affecté au groupe lui correspondant le plus ; soit le groupe avec la plus petite distance entre l'individu et le centre du groupe. Le centre du groupe est déterminé comme la moyenne de tous les individus. Comme une fonction multi-objectif est utilisée, le centre correspond au vecteur des moyennes de chaque dimension de la fonction multi-objectif. Les groupes sont ensuite triés selon un tri de dominance à l'aide des concepts de dominance de groupe ou de dominance partielle de groupe.

Soit deux groupes $g_1 = (g_{1,1}, g_{1,2} \cdots g_{1,m})$ et $g_2 = (g_{2,1}, g_{2,2} \cdots g_{2,m})$ dont la fonction multi-objectif possède m dimensions, g_1 domine g_2 si

$$\forall i = 1..m,\ g_{1,i} \geq g_{2,i} \text{ et } \exists i \in 1, 2 \cdots m \text{ tel que } g_{1,i} > g_{2,i} \tag{3.12}$$

Cette condition de domination de groupe est toutefois difficile à satisfaire en pratique. Dans le cas où aucune solution dominante n'est identifiée, alors g_1 domine partiellement g_2 si

$$|g_{1,i} > g_{2,i},\ \text{pour } i = 1..m| > |g_{1,i} < g_{2,i},\ \text{pour } i = 1..m| \tag{3.13}$$

Dans le cas où une égalité surviendrait pour le critère de dominance partiel, ce second critère de dominance partielle est utilisé :

$$\sum_{i=1}^{m} \frac{g_{2,i} - g_{1,i}}{g_{2,i}} > 0,\ \text{alors } g_1 \text{ domine partiellement } g_2 \tag{3.14}$$

Ensuite, l'ensemble d'individus élites est créé à partir des individus de la population. L'ensemble des élites est composé de 400 individus (40 %). Les élites sont sélectionnées à partir des clusters les plus dominants. Les observations étant non ordonnées dans un cluster, les individus provenant du dernier cluster sélectionné sont choisis au hasard. L'ensemble des élites est reporté tel quel à la prochaine génération.

Par la suite a lieu la phase de croisement et mutation pour compléter la prochaine génération. Pour effectuer le croisement (voir exemple au tableau 3.1), deux parents de l'ensemble des élites sont sélectionnés. Les enfants sont créés à partir des rangs de chaque chromosome dans les parents. Les chromosomes de plus petits rangs demeurent au début. Si un chromosome est présent dans les deux parents, alors son rang moyen est utilisé. Soit les deux individus suivants : $P_1 = (A, C, E, D)$ et $P_2 = (B, E, F, D)$ alors l'enfant serait composé des quatre chromosomes de plus petit rang $E = (A, B, C, E)$ ou (B, A, C, E), car A et B sont de rang égal. Notons que par notre implémentation deux chromosomes de rang ex aequo sont sélectionnés au hasard.

TABLE 3.1 – Rangs des chromosomes lors du croisement

Classe	A	B	C	D	E	F
Rang	1	1	2	4	2.5	3

Ensuite, une étape de mutation est appliquée afin de diversifier les solutions. La probabilité de mutation est de

$$\frac{1}{\text{nombre de chromosomes}} \tag{3.15}$$

de sorte à ce qu'en moyenne chaque individu mute une fois par génération. Une mutation consiste à remplacer un chromosome par un autre qui n'est pas inclus dans l'individu. Comme cette étape s'exécute distinctement du croisement, il peut s'agir d'un chromosome éliminé par le processus de croisement qui est immédiatement réintroduit. Suffisamment d'individus sont générés par croisement pour compléter la génération suivante. Dans notre cas, il s'agit de 600 individus (60 %).

3.3.5 Algorithme de classification hiérarchique

L'utilisation de la classification hiérarchique [Carlson *et al.*, 2011] permet également de prioriser les tests. La classification hiérarchique ne permet pas à elle seule d'ordonner les cas de test, mais seulement de les sélectionner. Les auteurs [Carlson *et al.*, 2011] ont ajouté une étape de tri afin d'obtenir une séquence de tests ordonnée.

L'approche commence par créer un certain nombre de groupes en utilisant la couverture des tests de ces classes pour établir la similarité entre les groupes. Au début, chaque classe du système forme un groupe. À chaque étape, les deux groupes les moins distants sont regroupés pour former un nouveau groupe. L'algorithme se termine lorsque le nombre de groupes restants est égal au nombre de groupes déterminés.

Au niveau de la mesure de distance entre deux groupes, nous avons la distance euclidienne entre la couverture des groupes [Carlson *et al.*, 2011]. La couverture d'un groupe se définit comme l'union des couvertures de chacune des classes du groupe. Soit G_i un groupe de classes ($g_{i,j}$ ses éléments) et *couv* la fonction qui retourne les classes couvertes directement et indirectement (couverture totale) par le test de la classe C, alors

$$couv(G_i) = \bigcup_{j=1}^{|G_i|} couv(g_{i,j}) \tag{3.16}$$

On écrit la couverture d'un groupe sous forme d'un vecteur binaire, où chaque classe est représentée par un bit, prenant la valeur 1 si l'élément est couvert et la valeur 0 si l'élément n'est pas couvert. La différence entre les couvertures est alors la distance entre ces vecteurs (appelé distance de Hamming [MacWilliams et Sloane, 1977]. Pour le regroupement, à chaque étape les deux groupes ayant la plus petite distance les séparant (couvrant le plus haut taux de classes similaires) sont regroupés.

Dans leur approche originale, Carlson et coll. sélectionnent le nombre de groupes formés en fonction du nombre de fonctionnalités du logiciel. Leur objectif est d'obtenir un groupe par fonctionnalité. Il nous est impossible de répliquer cette méthodologie dans notre approche, car cela demanderait une étude approfondie des fonctionnalités de chaque jeu de données, ce qui est hors du contexte. Nous avons comparé plusieurs critères de la littérature pour sélectionner le nombre de groupes à former à l'aide du package R *NbClust* [Charrad *et al.*, 2014]. La plupart des critères retournaient un résultat similaire. Nous avons donc sélectionné le critère cindex [Dalrymple-Alford, 1970] comme critère de référence. Le critère sélectionne le meilleur nombre de groupes basé sur la variance intragroupe. Le critère est paramétré pour retourner un nombre de groupes entre 4 et 10 % du nombre de classes (pour un système de 100 classes, au plus 10 groupes seront formés). L'utilisateur d'un indicateur connu nous permet d'éviter les problèmes de trop de petits groupes ou de trop peu de grands groupes.

Lorsque les groupes sont formés, les éléments sont ordonnés selon un critère donné. Dans l'article de Carlson et coll. [Carlson *et al.*, 2011], quatre critères ont été explorés. Le critère retenu pour la comparaison est celui basé sur la complexité du code. Soit c une classe du programme et C l'ensemble des classes du programme :

$$CC(c) = \frac{1}{2}\left(\frac{LOC(c)}{\max_{d \in C} LOC(d)} + \frac{CBO(c)}{\max_{d \in C} CBO(d)}\right) \tag{3.17}$$

Notons ici l'utilisation de la métrique CBO [Chidamber et Kemerer, 1994] plutôt que de la métrique DC telle qu'utilisée dans l'article original. La métrique DC compte le nombre de dépendances d'une méthode. Dans la présente méthodologie, nous travaillons plutôt au niveau des classes, alors la métrique calculant l'information correspondante pour les classes (nombre de dépendances entre classes) a été utilisée.

Nous avons choisi d'utiliser ce critère afin de permettre à l'algorithme de traiter à la fois l'aspect de la couverture et l'aspect de la pertinence des classes sélectionnées.

La maximisation de la couverture est obtenue en créant divers groupes de classes qui couvrent différents éléments du code source. La pertinence des classes sélectionnées (risque de contenir des fautes) est pour sa part obtenue par la mesure de complexité de code proposé, puisque les métriques LOC et CBO sont généralement reliées à la présence de fautes dans un logiciel [Basili *et al.*, 1996, Gyimothy *et al.*, 2005, He *et al.*, 2015].

Après l'ordonnancement, les classes sont sélectionnées dans l'ordre pour chaque groupe. D'abord, la première classe du premier groupe est sélectionnée, puis la première classe du deuxième groupe est sélectionnée et ainsi de suite. Les groupes n'ont pas de relation d'ordre de définie entre eux. Nous les ordonnons donc par ordre de création. Comme les classes sont initialement disposées en groupes d'un élément, ordonnés aléatoirement, nous obtenons au final des groupes ordonnés aléatoirement. En utilisant un nombre de groupes plus petit que le nombre de classes sélectionnées, il est assuré que les classes les plus importantes de chaque groupe seront sélectionnées. Notons ici qu'il est intéressant d'avoir des groupes équilibrés puisque chaque groupe apporte de l'information sur des parties complémentaires du logiciel. En sélectionnant dans tous les groupes, nous couvrons donc plusieurs parties du logiciel.

3.3.6 Résumé des approches de contrôle

Nous présentons, dans le tableau 3.2, un résumé des approches de contrôle utilisées dans ce mémoire.

3.4 Approche de priorisation développée — Score local

L'approche de priorisation développée, appelée score local (SL), repose sur les objectifs visés, c'est-à-dire :

1. Maximiser la couverture
2. Maximiser les fautes détectées

TABLE 3.2 – Approches de contrôle

Code	**Algorithme**	**Fonction d'évaluation**
ALEA-couv	Séquence aléatoire	Couverture directe en LOC
GL-couv	Glouton	Couverture directe en LOC
GL-comp	Glouton	Métrique de complexité du code
GL-risque (voir tableau 3.3)	Glouton	Risque de fautes selon un modèle d'apprentissage automatique
DG-couv	Descente de gradient	Couverture directe en LOC
CBGA-ES	Algorithme génétique	Couverture directe en LOC
		Métrique de complexité du code
		Simplicité des tests
CH	Classification hiérarchique	Couverture directe en LOC
		Métrique de complexité du code

3. Limiter l'effort de test

4. Être applicable en dehors du contexte des tests de régression

L'approche vérifie ces quatre critères. D'abord, l'algorithme évalue pour chaque classe un risque qu'elle contienne au moins une faute à l'aide de modèles de classification pré-entraînés. Ces modèles retournent une probabilité d'erreur. Ensuite, pour chaque classe, un score initial est déterminé. Ce score correspond à la probabilité que la classe contienne une faute additionnée à la probabilité pondérée que les types dépendants contiennent une faute. La pondération se fait en fonction du degré de couplage entre la classe et sa classe dépendante qui apparaît sous la forme d'un coefficient noté λ. Le degré de dépendance de la classe j à la classe i ($\lambda_{i,j}$) est estimé par l'équation suivante :

$$\lambda_{i,j} = \begin{cases} 1 \text{ , si appel}(j,i) \leq CIS(j) \\ \dfrac{\text{appel}(j,i)}{\text{CIS}(j)}, \text{ sinon} \end{cases} \tag{3.18}$$

Où appel(j,i) est le nombre d'appels de méthodes de j à partir de i et CIS(j) est la métrique *Classe Size Interface* pour la classe j. La métrique CIS retourne le nombre de méthodes publiques de la classe. Comme nous utilisons uniquement l'analyse statique de code, il est impossible de déterminer dans tous les cas la méthode de j appelée par i, et par conséquent, nous utilisons cette estimation dans tous les cas. Cette estimation repose sur l'hypothèse que tous les appels de j dans i se font sur des méthodes distinctes. Également, pour éviter de surpondérer, le coefficient est limité à 1. Les méthodes privées sont exclues dans l'estimation, car la seule façon de les couvrir

est par des appels internes qui se font par les méthodes publiques. Il serait incorrect de les considérer puisque les méthodes privées ne participent pas au couplage entrant.

Ensuite, la classe avec le plus haut score est sélectionnée comme classe candidate au test. La classe voit sa probabilité de contenir des fautes multipliées par 0.4. La valeur 0.4 a été déterminée en évaluant plusieurs seuils et en sélectionnant celui offrant les meilleurs résultats. Également, la probabilité qu'une classe dépendante contienne des fautes est révisée en étant multipliée par le facteur $1 - \lambda_{i,j}$. Donc, plus la dépendance est forte, plus on considère la classe couverte par les tests. Le score de chaque classe est ensuite mis à jour avec les valeurs des nouvelles probabilités. Comme notre approche actualise à chaque étape les scores, elle s'inscrit dans la stratégie « additionnelle » des approches de priorisation.

Également, pour limiter la recherche, pour être une candidate au test, une classe doit vérifier les deux conditions suivantes :

1. Avoir une probabilité de contenir des fautes d'au moins 5 %
2. Avoir un score d'au moins 0.5

Ces deux valeurs ont aussi été déterminées par une série d'expérimentations avec différents seuils et elles offraient les meilleurs résultats.

3.4.1 Algorithmes d'apprentissage automatique

Afin de déterminer la probabilité qu'une classe contienne des fautes, deux algorithmes d'apprentissage automatique sont utilisés (k plus proches voisins et forêt aléatoire). L'algorithme K-NN est classé par certains auteurs comme algorithme de classifcation [Sammut et Webb, 2011], mais il sera utilisé dans ce travail dans un contexte d'apprentissage automatique. Nous nous limitons à ces algorithmes, car c'est avec ces algorithmes que les meilleurs résultats ont été observés dans [Ouellet et Badri, 2021]. L'algorithme C5.0 offrait aussi d'excellents résultats, mais après une série initiale d'expérimentations, les résultats obtenus en combinant cet algorithme avec KNN et RF n'étaient pas meilleurs que KNN et RF combiné. Nous l'avons donc mis de côté pour les présents travaux. Les algorithmes d'apprentissage automatique sont des algorithmes qui apprennent à prédire un caractère sur un système et qui peuvent être utilisés pour prédire le même caractère sur d'autres systèmes. Dans la présente étude, nous

avons deux scénarios d'apprentissage, intra-système et inter-systèmes. L'apprentissage intra-système consiste à réaliser l'apprentissage visant à prédire les caractéristiques d'un système sur des versions différentes du même système. Dans la littérature, les versions précédentes sont utilisées pour réaliser la prédiction [He *et al.*, 2015, Rahman *et al.*, 2012]. Cette approche a pour conséquence de rendre notre approche basée sur l'historique. Nous souhaitons éviter cette dépendance. Nous utilisons l'approche intra-système afin d'obtenir un ensemble de données d'entraînement où les différences entre le système à prédire et les systèmes d'entraînement sont limitées. L'approche inter-systèmes, pour sa part, consiste à utiliser des systèmes différents pour l'entraînement des modèles. Cette approche mène généralement à des résultats moins bons que ceux de l'intra-système [Ouellet et Badri, 2021, Rahman *et al.*, 2012, Zimmermann *et al.*, 2009], mais mesure réellement l'efficacité de l'approche dans une situation réelle où des données similaires ne sont pas disponibles. Notons que l'approche inter-systèmes revient, en fait, à l'évaluation de l'extrapolation et que nous n'avons aucune garantie de performance dans ce contexte. Cela survient, par exemple, lors de la priorisation des tests sur la première version d'un nouveau système ou si les versions précédentes n'ont pas été suffisamment testées et utilisées pour avoir des données fiables sur les fautes. Remarquons, ici, que les versions différentes d'un même système ne sont pas exclues du processus de prédiction inter-systèmes. La prédiction inter-systèmes inclus donc les différentes versions d'un même système en plus de l'ensemble des versions des autres systèmes.

Autant dans le cas intra-système qu'inter-systèmes, au plus dix systèmes sont utilisés pour l'entraînement, ce qui mène au développement d'au plus dix modèles de prédiction. Dans le cas où plus de dix systèmes peuvent être utilisés, ceux utilisés sont sélectionnés au hasard. Ensuite, la probabilité qu'une classe contienne une faute est calculée pour chaque modèle. La probabilité initiale qu'une classe contienne une faute est la moyenne obtenue pour chacun des modèles.

Les prochaines sections présentent les deux algorithmes d'apprentissage automatique utilisés dans le cadre du présent travail. Nous nous sommes limités à deux algorithmes en raison des résultats obtenus dans [Ouellet et Badri, 2021] qui indiquent que ces deux algorithmes (KNN et forêt aléatoire) permettent la meilleure classification des fautes autant dans un contexte intra-systèmes qu'inter-systèmes. Nous résumons par la suite quelques aspects méthodologiques des travaux présentés dans [Ouellet et Badri, 2021] afin d'expliquer le processus de sélection des métriques.

K plus proches voisins

L'algorithme des k plus proches voisins (*k nearest neighbours* — KNN) [Choudhary *et al.*, 2018, Elish, 2014, Shatnawi, 2012, Sammut et Webb, 2011] projette chaque observation de l'ensemble d'apprentissage dans un espace de dimension égale au nombre de variables explicatives. La prédiction consiste à projeter l'observation dans l'espace obtenu puis de trouver ses voisins les plus proches. Le nombre de voisins considérés peut être ajusté sans contrainte, mais généralement un nombre impair de voisins est utilisé. Dans le cadre actuel, nous avons utilisé cinq voisins. Le choix a été fait après avoir testé différentes valeurs de nombre de voisins. Ensuite, un vote est pris entre chaque voisin pour déterminer quel groupe de voisins est le plus présent (ex. classes fautives ou non fautives). De ce vote, une probabilité que l'observation appartienne au groupe fautif (modalité positive dans nos expérimentations) est calculée. Cette probabilité est ensuite utilisée dans notre approche pour évaluer le risque initial qu'une classe soit fautive. L'utilisation d'un nombre impair de voisins empêche la présence d'ex aequo dans le cadre d'une classification binaire.

La métrique utilisée permet de combiner divers types de variables dans la génération de l'espace de représentation des observations lors de la phase d'entraînement. Elle permet aussi de considérer des variables continues qui ont des différences d'échelles sans subir de biais dû à cette différence. Par exemple, deux variables X et Y dont l'étendue est respectivement de 0 à 10 et de 0 à 100. Une variation de 5 pour X est, de façon relative, beaucoup plus importante qu'une variation de 5 pour Y. La métrique euclidienne ne tiendrait pas compte de cette différence. Nous avons utilisé la métrique HEOM (*Heteorogenous Euclidean Overlap Metric*) qui est définie comme suit :

$$\mathrm{HEOM}(\mathbf{p}, \mathbf{q}) = \frac{\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}}{|\max(X_i)) - \min(X_i)|} \tag{3.19}$$

avec $\mathbf{p}$ et $\mathbf{1}$ deux vecteurs de données de n composantes, et X_i la variable de la ième composante des vecteurs $\mathbf{p}$ et $\mathbf{q}$. La métrique HEOM est aussi définie pour des valeurs discrètes, mais nous n'en utilisons pas dans le cadre de cette étude, alors, nous laisserons cette partie de la définition de HEOM de côté. Nous avons utilisé l'implémentation décrite par Hechenbichler et Schliep [Hechenbichler et Schliep, 2004].

Forêt aléatoire

L'algorithme de forêt aléatoire (*Random Forest* — RF) [Sammut et Webb, 2011, Choudhary *et al.*, 2018, Moeyersoms *et al.*, 2015, Kaur et Malhotra, 2008] sélectionne un certain sous-ensemble de variables explicatives et d'observations afin d'entraîner un modèle d'arbre de décision. La procédure est répétée afin d'obtenir un certain nombre d'arbres. Nous avons déterminé, à l'aide d'expérimentations préliminaires où nous avons varié les paramètres, que cinquante arbres menaient à de bons résultats. Nous avons utilisé l'implémentation décrite par Liaw et Wiener dans [Liaw et Wiener, 2002].

Un arbre de décision est une structure de graphe ou chaque nœud de l'arbre comprend une décision par rapport à une variable et un seuil donné. Chaque feuille (sommet sans arête sortant) est associée à une classe. Initialement, on laisse croître les arbres jusqu'à ce que chaque feuille contienne que des éléments d'une même classe. Toutefois, dans notre méthodologie, afin de pouvoir traiter efficacement de grands ensembles, nous avons élagué les arbres (fusionné des feuilles) pour obtenir un minimum de cinq observations par feuille. Pour la prédiction, une observation traverse tous les arbres et est assignée à la classe correspondant au vote majoritaire. Dans le cadre de notre approche, une probabilité de bonne classification est dérivée des résultats des arbres et sert comme probabilité initiale qu'une classe a de contenir des fautes.

Sélection des métriques et mesures utilisées

Cinq métriques (orientées objet) et mesures (de centralité) ont été sélectionnées comme variables explicatives à partir d'un ensemble de dix-huit métriques et mesures par une méthode visant à assurer :

1. Un lien entre les variables explicatives (métriques et mesures) et la variable expliquée (présence)
2. Une différence marquée entre l'information apportée par les variables pour éliminer la colinéarité entre les variables explicatives.

La méthodologie complète de sélection est décrite dans [Ouellet et Badri, 2021]. Les métriques et mesures retenues sont CE, CL, KATZ, RFC et SLC.

La variable à prédire pour l'entraînement des modèles est une variable binaire appelée « présence » qui prend les valeurs suivantes pour une classe c.

$$\text{presence}(c) = \begin{cases} 0, \text{ si } c \text{ ne contient pas de fautes} \\ 1, \text{ si } c \text{ contient une faute ou plus} \end{cases} \tag{3.20}$$

Sur-échantillonnage

Durant la phase d'apprentissage, une étape de sur-échantillonnage est incluse afin d'équilibrer le nombre d'observations dans chaque classe (fautive et non fautive). Le sur-échantillonnage SMOTE [He et Garcia, 2009] consiste à générer des observations dites « synthétiques » à partir des observations existantes. Les observations synthétiques sont obtenues à partir des caractéristiques des observations réelles de la classe la moins populeuse. Elles sont générées à partir d'interpolation linéaire d'observations réelles. D'abord, une observation réelle est sélectionnée (x) et parmi ses cinq plus proches voisins, une autre observation réelle est choisie aléatoirement ($\hat{x}$). Ensuite, un coefficient δ est généré aléatoirement dans l'intervalle (0,1).

$$x_{\text{syn}} = x + \delta(\hat{x} - x) \tag{3.21}$$

avec x_{syn} l'observation synthétique générée. Cette approche de sur-échantillonnage a été sélectionnée dans [Ouellet et Badri, 2021], car elle améliore la qualité de la prédiction, particulièrement en contexte intra-système et inter-systèmes.

Variantes des algorithmes

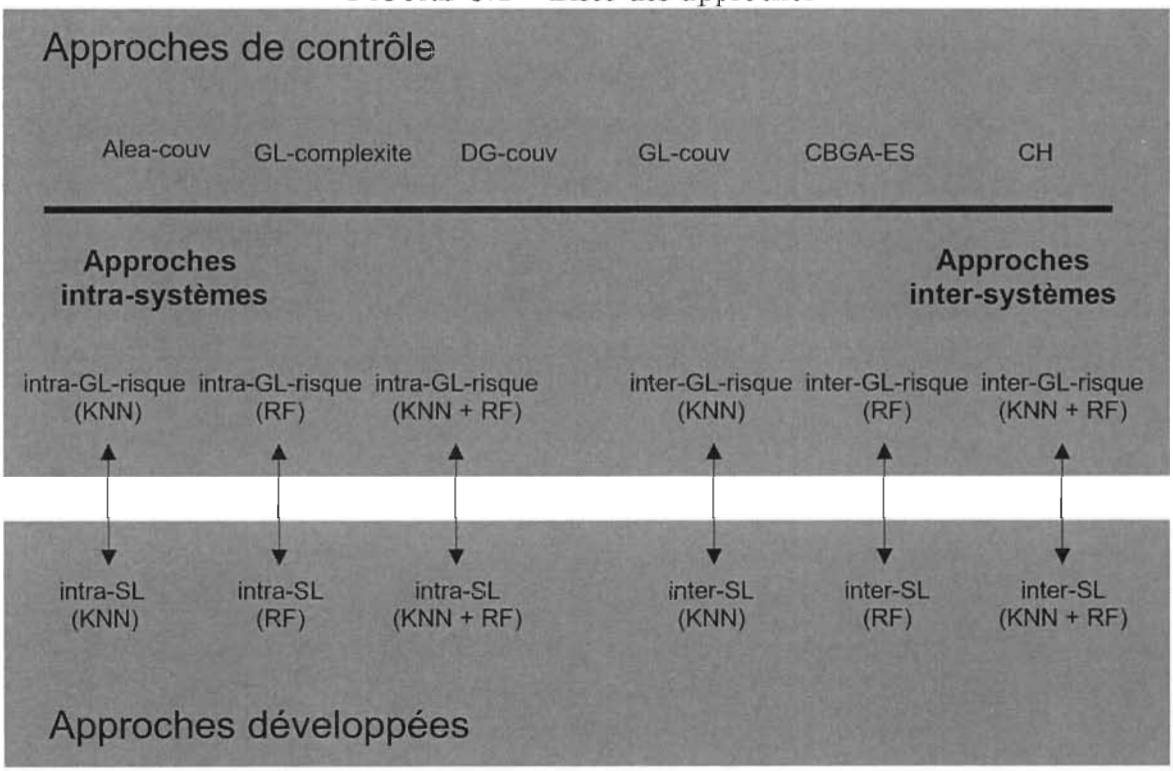
L'approche développée sera évaluée selon six variantes telles que présentées dans le tableau 3.3. En raison de l'utilisation des algorithmes d'apprentissage, l'algorithme glouton basé sur le risque aura également une variante pour chaque variation de notre approche. 6 variantes seront évaluées au total : 3, pour les variations d'algorithme d'apprentissage automatique utilisé, que l'on multiplie par 2, pour les méthodologies utilisées (intra et inter-systèmes). La sélection aléatoire des systèmes utilisés pour réaliser la prédiction est la même pour l'approche SL que l'algorithme de contrôle glouton. Nous avons choisi de procéder ainsi pour éliminer la variance due à la sé-

TABLE 3.3 – Variantes de l'approche de priorisation développée et de l'algorithme glouton basé sur le risque

Données d'entraînement	**Algorithme**	**Approche développée (SL)**	**Algorithme de contrôle glouton basé sur le risque**
Intra-système	KNN	Intra-SL (KNN)	Intra-GL-risque (KNN)
	RF	Intra-SL (RF)	Intra-GL-risque (RF)
	KNN et RF	Intra-SL (KNN+RF)	Intra-GL-risque (KNN+RF)
Inter-systèmes	KNN	Inter-SL (KNN)	Inter-GL-risque (KNN)
	RF	Inter-SL (RF)	Inter-GL-risque (RF)
	KNN et RF	Inter-SL (KNN+RF)	Inter-GL-risque (KNN+RF)

lection des données d'entraînement et pour pouvoir comparer l'approche SL avec la référence uniquement basée sur la prédiction faite par le modèle d'apprentissage automatique. Si l'ordonnancement naïf (glouton) à partir du modèle d'apprentissage donne de meilleurs résultats que les étapes supplémentaires de notre approche il n'y alors aucun gain au temps alloué aux étapes de calcul supplémentaires. La figure 3.1 résume les différences approches et les liens les unissant.

FIGURE 3.1 – Liste des approches



3.5 Métriques d'évaluation

L'évaluation des approches de priorisation se fait selon 4 axes : la couverture, la détection des fautes, l'efficience de la détection et la qualité de l'ordonnancement. La plupart de ces axes sont divisés selon trois types : total, direct et indirect. Le type direct consiste à mesurer la métrique d'évaluation sur les classes candidates au test. Soit M_d une métrique d'évaluation de type directe et T un ensemble ordonné de classes candidates au test.

$$M_d(T) = \sum_{t \in T} m_d(t) \tag{3.22}$$

Où m_d est la fonction permettant d'évaluer la métrique M_d sur une classe en particulier. Le type indirect est appliqué sur toutes les classes dont dépendent les classes candidates au test. Soit M_i une métrique d'évaluation de type indirect, T un ensemble ordonné de tests et G le graphe des dépendances du système.

$$M_i(T) = \sum_{\{u | \forall\ t \in T,\ (t,u) \in V(G) \wedge u \notin T\}} m_i(u) \tag{3.23}$$

Où m_i est la fonction permettant d'évaluer la métrique M_i sur une classe en particulier. La couverture indirecte est définie comme la couverture indirecte « propre ». Si une classe A se trouve dans la couverture indirecte de B et que A est candidate au test, alors A est exclue des calculs de la couverture indirecte. Toutes les fonctions permettant de calculer une métrique d'évaluation sont positives ($m \geq 0$), alors le type indirect tel que calculé dans le présent mémoire sous-estime les valeurs par rapport à la couverture des classes dépendantes aux classes candidates au test.

$$M_i(T) \leq \sum_{\{u | \forall\ t \in T,\ (t,u) \in V(G)\}} m_i(u) \tag{3.24}$$

Cette surestimation est due simplement aux relations suivantes :

$$|V(G)| \leq |V(G) \wedge u \notin V(G)| \tag{3.25}$$

$$m(v) \geq 0 \tag{3.26}$$

La couverture indirecte s'appuie sur l'hypothèse que tous les éléments dans une classe dépendante sont atteints. Par exemple, soit A et B deux classes avec A qui dépend de

B. La classe B est composée de 2 méthodes m_1 et m_2. Lors de l'exécution, A appelle seulement m_1 dans B, mais n'appelle pas m_2. Dans le calcul des mesures indirectes, les valeurs de m_2 seront aussi comptées. Cette hypothèse facilite grandement le calcul des métriques d'évaluation, mais surestime leurs valeurs. Cette surestimation est faite pour toutes les approches.

Le type total représente l'évaluation de la métrique sur toutes les classes candidates et les classes dont elles dépendent. Soit M une métrique d'évaluation de type total, T un ensemble ordonné de classes candidates au test et G le graphe des dépendances du système.

$$M(T) = \sum_{v \in \{u | \exists t \in T,\ (t,u) \in V(G) \vee u \in T\}} m(v) \tag{3.27}$$

Où m est la fonction permettant d'évaluer la métrique M sur une classe en particulier. On peut aussi vérifier que pour une métrique d'évaluation M_d et M_i ayant la même fonction m pour évaluer la métrique sur une classe.

$$\begin{aligned}
M_d(T) + M_i(T) &= \sum_{t \in T} m(t) + \sum_{\{u | \forall\ t \in T,\ (t,u) \in V(G)\ \wedge\ u \notin T\}} m(u) && (3.28)\\
&= \sum_{v \in \{u | \forall\ t \in T,\ (t,u) \in V(G)\ \wedge\ u \notin T\}\ \cup\ v \in T} m(u) && (3.29)\\
&= \sum_{v \in \{u | (\forall\ t \in T,\ (t,u) \in V(G)\ \wedge\ u \notin T)\ \vee\ v \in T\}} m(u) && (3.30)\\
&= \sum_{v \in \{u | (\forall\ t \in T,\ (t,u) \in V(G)\ \vee\ u \in T)\ \wedge\ (u \notin T\ \vee\ u \in T)\}} m(u) && (3.31)\\
&= \sum_{v \in \{u | \exists t \in T,\ (t,u) \in V(G)\ \vee\ u \in T\}} m(v) = M(T) && (3.32)
\end{aligned}$$

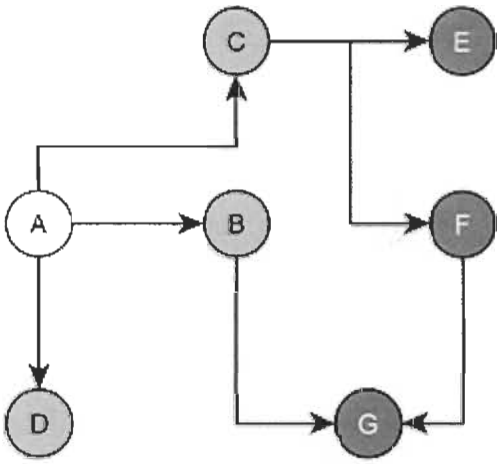
Par conséquent, la couverture totale correspond à la somme des couvertures directes et indirectes. Cette identité est due à la définition du type indirecte comme la couverture indirecte « propre ».

La principale raison qui nous amène à retenir cette approche est que nous ne possédons pas l'information sur la distribution des fautes à une granularité plus fine que celle des classes. Donc, pour estimer les fautes dans chaque méthode, il nous faudrait faire des estimations supplémentaires et nous appuyer sur des hypothèses pouvant être fausses. Pour cette raison, nous avons choisi de procéder de la façon décrite ci-dessus. Nous avons gardé cette façon de procéder pour toutes les métriques

d'évaluation afin d'assurer une uniformité dans la démarche et faciliter l'interprétation des résultats en adoptant une méthodologie unique.

Sur la figure 3.2, on peut voir une illustration des trois types de couverture. La couverture directe de la classe A se limite au sommet bleu sur le graphe. La couverture indirecte est composée des classes B, C et D identifiées en vert sur le graphe. La couverture totale de A est composée des classes A, B, C et D. Les classes en gris : E, F et G, sont ignorées dans tous les calculs de couverture pour A.

FIGURE 3.2 – Types de couverture



3.5.1 Couverture

La couverture des classes candidates est évaluée selon deux critères. Le premier est le nombre de classes couvertes. La couverture directe correspond au nombre de classes candidates au test. La fonction m pour évaluer cette métrique sur une classe candidate au test t dans un système de n classes est :

$$m(t) = \frac{1}{n} \times 100\% \tag{3.33}$$

La mesure relative permet de comparer les résultats entre les systèmes. Pour plusieurs approches, la mesure du nombre de classes couvertes directement était fixée comme paramètre de l'approche.

La couverture peut aussi être évaluée par le nombre de lignes de code couvertes. Dans ce cas, le nombre de lignes de code de chaque classe couverte est utilisé. La fonction m permettant d'évaluer cette métrique sur une classe candidate au test t dans un système constitué des classes de l'ensemble C est :

$$m(t) = \frac{\text{LOC}(t)}{\sum_{c \in C} \text{LOC}(c)} \times 100\% \tag{3.34}$$

Avec LOC la fonction qui retourne le nombre de lignes de code dans une classe.

Les métriques d'évaluation de la couverture directe servent également à estimer l'effort de test associé à un ensemble de classes candidates. Plus la taille de l'ensemble est importante, plus l'effort de test sera important.

3.5.2 Détection des fautes

La détection des fautes est évaluée selon le nombre de fautes trouvées pour les classes candidates au test. Comme nous ne disposons pas de suites de tests écrites, nous estimons la capacité de détection des tests par le nombre de fautes contenues dans chaque classe. Cette hypothèse implique que si les classes candidates sont testées, alors toutes leurs fautes seront révélées par le test. Il s'agit d'une estimation du meilleur cas, mais en l'absence de tests, nous ne pouvons pas évaluer dans quelle mesure ceux-ci seront efficaces. Également, l'efficacité des tests dépend de plusieurs facteurs externes à nos expérimentations. Nous nous en tennons donc à l'estimation du meilleur cas. Notons que ceci n'introduit pas de biais dans nos comparaisons, car toutes les approches sont évaluées dans leur meilleur cas.

La fonction m permettant d'évaluer le nombre de fautes dans une classe candidate au test t dans un système composé de l'ensemble des classes C est :

$$m(t) = \frac{\text{FAUTES}(t)}{\sum_{c \in C} \text{FAUTES}(c)} \times 100\% \tag{3.35}$$

Avec FAUTES la fonction qui retourne le nombre de fautes dans une classe t. Encore une fois, cette métrique est relative afin de permettre les comparaisons entre les systèmes.

3.5.3 Efficience de la détection

Les métriques d'évaluation de l'efficience de la détection servent à mesurer le rapport entre l'effort de test déployé et le nombre de fautes trouvées. Couvrir des parties du code lors des tests n'est pas un objectif en soi ; le but du processus de test est de trouver des fautes. Bien que l'on puisse s'attendre à ce que la couverture atteinte et le nombre de fautes détectées soient positivement corrélés, une haute couverture, qui ne trouve pas plus de fautes qu'une couverture de code plus faible, demande un effort de test supplémentaire et n'apporte aucun gain au final dans la qualité. L'évaluation de l'efficience permet de mettre en évidence ces situations.

Afin d'évaluer l'efficience de la détection, nous utilisons le rapport entre les métriques d'évaluation de couverture et les métriques d'évaluation de détection des fautes. Tous les ratios sont exprimés en nombre de fautes par millier de lignes de code couvertes directement. Nous l'exprimons en milliers de lignes de code pour obtenir des ratios près de 0 et alléger les notations. Pour chaque mesure de détection des fautes un ratio est calculé, soient le nombre de fautes directement trouvées par millier de lignes de code directement couvertes, le nombre de fautes indirectement trouvées par millier de lignes de code directement couvertes et le nombre total de fautes trouvées par millier de lignes de code directement couvertes.

La fonction m permettant d'évaluer l'efficience d'une classe candidate au test t d'un système C est :

$$m(t) = \frac{\text{FAUTES}(t)}{\text{couverture directe}(t)} \times 1000; \tag{3.36}$$

Un quatrième ratio, le rapport entre les deux premiers permet de déterminer si l'approche est plus efficace pour les fautes couvertes directement qu'indirectement.

Un ratio plus grand que 1 indique que la couverture directe est plus efficace que la couverture indirecte et un ratio plus petit que 1 indique l'inverse.

3.5.4 Qualité de l'ordonnancement

La métrique Average Percentage of Fault Detected (APFD) communément utilisée dans la littérature [Carlson *et al.*, 2011, Chen *et al.*, 2018, Li *et al.*, 2007, Rothermel *et al.*, 2001] a été utilisée pour évaluer la qualité de l'ordonnancement des classes candidates au test. La métrique APFD est définie comme suit pour l'évaluation sur des cas de test :

$$APFD = 1 - \sum_{i=1}^{m} \frac{TF_i}{mn} + \frac{1}{2n} \tag{3.37}$$

Avec m le nombre de fautes, n le nombre de classes et TF_i la position de la classe dans la suite de tests révélant la $i^{\text{ème}}$ faute.

Nous avons adapté la mesure de deux façons. D'abord, pour des raisons de calculabilité, nous proposons la formulation équivalente suivante :

$$\sum_{i=1}^{m} TF_i = \sum_{i=1}^{n} if_i \tag{3.38}$$

Avec f_i le nombre de fautes contenues dans la ième classe. Deuxièmement, comme toutes les fautes ne sont pas détectées par les classes candidates pas plus que toutes les classes présentes dans l'ensemble des classes candidates au test, la valeur de m est le nombre de fautes présentes dans l'ensemble de classes candidates et n le nombre de classes candidates.

$$APFD = 1 - \sum_{i=1}^{n} \frac{if_i}{mn} + \frac{1}{2n} \tag{3.39}$$

Les impacts de cette reformulation sont d'abord que l'APFD doit être interprétée conjointement avec le nombre de fautes détectées. Le fait que deux systèmes n'ont pas le même nombre de fautes ou des ensembles de classes candidates de même taille peut influencer la valeur de l'APFD. Ensuite, les valeurs obtenues sont plus faibles que celles dans la littérature, car dans plusieurs approches, les cas de test ne révélant pas d'erreur sont écartés tandis que de nombreuses classes candidates peuvent ne pas contenir

de fautes. De plus, l'APFD mesure l'ordonnancement seulement pour la détection directe. Il serait hasardeux d'adapter la mesure pour considérer la couverture indirecte et directe tout en restant borné dans l'intervalle [0, 1]. Nous tenons à respecter les bornes, car la mesure APFD, telle que définit ailleurs dans la littérature est toujours dans l'intervalle [0, 1].

Le tableau 3.4 présente toutes les métriques d'évaluation utilisées pour évaluer les approches de priorisation.

TABLE 3.4 – Liste des métriques d'évaluation selon l'axe évalué

Axe	Métrique d'évaluation
Taille	Couverture directe des classes
	Couverture directe des lignes de code
	Couverture indirecte des lignes de code
	Couverture totale des lignes de code
Détection des fautes	Fautes trouvées directement
	Fautes trouvées indirectement
	Total des fautes trouvées
Efficience de la détection	Efficience de la détection directe
	Efficience de la détection indirecte
	Efficience de la détection totale
	Rapport de la détection directe et de la détection indirecte
Qualité de l'ordonnancement	APFD

3.6 Systèmes utilisés et collecte des données

Afin de réaliser nos différentes expérimentations, les données de 5 systèmes ont été collectées. Pour chaque système, entre 3 et 5 versions du système ont été sélectionnées pour un total de 20 versions. Le tableau 3.5 présente les caractéristiques de chaque version. Nous décrivons aussi brièvement chaque système et son champ d'application.

- Ant est un outil assistant la compilation qui propose divers utilitaires de gestion de dépendances et d'assurance qualité pour les systèmes Java principalement. Il fait partie de l'Apache Software Foundation (ant.apache.org). Ce système a été étudié dans plusieurs travaux [Anwer *et al.*, 2017, Chen *et al.*, 2016, He *et al.*, 2015, Jureczko et Madeyski, 2010].

- Camel est un système qui permet la communication entre diverses applications en utilisant divers protocoles définis par l'utilisateur ou régis par des normes. Il s'agit également d'un projet de l'Apache Software Foundation (camel.apache.org) et a été étudié dans plusieurs travaux [Anwer *et al.*, 2017, He *et al.*, 2015, Jureczko et Madeyski, 2010].
- JEdit est un éditeur de texte libre dont le code source est public. Il est conçu pour supporter facilement des extensions et permet la coloration syntaxique de plus de 200 langages (www.jedit.org). Ce système a été étudié dans les travaux suivants : [Anwer *et al.*, 2017, He *et al.*, 2015, Jureczko et Madeyski, 2010, Kaur et Malhotra, 2008].
- POI permet de manipuler les fichiers créés par la suite de bureautique de Microsoft, Office. Il s'agit aussi d'un projet de l'Apache Software Foundation (poi.apache.org). Il a également été étudié dans plusieurs travaux [Anwer *et al.*, 2017, He *et al.*, 2015, Jureczko et Madeyski, 2010].
- Xalan permet de transcrire des documents du format XSLT vers d'autres standards XML dont HTML. Le projet fait partie de l'Apache Software Foundation (xalan.apache.org). Ce système a été étudié dans plusieurs travaux [Anwer *et al.*, 2017, He *et al.*, 2015, Jureczko et Madeyski, 2010].

La collecte des données s'est faite avec deux outils. Pour les métriques orientées objet et l'extraction des graphes de dépendances, nous avons utilisé l'outil CKJM [Spinellis, 2005]. Pour le calcul des mesures de centralité, nous les avons implémentées dans un outil que nous avons nous-même développé. Nous avons aussi développé l'outil pour appliquer les algorithmes de priorisation. La plupart des algorithmes ont été complètement implémentés dans notre outil, mais pour certains, notamment la classification hiérarchique et l'apprentissage automatique, nous avons utilisé des bibliothèques de l'outil R [Team, 2020] pour exécuter certains traitements.

TABLE 3.5 – Description des systèmes étudiés

		Taille		Fautes		
Système	**Version**	**Milliers de LOC**	**Nombre de classes**	**Nombre total de fautes**	**Pourcentage de classes fautives**	**Nombre moyen de fautes par classe**
Ant	1.3	38	126	135	48 %	1,1
	1.4	60	265	47	15 %	0,2
	1.5	95	401	35	8 %	0,1
	1.6	127	523	184	18 %	0,4
	1.7	234	1062	338	16 %	0,3
Camel	1.2	147	765	522	28 %	0,7
	1.4	216	1122	335	13 %	0,3
	1.6	249	1252	500	15 %	0,4
JEdit	3.2.1	146	508	382	18 %	0,8
	4.0	167	606	226	12 %	0,4
	4.1	178	644	217	12 %	0,3
	4.2	206	805	106	6 %	0,1
POI	1.5	58	294	342	48%	1,2
	2.0	97	380	39	10%	0,1
	2.5.1	124	460	496	54%	1,1
	3.0	135	531	500	53%	0,9
Xalan	2.4	225	813	156	14 %	0,2
	2.5	305	902	522	42 %	0,6
	2.6	422	1158	624	35 %	0,5
	2.7	442	1192	1212	75 %	1,0
Total	*20*	*3 670*	*13 789*	*6 903*	-	*10,6*
Moyenne	*4*	*184*	*689*	*345*	*27 %*	*0,5*

3.7 Conclusion du chapitre 3

Dans ce chapitre, les principaux concepts qui seront utilisés dans les expérimentations ont été introduits. D'abord, l'adaption du problème à un contexte plus large que les tests de régression a été présentée. Par la suite, les tests statistiques ont été détaillés. J'ai également introduit les approches de contrôle qui seront utilisés pour évaluer les résultats de l'approche développée. Ensuite, l'approche SL a été expliquée en détail. J'ai traité également des métriques d'évaluation qui seront utilisées pour évaluer les approches de priorisation. Finalement, le jeu de données utilisé à été présenté. La prochaine section traitera des résultats expérimentaux et de leur analyse en s'appuyant sur les concepts vus dans cette section.

Chapitre 4

Expérimentations

Le chapitre 4 présente d'abord quelques caractéristiques des systèmes étudiés. Ensuite, les résultats des approches de priorisation sans variante sont détaillés. Le chapitre continue en présentant les résultats des approches intra-systèmes puis inter-systèmes. Par la suite, quelques particularités du système Xalan sont explorées. Finalement, une synthèse de tous les résultats conclut le chapitre.

4.1 Caractéristiques des systèmes

Les statistiques descriptives des systèmes (taille et nombre de fautes) ont été présentées précédemment à la section 3.6. Dans cette section, les statistiques descriptives des métriques et mesures de centralité sont présentées, suivies par une étude de corrélation entre le couplage et la taille des systèmes.

4.1.1 Distribution des métriques

Nous présentons dans le tableau 4.1 une brève statistique descriptive des métriques utilisées dans l'implémentation de l'approche SL. Ces métriques donnent aussi une vue d'ensemble du couplage (CE, CL, SLC et KATZ), de la complexité (RFC) et de la

distribution des classes (CL, SLC) du point de vue des mesures de centralité. Les résultats sont agrégés par système, donc les moyennes des valeurs sont présentées, à l'exception de la médiane où la médiane des médianes des systèmes est présentée.

Le tableau 4.1 des statistiques descriptives montre que le couplage est très variable à l'intérieur d'un même système avec des coefficients de variation tous supérieurs à 1. Les médianes sont en revanche semblables entre les systèmes indiquant un comportement similaire du couplage. D'importantes variations entre les systèmes existent pour la mesure CL, mais les variations internes des systèmes sont moins prononcées avec des coefficients de variation inférieurs à 1. La mesure KATZ, et les métriques RFC et SLC ont toutes des coefficients de variation légèrement supérieurs à 1 avec quelques exceptions. Leurs moyennes et médianes varient également de façon significative entre les logiciels, illustrant une hétérogénéité dans les dispersions des valeurs de ces métriques.

4.1.2 Couplage et taille

Le couplage entrant joue un rôle très important dans notre approche. En effet, les classes candidates considèrent dans leur couverture indirecte toutes les classes dont elles dépendent. Également, dans l'exécution de l'approche, il y a une étape de propagation de la sélection d'une classe comme candidate qui réduit la probabilité que ses dépendances soient testées.

Les systèmes étudiés manifestent en moyenne un fort lien entre leur couplage moyen entrant et leur taille en nombre de classes. Nous avons calculé la corrélation entre ces facteurs globalement et par système. Les résultats sont présentés au tableau 4.2 et les données servant à la corrélation à la figure 4.1.

Les systèmes Ant, Camel et JEdit admettent tous les trois une corrélation négative importante signifiant que plus la taille du logiciel est élevée, plus faible devient le couplage entrant moyen. Pour le système POI, la taille et le couplage entrant moyens ne semble que faiblement liés et pour le système Xalan, le contraire se produit : le couplage entrant moyen augmente en même temps que la taille des versions. Aucun test de significativité n'a été effectué sur ces coefficients de corrélation en raison de la

TABLE 4.1 – Statistique descriptive des métriques OO et mesures de centralité

Métrique	Statistique	Ant	Camel	JEdit	POI	Xalan
CE	Moyenne	4,73	5,58	4,72	4,12	6,96
	Écart-type	4,85	6,00	6,00	8,00	9,17
	Médiane	3,00	4,00	3,00	3,00	4,00
	Coefficient variation	1,03	1,07	1,27	1,95	1,33
CL	Moyenne	10,34	13,74	33,58	7,64	18,89
	Écart-type	8,01	12,47	23,10	10,39	17,51
	Médiane	9,91	11,48	34,54	2,75	14,32
	Coefficient variation	0,77	0,91	0,69	1,36	0,94
KATZ	Moyenne	0,18	0,09	0,04	0,16	0,12
	Écart-type	0,18	0,10	0,07	0,16	0,15
	Médiane	0,10	0,05	0,03	0,13	0,07
	Coefficient variation	1,04	1,15	1,70	1,01	1,26
RFC	Moyenne	27,96	18,32	25,47	25,5	25,19
	Écart-type	29,77	21,28	42,91	31,85	33,99
	Médiane	16,50	11,00	13,00	20,00	13,50
	Coefficient variation	1,08	1,16	1,68	1,25	1,35
SLC	Moyenne	120329,66	177467,33	65005,25	63624,92	89907,92
	Écart-type	122538,93	183087,62	78632,61	65975,56	114432,91
	Médiane	54674,00	156288,50	40510,75	34240,50	48160,00
	Coefficient variation	0,94	1,03	1,2	1,04	1,28

trop petite taille des échantillons. Pour la réunion de tous les systèmes, une corrélation négative de -0.61 est observée et est significative au seuil de 5 % (notée en gras).

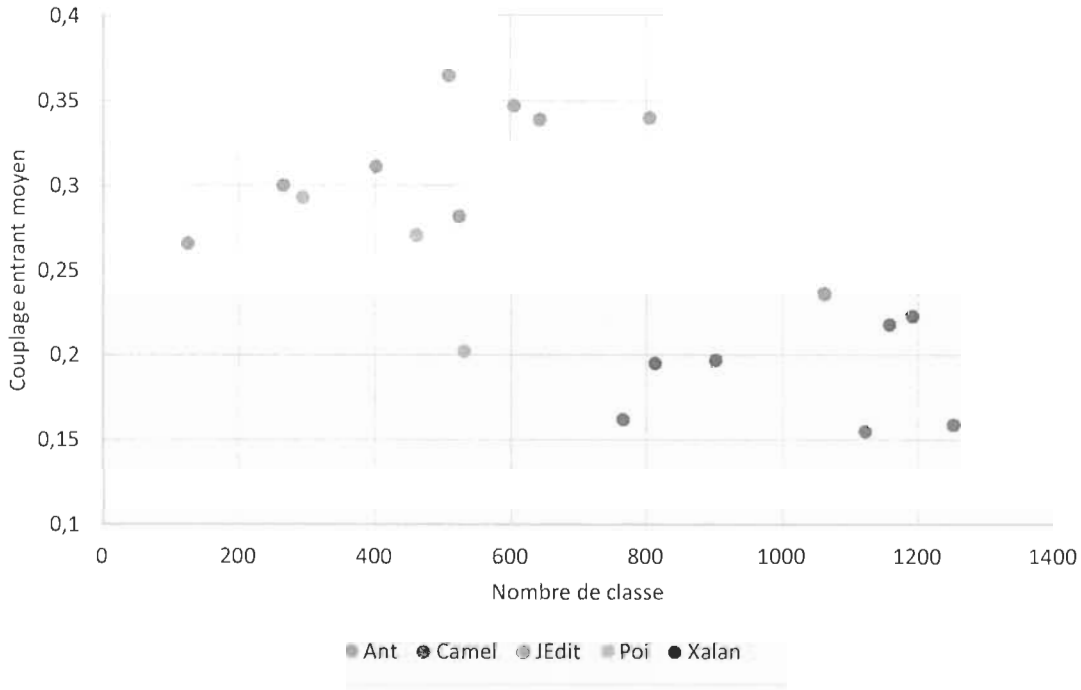
4.2 Approches de contrôle sans variante

La plupart des approches de contrôle que nous utilisons n'ont pas de variante, c'est-à-dire, qu'il n'est pas possible de proposer des approches utilisant le même algorithme avec des paramètres différents qui s'accordent avec ceux de l'approche développée. Afin d'éviter de répéter la présentation des résultats, ceux-ci sont groupés et présentés uniquement dans cette section. Les approches de contrôle sans variante sont

TABLE 4.2 – Corrélations entre le couplage entrant moyen et la taille des systèmes en nombre de classes

Système	Taille d'échantillon	Corrélation	p-valeur
Ant	5	-0,64	
Camel	3	-0,65	
JEdit	4	-0,80	
POI	4	0,09	
Xalan	4	0,99	
Tout	*20*	*-0,61*	***0,004***

FIGURE 4.1 – Couplage entrant moyen et la taille des systèmes en nombre de classes



couverture aléatoire (ALEA-couv), glouton pour la couverture (GL-couv), glouton pour la complexité (GL-comp), descente de gradient pour la couverture (DG-couv), classification hiérarchique (CH) et l'algorithme génétique (CBGA-ES). Tous ces algorithmes sont paramétrés pour sélectionner un ensemble de classes candidates au test de 15 % de la taille totale du système. Nous avons utilisé cette valeur, car il s'agit de la taille moyenne sélectionnée par l'approche SL, par conséquent, cela nous permet de comparer les approches comme elles proposent toutes un effort de test similaire.

4.2.1 Couverture

Les tableaux 4.3 à 4.5 présentent les informations pour la couverture des lignes de code atteinte par les approches de contrôle sans variante. On remarque qu'en moyenne les approches gloutonnes et l'approche de classification hiérarchique atteignent plus de 80 % de la couverture totale ; soit que 80 % des lignes de code du programme sont contenues dans les classes candidates où celles directement appelées par les candidates. Leur couverture directe se situe en moyenne entre 57 % et 62 %, ce qui indique que ces approches sélectionnent les classes volumineuses en termes de lignes de code comme candidates au test, car seules 15 % des classes sont sélectionnées. Elles font une utilisation limitée de la couverture indirecte (uniquement les classes directement appelées par les candidates). L'approche aléatoire suit une tendance proche du comportement aléatoire attendu. Elle atteint une couverture totale de 50 %, ce qui est surestimé par la répétition du processus qui assure de sélectionner une valeur au-dessus de la médiane, et une couverture directe de 15 % des lignes de code, ce qui est attendu, car seules 15 % des classes peuvent être sélectionnées comme candidates au test. Les approches de descente de gradient et d'algorithme génétique atteignent une couverture respective de 53 % et 61 % avec des couvertures directes plutôt basses de 19 % et 12 %. Elles atteignent alors leur couverture en utilisant la composante indirecte de la couverture.

Les différents systèmes admettent peu de variations dans la couverture atteinte. Les variations sont observées entre les approches. Les différences entre les systèmes sont donc dues à leur interaction avec les approches plutôt qu'à une caractéristique particulière du système.

TABLE 4.3 – Couverture totale (% LOC) des approches de contrôle sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	41,6 %	59,7 %	49,7 %	43,3 %	45,0 %	57,9 %
Ant 1.4	48,4 %	74,5 %	69,2 %	44,7 %	55,3 %	69,2 %
Ant 1.5	41,9 %	78,2 %	75,1 %	53,7 %	61,7 %	75,8 %
Ant 1.6	45,5 %	79,2 %	76,7 %	50,1 %	49,6 %	79,4 %
Ant 1.7	46,3 %	76,9 %	74,7 %	51,1 %	55,5 %	74,7 %
Camel 1.2	42,1 %	78,7 %	77,7 %	50,5 %	51,7 %	77,7 %
Camel 1.4	48,6 %	81,0 %	79,3 %	50,9 %	53,6 %	79,6 %
Camel 1.6	54,3 %	80,2 %	79,0 %	46,7 %	50,3 %	79,0 %
JEdit 3.2.1	45,0 %	93,6 %	89,7 %	70,2 %	75,0 %	89,7 %
JEdit 4.0	44,8 %	94,3 %	90,7 %	69,9 %	69,0 %	90,7 %
JEdit 4.1	62,5 %	91,9 %	91,0 %	71,4 %	72,4 %	91,0 %
JEdit 4.2	68,5 %	89,5 %	87,6 %	67,1 %	67,5 %	87,6 %
POI 1.5	49,3 %	88,5 %	92,5 %	65,5 %	77,9 %	92,5 %
POI 2.0	41,2 %	92,5 %	94,3 %	39,6 %	80,1 %	94,3 %
POI 2.5.1	25,6 %	92,3 %	94,6 %	40,7 %	82,3 %	94,6 %
POI 3.0	25,9 %	93,7 %	92,1 %	66,5 %	76,4 %	92,1 %
Xalan 2.4	54,1 %	87,5 %	82,2 %	45,2 %	60,6 %	82,2 %
Xalan 2.5	50,3 %	87,5 %	81,8 %	46,7 %	55,3 %	81,8 %
Xalan 2.6	42,2 %	87,5 %	82,0 %	43,9 %	42,6 %	82,0 %
Xalan 2.7	40,3 %	85,8 %	79,7 %	37,1 %	37,7 %	79,7 %
Moyenne	*45,9 %*	*84,7 %*	*82,0 %*	*52,7 %*	*61,0 %*	*82,6 %*

TABLE 4.4 – Couverture directe (% LOC) des approches de contrôle sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	14,5 %	44,2 %	40,3 %	17,7 %	9,1 %	38,9 %
Ant 1.4	16,3 %	55,1 %	52,4 %	21,2 %	10,3 %	52,4 %
Ant 1.5	17,8 %	60,0 %	56,9 %	12,9 %	13,5 %	57,5 %
Ant 1.6	20,9 %	59,4 %	57,2 %	16,0 %	9,0 %	57,4 %
Ant 1.7	17,0 %	59,8 %	58,4 %	17,6 %	13,4 %	58,6 %
Camel 1.2	12,3 %	54,3 %	52,0 %	17,8 %	14,6 %	52,0 %
Camel 1.4	14,2 %	56,1 %	54,4 %	16,6 %	12,4 %	54,6 %
Camel 1.6	16,7 %	57,0 %	55,4 %	14,0 %	10,7 %	55,4 %
JEdit 3.2.1	8,9 %	72,4 %	64,7 %	30,6 %	14,0 %	64,8 %
JEdit 4.0	8,6 %	72,9 %	66,0 %	24,2 %	9,8 %	66,0 %
JEdit 4.1	13,2 %	72,7 %	66,3 %	28,7 %	10,1 %	66,3 %
JEdit 4.2	19,5 %	69,7 %	65,6 %	15,5 %	11,0 %	65,6 %
POI 1.5	13,7 %	52,8 %	49,8 %	22,7 %	14,0 %	50,5 %
POI 2.0	19,6 %	59,6 %	56,2 %	22,3 %	13,4 %	56,2 %
POI 2.5.1	9,8 %	59,9 %	54,9 %	21,3 %	16,5 %	54,9 %
POI 3.0	10,4 %	61,3 %	58,0 %	25,1 %	13,3 %	58,0 %
Xalan 2.4	15,9 %	62,2 %	57,3 %	12,9 %	12,2 %	57,3 %
Xalan 2.5	18,3 %	66,0 %	59,9 %	17,0 %	14,6 %	60,1 %
Xalan 2.6	15,4 %	69,5 %	64,6 %	17,0 %	12,8 %	64,6 %
Xalan 2.7	12,7 %	68,3 %	62,8 %	16,8 %	10,1 %	62,8 %
Moyenne	*14,8 %*	*61,7 %*	*57,7 %*	*19,4 %*	*12,2 %*	*57,7 %*

TABLE 4.5 – Couverture indirecte (% LOC) des approches de contrôle sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	27,0 %	15,5 %	9,4 %	25,6 %	35,9 %	19,0 %
Ant 1.4	32,1 %	19,3 %	16,8 %	23,6 %	44,9 %	16,8 %
Ant 1.5	24,1 %	18,3 %	18,3 %	40,8 %	48,2 %	18,3 %
Ant 1.6	24,6 %	19,8 %	19,5 %	34,1 %	40,6 %	22,0 %
Ant 1.7	29,3 %	17,2 %	16,4 %	33,5 %	42,1 %	16,2 %
Camel 1.2	29,8 %	24,4 %	25,7 %	32,7 %	37,1 %	25,7 %
Camel 1.4	34,4 %	24,9 %	24,9 %	34,3 %	41,1 %	24,9 %
Camel 1.6	37,7 %	23,2 %	23,6 %	32,7 %	39,6 %	23,6 %
JEdit 3.2.1	36,0 %	21,2 %	25,0 %	39,7 %	60,9 %	24,9 %
JEdit 4.0	36,2 %	21,4 %	24,7 %	45,8 %	59,2 %	24,7 %
JEdit 4.1	49,3 %	19,2 %	24,7 %	42,7 %	62,4 %	24,7 %
JEdit 4.2	49,0 %	19,9 %	22,0 %	51,6 %	56,5 %	22,0 %
POI 1.5	35,6 %	35,6 %	42,7 %	42,8 %	63,9 %	42,0 %
POI 2.0	21,6 %	32,9 %	38,1 %	17,2 %	66,7 %	38,1 %
POI 2.5.1	15,8 %	32,4 %	39,7 %	19,4 %	65,8 %	39,7 %
POI 3.0	15,5 %	32,4 %	34,1 %	41,4 %	63,1 %	34,1 %
Xalan 2.4	38,2 %	25,3 %	24,9 %	32,3 %	48,4 %	24,9 %
Xalan 2.5	32,0 %	21,5 %	21,9 %	29,7 %	40,7 %	21,7 %
Xalan 2.6	26,8 %	18,0 %	17,4 %	26,9 %	29,9 %	17,4 %
Xalan 2.7	27,7 %	17,5 %	16,9 %	20,3 %	27,5 %	16,9 %
Moyenne	*31,1 %*	*23,0 %*	*24,3 %*	*33,3 %*	*48,7 %*	*24,9 %*

4.2.2 Détection des fautes

TABLE 4.6 – Détection des fautes directes et indirectes des approches de contrôle sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	36,3 %	54,8 %	48,1 %	37,8 %	47,4 %	57,0 %
Ant 1.4	29,8 %	51,1 %	38,3 %	40,4 %	48,9 %	38,3 %
Ant 1.5	31,4 %	82,9 %	82,9 %	48,6 %	60,0 %	82,9 %
Ant 1.6	47,8 %	85,9 %	82,6 %	50,5 %	51,6 %	83,7 %
Ant 1.7	49,1 %	82,2 %	81,4 %	60,4 %	60,1 %	81,4 %
Camel 1.2	49,2 %	84,7 %	83,3 %	52,7 %	53,4 %	83,3 %
Camel 1.4	63,0 %	86,6 %	86,3 %	67,2 %	63,0 %	86,3 %
Camel 1.6	59,4 %	83,2 %	84,2 %	63,2 %	61,6 %	84,2 %
JEdit 3.2.1	64,1 %	96,9 %	97,4 %	76,4 %	82,2 %	97,4 %
JEdit 4.0	74,8 %	96,9 %	95,6 %	80,5 %	75,2 %	95,6 %
JEdit 4.1	80,2 %	91,2 %	90,8 %	66,4 %	80,2 %	90,8 %
JEdit 4.2	76,4 %	97,2 %	97,2 %	77,4 %	82,1 %	97,2 %
POI 1.5	45,0 %	83,6 %	90,1 %	61,4 %	70,8 %	90,1 %
POI 2.0	41,0 %	69,2 %	84,6 %	35,9 %	64,1 %	84,6 %
POI 2.5.1	23,8 %	89,7 %	94,6 %	31,5 %	80,8 %	94,6 %
POI 3.0	32,4 %	92,0 %	92,0 %	72,2 %	76,4 %	92,0 %
Xalan 2.4	51,3 %	84,6 %	80,8 %	53,2 %	63,5 %	80,8 %
Xalan 2.5	55,0 %	67,8 %	63,4 %	52,5 %	53,8 %	63,4 %
Xalan 2.6	51,8 %	77,7 %	72,9 %	47,3 %	55,4 %	72,9 %
Xalan 2.7	41,7 %	68,8 %	65,0 %	37,6 %	41,4 %	65,1 %
Moyenne	*50,2 %*	*81,4 %*	*80,6 %*	*55,6 %*	*63,6 %*	*81,1 %*

Dans les tableaux 4.6 à 4.8, les résultats pour la détection des fautes des approches de contrôle sans variante sont présentés. Les approches gloutonnes et de classification hiérarchique détectent, en moyenne, au moins 80 % des fautes dans leur couverture directe ou indirecte. Ces trois approches couvrent en moyenne 50 % des fautes directement (c'est-à-dire que l'ensemble des classes candidates contient 50 % des fautes). Tout comme pour la couverture, ces approches misent sur la détection des fautes directe plutôt qu'indirecte. L'approche aléatoire détecte en moyenne 50 % des fautes, ce qui est attendu, car elle couvre près de 50 % des lignes de code au total (voir tableau 4.3). Encore une fois, environ 15 % des fautes sont trouvées dans la couverture directe, ce qui est conforme avec les résultats de la couverture directe des lignes de code qui est elle aussi près de 15 %. La descente de gradient et les algorithmes génétiques, pour leur part, couvrent respectivement 55 % et 64 % des fautes en moyenne. Pour la descente de gradient, le résultat est à peine meilleur que celui de l'approche aléatoire (+5 %) et meilleur de seulement 1.6 % pour la détection directe. Elles détectent, en moyenne, le même nombre de fautes directes, soit près de 15 %. Ces deux approches détectent plus de fautes grâce à la couverture indirecte des classes candidates aux tests identifiées.

À l'instar de la couverture, on remarque peu de différences entre les systèmes. Les variations observées proviennent donc majoritairement des algorithmes et de leur comportement lors de cas particuliers plutôt que des caractéristiques des systèmes.

TABLE 4.7 – Détection directe des fautes des approches de contrôle sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	12,6 %	39,3 %	39,3 %	11,9 %	14,1 %	33,3 %
Ant 1.4	19,1 %	40,4 %	27,7 %	14,9 %	17,0 %	27,7 %
Ant 1.5	5,7 %	62,9 %	62,9 %	14,3 %	14,3 %	62,9 %
Ant 1.6	17,4 %	69,0 %	68,5 %	8,2 %	7,1 %	68,5 %
Ant 1.7	16,3 %	71,0 %	69,2 %	20,7 %	16,9 %	69,2 %
Camel 1.2	16,3 %	41,4 %	43,7 %	14,8 %	12,5 %	43,7 %
Camel 1.4	10,7 %	54,6 %	57,6 %	24,2 %	13,7 %	57,6 %
Camel 1.6	10,2 %	47,8 %	57,4 %	11,2 %	12,4 %	57,4 %
JEdit 3.2.1	5,8 %	77,7 %	70,9 %	13,9 %	25,4 %	71,2 %
JEdit 4.0	11,9 %	81,0 %	77,4 %	11,9 %	6,6 %	77,4 %
JEdit 4.1	10,1 %	76,5 %	71,0 %	16,6 %	11,5 %	71,0 %
JEdit 4.2	16,0 %	84,0 %	86,8 %	9,4 %	12,3 %	86,8 %
POI 1.5	14,9 %	39,5 %	37,4 %	23,4 %	17,3 %	37,7 %
POI 2.0	7,7 %	41,0 %	48,7 %	10,3 %	20,5 %	48,7 %
POI 2.5.1	12,1 %	31,3 %	26,4 %	11,3 %	17,5 %	26,4 %
POI 3.0	15,0 %	40,8 %	41,0 %	23,8 %	16,4 %	41,0 %
Xalan 2.4	13,5 %	51,9 %	48,1 %	9,0 %	11,5 %	48,1 %
Xalan 2.5	19,3 %	31,4 %	30,5 %	19,3 %	14,4 %	30,7 %
Xalan 2.6	17,8 %	44,4 %	43,3 %	13,9 %	14,1 %	43,3 %
Xalan 2.7	12,9 %	27,0 %	27,6 %	14,5 %	14,8 %	27,7 %
Moyenne	*13,3 %*	*52,6 %*	*51,8 %*	*14,9 %*	*14,5 %*	*51,5 %*

TABLE 4.8 – Détection indirecte des fautes des approches de contrôle sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	23,7 %	15,6 %	8,9 %	25,9 %	33,3 %	23,7 %
Ant 1.4	10,6 %	10,6 %	10,6 %	25,5 %	31,9 %	10,6 %
Ant 1.5	25,7 %	20,0 %	20,0 %	34,3 %	45,7 %	20,0 %
Ant 1.6	30,4 %	16,8 %	14,1 %	42,4 %	44,6 %	15,2 %
Ant 1.7	32,8 %	11,2 %	12,1 %	39,6 %	43,2 %	12,1 %
Camel 1.2	33,0 %	43,3 %	39,7 %	37,9 %	41,0 %	39,7 %
Camel 1.4	52,2 %	31,9 %	28,7 %	43,0 %	49,3 %	28,7 %
Camel 1.6	49,2 %	35,4 %	26,8 %	52,0 %	49,2 %	26,8 %
JEdit 3.2.1	58,4 %	19,1 %	26,4 %	62,6 %	56,8 %	26,2 %
JEdit 4.0	62,8 %	15,9 %	18,1 %	68,6 %	68,6 %	18,1 %
JEdit 4.1	70,0 %	14,7 %	19,8 %	49,8 %	68,7 %	19,8 %
JEdit 4.2	60,4 %	13,2 %	10,4 %	67,9 %	69,8 %	10,4 %
POI 1.5	30,1 %	44,2 %	52,6 %	38,0 %	53,5 %	52,3 %
POI 2.0	33,3 %	28,2 %	35,9 %	25,6 %	43,6 %	35,9 %
POI 2.5.1	11,7 %	58,5 %	68,1 %	20,2 %	63,3 %	68,1 %
POI 3.0	17,4 %	51,2 %	51,0 %	48,4 %	60,0 %	51,0 %
Xalan 2.4	37,8 %	32,7 %	32,7 %	44,2 %	51,9 %	32,7 %
Xalan 2.5	35,6 %	36,4 %	33,0 %	33,1 %	39,5 %	32,8 %
Xalan 2.6	34,0 %	33,3 %	29,6 %	33,3 %	41,3 %	29,6 %
Xalan 2.7	28,9 %	41,8 %	37,4 %	23,1 %	26,7 %	37,4 %
Moyenne	*36,9 %*	*28,7 %*	*28,8 %*	*40,8 %*	*49,1 %*	*29,6 %*

4.2.3 Efficience de la détection

Dans les tableaux 4.9 à 4.12 est présentée l'efficience de diverses catégories de détection des fautes. En général, les approches trouvent moins de fautes en moyenne par millier de lignes de code que l'approche aléatoire. C'est un résultat qui s'explique par le faible nombre de lignes de code couvertes par l'approche aléatoire. Bien que cette dernière sélectionne 15 % des classes comme les autres approches, ces classes sont généralement de plus petite taille comme montré au tableau 4.4. Il faut donc prendre cette meilleure efficience apparente de l'approche aléatoire avec prudence. En effet, la présence d'un plus petit ensemble à tester (nombre de ligne de code plus faible) permet d'atteindre des valeurs d'efficience similaire aux autres approches tout en trouvant moins de fautes. À l'exception de l'algorithme génétique et de la descente de gradient, les autres approches trouvent un nombre semblable de fautes par millier de lignes de code et ont une couverture directe similaire. Les algorithmes génétiques couvrent moins de lignes de code directement, donc ceci peut expliquer le plus haut ratio.

Ces approches, soit la descente de gradient et l'algorithme génétique, ont toutefois une meilleure détection indirecte par millier de lignes de code des classes candidates (tableau 4.11). Ceci est directement lié au fait que ces deux approches détectent beaucoup de fautes de façon indirecte (tableau 4.8). L'approche aléatoire a un ratio de détection indirect supérieur à la plupart des approches (3 à 4 fautes de plus par milliers de lignes de code), mis à part l'algorithme génétique. Au niveau de la détection totale par milliers de lignes de code montrée au tableau 4.9, seul l'algorithme génétique (10,51) permet d'atteindre une meilleure efficience que l'approche aléatoire (7,47). Ceci démontre que la plupart des approches ont de la difficulté à cibler les classes pertinentes et qu'une bonne partie de l'effort de test investit ne permet pas de révéler des fautes.

Toutefois, le critère d'efficience n'est pas l'unique critère à prendre en considération. L'algorithme génétique est également l'une des approches qui détectent le moins de fautes. L'hypothèse formulée ici est que l'algorithme génétique se concentre sur des classes où la présence de fautes est plus vraisemblable, mais est trop restrictif et n'inclut pas de classes où les fautes sont moins probables. Par conséquent, l'algorithme génétique est plus efficace, car il ne marque comme candidates au test que les classes

TABLE 4.9 – Nombre de fautes détectées directement et indirectement par millier de lignes de code pour les approches sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	8,89	4,41	4,26	7,58	18,52	5,22
Ant 1.4	1,43	0,72	0,57	1,49	3,70	0,57
Ant 1.5	0,65	0,51	0,54	1,39	1,63	0,53
Ant 1.6	3,31	2,09	2,09	4,57	8,27	2,11
Ant 1.7	4,17	1,98	2,01	4,95	6,45	2,00
Camel 1.2	14,14	5,52	5,68	10,46	12,97	5,68
Camel 1.4	6,87	2,40	2,46	6,29	7,86	2,45
Camel 1.6	7,16	2,93	3,05	9,06	11,56	3,05
JEdit 3.2.1	18,70	3,49	3,93	6,53	15,27	3,92
JEdit 4.0	11,72	1,80	1,96	4,51	10,41	1,96
JEdit 4.1	7,43	1,53	1,67	2,82	9,70	1,67
JEdit 4.2	2,02	0,72	0,76	2,57	3,86	0,76
POI 1.5	19,34	9,29	10,61	15,90	29,70	10,46
POI 2.0	0,84	0,46	0,60	0,64	1,91	0,60
POI 2.5.1	9,70	5,97	6,86	5,89	19,56	6,86
POI 3.0	11,57	5,56	5,87	10,65	21,25	5,87
Xalan 2.4	2,24	0,94	0,98	2,85	3,60	0,98
Xalan 2.5	5,15	1,76	1,81	5,30	6,34	1,81
Xalan 2.6	4,97	1,65	1,67	4,11	6,43	1,67
Xalan 2.7	9,04	2,76	2,84	6,12	11,23	2,84
Moyenne	*7,47*	*2,83*	*3,01*	*5,69*	*10,51*	*3,05*

où il est plus vraisemblable d'observer des fautes, ce qui réduit l'effort de test, mais passe à côté de plusieurs classes possiblement fautives.

Les algorithmes les plus efficaces dans leur détection des fautes sont ceux qui font usage de la détection indirecte (voir tableau 4.12), comme l'approche aléatoire (5,34), la descente de gradient (4,02) et l'algorithme génétique (8,04). Leur ratio d'efficience directe comparé à celui de l'efficience indirecte est inférieur à 1. Les approches les plus performantes au niveau de la détection des fautes sont celles qui misent sur la détection directe (ratio supérieur à 1). Ces approches détectent en moyenne environ 2.5 fois plus de fautes directement qu'indirectement.

La conclusion ici est que peu d'approches semblent exploiter la détection indirecte, autrement dit, considérer l'information des classes dont une classe dépend dans la priorisation. Cette conclusion est consolidée par le fait que les approches misant sur la détection directe sont celles avec la meilleure détection des fautes totales. La conséquence observée est une perte dans l'efficience du test où le nombre de fautes trouvées par lignes de code testées est plus faible. Il n'est pas surprenant que les ap-

TABLE 4.10 – Nombre de fautes détectées directement par millier de lignes de code pour les approches sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	3,08	3,16	3,47	2,38	5,50	3,05
Ant 1.4	0,92	0,57	0,41	0,55	1,29	0,41
Ant 1.5	0,12	0,39	0,41	0,41	0,39	0,40
Ant 1.6	1,2	1,68	1,73	0,74	1,13	1,73
Ant 1.7	1,38	1,71	1,71	1,70	1,81	1,70
Camel 1.2	4,68	2,7	2,98	2,93	3,02	2,98
Camel 1.4	1,17	1,51	1,64	2,26	1,71	1,64
Camel 1.6	1,23	1,68	2,08	1,61	2,33	2,08
JEdit 3.2.1	1,68	2,80	2,86	1,19	4,72	2,87
JEdit 4.0	1,87	1,50	1,59	0,67	0,92	1,59
JEdit 4.1	0,94	1,28	1,31	0,71	1,39	1,31
JEdit 4.2	0,42	0,62	0,68	0,31	0,58	0,68
POI 1.5	6,41	4,39	4,41	6,06	7,24	4,38
POI 2.0	0,16	0,28	0,35	0,18	0,61	0,35
POI 2.5.1	4,93	2,08	1,92	2,12	4,24	1,92
POI 3.0	5,36	2,47	2,62	3,51	4,56	2,62
Xalan 2.4	0,59	0,58	0,58	0,48	0,66	0,58
Xalan 2.5	1,81	0,82	0,87	1,95	1,69	0,87
Xalan 2.6	1,71	0,94	0,99	1,21	1,63	0,99
Xalan 2.7	2,79	1,08	1,21	2,36	4,00	1,21
Moyenne	*2,12*	*1,61*	*1,69*	*1,67*	*2,47*	*1,67*

proches ne tiennent pas compte du caractère indirect, car elles n'ont pas été conçues pour le faire.

TABLE 4.11 – Nombre de fautes détectées indirectement par millier de lignes de code pour les approches sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	5,81	1,25	0,79	5,2	13,02	2,17
Ant 1.4	0,51	0,15	0,16	0,94	2,41	0,16
Ant 1.5	0,53	0,12	0,13	0,98	1,24	0,13
Ant 1.6	2,11	0,41	0,36	3,83	7,13	0,38
Ant 1.7	2,79	0,27	0,30	3,25	4,64	0,30
Camel 1.2	9,46	2,82	2,70	7,53	9,95	2,70
Camel 1.4	5,69	0,88	0,82	4,03	6,15	0,81
Camel 1.6	5,93	1,25	0,97	7,46	9,23	0,97
JEdit 3.2.1	17,02	0,69	1,07	5,34	10,55	1,05
JEdit 4.0	9,84	0,30	0,37	3,84	9,49	0,37
JEdit 4.1	6,49	0,25	0,36	2,12	8,31	0,36
JEdit 4.2	1,59	0,10	0,08	2,26	3,28	0,08
POI 1.5	12,94	4,91	6,20	9,84	22,46	6,08
POI 2.0	0,68	0,19	0,26	0,46	1,30	0,26
POI 2.5.1	4,77	3,89	4,94	3,78	15,31	4,94
POI 3.0	6,21	3,10	3,26	7,14	16,69	3,26
Xalan 2.4	1,65	0,36	0,39	2,37	2,95	0,39
Xalan 2.5	3,34	0,95	0,94	3,34	4,65	0,93
Xalan 2.6	3,26	0,71	0,68	2,9	4,79	0,68
Xalan 2.7	6,25	1,68	1,63	3,76	7,22	1,63
Moyenne	*5,34*	*1,21*	*1,32*	*4,02*	*8,04*	*1,38*

TABLE 4.12 – Ratio de l'efficience directe par rapport à l'efficience indirecte pour les approches sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	0,53	2,52	4,42	0,46	0,42	1,41
Ant 1.4	1,80	3,8	2,6	0,58	0,53	2,60
Ant 1.5	0,22	3,14	3,14	0,42	0,31	3,14
Ant 1.6	0,57	4,10	4,85	0,19	0,16	4,50
Ant 1.7	0,50	6,32	5,71	0,52	0,39	5,71
Camel 1.2	0,49	0,96	1,1	0,39	0,3	1,1
Camel 1.4	0,21	1,71	2,01	0,56	0,28	2,01
Camel 1.6	0,21	1,35	2,14	0,22	0,25	2,14
JEdit 3.2.1	0,10	4,07	2,68	0,22	0,45	2,72
JEdit 4.0	0,19	5,08	4,27	0,17	0,10	4,27
JEdit 4.1	0,14	5,19	3,58	0,33	0,17	3,58
JEdit 4.2	0,27	6,36	8,36	0,14	0,18	8,36
POI 1.5	0,50	0,89	0,71	0,62	0,32	0,72
POI 2.0	0,23	1,45	1,36	0,4	0,47	1,36
POI 2.5.1	1,03	0,53	0,39	0,56	0,28	0,39
POI 3.0	0,86	0,80	0,80	0,49	0,27	0,8
Xalan 2.4	0,36	1,59	1,47	0,2	0,22	1,47
Xalan 2.5	0,54	0,86	0,92	0,58	0,36	0,94
Xalan 2.6	0,52	1,33	1,46	0,42	0,34	1,46
Xalan 2.7	0,45	0,64	0,74	0,63	0,55	0,74
Moyenne	*0,49*	*2,64*	*2,64*	*0,41*	*0,32*	*2,47*

4.2.4 Qualité de l'ordonnancement

Les résultats pour la métrique APFD sont présentés au tableau 4.13. L'approche aléatoire obtient un APFD d'environ 50 %, ce qui revient à trier aléatoirement les fautes (voir annexe A). Les approches de descente de gradient et d'algorithme génétique n'apportent pas d'amélioration par rapport à l'ordonnancement aléatoire. Les autres approches ont une APFD légèrement supérieure à 60 %. Les valeurs sont plus basses que ce qui peut être observé ailleurs dans la littérature en raison de notre considération des classes candidates au test plutôt que des cas de test et de la limitation à la détection directe. Nous observons aussi une dispersion modérée avec des résultats variant de 30 % à 75 %. Nous la qualifions de modérée, car nous n'atteignons par des valeurs extrêmes près de 0 % ou 100 %. Les différences ne peuvent pas non plus être expliquées par les caractéristiques des systèmes, car JEdit par exemple affiche parmi les pires résultats pour la descente de gradient et les meilleurs pour les algorithmes gloutons ou la classification hiérarchique. Les différences observées sont donc liées aux algorithmes utilisés.

Nous avons testé l'influence de la taille de l'ensemble des classes candidates et du nombre de fautes directement détectées sur l'APFD en utilisant l'équation de régression linéaire suivante pour chaque approche :

$$\begin{aligned} APFD = b_0 + b_1 \times \text{ taille ensemble candidat } + \\ b_2 \times \text{ nombre de fautes détectées directement} \end{aligned} \tag{4.1}$$

Dans tous les cas, la régression s'est avérée non significative globalement et pour chaque variable. Par conséquent, il n'y a aucune influence significative des différentes tailles d'ensembles de candidates ou du nombre de fautes détectées directement sur l'APFD.

L'annexe B présente les résultats sous forme de graphique (figure B.1) les métriques d'évaluation liées aux cinq axes des questions de recherche.

TABLE 4.13 – APFD pour les approches sans variante

Système	ALEA-couv	GL-couv	GL-comp	DG-couv	CBGA-ES	CH
Ant 1.3	50,6 %	55,6 %	46,8 %	35,9 %	53,2 %	55.6 %
Ant 1.4	44,2 %	31,8 %	39,9 %	46,9 %	45,3 %	39,9 %
Ant 1.5	35,5 %	60,6 %	53,2 %	47,8 %	52,2 %	54,4 %
Ant 1.6	57,6 %	60,2 %	58,3 %	63,4 %	51,0 %	57,9 %
Ant 1.7	39,1 %	65,4 %	65,0 %	53,1 %	42,0 %	65,2 %
Camel 1.2	55,3 %	71,0 %	67,6 %	50,7 %	54,1 %	67,7 %
Camel 1.4	55,3 %	63,4 %	63,6 %	48,7 %	45,9 %	63,8 %
Camel 1.6	50,9 %	65,0 %	67,6 %	51,8 %	59,9 %	67,7 %
JEdit 3.2.1	71,2 %	69,2 %	66,5 %	44,5 %	50,8 %	66,8 %
JEdit 4.0	50,7%	72,1%	71,9%	45,8%	53,4%	71,9%
JEdit 4.1	34,2 %	72,0 %	71,9 %	38,9 %	39,5 %	71,9 %
JEdit 4.2	36,8 %	72,2 %	72,7 %	34,1 %	48,3 %	72,7 %
POI 1.5	49,7%	58,6%	62,6%	49,4%	43,2%	62,7%
POI 2.0	67,0 %	66,3 %	61,1 %	59,6 %	66,4 %	61,0 %
POI 2.5.1	48,9 %	58,9 %	64,0 %	54,0 %	46,2 %	64,0 %
POI 3.0	52,8 %	64,9 %	64,9 %	44,9 %	53,7 %	64,7 %
Xalan 2.4	41,9 %	61,5 %	59,9 %	59,2 %	57,3 %	59,8 %
Xalan 2.5	44,3 %	56,7 %	58,7 %	49,8 %	49,5 %	58,6 %
Xalan 2.6	47,4 %	57,4 %	56,4 %	50,8 %	55,8 %	56,4 %
Xalan 2.7	49,0 %	54,2 %	53,0 %	53,2 %	49,8 %	52,9 %
Moyenne	*49,1 %*	*61,9 %*	*61,3 %*	*49,1 %*	*50,9 %*	*61,8 %*

4.3 Prédiction intra-système

Dans la prédiction intra-système, les jeux de données utilisés pour l'entraînement sont limités aux données provenant des différentes versions du même système. Cette méthodologie permet d'assumer une certaine homogénéité entre les jeux de données, comme ils proviennent tous du même système. Certaines ressemblances existent entre les versions, notamment dans la structure du programme, car les versions subséquentes d'un système reposent sur les versions antérieures. De plus, certains facteurs de variance entre les systèmes, comme les changements dans l'équipe de programmation et dans les méthodes de conception utilisées sont amoindris dans la prédiction intra-système, puisque l'équipe de développement est la même (ou suit, en cas de changement, les mêmes approches).

4.3.1 Approches de priorisation

Les prochaines sections présentent les résultats pour chaque catégorie de métriques d'évaluation pour les variantes de l'approche SL dans le contexte intra-système soient intra-SL (KNN), intra-SL (RF) et intra-SL (KNN+RF).

Couverture

Pour la couverture (tableaux 4.14 à 4.16), intra-SL (KNN) et intra-SL (RF) proposent une couverture assez similaire en moyenne (81,4 % et 81,6 %). Cependant, on peut observer des différences plus importantes lorsque l'on s'intéresse à un système en particulier. L'approche intra-SL (KNN) possède une couverture directe légèrement plus faible, soit de 1 % de moins, que celle de l'approche intra-SL (RF), mais une couverture indirecte un peu plus importante. L'approche intra-SL (KNN + RF) offre une couverture légèrement supérieure tant au niveau de la couverture directe qu'indirecte et atteint 84,7 % en couverture totale, soit 3 % de plus que les approches intra-SL impliquant un seul algorithme de détection. Les approches de contrôle issues des modèles de prédictions (Intra-GL-risque) offrent, pour leur part, une couverture totale nettement inférieure, de plus de 15 %, à leur approche développée utilisant le ou les

mêmes algorithmes d'apprentissage automatique. En revanche, la couverture directe des approches de contrôle est plus importante, donc les approches intra-SL développées ont tendance à produire des ensembles de classes candidates au test plus petits. La couverture indirecte des approches de contrôle intra-GL-risque est nettement inférieure (par plus de 13 %) à celle des approches intra-SL. Ces différences illustrent que l'approche SL utilise bien la couverture indirecte pour atteindre une couverture totale intéressante.

TABLE 4.14 – Couverture totale (% LOC) des approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	57,5 %	60,9 %	63,9 %	85,4 %	87,5 %	87,8 %
Ant 1.4	69,1 %	67,9 %	67,9 %	85,6 %	87,5 %	88,0 %
Ant 1.5	66,7 %	67,2 %	67,2 %	77,5 %	83,5 %	84,6 %
Ant 1.6	72,8 %	72,4 %	72,9 %	81,0 %	82,1 %	79,5 %
Ant 1.7	70,6 %	70,0 %	69,7 %	73,7 %	75,8 %	73,2 %
Camel 1.2	48,6 %	49,8 %	52,2 %	80,6 %	85,5 %	85,5 %
Camel 1.4	48,2%	46,5%	49,6%	75,4%	83,0%	78,5%
Camel 1.6	53,0%	48,8%	50,6%	73,9%	85,6%	81,7%
JEdit 3.2.1	89,6%	82,7%	92,0%	82,4%	84,6%	86,8%
JEdit 4.0	89,6%	83,4%	67,9%	83,1%	65,2%	83,6%
JEdit 4.1	89,2 %	67,7 %	68,0 %	82,3 %	68,0 %	81,6 %
JEdit 4.2	69,9 %	70,1 %	70,7 %	78,5 %	75,9 %	86,3 %
POI 1.5	78,2%	67,4%	80,9%	95,5%	95,9%	96,6%
POI 2.0	82,8%	68,7%	82,1%	94,7%	96,1%	95,8%
POI 2.5.1	81,3%	67,4%	78,4%	93,9%	94,4%	94,9%
POI 3.0	75,1 %	70,8 %	75,0 %	91,7 %	92,1 %	92,7 %
Xalan 2.4	69,9 %	76,1 %	75,8 %	97,8 %	95,6 %	95,3 %
Xalan 2.5	59,1 %	62,6 %	61,0 %	82,3 %	75,3 %	77,4 %
Xalan 2.6	50,3 %	50,6 %	52,0 %	59,5 %	59,1 %	87,7 %
Xalan 2.7	52,3 %	49,5 %	52,3 %	56,6 %	56,2 %	56,6 %
Moyenne	*68,7 %*	*65,0 %*	*67,5 %*	*81,6 %*	*81,4 %*	*84,7 %*

Le tableau 4.17 présente les différentes tailles des ensembles de classes candidates au test par système. Toutes les approches de contrôle utilisent la taille de référence de 15 %. On peut observer qu'en moyenne, les approches intra-SL sélectionnent un nombre de classes similaire à celui des approches de contrôle présentées dans la colonne « Référence 15 % » utilisée par les trois approches intra-GL-risque.

Le nombre de classes sélectionnées varie cependant de façon très différente selon les différents systèmes. Nous avons calculé la différence entre le nombre de classes sélectionnées par les approches intra-SL et par la référence de 15 % utilisée par les approches de contrôle. Cette différence est appelée Δ_t. Nous avons calculé la corrélation entre les Δ_t et la taille des systèmes en nombre de classes (voir tableau 4.18) afin

TABLE 4.15 – Couverture directe (% LOC) des approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	27,4 %	30,5 %	32,4 %	42,4 %	44,8 %	44,5 %
Ant 1.4	48,0 %	48,0 %	47,6 %	40,1 %	42,6 %	42,9 %
Ant 1.5	45,8 %	46,2 %	46,9 %	33,2 %	34,8 %	33,8 %
Ant 1.6	44,5 %	46,8 %	46,0 %	32,2 %	31,5 %	33,3 %
Ant 1.7	48,6 %	50,2 %	49,7 %	23,9 %	24,8 %	23,4 %
Camel 1.2	27,3 %	29,0 %	30,0 %	36,1 %	37,6 %	38,3 %
Camel 1.4	29,6%	28,3%	30,4%	29,7%	32,1%	30,2%
Camel 1.6	31,7%	30,0%	33,4%	25,7%	33,8%	30,1%
JEdit 3.2.1	53,4%	40,9%	56,6%	28,4%	38,4%	40,6%
JEdit 4.0	56,4%	42,7%	42,8%	28,3%	25,1%	27,9%
JEdit 4.1	55,7 %	43,8 %	43,4 %	26,1 %	27,0 %	27,3 %
JEdit 4.2	43,5 %	44,8 %	44,6 %	29,0 %	31,7 %	35,6 %
POI 1.5	31,3%	30,6%	37,4%	24,3%	24,7%	25,3%
POI 2.0	38,4%	38,2%	40,0%	30,6%	30,1%	29,8%
POI 2.5.1	35,1%	33,9%	35,0%	34,8%	36,4%	35,2%
POI 3.0	36,0 %	37,0 %	37,6 %	32,9 %	32,6 %	33,6 %
Xalan 2.4	34,4 %	43,6 %	43,3 %	45,0 %	43,8 %	43,3 %
Xalan 2.5	35,0 %	38,7 %	36,6 %	40,0 %	33,7 %	35,1 %
Xalan 2.6	31,6 %	31,7 %	33,7 %	28,3 %	25,9 %	57,5 %
Xalan 2.7	33,7 %	33,4 %	34,8 %	24,5 %	24,0 %	23,6 %
Moyenne	*39,4 %*	*38,4 %*	*40,1 %*	*31,8 %*	*32,8 %*	*34,6 %*

TABLE 4.16 – Couverture indirect (% LOC) des approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	30,1 %	30,4 %	31,5 %	42,9 %	42,6 %	43,3 %
Ant 1.4	21,1 %	19,8 %	20,4 %	45,5 %	44,9 %	45,1 %
Ant 1.5	20,9 %	21,0 %	20,3 %	44,3 %	48,6 %	50,8 %
Ant 1.6	28,3 %	25,6 %	26,9 %	48,8 %	50,6 %	46,2 %
Ant 1.7	22,0 %	19,8 %	20,0 %	49,7 %	51,0 %	49,8 %
Camel 1.2	21,3 %	20,8 %	22,1 %	44,6 %	47,9 %	47,1 %
Camel 1.4	18,6%	18,2%	19,2%	45,7%	50,9%	48,2%
Camel 1.6	21,3%	18,8%	17,2%	48,2%	51,9%	51,6%
JEdit 3.2.1	36,2%	41,9%	35,4%	54,0%	46,2%	46,2%
JEdit 4.0	33,2%	40,7%	25,1%	54,8%	40,2%	55,7%
JEdit 4.1	33,5 %	23,9 %	24,7 %	56,2 %	40,9 %	54,3 %
JEdit 4.2	26,4 %	25,3 %	26,1 %	49,5 %	44,2 %	50,7 %
POI 1.5	46,8 %	36,8 %	43,5 %	71,2 %	71,3 %	71,2 %
POI 2.0	44,4 %	30,4 %	42,1 %	64,1 %	66,0 %	66,0 %
POI 2.5.1	46,2 %	33,5 %	43,4 %	59,1 %	58,0 %	59,7 %
POI 3.0	39,1 %	33,8 %	37,4 %	58,8 %	59,5 %	59,1 %
Xalan 2.4	35,6 %	32,5 %	32,5 %	52,7 %	51,8 %	52,0 %
Xalan 2.5	24,0 %	23,9 %	24,5 %	42,3 %	41,6 %	42,3 %
Xalan 2.6	18,7 %	18,9 %	18,2 %	31,2 %	33,2 %	30,3 %
Xalan 2.7	18,7 %	16,1 %	17,5 %	32,1 %	32,2 %	33,0 %
Moyenne	*29,3 %*	*26,6 %*	*27,4 %*	*49,8 %*	*48,7 %*	*50,1 %*

TABLE 4.17 – Taille des ensembles de classes candidates au test pour les approches intra-système

Système	Référence 15 %	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	19	36	38	38
Ant 1.4	40	52	55	55
Ant 1.5	60	66	71	67
Ant 1.6	78	79	69	73
Ant 1.7	159	113	111	107
Camel 1.2	115	153	156	155
Camel 1.4	168	176	199	168
Camel 1.6	188	178	206	179
JEdit 3.2.1	76	56	37	46
JEdit 4.0	91	61	48	52
JEdit 4.1	97	66	60	61
JEdit 4.2	121	99	100	96
POI 1.5	44	41	47	45
POI 2.0	57	44	45	41
POI 2.5.1	69	58	58	56
POI 3.0	80	52	52	53
Xalan 2.4	122	271	247	250
Xalan 2.5	135	197	187	188
Xalan 2.6	174	252	232	273
Xalan 2.7	179	150	137	147
Moyenne	*104*	*110*	*108*	*108*

de vérifier si la taille originale des systèmes influence la taille de l'ensemble candidat sélectionné par intra-SL.

$$\Delta_T = |C_{SL}| - 0.15n \tag{4.2}$$

où n est le nombre de classes dans un système et $|C_{SL}|$ est le nombre de classes candidates aux tests retournées par l'approche SL.

Pour tous les systèmes sauf JEdit, une forte corrélation négative est observée. Autrement dit, les différences entre les tailles des ensembles de classes candidates deviennent de plus en plus importantes au fur et à mesure que la taille augmente. Les approches intra-SL (KNN), intra-SL (RF) et intra-SL (KNN+RF) tendent à sélectionner plus de classes sur de plus petits systèmes et moins sur les systèmes de plus grande taille que le seuil de référence. Notons, cependant, une inversion de tendance pour JEdit. L'approche intra-SL (KNN) ne manifeste pas de relation avec la taille et dans les autres cas la tendance est inversée en raison du coefficient de corrélation positif. Nuançons toutefois l'ensemble des résultats du tableau 4.18 en soulignant la faible taille des échantillons (5 pour Ant, 3 pour Camel et 4 pour le autres systèmes). Ces faibles tailles nous empêchent également d'utiliser un test statistique pour établir la significativité des corrélations.

TABLE 4.18 – Corrélations entre la différence de taille de l'ensemble de classes candidates et la taille du système (nombre de classes) par système pour les approches intra-système

Système	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant	-0,98	-0,98	-0,99
Camel	-0,99	-0,94	-1
JEdit	-0,01	0,87	0,48
POI	-0,89	-0,93	-0,91
Xalan	-0,76	-0,79	-0,6

Détection des fautes

Les approches (variantes) SL proposées (KNN, RF et KNN + RF) détectent en moyenne plus de 90 % des fautes dans les systèmes, de façon directe ou indirecte (tableau 4.19). Les approches gloutonnes intra-GL-risque, pour leur part, détectent en moyenne entre 75 et 80 % des fautes. L'approche de forêt aléatoire seule offre des résultats inférieurs pour les approches gloutonnes, mais pas pour SL.

TABLE 4.19 – Détection des fautes directes et indirectes dans les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	58,5 %	58,5 %	63,0 %	91,1 %	91,9 %	91,9 %
Ant 1.4	51,1 %	48,9 %	46,8 %	89,4 %	85,1 %	89,4 %
Ant 1.5	85,7 %	85,7 %	85,7 %	94,3 %	97,1 %	94,3 %
Ant 1.6	85,3 %	87,0 %	85,9 %	96,2 %	94,6 %	95,1 %
Ant 1.7	77,5 %	78,7 %	81,1 %	86,1 %	87,9 %	86,1 %
Camel 1.2	69,2 %	66,3 %	70,3 %	89,8 %	95,8 %	94,6 %
Camel 1.4	86,9%	83,3%	88,1%	94,6%	95,8%	95,5%
Camel 1.6	89,6%	86,8%	88,8%	91,6%	96,4%	95,2%
JEdit 3.2.1	98,2%	96,1%	97,9%	96,6%	94,2%	96,6%
JEdit 4.0	97,3%	96,9%	97,3%	99,6%	97,3%	97,8%
JEdit 4.1	94,0 %	91,7 %	93,5 %	96,8 %	93,5 %	97,7 %
JEdit 4.2	99,1 %	98,1 %	99,1 %	98,1 %	98,1 %	98,1 %
POI 1.5	72,5 %	57,3 %	72,8 %	93,9 %	93,6 %	96,5 %
POI 2.0	61,5 %	48,7 %	61,5 %	89,7 %	94,9 %	89,7 %
POI 2.5.1	92,5 %	75,4 %	90,7 %	97,0 %	96,4 %	96,8 %
POI 3.0	86,6 %	82,6 %	86,4 %	95,8 %	95,6 %	96,2 %
Xalan 2.4	78,2 %	82,1 %	83,3 %	100,0 %	100,0 %	100,0 %
Xalan 2.5	69,0 %	69,2 %	69,3 %	84,7 %	83,9 %	84,7 %
Xalan 2.6	65,7 %	66,2 %	67,9 %	79,8 %	76,8 %	86,7 %
Xalan 2.7	61,1 %	59,9 %	61,2 %	63,9 %	64,4 %	65,3 %
Moyenne	*79,0 %*	*76,0 %*	*79,5 %*	*91,4 %*	*91,7 %*	*92,4 %*

Au niveau de la répartition entre la détection directe et indirecte (tableau 4.20 et tableau 4.21), les approches intra-SL détectent près de 40 % des fautes directement et 50 % indirectement, ce qui indique une faible propension à miser sur la couverture indirecte. Les approches gloutonnes détectent les fautes principalement de façon directe. En effet, environ 50 % des fautes sont détectées directement par les approches intra-GL-risque et seulement un peu plus de 25 % des fautes sont détectées indirectement.

Un élément à remarquer est également la constance avec laquelle les fautes sont détectées par l'approche intra-SL. À l'exception du système Xalan, l'approche intra-SL détecte au moins 90 % des fautes dans presque tous les cas. Pour les approches de contrôle, y compris celles présentées à la section 4.2, les valeurs pouvaient grandement varier entre les systèmes (Ant 1.5 à 71.4 % et Camel 1.2 à 35,2 % pour Intra-GL-risque (KNN)) et même entre les versions d'un système (POI 2.0 à 15,4 % et POI 3.0 à 40,0 % pour Intra-GL-risque (RF)). L'approche intra-SL produit des résultats généralement plus constants. Une possible explication de cette constance provient de la bonne capacité des modèles d'apprentissage à prédire les fautes en intra-système [Ouellet et Badri, 2021]. La stabilité est plus remarquable avec SL, car elle considère aussi les caractéristiques des classes avoisinantes, ce qui, même si SL n'identifie pas directement une classe fautive avec une haute probabilité d'erreur, lui permet de la considérer dans la sélection des classes candidates aux tests.

TABLE 4.20 – Détection des fautes directes dans les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	31,9 %	38,5 %	38,5 %	54,8 %	54,8 %	54,8 %
Ant 1.4	38,3 %	40,4 %	38,3 %	53,2 %	51,1 %	55,3 %
Ant 1.5	71,4 %	68,6 %	68,6 %	62,9 %	65,7 %	62,9 %
Ant 1.6	63,6 %	70,1 %	67,9 %	36,4 %	35,3 %	40,2 %
Ant 1.7	61,2 %	64,2 %	65,7 %	35,8 %	39,1 %	38,2 %
Camel 1.2	35,2 %	37,2 %	38,7 %	35,1 %	40,0 %	39,1 %
Camel 1.4	54,3%	56,7%	61,2%	35,2%	38,8%	37,3%
Camel 1.6	52,4%	56,6%	62,8%	26,6%	34,4%	31,2%
JEdit 3.2.1	78,8%	76,4%	78,8%	48,4%	46,1%	45,0%
JEdit 4.0	83,6%	85,0%	85,8%	62,8%	55,8%	60,2%
JEdit 4.1	82,9 %	80,2 %	82,0 %	45,2 %	47,5 %	47,9 %
JEdit 4.2	88,7 %	84,9 %	90,6 %	50,9 %	55,7 %	55,7 %
POI 1.5	36,8%	33,3%	41,2%	27,8%	26,9%	28,1%
POI 2.0	20,5%	15,4%	20,5%	33,3%	38,5%	35,9%
POI 2.5.1	31,3 %	33,1 %	31,9 %	17,7 %	18,3 %	17,7 %
POI 3.0	40,2 %	40,0 %	42,4 %	25,6 %	25,2 %	26,0 %
Xalan 2.4	34,6 %	50,0 %	44,9 %	45,5 %	47,4 %	48,1 %
Xalan 2.5	33,5 %	36,6 %	34,5 %	34,3 %	34,3 %	33,7 %
Xalan 2.6	37,2 %	39,6 %	39,6 %	32,9 %	27,7 %	42,0 %
Xalan 2.7	23,0 %	24,9 %	24,3 %	17,3 %	17,0 %	17,5 %
Moyenne	*50,0 %*	*51,6 %*	*52,9 %*	*39,1 %*	*40,0 %*	*40,8 %*

TABLE 4.21 – Détection des fautes indirectes dans les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	26,7 %	20,0 %	24,4 %	36,3 %	37,0 %	37,0 %
Ant 1.4	12,8 %	8,5 %	8,5 %	36,2 %	34,0 %	34,0 %
Ant 1.5	14,3 %	17,1 %	17,1 %	31,4 %	31,4 %	31,4 %
Ant 1.6	21,7 %	16,8 %	17,9 %	59,8 %	59,2 %	54,9 %
Ant 1.7	16,3 %	14,5 %	15,4 %	50,3 %	48,8 %	47,9 %
Camel 1.2	33,9 %	29,1 %	31,6 %	54,8 %	55,7 %	55,6 %
Camel 1.4	32,5%	26,6%	26,9%	59,4%	57,0%	58,2%
Camel 1.6	37,2%	30,2%	26,0%	65,0%	62,0%	64,0%
JEdit 3.2.1	19,4%	19,6%	19,1%	48,2%	48,2%	51,6%
JEdit 4.0	13,7%	11,9%	11,5%	36,7%	41,6%	37,6%
JEdit 4.1	11,1 %	11,5 %	11,5 %	51,6 %	46,1 %	49,8 %
JEdit 4.2	10,4 %	13,2 %	8,5 %	47,2 %	42,5 %	42,5 %
POI 1.5	35,7%	24,0%	31,6%	66,1%	66,7%	68,4%
POI 2.0	41,0 %	33,3 %	41,0 %	56,4 %	56,4 %	53,8 %
POI 2.5.1	61,3 %	42,3 %	58,9 %	79,2 %	78,0 %	79,0 %
POI 3.0	46,4 %	42,6 %	44,0 %	70,2 %	70,4 %	70,2 %
Xalan 2.4	43,6 %	32,1 %	38,5 %	54,5 %	52,6 %	51,9 %
Xalan 2.5	35,4 %	32,6 %	34,9 %	50,4 %	49,6 %	51,0 %
Xalan 2.6	28,5 %	26,6 %	28,4 %	47,0 %	49,0 %	44,7 %
Xalan 2.7	38,1 %	35,0 %	37,0 %	46,5 %	47,4 %	47,9 %
Moyenne	*29,0 %*	*24,4 %*	*26,6 %*	*52,4 %*	*51,7 %*	*51,6 %*

Efficience de la détection

Par rapport à l'efficience de la détection totale des fautes (tableau 4.22), les approches intra-SL trouvent environ 6 fautes par milliers de lignes de code dans l'ensemble de candidates au test. Ces valeurs sont supérieures à celles observées pour les approches gloutonnes. Notons toutefois une grande variabilité dans les résultats où, pour certains systèmes, une très grande efficience est atteinte (POI 1.5 qui détecte 22 fautes par millier de lignes de code). Ces systèmes contiennent généralement plus de fautes. Néanmoins, les différences sont bien présentes entre les versions d'un même système. L'efficience directe (tableau 4.23) est très similaire entre les algorithmes gloutons et les approches intra-SL. Les approches intra-SL se distinguent par leur plus grande efficience dans la détection indirecte des fautes (tableau 4.24) détectant près de 2 fautes indirectement par millier de lignes de code. Cette différence est directement liée au fait que les approches SL détectent plus de fautes, en particulier indirectement (tableau 4.21)

TABLE 4.22 – Nombre de fautes détectées directement et indirectement par millier de lignes de code pour les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	7,60	6,83	6,92	7,64	7,29	7,34
Ant 1.4	0,83	0,80	0,77	1,74	1,56	1,63
Ant 1.5	0,69	0,68	0,67	1,05	1,03	1,03
Ant 1.6	2,77	2,69	2,70	4,32	4,34	4,13
Ant 1.7	2,30	2,26	2,35	5,19	5,12	5,31
Camel 1.2	8,97	8,08	8,29	8,82	9,02	8,74
Camel 1.4	4,56	4,57	4,50	4,94	4,64	4,91
Camel 1.6	5,68	5,81	5,33	7,14	5,72	6,34
JEdit 3.2.1	4,80	6,14	4,52	8,86	6,40	6,22
JEdit 4.0	2,34	3,07	3,08	4,77	5,26	4,75
JEdit 4.1	2,06	2,55	2,63	4,52	4,22	4,36
JEdit 4.2	1,17	1,13	1,15	1,75	1,60	1,42
POI 1.5	13,59	11,01	11,42	22,67	22,28	22,37
POI 2.0	0,64	0,51	0,62	1,17	1,26	1,21
POI 2.5.1	10,51	8,85	10,33	11,11	10,55	10,95
POI 3.0	8,90	8,27	8,51	10,77	10,85	10,61
Xalan 2.4	1,58	1,30	1,33	1,54	1,58	1,60
Xalan 2.5	3,37	3,06	3,25	3,63	4,27	4,13
Xalan 2.6	3,08	3,09	2,98	4,17	4,38	2,23
Xalan 2.7	4,98	4,91	4,82	7,15	7,35	7,59
Moyenne	*4,52*	*4,28*	*4,31*	*6,15*	*5,94*	*5,84*

Au niveau du ratio entre l'efficience directe et indirecte (tableau 4.25), les ratios plus petits que 1 indiquent que les approches intra-SL ont une efficience indirecte

TABLE 4.23 – Nombre de fautes détectées directement par millier de lignes de code pour les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	4,14	4,50	4,23	4,60	4,35	4,38
Ant 1.4	0,62	0,66	0,63	1,04	0,94	1,01
Ant 1.5	0,57	0,55	0,54	0,70	0,69	0,69
Ant 1.6	2,07	2,17	2,13	1,63	1,62	1,75
Ant 1.7	1,82	1,85	1,91	2,16	2,28	2,36
Camel 1.2	4,57	4,53	4,56	3,44	3,77	3,61
Camel 1.4	2,85	3,11	3,13	1,84	1.88	1,92
Camel 1.6	3,32	3,79	3,77	2,07	2.04	2,08
JEdit 3.2.1	3,85	4,88	3,63	4,44	3,13	2,90
JEdit 4.0	2,01	2,69	2,71	3,01	3,01	2,92
JEdit 4.1	1,82	2,23	2,31	2,11	2,14	2,14
JEdit 4.2	1,05	0,98	1,05	0,91	0,91	0,81
POI 1.5	6,90	6,40	6,47	6,71	6,40	6,51
POI 2.0	0,21	0,16	0,21	0,44	0,51	0,48
POI 2.5.1	3,55	3,88	3,63	2,03	2,01	2,01
POI 3.0	4,13	4,00	4,18	2,88	2,86	2,87
Xalan 2.4	0,70	0,79	0,72	0,70	0,75	0,77
Xalan 2.5	1,64	1,62	1,62	1,47	1,74	1,64
Xalan 2.6	1,74	1,85	1,74	1,72	1,58	1,08
Xalan 2.7	1,87	2,04	1,91	1,94	1,94	2,03
Moyenne	*2,47*	*2,63*	*2,55*	*2,29*	*2,23*	*2,20*

plus forte que directe, contrairement aux approches gloutonnes. Une raison pouvant expliquer cette différence est la taille des ensembles de classes candidates aux tests. Dans les systèmes d'envergure, les ensembles sont plus petits, laissant une plus grande proportion de classes non directement couvertes. Si, de façon relative, la taille de l'ensemble candidat est grande, cela laisse moins de place à la couverture indirecte, car si une classe est couverte directement et indirectement, elle n'est comptée que dans la couverture directe.

TABLE 4.24 – Nombre de fautes détectées indirectement par millier de lignes de code pour les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	3,46	2,33	2,69	3,04	2,94	2,96
Ant 1.4	0,21	0,14	0,14	0,71	0,62	0,62
Ant 1.5	0,11	0,14	0,13	0,35	0,33	0,34
Ant 1.6	0,71	0,52	0,56	2,68	2,72	2,38
Ant 1.7	0,48	0,42	0,45	3,03	2,84	2,96
Camel 1.2	4,40	3,55	3,73	5,38	5,25	5,13
Camel 1.4	1,71	1,46	1,37	3,10	2,76	2,99
Camel 1.6	2,36	2,02	1,56	5,07	3,68	4,26
JEdit 3.2.1	0,95	1,25	0,88	4,42	3,27	3,32
JEdit 4.0	0,33	0,38	0,36	1,76	2,25	1,83
JEdit 4.1	0,24	0,32	0,32	2,41	2,08	2,22
JEdit 4.2	0,12	0,15	0,10	0,84	0,69	0,62
POI 1.5	6,68	4,61	4,96	15,96	15,87	15,86
POI 2.0	0,43	0,35	0,41	0,74	0,75	0,72
POI 2.5.1	6,96	4,97	6,70	9,08	8,54	8,94
POI 3.0	4,77	4,26	4,33	7,89	7,99	7,74
Xalan 2.4	0,88	0,51	0,61	0,84	0,83	0,83
Xalan 2.5	1,73	1,44	1,63	2,16	2,52	2,48
Xalan 2.6	1,34	1,24	1,24	2,45	2,80	1,15
Xalan 2.7	3,10	2,87	2,91	5,21	5,41	5,55
Moyenne	*2,05*	*1,65*	*1,76*	*3,86*	*3,71*	*3,65*

TABLE 4.25 – Ratio de l'efficience directe par rapport à l'efficience indirecte pour les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	1,19	1,93	1,58	1,51	1,48	1,48
Ant 1.4	3,00	4,75	4,50	1,47	1,50	1,63
Ant 1.5	5,00	4,00	4,00	2,00	2,09	2,00
Ant 1.6	2,93	4,16	3,79	0,61	0,60	0,73
Ant 1.7	3,76	4,43	4,27	0,71	0,80	0,80
Camel 1.2	1,04	1,28	1,22	0,64	0,72	0,70
Camel 1.4	1,67	2,13	2,28	0,59	0,68	0,64
Camel 1.6	1,41	1,87	2,42	0,41	0,55	0,49
JEdit 3.2.1	4,07	3,89	4,12	1,01	0,96	0,87
JEdit 4.0	6,10	7,11	7,46	1,71	1,34	1,60
JEdit 4.1	7,50	6,96	7,12	0,88	1,03	0,96
JEdit 4.2	8,55	6,43	10,67	1,08	1,31	1,31
POI 1.5	1,03	1,39	1,31	0,42	0,40	0,41
POI 2.0	0,50	0,46	0,50	0,59	0,68	0,67
POI 2.5.1	0,51	0,78	0,54	0,22	0,24	0,22
POI 3.0	0,87	0,94	0,96	0,36	0,36	0,37
Xalan 2.4	0,79	1,56	1,17	0,84	0,90	0,93
Xalan 2.5	0,95	1,12	0,99	0,68	0,69	0,66
Xalan 2.6	1,30	1,49	1,40	0,70	0,57	0,94
Xalan 2.7	0,60	0,71	0,66	0,37	0,36	0,37
Moyenne	*2,64*	*2,87*	*3,05*	*0,84*	*0,86*	*0,89*

Qualité de l'ordonnancement

Pour l'ordonnancement des classes candidates au test (tableau 4.26), les approches intra-SL ont un score d'APFD très similaire, près de 65 %. Les approches gloutonnes ont un score légèrement inférieur à près de 60 %. Il y a relativement peu de variance dans les résultats moyens pour l'approche intra-SL (valeurs entre 64,4 % et 66,6 %) et intra-GL-risque (valeurs entre 60,8 % et 61,3 %). Les valeurs d'APFD sont situées dans un intervalle d'environ 10 % de pourcentage autour de la moyenne. Tout comme dans la section 4.2.4, nous avons testé le lien entre l'APFD et le nombre de fautes et la taille de l'ensemble candidat, comme ces valeurs varient entre les systèmes. Nous avons validé ce lien avec l'équation de régression linéaire suivante :

$$\begin{aligned} APFD = b_0 + b_1 \times \text{ taille ensemble candidat } + \\ b_2 \times \text{ nombre de fautes détectées directement} \end{aligned} \tag{4.3}$$

Pour les approches gloutonnes de KNN et de la combinaison de KNN et RF, la régression est significative et les tests marginaux des deux variables sont significatifs. Pour les autres situations, la régression est non significative. Alors, dans les deux cas où la régression s'est avérée significative, les APFD et la moyenne de l'APFD sont des résultats à considérer avec prudence, car les facteurs qui ne devraient pas avoir une influence sur la valeur de l'APFD, soit la taille de l'ensemble candidat et le nombre de fautes détectées directement, ont dans les faits une influence. Les valeurs incluent donc un biais important et doivent être interprétées en considérant que plus le nombre de classes est faible, plus l'APFD sera élevée. Toutefois, la valeur des coefficients est faible donc l'ampleur de la taille de l'effet de ce lien non-désiré est également faible.

TABLE 4.26 – APFD pour les approches intra-système

Système	Intra-GL-risque (KNN)	Intra-GL-risque (RF)	Intra-GL-risque (KNN + RF)	Intra-SL (KNN)	Intra-SL (RF)	Intra-SL (KNN + RF)
Ant 1.3	57,0 %	55,3 %	54,6 %	58,2 %	61,3 %	61,5 %
Ant 1.4	53,8 %	54,0 %	57,8 %	52,0 %	54,7 %	52,8 %
Ant 1.5	58,9 %	67,8 %	64,5 %	61,1 %	61,3 %	61,9 %
Ant 1.6	54,1 %	54,9 %	56,5 %	58,0 %	57,9 %	57,3 %
Ant 1.7	62,0 %	60,0 %	59,3 %	59,7 %	61,5 %	59,9 %
Camel 1.2	58,6 %	63,6 %	65,0 %	74,6 %	62,4 %	68,0 %
Camel 1.4	57,7%	64,4%	59,6%	73,3%	66,1%	68,3%
Camel 1.6	62,6%	62,0%	61,3%	74,4%	70,1%	74,1%
JEdit 3.2.1	72,8%	78,1%	77,7%	75,3%	75,0%	74,1%
JEdit 4.0	72,8%	76,9%	77,0%	65,0%	67,0%	63,5%
JEdit 4.1	68,0 %	74,9 %	73,4 %	74.9 %	67,4 %	63,2 %
JEdit 4.2	70,8 %	74,2 %	70,3 %	76,2 %	80,0 %	78,3 %
POI 1.5	67,4%	71,4%	66,6%	66,2%	73,8%	68,9%
POI 2.0	55,0%	69,6%	41,7%	69,3%	54,4%	54,2%
POI 2.5.1	57,9 %	59,8 %	58,5 %	65,6 %	66,2 %	64,9 %
POI 3.0	67,9 %	65,9 %	66,1 %	70,9 %	71,3 %	72,0 %
Xalan 2.4	54,2 %	48,1 %	49,6 %	72,3 %	72,2 %	71,4 %
Xalan 2.5	55,3 %	53,4 %	56,5 %	64,6 %	62,0 %	62,9 %
Xalan 2.6	56,3 %	57,0 %	57,3 %	64,4 %	70,0 %	55,7 %
Xalan 2.7	53,1 %	50,6 %	52,1 %	55,3 %	54,5 %	55,1 %
Moyenne	*60,8 %*	*60,8 %*	*61,3 %*	*66,6 %*	*65,5 %*	*64,4 %*

4.3.2 Comparaison des approches de priorisation

La présente section offre des comparaisons entre les approches intra-SL afin de sélectionner la meilleure approche intra-SL. Ensuite, cette approche est comparée avec les approches de contrôle dont les résultats ont été présentés à la section 4.2.

Comparaison des approches développées

Le tableau 4.27 présente les différences, pour chaque métrique d'évaluation, entre les 3 approches intra-SL afin de vérifier si l'utilisation d'un des trois algorithmes pour estimer la probabilité des fautes permet d'établir de meilleurs résultats. La colonne p-valeur présente les p-valeur des tests effectués entre chaque paire d'approches SL. Le test de Student est utilisé pour comparer les moyennes et celui de Fisher pour les variances. On observe des différences significatives de moyenne majoritaire entre KNN et KNN + RF où KNN + RF offre de meilleurs résultats par rapport à plusieurs indicateurs. Au niveau de la couverture, on remarque une légère avance significative d'intra-SL (KNN + RF) , même si cela conduit à un ensemble de classes légèrement plus important en termes de lignes de code. Intra-SL (KNN + RF) détecte également un peu plus de fautes (0,5 % de plus que intra-SL (KNN) et 3,5 % de plus qu'intra-SL (RF)). En revanche, son efficience est moindre (par 0,31 faute par mille lignes de code), particulièrement l'efficience indirecte. Finalement, la qualité de l'ordonnancement est supérieure pour l'approche utilisant seulement KNN, même si encore une fois, aucune différence significative n'est notée entre les approches. L'approche intra-SL (KNN + RF) semble offrir les meilleurs résultats au prix de l'efficience des tests.

Au niveau de la dispersion interne (entre les versions d'un même système), les écarts-types de certaines métriques d'évaluation ont été calculés. Nous nous sommes limités aux mesures où la taille de l'ensemble des classes candidates et la taille du système n'influençaient pas la valeur de la métrique, car chaque version possède un nombre de classes différent. Ce critère exclu la plupart des mesures d'efficience. Aucune approche ne présente de différence significative au niveau de sa variance vis-à-vis des autres approches. L'approche intra-SL (KNN + RF), sauf au niveau de la couverture directe, est celle présentant la plus petite variation dans la majorité des cas. Un plus petit écart-type indique des résultats plus constants entre les systèmes et,

TABLE 4.27 – Comparaison des approches intra-SL par métrique d'évaluation

	Moyennes			p-valeur moyennes		
Métrique d'évaluation	**Intra-SL (KNN)**	**Intra-SL (RF)**	**Intra-SL (KNN + RF)**	**KNN / RF**	**KNN / KNN + RF**	**RF / KNN + RF**
Couverture totale (LOC)	81,6%	81,4%	84,7%	0,466	**0,019**	**0,039**
Couverture directe (LOC)	31,8%	32,8%	34,6%	0,103	**0,039**	0,134
Couverture indirecte (LOC)	49,8%	48,7%	50,1%	0,185	0,284	0,091
Taille ensemble candidat	110	108	108	0,219	0,084	0,468
Fautes détectées total	91,4%	91,7%	92,4%	0,362	**0,018**	0,140
Fautes détectées direct	39,1%	40,0%	40,8%	0,136	**0,003**	0,160
Fautes détectées indirect	52,4%	51,7%	51,6%	0,099	0,053	0,409
Efficience détection total	6,15	5,94	5,84	0,081	**0,033**	0,225
Efficience détection direct	2,29	2,23	2,20	0,192	0,137	0,167
Efficience détection indirect	3,86	3,71	3,65	0,073	**0,011**	0,264
Ratio efficience directe/indirecte	0,84	0,86	0,89	0,207	**0,012**	0,157
APFD	66,6%	65,5%	64,4%	0,192	**0,023**	0,115
	Écarts-types			**p-valeur variances**		
Métrique d'évaluation	**Intra-SL (KNN)**	**Intra-SL (RF)**	**Intra-SL (KNN + RF)**	**KNN / RF**	**KNN / KNN + RF**	**RF / KNN + RF**
Couverture totale (LOC)	10,8%	12,0%	9,3%	0,451	0,320	0,137
Couverture directe (LOC)	6,3%	6,5%	8,3%	0,796	0,158	0,214
Couverture indirecte (LOC)	9,7%	9,8%	9,7%	0,933	0,970	0,942
Taille ensemble candidat	7268,8%	7141,8%	7265,3%	0,681	0,986	0,708
Fautes détectées total	8,3%	8,5%	7,7%	0,755	0,201	0,250
Fautes détectées direct	13,4%	13,1%	13,1%	0,735	0,629	0,971
Fautes détectées indirect	12,2%	11,9%	12,3%	0,593	0,814	0,438
Ratio efficience directe/indirecte	48,7%	47,3%	48,3%	0,640	0,854	0,730
APFD	7,4%	7,2%	7,3%	0,873	0,964	0,872

par conséquent, une plus grande fiabilité des résultats. Nous sélectionnons l'approche intra-SL (KNN + RF) comme approche permettant d'atteindre la meilleure sélection et priorisation des classes candidates aux tests dans le contexte intra-système. Elle sera utilisée pour la comparaison avec les approches de contrôle.

L'annexe B présente les résultats sous forme de graphique (figure B.2) les métriques d'évaluation liées aux cinq axes des questions de recherche.

Comparaison aux approches de contrôle

L'approche intra-SL (KNN + RF) est considérée comme la meilleure approche parmi les trois approches intra-SL développées pour le contexte intra-système. Elle est ici comparée aux approches de contrôle sans variante et à l'approche gloutonne utilisant KNN et RF (car il s'agit de son approche de type intra-GL-risque correspondante). Dans les tableaux ci-dessous (tableau 4.28 au tableau 4.32), un nombre en caractère gras indique une différence significative au niveau de confiance de 95 % avec la valeur correspondante de l'approche intra-SL (KNN + RF).

TABLE 4.28 – Comparaison de la couverture d'intra-SL (KNN+RF) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Intra-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Intra-SL (KNN+RF)
Couverture totale (LOC)	Ant	44,7 %	73,7 %	69,1 %	68,3 %	48,6 %	53,4 %	71,4 %	82,6 %
	Camel	48,4 %	80,0 %	78,7 %	50,8 %	49,4 %	51,8 %	78,7 %	81,9 %
	JEdit	55,2 %	92,3 %	89,7 %	74,6 %	69,7 %	71,0 %	89,7 %	84,6 %
	POI	35,5%	91,7%	93,4%	79,1%	53,0%	79,2%	93,4%	95,0%
	Xalan	46,7 %	87,1 %	81,4 %	60,3 %	43,2 %	49,0 %	81,4 %	79,3 %
	Total	***45,9 %***	*84,7 %*	*82,0 %*	***67,5 %***	***52,7 %***	***61,0 %***	*82,6 %*	*84,7 %*
Couverture directe (LOC)	Ant	17,3 %	55,7 %	53,0 %	44,5 %	17,1 %	11,1 %	53,0 %	35,6 %
	Camel	14,4 %	55,8 %	53,9 %	31,3 %	16,1 %	12,6 %	54,0 %	32,9 %
	JEdit	12,6 %	71,9 %	65,6 %	46,8 %	24,7 %	11,2 %	65,7 %	32,8 %
	POI	13,3%	58,4%	54,7%	37,5%	22,8%	14,3%	54,9%	31,0%
	Xalan	15,6 %	66,5 %	61,1 %	37,1 %	15,9 %	12,4 %	61,2 %	39,9 %
	Total	***14,8 %***	***61,7 %***	***57,7 %***	*40,1 %*	***19,4 %***	***12,2 %***	***57,7 %***	*34,6 %*
Couverture indirecte (LOC)	Ant	27,4 %	18,0 %	16,1 %	23,8 %	31,5 %	42,3 %	18,4 %	47,0 %
	Camel	34,0 %	24,2 %	24,7 %	19,5 %	33,2 %	39,3 %	24,8 %	49,0 %
	JEdit	42,6 %	20,4 %	24,1 %	27,8 %	44,9 %	59,7 %	24,1 %	51,7 %
	POI	22,1%	33,3%	38,6%	41,6%	30,2%	64,9%	38,5%	64,0%
	Xalan	31,2 %	20,6 %	20,3 %	23,2 %	27,3 %	36,6 %	20,2 %	39,4 %
	Total	***31,1 %***	***23,0 %***	***24,3 %***	***27,4 %***	***33,3 %***	*48,7 %*	***24,9 %***	*50,1 %*

Pour la couverture (tableau 4.28), l'approche intra-SL (KNN + RF) présente une amélioration significative de la couverture totale atteinte par rapport à 4 approches de contrôle, soit l'approche aléatoire (+ 38,8 %), l'algorithme glouton associé au risque (+

17,2 %), la descente de gradient (+ 32,0 %) et l'algorithme génétique (+ 23,7 %). Au niveau de la couverture directe, l'approche atteint une couverture directe supérieure à trois approches (aléatoire, descente de gradient et algorithme génétique), mais significativement inférieure à trois approches soit l'approche glouton pour la couverture (-27,1 %), l'approche glouton de la complexité (-23,1 %) et la classification hiérarchique (-23,1 %). Notons ici, que la couverture directe, bien qu'intéressante à obtenir car elle facilite sans doute la correction des fautes détectées par les tests, représente aussi l'effort de test associé à l'ensemble des classes candidates. Donc, l'approche intra-SL (KNN + RF) possède un effort de test significativement plus faible que les trois approches de contrôle précédemment identifiées. Notons que ces réductions n'ont pas lieu pour des approches pour lesquelles l'approche développée augmentait la couverture totale. Notre approche améliore la couverture totale ou réduit l'effort, mais elle ne permet pas de réaliser les deux objectifs en même temps. L'effort de test est calculé par la couverture directe de notre approche (une grande couverture directe augmente l'effort de test) et la couverture directe contribue à la couverture total (en l'augmentant). Notre observation s'appuie sur le fait la couverture directe influence négativement l'effort de test et positivement la couverture totale. L'approche développée permet donc d'obtenir une amélioration nette par rapport aux approches de contrôle.

Pour les tailles d'ensemble de tests (tableau 4.29), aucune différence significative n'est observée sur la moyenne (104 pour la référence comparativement à 108 pour l'approche intra-SL). On remarque que pour les systèmes Ant, Camel et POI, la taille moyenne des approches est similaire. Cependant, pour les deux autres systèmes, des différences importantes existent, soit négative pour JEdit et positive pour Xalan. Ce comportement très hétérogène entre les systèmes indique des différences importantes entre les divers systèmes sur le comportement de l'approche développée.

TABLE 4.29 – Comparaison de la taille de l'ensemble de classes candidates au test de l'approche en contexte intra-système aux approches de contrôle

Métrique d'évaluation	**Système**	**Référence 15 % - Approches de contrôle**	**Intra-SL (KNN + RF)**
Taille de l'ensemble de classes candidates aux tests	Ant	71	68
	Camel	157	167
	JEdit	96	64
	POI	63	49
	Xalan	153	215
	Total	*104*	*108*

Pour la détection des fautes (tableau 4.30), l'approche intra-SL (KNN + RF) améliore significativement, dans tous les cas, le pourcentage de fautes détectées. On remarque, toutefois, une légère baisse de performance pour le système Xalan. Cette baisse est un peu plus notée pour l'approche développée, mais présente pour toutes les approches de contrôle à l'exception de l'approche aléatoire. La principale cause, pour expliquer cette différence dans notre approche, vient des différences identifiées dans la distribution des métriques de Xalan par rapport aux autres systèmes à la section 4.1. Ces différences modifient le comportement de l'approche qui s'appuie sur les métriques présentant une différence pour Xalan et rendent l'approche moins efficace dans certains cas. Les raisons de ces différences sont expliquées en détail à la section 4.5 et la raison pour laquelle une performance plus faible est observée pour Xalan y est détaillée.

TABLE 4.30 – Comparaison de la détection des fautes d'intra-SL (KNN+RF) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Intra-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Intra-SL (KNN+RF)
Détection de fautes total	Ant	38,9 %	71,4 %	66,7 %	72,5 %	47,5 %	53,6 %	68,6 %	91,3 %
	Camel	57,2 %	84,8 %	84,6 %	82,4 %	61,0 %	59,3 %	84,6 %	95,1 %
	JEdit	73,9 %	95,5 %	95,2 %	97,0 %	75,2 %	79,9 %	95,2 %	97,5 %
	POI	35,60%	83,60%	90,30%	77,90%	50,20%	73,00%	90,30%	94,80%
	Xalan	49,9 %	74,7 %	70,5 %	70,5 %	47,6 %	53,5 %	70,5 %	84,2 %
	Total	***50,2 %***	***81,4 %***	***80,6 %***	***79,5 %***	***55,6 %***	***63,6 %***	***81,1 %***	*92,4 %*
Détection de fautes directes	Ant	14,2 %	56,5 %	53,5 %	55,8 %	14,0 %	13,9 %	52,3 %	50,3 %
	Camel	12,4 %	47,9 %	52,9 %	54,2 %	16,7 %	12,9 %	52,9 %	35,9 %
	JEdit	11,0 %	79,8 %	76,5 %	84,3 %	13,0 %	14,0 %	76,6 %	52,2 %
	POI	12,40%	38,10%	38,40%	34,00%	17,20%	17,90%	38,50%	26,90%
	Xalan	15,9 %	38,7 %	37,4 %	35,8 %	14,2 %	13,7 %	37,4 %	35,3 %
	Total	***13,3 %***	***52,6 %***	***51,8 %***	***52,9 %***	***14,9 %***	***14,5 %***	***51,5 %***	*40,8 %*
Détection de fautes indirectes	Ant	24,7 %	14,9 %	13,2 %	16,7 %	33,6 %	39,7 %	16,3 %	41,1 %
	Camel	44,8 %	36,9 %	31,7 %	28,2 %	44,3 %	46,5 %	31,7 %	59,3 %
	JEdit	62,9 %	15,7 %	18,7 %	12,7 %	62,2 %	66,0 %	18,6 %	45,4 %
	POI	23,10%	45,50%	51,90%	43,90%	33,10%	55,10%	51,80%	67,90%
	Xalan	34,1 %	36,1 %	33,2 %	34,7 %	33,5 %	39,8 %	33,1 %	48,9 %
	Total	***36,9 %***	***28,7 %***	***28,8 %***	***26,6 %***	***40,8 %***	*49,1 %*	***29,6 %***	*51,6 %*

Pour la détection directe, l'approche intra-SL (KNN + RF) augmente significativement le pourcentage de fautes détectées directement par rapport à l'approche aléatoire, la descente de gradient et l'algorithme génétique. Les quatre autres approches de contrôle détectent significativement plus de fautes directement que l'approche intra-SL.

Pour trois de ces quatre approches, soit l'algorithme glouton pour la couverture, la complexité et l'algorithme de classification hiérarchique, ces approches ont une couverture directe significativement plus importante, ce qui est de facto relié à une plus

grande détection des fautes directes. Pour la quatrième approche, l'algorithme glouton associé au risque de faute, la différence est due au fait suivant : les deux approches ont le même point de départ, c'est-à-dire la probabilité d'erreur, mais l'approche SL diminue la probabilité d'une classe d'être sélectionnée si elle est indirectement couverte. Par conséquent, il est normal d'observer moins de fautes directement couvertes, car dans l'approche par les algorithmes gloutons les classes fautives indirectement couvertes ne voient pas leur probabilité d'être sélectionnées réassignée avec l'évolution de l'ensemble des classes candidates sélectionnées. L'approche intra-SL (KNN + RF) améliore aussi la détection indirecte de façon significative pour toutes les approches sauf celle de l'algorithme génétique. Ce dernier résultat était attendu, car l'approche développée est la seule qui mise dans son fonctionnement sur la couverture indirecte.

TABLE 4.31 – Comparaison de l'efficience de la détection des fautes d'intra-SL (KNN+RF) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Intra-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Intra-SL (KNN+RF)
Efficience détection totale	Ant	3,69	1,94	1,89	2,68	4,00	7,71	2,09	3,89
	Camel	9,39	3,61	3,73	6,04	8,61	10,8	3,73	6,66
	JEdit	9,97	1,89	2,08	2,84	4,11	9,81	2,08	4,19
	POI	10,36	5,32	5,99	7,72	8,27	18,1	5,95	11,28
	Xalan	5,35	1,78	1,82	3,10	4,60	6,90	1,82	3,89
	Total	*7,47*	***2,83***	***3,01***	*4,31*	*5,69*	***10,51***	***3,05***	*5,84*
Efficience détection directe	Ant	1,34	1,50	1,55	1,89	1,15	2,02	1,46	2,04
	Camel	2,36	1,96	2,23	3,82	2,27	2,35	2,23	2,54
	JEdit	1,23	1,55	1,61	2,43	0,72	1,90	1,61	2,19
	POI	4,21	2,30	2,32	3,62	2,97	4,16	2,32	2,97
	Xalan	1,72	0,86	0,91	1,50	1,50	2,00	0,91	1,38
	Total	*2,12*	*1,61*	*1,69*	*2,55*	*1,67*	*2,47*	*1,67*	*2,20*
Efficience détection indirecte	Ant	2,35	0,44	0,35	0,79	2,84	5,69	0,63	1,85
	Camel	7,03	1,65	1,50	2,22	6,34	8,44	1,50	4,13
	JEdit	8,74	0,33	0,47	0,42	3,39	7,91	0,47	2,00
	POI	6,15	3,02	3,66	4,10	5,30	13,94	3,63	8,32
	Xalan	3,63	0,92	0,91	1,60	3,09	4,90	0,91	2,51
	Total	*5,34*	***1,21***	***1,32***	***1,76***	*4,02*	***8,04***	***1,38***	*3,65*

Le tableau 4.31 montre les moyennes par système pour les métriques d'évaluation liées à l'efficience. Au niveau de l'efficience totale, l'approche intra-SL (KNN + RF) est significativement plus efficiente que l'approche gloutonne basée sur la couverture (+ 3,01), celle basée sur la complexité (+ 2,83) et l'approche de classification hiérarchique (+ 2,79). Elle est, toutefois, significativement moins efficiente que l'algorithme génétique (- 4,67). On peut expliquer cette tendance par le plus faible nombre de lignes de code sélectionnées par l'algorithme génétique. Il est alors plus facile de détecter de façon efficiente les fautes, car l'effort de test déployé est moins grand. Aucune ap-

proche ne se distingue significativement des autres par rapport à l'efficience directe. Au niveau de l'efficience indirecte, la tendance est la même que l'efficience totale sauf pour l'algorithme glouton basé sur le risque qui est significativement moins efficient (+ 1,89) que l'approche intra-SL (KNN + RF). La valeur d'efficience plus petite (mais l'écart n'est pas assez important pour être significatif), entre SL et les approches aléatoire et de descente de gradient, s'explique par la plus faible de l'ensemble de classes candidates sélectionnées par ces algorithmes.

Comme mentionné précédemment, le critère d'efficience ne doit pas être interprété seul, mais plutôt conjointement avec le nombre de fautes détectées (une haute détection des fautes aide l'efficience) et la taille de l'ensemble candidat (une forte taille nuit à l'efficience). Pour deux séquences de test détectant le même nombre de fautes, la plus grande des deux aura une efficience plus basse, car elle consomme plus de ressources pour parvenir au même résultat. De même, deux séquences pourraient avoir la même efficience, mais l'une peut détecter la moitié des fautes en consommant la moitié des ressources. Nous n'avons pas effectué de comparaison sur le ratio entre efficience directe et indirecte puisqu'il s'agit d'une donnée intéressante à des fins de description, mais qui ne révèle rien sur l'efficacité de l'approche.

TABLE 4.32 – Comparaison de l'APFD d'intra-SL (KNN+RF) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Intra-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Intra-SL (KNN+RF)
APFD	Ant	45,4 %	54,7 %	52,6 %	58,6 %	49,4 %	48,7 %	54,6 %	58,7 %
	Camel	53,8 %	66,4 %	66,3 %	62,0 %	50,4 %	53,3 %	66,4 %	70,1 %
	JEdit	48,2 %	71,4 %	70,8 %	74,6 %	40,8 %	48,0 %	70,8 %	69,8 %
	POI	54,6%	62,2%	63,1%	58,2%	52,0%	52,4%	63,1%	65,0%
	Xalan	45,6 %	57,5 %	57,0 %	53,9 %	53,30 %	53,1 %	56,9 %	61,3 %
	Total	***49,1 %***	*61,9 %*	*61,3 %*	*61,3 %*	***49,1 %***	***50,9 %***	*61,8 %*	*64,4 %*

Pour l'APFD (tableau 4.32), nous remarquons que l'approche intra-SL (KNN + RF) permet un ordonnancement significativement meilleur des classes candidates que l'approche aléatoire (+ 15,3 %), la descente de gradient (+ 15,3 %) et l'algorithme génétique (+ 13,5 %). La valeur moyenne de notre approche au niveau de l'APFD est également supérieure aux moyennes des autres approches par environ 3 %, mais cet écart n'est pas suffisant pour obtenir une amélioration statistiquement significative. Les valeurs sont généralement consistantes entre les systèmes de sorte que les différences observées sur la moyenne de tous les systèmes peuvent aussi être observées sur la moyenne de chaque système considéré séparément.

Le fait que intra-SL présente une meilleure APFD illustre bien qu'elle parvient à résoudre de façon intéressante le problème de priorisation. Les comparatifs sur la couverture, le nombre de fautes détectées et l'efficience pour leur part illustrent bien que l'approche SL résoud bien le problème de sélection des classes candidates au test.

4.4 Prédiction inter-systèmes

La présente section décrit les approches développées avec les modèles d'apprentissage automatique dans un contexte d'apprentissage inter-systèmes. L'apprentissage inter-systèmes consiste à utiliser des données d'entraînement provenant de divers systèmes afin de réaliser la phase d'apprentissage. La phase de prédiction est effectuée sur un système différent de tous ceux utilisés lors de l'apprentissage. Dans nos expérimentations, nous avons considéré 10 systèmes d'entraînement pour chaque prédiction. Les systèmes d'entraînement peuvent être des systèmes différents ou des versions différentes du même système.

4.4.1 Approches de priorisation

Les prochaines sections détaillent les résultats obtenus par les approches inter-SL pour chacune des trois méthodes de prédiction des fautes (KNN, RF et KNN + RF), ainsi que pour les algorithmes gloutons associés respectivement à ces méthodes. Comme précédemment, les approches sont évaluées selon quatre axes : la couverture, la détection des fautes, l'efficience de la détection et la qualité de l'ordonnancement.

Couverture

La couverture des approches inter-SL est présentée du tableau 4.33 au tableau 4.35. On remarque que les approches inter-SL ont une couverture totale près de 86 % alors que les approches gloutonnes de contrôle atteignent seulement 73 % de couverture. La couverture des approches inter-SL est principalement indirecte (35 % directe et 50 % indirecte) tandis que pour les approches de contrôle, la couverture est plutôt directe (43 % direct et 30 % indirect). La principale raison expliquant cette différence est que les algorithmes de contrôle ne considèrent pas la couverture indirecte. Les approches inter-SL misent donc efficacement sur la couverture indirecte pour réduire l'effort de test. On le constate par une couverture directe, en lignes de code, moins élevée pour les approches inter-SL que les approches de contrôle gloutonnes. On remarque généralement une homogénéité entre les systèmes, à l'exception de Xalan qui manifeste

des comportements différents. Généralement, les résultats sont plus faibles sur les versions du système Xalan. Les raisons de ces différences sont expliquées à la section 4.5.

TABLE 4.33 – Couverture totale (% LOC) des approches de priorisations inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	51,5 %	56,2 %	54,2 %	95,5 %	92,7 %	95,3 %
Ant 1.4	66,5 %	69,5 %	66,4 %	93,2 %	88,3 %	94,9 %
Ant 1.5	63,6 %	66,0 %	63,5 %	90,1 %	88,6 %	88,0 %
Ant 1.6	70,3 %	72,3 %	70,3 %	91,5 %	91,5 %	92,9 %
Ant 1.7	68,2 %	71,9 %	71,6 %	88,7 %	84,7 %	92,0 %
Camel 1.2	71,2 %	71,9 %	70,3 %	92,0 %	91,5 %	93,3 %
Camel 1.4	63,7%	63,3%	70,3%	89,2%	87,5%	93,1%
Camel 1.6	71,2%	72,7%	69,7%	88,1%	87,2%	92,7%
JEdit 3.2.1	92,8 %	91,5 %	89,1 %	92,4 %	89,4 %	89,1 %
JEdit 4.0	90,2%	89,3%	93,4%	94,5%	86,1%	86,9%
JEdit 4.1	90,0 %	91,8 %	91,6 %	93,7 %	91,0 %	90,7 %
JEdit 4.2	87,1 %	80,0 %	87,2 %	90,0 %	90,8 %	79,8 %
POI 1.5	85,6%	78,3%	81,3%	93,7%	95,1%	90,8%
POI 2.0	88,4%	87,3%	91,6%	83,5%	95,8%	96,1%
POI 2.5.1	81,9%	84,8%	85,1%	95,3%	94,0%	93,6%
POI 3.0	75,9 %	83,0 %	77,1 %	95,3 %	93,8 %	84,2 %
Xalan 2.4	71,3 %	77,0 %	71,6 %	88,3 %	88,2 %	89,9 %
Xalan 2.5	57,8 %	55,6 %	57,5 %	67,7 %	66,7 %	68,1 %
Xalan 2.6	51,9 %	49,7 %	49,0 %	52,7 %	51,8 %	51,8 %
Xalan 2.7	51,1 %	48,4 %	48,1 %	51,0 %	51,4 %	54,3 %
Moyenne	*72,5 %*	*73,0 %*	*72,9 %*	*86,3 %*	*85,3 %*	*85,9 %*

La taille des différents ensembles de classes candidates au test est donnée au tableau 4.36. La colonne « Référence 15 % » indique la taille de l'ensemble des classes candidates pour les approches de contrôle (elles utilisent toutes le même seuil de 15 % qui est la valeur moyenne des seuils pour l'approche inter-SL). On remarque une variabilité importante dans le nombre de classes candidates au test. En moyenne, inter-SL (RF) est celle qui produit les plus petits ensembles de classes candidates au test. Il est étonnant qu'inter-SL (RF) atteigne une couverture totale et directe similaire aux autres variantes de SL malgré qu'elle sélectionne moins de classes comme candidates aux tests. Cependant, lorsqu'on regarde par système, des différences importantes sont observées. Ces différences seront étudiées plus en détail dans la section 4.4.1.

En analysant la corrélation entre la différence de taille des approches inter-SL et des approches de contrôle (seuil de 15 %) avec la taille des systèmes en nombre de classes (tableau 4.37), on remarque que la plupart des coefficients de corrélation sont

TABLE 4.34 – Couverture directe (% LOC) des approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	32,4 %	31,3 %	34,2 %	54,3 %	50,8 %	54,2 %
Ant 1.4	43,7 %	45,1 %	43,8 %	46,4 %	40,7 %	52,3 %
Ant 1.5	42,3 %	44,9 %	41,8 %	38,9 %	36,2 %	37,6 %
Ant 1.6	45,4 %	47,3 %	45,7 %	41,5 %	38,8 %	40,0 %
Ant 1.7	47,4 %	49,7 %	49,7 %	37,4 %	33,4 %	42,3 %
Camel 1.2	41,7 %	40,2 %	38,0 %	38,5 %	39,2 %	40,1 %
Camel 1.4	34,3%	37,3%	41,7%	35,1%	34,3%	37,5%
Camel 1.6	41,5%	43,8%	40,4%	35,4%	33,0%	39,5%
JEdit 3.2.1	60,1 %	67,7 %	56,0 %	48,8 %	44,7 %	41,7 %
JEdit 4.0	58,4%	52,4%	67,0%	49,2%	36,6%	37,7%
JEdit 4.1	55,3 %	65,0 %	57,6 %	47,3 %	43,1 %	43,6 %
JEdit 4.2	55,0 %	47,8 %	57,0 %	40,7 %	40,6 %	29,0 %
POI 1.5	40,7%	42,5%	44,9%	22,4%	23,0%	15,5%
POI 2.0	39,7%	41,6%	42,3%	15,5%	32,4%	30,2%
POI 2.5.1	33,4%	38,1%	36,9%	36,0%	34,3%	34,5%
POI 3.0	32,0 %	39,9 %	37,5 %	35,4 %	34,1 %	23,8 %
Xalan 2.4	36,4 %	47,0 %	40,1 %	32,7 %	32,8 %	33,6 %
Xalan 2.5	34,1 %	34,7 %	29,0 %	24,8 %	21,3 %	23,9 %
Xalan 2.6	33,7 %	32,0 %	29,3 %	18,8 %	18,8 %	18,0 %
Xalan 2.7	31,7 %	32,5 %	29,6 %	18,1 %	18,2 %	22,3 %
Moyenne	*42,0 %*	*44,0 %*	*43,1 %*	*35,8 %*	*34,3 %*	*34,9 %*

TABLE 4.35 – Couverture indirecte (% LOC) des approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	19,1 %	24,9 %	20,0 %	41,3 %	41,9 %	41,1 %
Ant 1.4	22,9 %	24,4 %	22,7 %	46,8 %	47,6 %	42,6 %
Ant 1.5	21,3 %	21,1 %	21,7 %	51,2 %	52,4 %	50,4 %
Ant 1.6	24,9 %	25,0 %	24,6 %	50,0 %	52,7 %	52,9 %
Ant 1.7	20,8 %	22,3 %	21,9 %	51,3 %	51,4 %	49,6 %
Camel 1.2	29,6 %	31,7 %	32,3 %	53,5 %	52,4 %	53,2 %
Camel 1.4	29,4%	25,9%	28,6%	54,1%	53,3%	55,6%
Camel 1.6	29,8%	28,9%	29,3%	52,8%	54,1%	53,2%
JEdit 3.2.1	32,7 %	23,9 %	33,1 %	43,7 %	44,7 %	47,4 %
JEdit 4.0	31,8%	36,9%	26,5%	45,2%	49,5%	49,2%
JEdit 4.1	34,7 %	26,8 %	34,0 %	46,4 %	47,9 %	47,2 %
JEdit 4.2	32,1 %	32,2 %	30,1 %	49,4 %	50,2 %	50,9 %
POI 1.5	44,8 %	35,9 %	36,4 %	71,3 %	72,1 %	75,3 %
POI 2.0	48,7 %	45,7 %	49,2 %	68,1 %	63,4 %	65,9 %
POI 2.5.1	48,4 %	46,7 %	48,2 %	59,3 %	59,7 %	59,1 %
POI 3.0	43,9 %	43,1 %	39,7 %	59,9 %	59,7 %	60,4 %
Xalan 2.4	34,9 %	30,0 %	31,5 %	55,6 %	55,4 %	56,3 %
Xalan 2.5	23,6 %	20,8 %	28,6 %	42,9 %	45,4 %	44,2 %
Xalan 2.6	18,1 %	17,7 %	19,7 %	34,0 %	33,1 %	33,7 %
Xalan 2.7	19,4 %	15,9 %	18,5 %	32,9 %	33,1 %	31,9 %
Moyenne	*30,6 %*	*29,0 %*	*29,8 %*	*50,5 %*	*51,0 %*	*51,0 %*

négatifs, ce qui indique que plus le système est de taille importante, moins l'ensemble candidat identifié par SL est important, d'un point de vue relatif. Comme il ne s'agit pas d'étude de comparaison et que les coefficients sont calculés pour chaque système, nous ne sommes pas en mesure de prédire la taille des ensembles candidats à partir de la taille des systèmes. La tendance suit celle identifiée pour l'algorithme intra-SL dont les résultats sont présentés au tableau 4.18.

TABLE 4.36 – Taille des ensembles de classes candidates au test pour les approches inter-systèmes

Système	Référence 15 %	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	19	60	48	57
Ant 1.4	40	72	62	88
Ant 1.5	60	102	94	93
Ant 1.6	78	127	121	129
Ant 1.7	159	214	175	272
Camel 1.2	115	162	152	152
Camel 1.4	168	190	175	220
Camel 1.6	188	183	170	225
JEdit 3.2.1	76	77	45	52
JEdit 4.0	91	107	41	40
JEdit 4.1	97	86	52	69
JEdit 4.2	121	85	97	43
POI 1.5	44	34	34	23
POI 2.0	57	31	43	51
POI 2.5.1	69	53	57	58
POI 3.0	80	59	63	53
Xalan 2.4	122	105	106	121
Xalan 2.5	135	100	90	109
Xalan 2.6	174	107	107	111
Xalan 2.7	179	107	109	141
Moyenne	*104*	*103*	*92*	*105*

TABLE 4.37 – Corrélations entre la différence de taille de l'ensemble de classes candidates et la taille du système (nombre de classes) par système pour les approches inter-systèmes

Système	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant	0,83	-0,39	0,92
Camel	-0,96	-0,98	0,26
JEdit	-0,81	0,39	-0,86
POI	-0,45	-0,81	-0,27
Xalan	-1	-0,96	-0,86

Détection des fautes

Pour la détection des fautes, les approches inter-SL détectent plus de 90 % des fautes dans les systèmes (tableau 4.38). Les approches de contrôle gloutonnes pour leur part détectent environ 80 % des erreurs. Les approches inter-SL misent particulièrement sur la détection indirecte, comme seuls 38 % des fautes sont détectées directement pour inter-SL (KNN) et pour inter-SL (RF) (voir tableau 4.39), tandis que la valeur descend à 18 % pour inter-SL (KNN + RF). La même tendance à la baisse est observée du côté des approches de contrôle gloutonnes où elles ne détectent que 24 % des fautes directement pour la combinaison de KNN et RF, tandis que si chacun des algorithmes est pris individuellement, environ 50 % des fautes sont détectées directement. Cette différence marquée est explicable par une sous-performance à prédire les fautes de plus de 10 % de l'algorithme combinant RF et KNN dans 13 des 20 versions. Des sous-performances sont observées avec les 5 systèmes.

Les résultats entre inter-SL (KNN) et inter-SL (RF) étant relativement uniformes, il serait peu probable que la différence observée soit uniquement due à la sélection des ensembles d'entraînement. En effet, pour chaque approche, l'ensemble d'entraînement a été sélectionné indépendamment. Un autre aspect surprenant est que cette différence ne se remarque pas sur la détection totale des fautes, par conséquent, inter-SL (KNN + RF) même s'il éprouve des difficultés remarquables à identifier directement les classes fautives, sélectionne tout de même des classes à proximité de sorte que la diminution de la détection directe est compensée par une augmentation de la détection indirecte. En effet, l'approche inter-SL (KNN + RF) détecte 74 % des fautes indirectement contre seulement 53 % pour les autres approches (tableau 4.40). Le même constat peut être fait sur l'algorithme de contrôle glouton où l'approche combinant RF et KNN détecte indirectement 57 % des fautes contre environ 28 % pour les autres variantes.

Bien que les données soient trop peu nombreuses pour établir une conclusion, cela semble indiquer une certaine robustesse du modèle inter-SL (KNN + RF), et de l'approche inter-SL en général, pour sélectionner des classes pertinentes à tester, et ce, même si les modèles de prédiction ne sont pas tout à fait justes. Nous attribuons la robustesse aux modèles d'apprentissage automatique utilisés plutôt qu'à l'approche, car cette même transition entre la détection directe et indirecte est observable avec l'approche de contrôle gloutonne. Dans notre cas, le manque de justesse est certaine-

ment attribuable en grande partie à l'extrapolation effectuée par les modèles, et non à des caractéristiques internes des modèles. Si les modèles étaient non justes, l'approche de contrôle gloutonne aurait sûrement moins bien performé au niveau de la détection. On remarque aussi que pour certains systèmes des sauts importants dans la prédiction et la détection directe entre les différentes variantes utilisant différents algorithmes. Par exemple, JEdit 4.2 où inter-GL-risque (KNN) ou inter-GL-risque (RF) détectent plus de 80 % des fautes et la combinaison KNN + RF trouve à peine 11,7 % de fautes directement. Des problèmes de qualité de prédiction auraient aussi affecté les variantes utilisant un seul algorithme. Nous avons également répété trois fois cette expérimentation pour s'assurer qu'il ne s'agisse pas simplement d'un problème de conditions initiales des algorithmes.

TABLE 4.38 – Détection des fautes directes et indirectes dans les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	50,4 %	58,5 %	57,8 %	98,5 %	97,0 %	97,0 %
Ant 1.4	44,7 %	51,1 %	53,2 %	89,4 %	89,4 %	93,6 %
Ant 1.5	85,7 %	85,7 %	85,7 %	97,1 %	97,1 %	97,1 %
Ant 1.6	84,2 %	85,3 %	82,6 %	97,8 %	97,3 %	97,8 %
Ant 1.7	75,1 %	79,3 %	80,2 %	95,0 %	92,6 %	98,2 %
Camel 1.2	86,4 %	84,1 %	80,7 %	95,2 %	95,0 %	95,0 %
Camel 1.4	84,5 %	87,2 %	86,6 %	94,0 %	93,1 %	94,6 %
Camel 1.6	92,0 %	89,2 %	88,4 %	95,0 %	95,0 %	96,8 %
JEdit 3.2.1	98,4 %	95,3 %	96,1 %	99,5 %	96,1 %	97,6 %
JEdit 4.0	96, 5%	95,10%	96,0 %	100,0 %	96,0 %	96,0 %
JEdit 4.1	92,6 %	92,2 %	96,8 %	96,8 %	96,3 %	96,8 %
JEdit 4.2	99,1 %	99,1 %	99,1 %	98,1 %	99,1 %	95,3 %
POI 1.5	84,5%	64,3 %	69,9 %	92,1 %	95,3 %	90,4 %
POI 2.0	79,5%	71,8 %	87,2 %	87,2 %	89,7 %	92,3 %
POI 2.5.1	90,9 %	93,3 %	92,9 %	95,0 %	97,2 %	96,8 %
POI 3.0	86,2 %	91,2 %	89,0 %	95,6 %	96,6 %	94,2 %
Xalan 2.4	76,9 %	85,3 %	77,6 %	91,7 %	91,7 %	94,2 %
Xalan 2.5	69,5 %	62,1 %	69,2 %	74,9 %	73,8 %	76,6 %
Xalan 2.6	67,1 %	67,8 %	66,0 %	70,4 %	68,4 %	69,2 %
Xalan 2.7	61,9 %	58,7 %	60,0 %	61,1 %	61,8 %	65,0 %
Moyenne	*80,3 %*	*79,8 %*	*80,7 %*	*91,2 %*	*90,9 %*	*91,7 %*

TABLE 4.39 – Détection des fautes directes dans les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	31,1 %	40,0 %	39,7 %	63,7 %	61,5 %	68,3 %
Ant 1.4	36,2 %	40,4 %	7,2 %	55,3 %	53,2 %	10,2 %
Ant 1.5	68,6 %	68,6 %	6,2 %	60,0 %	57,1 %	5,5 %
Ant 1.6	64,1 %	70,7 %	22,9 %	41,3 %	40,8 %	13,2 %
Ant 1.7	56,2 %	61,5 %	20,2 %	50,0 %	47,6 %	17,2 %
Camel 1.2	48,7 %	42,5 %	27,5 %	37,2 %	36,2 %	25,4 %
Camel 1.4	49,0 %	63,9 %	17,6 %	34,6 %	36,4 %	10,7 %
Camel 1.6	61,0 %	57,6 %	23,1 %	30,0 %	32,6 %	13,7 %
JEdit 3.2.1	79,1 %	77,7 %	59,4 %	51,8 %	40,1 %	36,8 %
JEdit 4.0	75,7 %	81,9 %	28,5 %	55,3 %	48,2 %	17,3 %
JEdit 4.1	76,0 %	77,9 %	26,4 %	50,2 %	46,5 %	16,8 %
JEdit 4.2	84,0 %	80,2 %	11,7 %	52,8 %	55,7 %	6,6 %
POI 1.5	41,5 %	36,3 %	42,5 %	25,1 %	26,6 %	24,8 %
POI 2.0	46,2 %	33,3 %	5,0 %	30,8 %	35,9 %	3,7 %
POI 2.5.1	28,8 %	31,7 %	30,0 %	16,7 %	17,9 %	19,1 %
POI 3.0	35,4 %	41,8 %	38,0 %	23,2 %	26,4 %	19,2 %
Xalan 2.4	46,8 %	53,2 %	9,6 %	36,5 %	37,8 %	7,4 %
Xalan 2.5	35,8 %	32,4 %	20,6 %	22,8 %	21,6 %	13,9 %
Xalan 2.6	34,9 %	41,2 %	19,4 %	21,5 %	21,5 %	11,6 %
Xalan 2.7	22,5 %	23,8 %	23,5 %	13,4 %	13,8 %	16,8 %
Moyenne	*51,1 %*	*52,8 %*	*24,0 %*	*38,6 %*	*37,9 %*	*17,9 %*

TABLE 4.40 – Détection des fautes indirectes dans les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	19,3 %	18,5 %	18,1 %	34,8 %	35,6 %	28,8 %
Ant 1.4	8,5 %	10,6 %	46,0 %	34,0 %	36,2 %	83,4 %
Ant 1.5	17,1 %	17,1 %	79,5 %	37,1 %	40,0 %	91,7 %
Ant 1.6	20,1 %	14,7 %	59,7 %	56,5 %	56,5 %	84,6 %
Ant 1.7	18,9 %	17,8 %	60,0 %	45,0 %	45,0 %	81,0 %
Camel 1.2	37,7 %	41,6 %	53,2 %	58,0 %	58,8 %	69,7 %
Camel 1.4	35,5 %	23,3 %	68,9 %	59,4 %	56,7 %	83,9 %
Camel 1.6	31,0 %	31,6 %	65,3 %	65,0 %	62,4 %	83,1 %
JEdit 3.2.1	19,4 %	17,5 %	36,6 %	47,6 %	56,0 %	60,8 %
JEdit 4.0	20,8%	13,3%	67,5%	44,7%	47,8%	78,7%
JEdit 4.1	16,6 %	14,3 %	70,4 %	46,5 %	49,8 %	80,0 %
JEdit 4.2	15,1 %	18,9 %	87,4 %	45,3 %	43,4 %	88,7 %
POI 1.5	43,0%	28,1%	27,4%	67,0%	68,7%	65,5%
POI 2.0	33,3 %	38,5 %	82,2 %	56,4 %	53,8 %	88,6 %
POI 2.5.1	62,1 %	61,7 %	62,9 %	78,2 %	79,2 %	77,6 %
POI 3.0	50,8 %	49,4 %	51,0 %	72,4 %	70,2 %	75,0 %
Xalan 2.4	30,1 %	32,1 %	68,0 %	55,1 %	53,8 %	86,9 %
Xalan 2.5	33,7 %	29,7 %	48,5 %	52,1 %	52,1 %	62,8 %
Xalan 2.6	32,2 %	26,6 %	46,6 %	48,9 %	47,0 %	57,7 %
Xalan 2.7	39,4 %	34,8 %	36,5 %	47,7 %	48,0 %	48,2 %
Moyenne	*29,2 %*	*27,0 %*	*56,8 %*	*52,6 %*	*53,1 %*	*73,8 %*

Efficience de la détection

Pour l'efficience par rapport à la détection totale des fautes (tableau 4.41), les approches inter-SL offrent une meilleure efficience que les approches de contrôle gloutonnes. Elles détectent en moyenne environ 6 fautes par 1000 lignes de code testées. Toutefois, cet indicateur est très variable selon les systèmes, car il dépend du nombre de fautes que contient le système. En effectuant une comparaison système par système, les approches inter-SL sont généralement plus efficientes dans la détection. Cet avantage est léger dans certains cas (Ant1.3 et 1.5 par exemple) ou très prononcé pour d'autres systèmes (POI 1.5 ou Xalan 2.7).

Pour l'efficience directe seulement (tableau 4.42), les approches offrent des résultats comparables les uns aux autres, non seulement en moyenne, mais aussi pour chaque système. L'approche inter-SL offre une efficience directe légèrement inférieure à celle de l'approche de contrôle gloutonne. Au niveau de l'efficience indirecte (tableau 4.43), l'approche inter-SL produit des résultats nettement supérieurs à ceux de l'approche de contrôle gloutonne, illustrant de nouveau la propension aux approches SL de tirer profit de la couverture indirecte. L'efficience est plus grande pour l'algorithme KNN + RF, différence explicable par le fait qu'inter-SL (KNN + RF) détecte la vaste majorité des fautes de façon indirecte (en moyenne 73 % des fautes). Cette différence n'intervient toutefois pas dans l'efficience des approches gloutonnes associées.

Le tableau 4.44 présente le ratio entre l'efficience directe et indirecte. Pour les approches inter-SL, le ratio est en faveur de l'efficience indirecte (ratio inférieur à 1), plus de fautes indirectes sont détectées par lignes de code testées. Pour les approches de comparaison gloutonnes, le ratio est du côté de l'efficience directe (ratio supérieur à 1), illustrant que l'approche inter-SL tire généralement profit de l'efficience indirecte. Même lorsque le ratio pour inter-SL est supérieur à 1, comme dans le cas du système Ant, les approches inter-GL-risque admettent des ratios pour la plupart largement supérieurs, confirmant le fait que les approches inter-SL misent sur la détection indirecte.

TABLE 4.41 – Nombre de fautes détectées directement et indirectement par millier de lignes de code pour les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	5,54	6,64	6,02	6,46	6,80	6,37
Ant 1.4	0,80	0,89	0,95	1,50	1,72	1,40
Ant 1.5	0,75	0,70	0,76	0,92	0,99	0,95
Ant 1.6	2,69	2,61	2,61	3,41	3,62	3,54
Ant 1.7	2,29	2,30	2,33	3,67	4,00	3,35
Camel 1.2	7,34	7,41	7,52	8,76	8,59	8,38
Camel 1.4	3,82	3,63	3,22	4,16	4,22	3,92
Camel 1.6	4,45	4,09	4,39	5,39	5,77	4,92
JEdit 3.2.1	4,27	3,68	4,48	5,32	5,61	6,11
JEdit 4.0	2,24	2,46	1,94	2,75	3,56	3,45
JEdit 4.1	2,04	1,73	2,05	2,50	2,72	2,71
JEdit 4.2	0,93	1,07	0,90	1,24	1,26	1,70
POI 1.5	12,18	8,89	9,14	24,15	24,30	34,31
POI 2.0	0,80	0,69	0,82	2,26	1,11	1,22
POI 2.5.1	10,83	9,77	10,03	10,52	11,29	11,19
POI 3.0	9,96	8,46	8,80	9,99	10,49	14,65
Xalan 2.4	1,46	1,26	1,34	1,94	1,94	1,94
Xalan 2.5	3,49	3,06	4,09	5,17	5,94	5,49
Xalan 2.6	2,94	3,13	3,34	5,55	5,39	5,68
Xalan 2.7	5,35	4,95	5,55	9,27	9,29	7,98
Moyenne	*4,21*	*3,87*	*4,01*	*5,75*	*5,93*	*6,46*

TABLE 4.42 – Nombre de fautes détectées directement par millier de lignes de code pour les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	3,42	4.54	3,86	4,18	4,31	4,18
Ant 1.4	0,65	0,70	0,72	0,93	1,02	0,86
Ant 1.5	0,60	0,56	0,63	0,57	0,58	0,62
Ant 1.6	2,04	2,16	2,06	1,44	1,52	1,36
Ant 1.7	1,71	1,79	1,84	1,93	2,06	1,84
Camel 1.2	4,13	3,75	3,75	3,42	3,27	3,28
Camel 1.4	2,22	2,66	2,20	1,53	1,65	1,48
Camel 1.6	2,95	2,64	2,87	1,70	1,98	1,75
JEdit 3.2.1	3,43	3,00	3,68	2,77	2,34	3,07
JEdit 4.0	1,75	2,11	1,55	1,52	1,79	1,67
JEdit 4.1	1,68	1,46	1,66	1,30	1,32	1,39
JEdit 4.2	0,79	0,86	0,80	0,67	0,71	0,89
POI 1.5	5,99	5,01	4,78	6,59	6,78	8,10
POI 2.0	0,47	0,32	0,46	0,80	0,44	0,48
POI 2.5.1	3,43	3,31	3,00	1,85	2,08	2,05
POI 3.0	4,09	3,88	3,99	2,43	2,87	3,17
Xalan 2.4	0,89	0,78	0,86	0,77	0,80	0,79
Xalan 2.5	1,80	1,60	2,11	1,57	1,74	1,72
Xalan 2.6	1,53	1,90	1,82	1,69	1,69	1,76
Xalan 2.7	1,95	2,01	2,14	2,04	2,07	2,02
Moyenne	*2,28*	*2,25*	*2,24*	*1,99*	*2,05*	*2,12*

TABLE 4.43 – Nombre de fautes détectées indirectement par millier de lignes de code pour les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	2,12	2,10	2,16	2,28	2,49	2,19
Ant 1.4	0,15	0,18	0,23	0,57	0,69	0,54
Ant 1.5	0,15	0,14	0,13	0,35	0,41	0,34
Ant 1.6	0,64	0,45	0,55	1,97	2,11	2,18
Ant 1.7	0,58	0,52	0,49	1,74	1,94	1,50
Camel 1.2	3,21	3,66	3,77	5,34	5,32	5,10
Camel 1.4	1,61	0,97	1,02	2,63	2,57	2,43
Camel 1.6	1,50	1,45	1,52	3,69	3,79	3,17
JEdit 3.2.1	0,84	0,68	0,79	2,55	3,27	3,05
JEdit 4.0	0,48	0,34	0,39	1,23	1,77	1,78
JEdit 4.1	0,37	0,27	0,39	1,20	1,41	1,32
JEdit 4.2	0,14	0,20	0,09	0.57	0,55	0,81
POI 1.5	6,20	3,88	4,36	17,55	17,52	26,20
POI 2.0	0,34	0,37	0,36	1,46	0,67	0,75
POI 2.5.1	7,40	6,45	7,03	8,67	9,20	9,14
POI 3.0	5,87	4,58	4,80	7,57	7,63	11,48
Xalan 2.4	0,57	0,47	0,48	1,17	1,14	1,15
Xalan 2.5	1,69	1,47	1,98	3,60	4,20	3,77
Xalan 2.6	1,41	1,23	1,51	3,85	3,70	3,92
Xalan 2.7	3,40	2,94	3,41	7,23	7,22	5,95
Moyenne	*1,93*	*1,62*	*1,77*	*3,76*	*3,88*	*4,34*

TABLE 4.44 – Ratio de l'efficience directe par rapport à l'efficience indirecte pour les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	1,62	2,16	1,79	1,83	1,73	1,91
Ant 1.4	4.25	3,80	3,17	1,63	1,47	1,59
Ant 1.5	4,00	4,00	5,00	1,62	1,43	1,83
Ant 1.6	3,19	4,81	3,75	0,73	0,72	0,62
Ant 1.7	2,97	3,47	3,75	1,11	1,06	1,23
Camel 1.2	1,29	1,02	1,00	0,64	0,62	0,64
Camel 1.4	1,38	2,74	2,15	0,58	0,64	0,61
Camel 1.6	1,97	1,82	1,89	0,46	0,52	0,55
JEdit 3.2.1	4,08	4,43	4,65	1,09	0,71	1,01
JEdit 4.0	3,64	6,17	3,93	1,24	1,01	0,94
JEdit 4.1	4,58	5,45	4,25	1,08	0,94	1,06
JEdit 4.2	5,56	4,25	8,55	1,17	1,28	1,10
POI 1.5	0,97	1,29	1,10	0,38	0,39	0,31
POI 2.0	1,38	0,87	1,27	0,55	0,67	0,64
POI 2.5.1	0,46	0,51	0,43	0,21	0,23	0,22
POI 3.0	0,70	0,85	0,83	0,32	0,38	0,28
Xalan 2.4	1,55	1,66	1,81	0,66	0,70	0,69
Xalan 2.5	1,06	1,09	1,06	0,44	0,42	0,45
Xalan 2.6	1,08	1,55	1,20	0,44	0,46	0,45
Xalan 2.7	0,57	0,68	0,63	0,28	0,29	0,34
Moyenne	*2,32*	*2,63*	*2,61*	*0,82*	*0,78*	*0,82*

Qualité de l'ordonnancement

Les APFD des approches sont présentées au tableau 4.45. En général, les valeurs d'APFD sont assez constantes pour l'approche inter-SL, et ce, pour tous les algorithmes d'apprentissage. Elles s'établissent généralement entre 58 % et 72 % pour une moyenne de 66 %. Les plus basses performances sont observées sur le système Xalan pour notre approche. Pour l'approche de contrôle gloutonne, le système Xalan a lui aussi de moins hauts scores d'APFD. Pour l'approche de contrôle, les APFD sont plus variantes (généralement entre 53 % et 77 %), mais ont une moyenne près de 65 %. L'approche inter-SL semble produire des résultats parfois moins bons (en moyenne tout aussi performant), mais plus stables, donc moins sujets aux variations entre les systèmes. Cette plus grande stabilité indique des résultats d'une qualité plus prévisible et donc plus fiable, car ils semblent peu influencés par le système. Ils peuvent indiquer que les résulats pourraient se généraliser à d'autres systèmes, bien que davantage d'expérimentations soient nécessaires pour le confirmer. À ce stade, cela représente déjà un avantage pour l'algorithme inter-SL (KNN + RF) même si les valeurs sont comparables aux autres approches inter-SL.

Tout comme les évaluations de l'APFD précédentes, nous avons testé la dépendance de l'APFD à la taille et au nombre de fautes détectées directement des systèmes, mesures qui peuvent influencer les résultats. L'évaluation de la dépendance s'est faite à l'aide de la régression linéaire suivante :

$$\begin{aligned} APFD = b_0 + b_1 \times \text{ taille ensemble candidat} + \\ b_2 \times \text{ nombre de fautes détectées directement} \end{aligned} \tag{4.4}$$

Dans tous les cas, tant les tests globaux que marginaux n'ont révélé aucune association entre l'APFD et la taille de l'ensemble candidat ou le nombre de fautes détectées directement.

TABLE 4.45 – APFD pour les approches inter-systèmes

Système	Inter-GL-risque (KNN)	Inter-GL-risque (RF)	Inter-GL-risque (KNN + RF)	Inter-SL (KNN)	Inter-SL (RF)	Inter-SL (KNN + RF)
Ant 1.3	58,2 %	62,1%	47,9 %	65,3 %	61,5 %	63,2 %
Ant 1.4	72,3 %	59,1%	60,4 %	62,5 %	56,4 %	67,7 %
Ant 1.5	65,4 %	62,1%	56,5 %	71,5 %	70,3 %	67,5 %
Ant 1.6	53,2 %	59,2%	52,4 %	65,2 %	64,8 %	64,3 %
Ant 1.7	56,1 %	62,9%	51,3 %	65,0 %	61,8 %	70,2 %
Camel 1.2	63,3 %	67,4%	67,1 %	67,1 %	65,7 %	65,2 %
Camel 1.4	65,0%	69,1%	67,6%	63,5%	64,4%	64,7%
Camel 1.6	57,1%	64,5%	64,9%	69,0%	65,7%	65,8%
JEdit 3.2.1	75,5 %	76,0%	76,6 %	73,1 %	79,6 %	75,4 %
JEdit 4.0	79,3%	79,2%	79,9%	79,9%	66,4%	60,2%
JEdit 4.1	77,8 %	78,0%	76,3 %	75,2 %	67,5 %	72,5 %
JEdit 4.2	80,3 %	75,5%	76,8 %	81,6 %	82,0 %	72,7 %
POI 1.5	62,9%	63,2%	69,0%	68,3%	67,2%	69,8%
POI 2.0	65,5%	81,6%	66,1%	53,0%	56,1%	63,7%
POI 2.5.1	62,9 %	63,0%	64,3 %	64,7 %	63,0 %	66,0 %
POI 3.0	74,6 %	68,8%	68,9 %	70,9 %	71,2 %	73,3 %
Xalan 2.4	58,0 %	64,5%	62,5 %	64,1 %	62,6 %	65,0 %
Xalan 2.5	60,1 %	58,7%	57,2 %	64,4 %	62,8 %	62,0 %
Xalan 2.6	60,3 %	59,8%	56,3 %	60,9 %	60,6 %	58,6 %
Xalan 2.7	51,4 %	52,8%	49,9 %	54,6 %	53,5 %	55,0 %
Moyenne	*65,0 %*	*66,4%*	*63,6 %*	*67,0 %*	*65,2 %*	*66,2 %*

4.4.2 Comparaison des approches de priorisation

La présente section compare les différentes approches inter-SL développées puis identifie celle qui offre les meilleurs résultats. La comparaison est basée sur la moyenne des indicateurs détaillés aux sections précédentes ainsi que sur leur variance. Le test de Student est utilisé pour comparer les moyennes et celui de Fisher les variances. L'approche de type inter-SL identifiée comme offrant les meilleurs résultats est ensuite comparée aux approches de contrôle afin d'évaluer l'amélioration apportée par l'approche inter-SL. Chaque paire d'approches est comparée et la p-valeur est indiquée dans la colonne éponyme. Dans tous les tableaux de cette section, les valeurs en caractère gras indiquent un test significatif au seuil de 5 %.

Comparaison des approches développées

Le tableau 4.46 compare les approches inter-SL selon l'algorithme utilisé pour obtenir la probabilité qu'une classe soit fautive. Pour les métriques d'évaluation de couverture, les trois approches inter-SL ont des performances similaires. Pour la détection des fautes, le nombre total de fautes détectées est similaire, mais l'approche combinant KNN et RF détecte en moyenne significativement moins de fautes directement (-23,1 % par rapport à inter-SL (KNN) et -22,1 % par rapport à interSL (RF)) et significativement plus de fautes indirectement que les deux autres approches (+23,3 % par rapport à inter-SL (KNN) et + 22,5 % par rapport à inter-SL (RF)). L'utilisation de KNN seul ou RF seul mène au même résultat.

Nous avons éliminé le biais de sélection des données d'entraînement en répliquant quatre fois l'apprentissage par inter-SL (KNN + RF) afin de valider que le faible taux de détection direct, différent des autres variantes de l'approche inter-SL, n'est pas seulement dû à la sélection des données d'entraînement. Nous avons obtenu dans toutes les réplications de l'entraînement (que nous ne présentons pas par soucis d'espace) des résultats similaires à ceux présentés au tableau 4.46. Notons toutefois une différence : les performances varient entre les systèmes selon la sélection. Autrement dit, dans le premier apprentissage la prédiction directe peut être bonne pour un système X, mais n'était pas bonne lors du second apprentissage. En moyenne, toutefois, les résultats varient peu.

Pour l'efficience, les trois approches offrent des performances similaires. L'approche inter-SL (KNN) semble légèrement moins efficiente, tant directement qu'indirectement que les deux autres approches, mais cette différence n'est pas statistiquement significative. L'approche avec l'algorithme RF repose légèrement plus sur l'efficience indirecte que les deux autres approches. Les valeurs d'APFD sont similaires entre les 3 algorithmes, mais inter-SL (KNN) a une APFD légèrement supérieure.

Pour la dispersion, seules les métriques d'évaluation dont les échelles sont comparables ont été évaluées. Aucun algorithme ne se distingue par une dispersion plus ou moins grande que ses voisins. En général, inter-SL (RF) offre une dispersion plus faible des résultats indiquant une plus grande constance des données obtenues. Toutefois, cet écart n'est pas statistiquement significatif.

La meilleure approche est inter-SL (KNN), car elle fournit une légère amélioration de la couverture atteinte. Elle demande un effort de test légèrement plus grand que les autres approches et n'est pas aussi efficiente que les autres approches. Cependant, l'approche utilisant l'algorithme KNN + RF repose trop sur la détection indirecte et manque de précision (comme mentionné précédemment, ce manque de précision n'est pas dû à la sélection des données d'entraînement). L'approche avec KNN est également celle qui a la meilleure détection directe des fautes. Finalement, sa valeur d'AFPD est légèrement supérieure à celles des autres approches. Le seul point plus négatif à sélectionner inter-SL (KNN) est qu'il s'agit de l'algorithme pour lequel les résultats varient le plus entre les systèmes. Ceci est vrai en particulier pour la couverture directe, la taille de l'ensemble de test et le ratio d'efficience direct/indirect où des différences significatives avec les autres algorithmes ont été opbservées.

L'annexe B présente les résultats sous forme de graphique (figure B.3) les métriques d'évaluation liées aux cinq axes des questions de recherche.

Comparaison aux approches de contrôle

L'approche inter-SL (KNN) est celle qui offre les meilleurs résultats lorsque nous comparons la valeur de ses métriques d'évaluation des trois approches inter-SL. Inter-SL (KNN) est comparée dans cette section aux approches de contrôle sans variantes présentées à la section 4.2 et à l'approche inter-GL-risque (KNN) qui est l'approche

TABLE 4.46 – Comparaison des approches développées inter-systèmes par métrique d'évaluation

	Moyennes			p-valeur moyennes		
Métrique d'évaluation	**Inter-SL (KNN)**	**Inter-SL (RF)**	**Inter-SL (KNN + RF)**	KNN / RF	KNN / KNN + RF	RF / KNN + RF
Couverture totale (LOC)	86,3%	85,3%	85,9%	0,118	0,351	0,295
Couverture directe (LOC)	35,8%	34,3%	34,9%	0,097	0,252	0,332
Couverture indirecte (LOC)	50,5%	51,0%	51,0%	0,095	0,129	0,495
Taille ensemble candidat	103	92	105	**0,005**	0,360	**0,023**
Fautes détectées total	95,4%	95,2%	95,6%	0,240	0,174	0,076
Fautes détectées direct	42,4%	41,4%	19,3%	0,191	**< 0,001**	**< 0,001**
Fautes détectées indirect	53,0%	53,8%	76,3%	0,224	**< 0,001**	**< 0,001**
Efficience détection total	5,75	5,93	6,46	**0,026**	0,101	0,168
Efficience détection direct	1,99	2,05	2,12	0,073	0,054	0,185
Efficience détection indirect	3,76	3,88	4,34	0,052	0,114	0,171
Ratio efficience directe/indirecte	0,82	0,78	0,82	0,076	0,478	0,096
APFD	68,5%	66,5%	67,6%	**0,023**	0,271	0,183
	Écart-types			**p-valeur variances**		
Métrique d'évaluation	**Inter-SL (KNN)**	**Inter-SL (RF)**	**Inter-SL (KNN + RF)**	KNN / RF	KNN / KNN + RF	RF / KNN + RF
Couverture totale (LOC)	13,3%	13,0%	12,9%	0,785	0,780	0,926
Couverture directe (LOC)	11,1%	8,6%	10,4%	**0,028**	0,661	0,140
Couverture indirecte (LOC)	9,7%	9,2%	10,1%	0,218	0,427	**0,035**
Taille ensemble candidat	50,73	47,09	68,58	0,411	**0,003**	**0,000**
Fautes détectées total	10,4%	10,4%	9,6%	0,913	0,164	0,203
Fautes détectées direct	15,4%	13,7%	14,2%	0,045	0,706	0,894
Fautes détectées indirect	11,9%	11,2%	15,9%	0,294	0,227	0,150
Ratio efficience directe/indirecte	0,82	0,78	0,82	**0,022**	0,563	**0,012**
APFD	7,2%	7,0%	5,2%	0,842	0,108	0,082

de contrôle gloutonne utilisant le même algorithme de prédiction des fautes que l'approche inter-SL (KNN). Du tableau 4.47 au tableau 4.51, les valeurs en caractères gras montrent une différence significative au seuil de 5 % entre la valeur et celle de l'approche inter-SL . Les différences sont testées seulement pour le total, car en raison du faible nombre de versions pour chaque système, les tests risqueraient d'avoir une faible puissance.

Pour la couverture (tableau 4.47), l'approche inter-SL [1] offre une couverture totale significativement plus importante que l'approche aléatoire (+ 40,4 %), inter-GL-risque (+ 13,8 %), la descente de gradient (+ 33,6 %) et l'algorithme génétique (+ 25,3 %). Pour les autres approches, la couverture totale est semblable. Au niveau de la couverture directe, l'approche inter-SL offre une couverture directe significativement plus importante que l'approche aléatoire (+ 21,8 %), la descente de gradient (+ 16,4 %) et l'algorithme génétique (+ 23,6 %). L'approche inter-SL offre toutefois une couverture directe significativement plus petite que les deux approches de contrôle gloutonnes (couverture et complexité par environ 23 %) et la classification hiérarchique (- 21,9 %), ce qui signifie une réduction de l'effort de test par rapport à ces approches. Pour toutes les approches, sauf l'algorithme génétique, la couverture indirecte d'inter-SL est significativement plus importante, par environ 23 %, que celle des approches de contrôle. Ceci indique clairement que l'approche inter-SL mise sur la couverture indirecte pour atteindre une bonne priorisation et une bonne détection des fautes.

Pour la taille des ensembles candidats (tableau 4.48), on remarque que selon les systèmes, la taille de l'ensemble candidat est plus ou moins élevée que la taille utilisée pour toutes les approches de contrôle (Référence 15 %). Pour les systèmes Ant et Camel, l'approche inter-SL tend à produire des ensembles de classes candidates au test de taille plus élevée que la référence de 15 %, tandis qu'elle est similaire pour JEdit et inférieure pour POI et Xalan.

Le tableau 4.49 montre les différences dans la détection des fautes entre les approches de contrôle et inter-SL. Dans tous les cas, l'approche inter-SL détecte significativement plus de fautes que les approches de contrôle (entre 10 % et 40 %). Pour la détection de fautes directes, les différences observées sont les mêmes que pour la

1. Nous omettrons la référence à l'algorithme dans cette section pour alléger le texte, mais nous faisons toujours référence, sauf indication contraire à inter-SL (KNN).

TABLE 4.47 – Comparaison de la couverture d'inter-SL (KNN) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Inter-GL-risque (KNN)	DG-couv	CBGA-ES	CH	Inter-SL (KNN+RF)
Couverture totale (LOC)	Ant	44,7 %	73,7 %	69,1 %	64,0 %	48,6 %	53,4 %	71,4 %	91,8 %
	Camel	48,4 %	80,0 %	78,7 %	68,7 %	49,4 %	51,8 %	78,7 %	89,8 %
	JEdit	55,2 %	92,3 %	89,7 %	90,1 %	69,7 %	71,0 %	89,7 %	92,7 %
	POI	35,5%	91,7%	93,4%	82,9%	53,0%	79,2%	93,4%	92,0%
	Xalan	46,7 %	87,1 %	81,4 %	58,0 %	43,2 %	49,0 %	81,4 %	64,9 %
	Total	***45,9 %***	*84,7 %*	*82,0 %*	***72,5 %***	***52,7 %***	***61,0 %***	*82,6 %*	*86,3 %*
Couverture directe (LOC)	Ant	17,3 %	55,7 %	53,0 %	42,2 %	17,1 %	11,1 %	53,0 %	43,7 %
	Camel	14,4 %	55,8 %	53,9 %	39,1 %	16,1 %	12,6 %	54,0 %	36,3 %
	JEdit	12,6 %	71,9 %	65,6 %	57,2 %	24,7 %	11,2 %	65,7 %	46,5 %
	POI	13,3%	58,4%	54,7%	36,5%	22,8%	14,3%	54,9%	27,3%
	Xalan	15,6 %	66,5 %	61,1 %	34,0 %	15,9 %	12,4 %	61,2 %	23,6 %
	Total	***14,8 %***	***61,7 %***	***57,7 %***	*42,0 %*	***19,4 %***	***12,2 %***	***57,7 %***	*35,8 %*
Couverture indirecte (LOC)	Ant	27,4 %	18,0 %	16,1 %	21,8 %	31,5 %	42,3 %	18,4 %	48,1 %
	Camel	34,0 %	24,2 %	24,7 %	29,6 %	33,2 %	39,3 %	24,8 %	53,5 %
	JEdit	42,6 %	20,4 %	24,1 %	32,9 %	44,9 %	59,7 %	24,1 %	46,2 %
	POI	22,1%	33,3%	38,6%	46,5%	30,2%	64,9%	38,5%	64,7%
	Xalan	31,2 %	20,6 %	20,3 %	24,0 %	27,3 %	36,6 %	20,2 %	41,3 %
	Total	***31,1 %***	***23,0 %***	***24,3 %***	***30,6 %***	***33,3 %***	*48,7 %*	***24,9 %***	*50,5 %*

TABLE 4.48 – Comparaison de la taille de l'ensemble de classes candidates au test de l'approche en contexte inter-systèmes aux approches de contrôle

Métrique d'évaluation	Système	Référence 15 %	Inter-SL (KNN)
Taille de l'ensemble candidat aux tests	Ant	71	115
	Camel	157	178
	JEdit	96	89
	POI	63	44
	Xalan	153	105
	Total	*104*	*103*

couverture directe présentée au tableau 4.47 ; l'approche aléatoire, de descente de gradient et l'algorithme génétique détectent significativement moins de fautes directes qu'inter-SL, tandis que les trois approches gloutonnes et la classification hiérarchique détectent significativement plus de fautes directes qu'inter-SL. Pour la détection indirecte des fautes, la tendance est également la même qu'avec la couverture indirecte (tableau 4.47) ; inter-SL détecte indirectement plus de fautes que toutes les approches sauf pour l'algorithme génétique. Cette tendance à obtenir les mêmes différences pour la couverture et la détection des fautes est la même que celle observée avec intra-SL (KNN + RF). Ce résultat contribue à montrer que l'approche SL, globalement, détecte plus de fautes que les approches de contrôle et utilise la couverture indirecte adéquatement pour y parvenir. Il s'agit de notre explication la plus plausible, car dans les deux cas, les résultats similaires ont été obtenus à partir de méthodologies différentes sur 2 aspects : sélection des ensembles d'entraînement (intra et inter) et algorithmes d'apprentissage automatique pour la prédiction des fautes (KNN + RF et KNN seul).

TABLE 4.49 – Comparaison de la détection des fautes d'inter-SL (KNN) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Inter-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Inter-SL (KNN+RF)
Détection de fautes total	Ant	38,9 %	71,4 %	66,7 %	68,0 %	47,5 %	53,6 %	68,6 %	95,6 %
	Camel	57,2 %	84,8 %	84,6 %	87,6 %	61,0 %	59,3 %	84,6 %	94,7 %
	JEdit	73,9 %	95,5 %	95,2 %	96,6 %	75,2 %	79,9 %	95,2 %	98,6 %
	POI	35,6%	83,6%	90,3%	85,3%	50,2%	73,0%	90,3%	92,5%
	Xalan	49,9 %	74,7 %	70,5 %	68,9 %	47,6 %	53,5 %	70,5 %	74,5 %
	Total	***50,2 %***	***81,4 %***	***80,6 %***	***80,3 %***	***55,6 %***	***63,6 %***	***81,1 %***	*91,2 %*
Détection de fautes directes	Ant	14,2 %	56,5 %	53,5 %	51,2 %	14,0 %	13,9 %	52,3 %	54,1 %
	Camel	12,4 %	47,9 %	52,9 %	52,9 %	16,7 %	12,9 %	52,9 %	33,9 %
	JEdit	11,0 %	79,8 %	76,5 %	78,7 %	13,0 %	14,0 %	76,6 %	52,6 %
	POI	12,4%	38,1%	38,4%	38,0%	17,2%	17,9%	38,5%	24,0%
	Xalan	15,9 %	38,7 %	37,4 %	35,0 %	14,2 %	13,7 %	37,4 %	23,6 %
	Total	***13,3 %***	***52,6 %***	***51,8 %***	***51,1 %***	***14,9 %***	***14,5 %***	***51,5 %***	*38,6 %*
Détection de fautes indirectes	Ant	24,7 %	14,9 %	13,2 %	16,8 %	33,6 %	39,7 %	16,3 %	41,5 %
	Camel	44,8 %	36,9 %	31,7 %	34,8 %	44,3 %	46,5 %	31,7 %	60,8 %
	JEdit	62,9 %	15,7 %	18,7 %	18,0 %	62,2 %	66,0 %	18,6 %	46,0 %
	POI	23,1%	45,5%	51,9%	47,3%	33,1%	55,1%	51,8%	68,5%
	Xalan	34,1 %	36,1 %	33,2 %	33,9 %	33,5 %	39,8 %	33,1 %	51,0 %
	Total	***36,9 %***	***28,7 %***	***28,8 %***	***29,2 %***	***40,8 %***	*49,1 %*	***29,6 %***	*52,6 %*

Le tableau 4.50 montre les différences entre les taux d'efficience des algorithmes. Pour l'efficience totale, l'approche inter-SL a une meilleure efficience que l'approche gloutonne basée sur la couverture (+ 2,92 fautes par millier de lignes de code couvertes), celle basée sur la complexité (+ 2,74) et la classification hiérarchique (+ 2,70). Cependant, l'algorithme génétique est plus efficient qu'inter-SL (- 4,76 fautes par millier de lignes de code couvertes). Les efficiences directes de toutes les approches

sont très similaires, de même que les efficiences indirectes. L'approche inter-SL est significativement plus efficiente indirectement que les trois approches gloutonnes et l'algorithme de classification, mais significativement moins efficiente que l'algorithme génétique. Notons toutefois que l'algorithme génétique trouve moins de fautes que l'approche inter-SL (par 27,6 % comme présenté au tableau 4.49).

TABLE 4.50 – Comparaison de l'efficience de la détection des fautes d'inter-SL (KNN) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Inter-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Inter-SL (KNN+RF)
Efficience détection totale	Ant	3,69	1,94	1,89	2,41	4,00	7,71	2,09	3,19
	Camel	9,39	3,61	3,73	5,21	8,61	10,80	3,73	6,10
	JEdit	9,97	1,89	2,08	2,37	4,11	9,81	2,08	2,95
	POI	10,36	5,32	5,99	8,44	8,27	18,10	5,95	11,73
	Xalan	5,35	1,78	1,82	3,31	4,60	6,90	1,82	5,48
	Total	*7,47*	***2,83***	***3,01***	*4,21*	*5,69*	***10,51***	***3,05***	*5,75*
Efficience détection directe	Ant	1,34	1,50	1,55	1,68	1,15	2,02	1,46	1,81
	Camel	2,36	1,96	2,23	3,10	2,27	2,35	2,23	2,22
	JEdit	1,23	1,55	1,61	1,91	0,72	1,90	1,61	1,57
	POI	4,21	2,30	2,32	3,49	2,97	4,16	2,32	2,92
	Xalan	1,72	0,86	0,91	1,54	1,50	2,00	0,91	1,52
	Total	*2,12*	*1,61*	*1,69*	*2,28*	*1,67*	*2,47*	*1,67*	*1,99*
Efficience détection indirecte	Ant	2,35	0,44	0,35	0,73	2,84	5,69	0,63	1,38
	Camel	7,03	1,65	1,50	2,11	6,34	8,44	1,50	3,89
	JEdit	8,74	0,33	0,47	0,46	3,39	7,91	0,47	1,39
	POI	6,15	3,02	3,66	4,95	5,30	13,94	3,63	8,81
	Xalan	3,63	0,92	0,91	1,77	3,09	4,90	0,91	3,96
	Total	*5,34*	***1,21***	***1,32***	***1,93***	*4,02*	***8,04***	***1,38***	*3,76*

Pour l'APFD (tableau 4.51), l'approche inter-SL (KNN) offre un ordonnancement significativement supérieur à toutes les approches de contrôle, par en moyenne 11,3 %, sauf pour inter-GL-risque où les résultats sont similaires (différence de seulement 2,0 %). L'utilisation de la probabilité d'erreur est donc un facteur qui semble permettre une bonne priorisation des classes candidates au test. L'approche inter-SL améliore, mais de façon non significative, l'ordonnancement des classes candidates par rapport à l'approche gloutonne sur le risque.

TABLE 4.51 – Comparaison de l'APFD d'inter-SL (KNN) aux approches de contrôle

Métrique d'évaluation	Système	ALEA-couv	GL-couv	GL-comp	Inter-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH	Inter-SL (KNN+RF)
APFD	Ant	45,4 %	54,7 %	52,6 %	61,0 %	49,4 %	48,7 %	54,6 %	65,9 %
	Camel	53,8 %	66,4 %	66,3 %	61,8 %	50,4 %	53,3 %	66,4 %	66,5 %
	JEdit	48,2 %	71,4 %	70,8 %	78,2 %	40,8 %	48,0 %	70,8 %	77,4 %
	POI	54,6%	62,2%	63,1%	66,5%	52,0%	52,4%	63,1%	64,2%
	Xalan	45,6 %	57,5 %	57,0 %	57,5 %	53,3 %	53,1 %	56,9 %	61,0 %
	Total	***49,1 %***	***61,9 %***	***61,3 %***	*65,0 %*	***49,1 %***	***50,9 %***	***61,8 %***	*67,0 %*

4.5 Caractéristiques influençant notre approche de priorisation

Certaines caractéristiques des systèmes ont une influence sur les résultats de l'approche SL. Nous avons calculé la corrélation entre la taille du système et les valeurs des métriques orientées objets et des mesures de centralité employées, ainsi qu'avec le niveau de couplage moyen. On observe à partir du tableau 4.52 que pour la plupart des systèmes, une augmentation de la taille entraîne un changement consistant dans les caractéristiques, à l'exception notable du système Xalan. Nous avons identifié, dans le tableau 4.52, en gras, les valeurs pour lesquelles la différence est majeure avec la tendance générale. En raison de la faible taille des échantillons, nous ne pouvons pas effectuer un test statistique pour mesurer la différence. Nous pouvons seulement qualifier la différence.

TABLE 4.52 – Corrélation entre la taille des systèmes en nombre de classes et leurs caractéristiques

Métrique	Ant	Camel	Jedit	POI	Xalan
CE	**-0,31**	0,98	0,96	0,96	**-1**
CL	0,95	0,99	0,99	0,98	**-0,98**
KATZ	-0,82	-1	**0,47**	-0,97	-1
RFC	**-0,41**	**1**	**-0,77**	**0,87**	**-0,99**
SLC	0,99	0,98	1	0,98	**-0,92**
Couplage moyen	-0,64	-0,65	-0,8	-0,69	**0,99**

On retrouve que le système Xalan a une tendance à varier différemment des autres systèmes : plus sa taille augmente, plus la valeur moyenne de ses métriques diminue alors qu'elle augmente généralement pour les autres systèmes. Nous attribuons à la tendance manifestée par RFC un caractère hétérogène (ne suit pas une tendance constante pour tous les systèmes), car nous n'arrivons pas à dégager une tendance générale de la corrélation de cette métrique avec la taille des systèmes. En effet, selon le système, le coefficient de corrélation varie de -0.99 (Xalan) à 1.00 (Camel).

La variation différente des métriques et mesures avec la taille peut expliquer pourquoi certaines prédictions utilisant Xalan donnent des résultats moins précis qu'avec les autres systèmes. Le couplage moyen est également une caractéristique importante de l'approche SL, car plus le couplage moyen est élevé plus une classe moyenne a de voisins et par conséquent, cela cause une diminution plus rapide des scores, car chaque

sélection a un impact sur davantage de classes. Cela peut expliquer la diminution de la taille de l'ensemble de classes candidates remarquées au tableau 4.17 entre les versions successives de Xalan. La sélection de moins de classes candidates peut également expliquer les variations observées dans la couverture inter-SL présentées du tableau 4.33 au tableau 4.35.

4.6 Résumé des comparaisons aux approches de contrôle

Le tableau 4.53 résume les différences significatives trouvées entre l'approche développée et les approches de contrôle dans le contexte intra-système. Le symbole plus « + » indique que la valeur du critère augmente significativement pour l'approche développée et le moins « - » indique une réduction significative de la valeur.

L'approche intra-SL améliore significativement au moins trois critères par rapport aux approches de contrôle, sauf pour l'approche intra-GL-risque, qui est, du point de vue des données, sûrement l'approche la plus près de l'approche intra-SL. Pour quatre des approches de contrôle, aucun critère ne s'est détérioré par rapport à l'approche intra-SL. L'approche intra-SL offre une couverture significativement plus importante que quatre des approches de contrôle. Dans les autres cas, l'amélioration est présente, mais non significative. La couverture totale est utilisée pour parvenir à ce constat. L'effort de test (couverture directe) pour sa part est plus grand pour l'approche intra-SL que trois des approches de contrôle, ce qui implique la détérioration significative de ce critère. Le critère effort de test est celui qui se détériore (où l'approche intra-SL offre de moins bons résultats que les approches de contrôle) le plus fréquemment parmi tous les critères. Notons, toutefois, que pour chaque algorithme où il y a une augmentation significative de l'effort de test, il y a également une augmentation significative de la couverture et de la détection des fautes. Pour trois approches, intra-SL demande un effort de test significativement moins élevé (critère amélioré). Notons aussi que dans ces cas, la couverture de ces approches était toujours similaire à intra-SL (pas d'amélioration ou de détérioration). Pour la détection des fautes (détection totale), l'approche intra-SL détecte significativement plus de fautes que toutes les approches de contrôle. L'efficience est aussi améliorée significativement

pour trois approches, mais détériorée pour une approche. Notons que CBGA-ES a une couverture directe faible comparée aux autres approches, ce qui facilite l'obtention d'une haute efficience. Finalement, l'ordonnancement des classes candidates au test est significativement amélioré par intra-SL pour trois des approches de contrôle.

On peut donc conclure, que pour la majorité des contextes l'approche intra-SL performe mieux que les approches de contrôle. Le cas où cette conclusion ne s'applique pas est celui où le nombre de ressources est très limité, et par conséquent, le critère le plus important devient l'effort de test. Dans ce cas, l'algorithme CBGA-ES sélectionne un ensemble de classes candidates au test demandant un effort de test plus faible et avec une meilleure efficience. En revanche, comme l'ordonnancement est tout de même meilleur pour l'approche intra-SL, il faudrait pouvoir être assuré de tester intégralement l'ensemble candidat soumis par CBGA-ES. Autrement, en cas de troncature, intra-SL pourrait toujours produire un meilleur résultat en raison du meilleur ordre de soumission des candidats.

TABLE 4.53 – Résumé de la comparaison d'intra-SL (KNN + RF) aux approches de contrôle

Critère	ALEA-couv	GL-couv	GL-comp	Intra-GL-risque (KNN+RF)	DG-couv	CBGA-ES	CH
Couverture	+			+	+	+	
Effort de test	+	-	-		+	+	-
Détection des fautes	+	+	+	+	+	+	+
Efficience		+	+			-	+
Ordonnancement	+				+	+	
Nb. critères améliorés significativement	*3*	*3*	*3*	*2*	*3*	*3*	*3*
Nb. critères détériorés significativement	*1*	*0*	*0*	*0*	*1*	*2*	*0*

Le tableau 4.54, qui présente les résultats analogues au tableau 4.53 pour l'approche inter-SL (KNN), montre que l'approche inter-SL (KNN) apporte une amélioration similaire par rapport aux approches de contrôle à celle notée pour l'approche intra-SL (KNN + RF). L'interprétation du tableau 4.54 est très similaire à l'interprétation du tableau 4.53. C'est pourquoi nous n'analysons que les différences entre ces deux tableaux. Pour tous les critères, sauf l'ordonnancement, les améliorations et détériorations sont les mêmes qu'avec intra-SL. Pour l'ordonnancement, l'approche inter-SL améliore significativement les résultats par rapport à toutes les approches de contrôle sauf pour inter-GL-risque. Notons que tout de même deux critères sont

améliorés par rapport à inter-GL-risque, soit la plus grande couverture atteinte et la meilleure détection des fautes. De plus, inter-SL ne performe jamais moins bien qu'inter-GL-risque.

Globalement, inter-SL offre de meilleurs résultats de priorisation des classes candidates au test que toutes les approches de contrôle avec encore une fois une exception pour CBGA-ES. Comme précédemment, un exemple où cette exception peut s'appliquer est le cas où les ressources allouées aux tests sont limitées, ce qui impose une contrainte sur l'effort de test. L'algorithme CBGA-ES sélectionne des ensembles de classes candidates demandant un effort de test moins important pour tester l'ensemble des classes candidates, de même qu'une meilleure efficience, critères importants dans cette situation.

TABLE 4.54 – Résumé de la comparaison d'inter-SL (KNN) aux approches de contrôle

Critère	ALEA-couv	GL-couv	GL-comp	Inter-GL-risque (KNN)	DG-couv	CBGA-ES	CH
Couverture	+			+	+	+	
Effort de test	+	-	-		+	+	-
Détection des fautes	+	+	+	+	+	+	+
Efficience		+	+			-	+
Ordonnancement	+	+	+		+	+	+
Nb. critères améliorés significativement	*3*	*4*	*4*	*2*	*3*	*3*	*4*
Nb. critères détériorés significativement	*1*	*0*	*0*	*0*	*1*	*2*	*0*

Nous pouvons maintenant apporter des réponses aux questions de recherche présentées dans l'introduction. Pour la QR1, l'approche développée propose-t-elle une meilleure couverture que les approches de contrôle, nous avons démontré que l'approche SL offre une meilleure couverture que plusieurs des approches de contrôle. Nous répondons à la QR2, est-ce que l'approche développée permet de détecter plus de fautes que les approches de contrôle, par l'affirmative. Notre approche de priorisation détecte plus de fautes, pour une taille semblable d'ensemble de classes candidates que les approches de contrôle. Ce résultat est obtenu pour toutes les approches de contrôle. Au niveau de la consommation de ressources, nous avons évalué cette question de recherche au travers de l'effort de test (ressources utilisées pour tester l'ensemble de classes candidates au test), mesuré par la couverture directe. La conclusion que nous dressons sur cette question de recherche est partagée. Isolée des autres facteurs, notre

approche consomme parfois moins et parfois plus de ressources. Lorsque nous croisons ce facteur avec la couverture, nous remarquons qu'un effort plus grand est associé à une meilleure couverture. Par conséquent, nous ne considérons pas comme problématique que l'approche développée augmente les ressources nécessaires par rapport à certaines approches de contrôle (effort de test de SL plus grand), car les résultats, au niveau du nombre de fautes détectées, s'en trouvent améliorés. Autrement dit, le compromis effort de test et qualité du test est préservé et possiblement amélioré. Pour la QR4, l'approche développée est-elle plus efficiente dans la recherche de fautes que les approches de contrôle, nous remarquons un gain ou une efficience équivalente avec la plupart des approches. Dans un cas, l'efficience diminue, mais en raison de facteurs explicités précédemment, ce résultat est à considérer avec prudence. Nous concluons donc tout de même que l'approche SL est plus efficiente que les approches de contrôle. Nous répondons à la QR5, l'approche développée ordonne-t-elle mieux les classes que les approches de contrôle, encore une fois par l'affirmative. Nous avons noté une amélioration par rapport aux résultats obtenus pour presque toutes les approches de contrôle.

Par rapport à la question de recherche principale, l'approche développée est-elle meilleure que les approches de contrôle, nous pouvons conclure qu'effectivement l'approche SL est meilleure que les approches de contrôle. Pour quatre des questions de recherche détaillées, nous avons répondu clairement par l'affirmative. Pour l'une des questions de recherche, nous ne pouvons pas affirmer que l'approche SL offre de meilleurs résultats que les approches de contrôle ni affirmer que notre approche offre des résultats moins bons que ceux des approches de contrôle. Le résultat est plus circonstanciel pour la QR3 et demanderait des études plus approfondies afin de pouvoir y répondre avec certitude.

4.7 Limitations et validité des résultats

Les méthodes d'analyse utilisées ont été appliquées sur de petits ensembles de données limitant la puissance des tests. Autrement dit, à certaines occasion l'hypothèse nulle n'a pas été rejetée alors qu'elle aurait dû l'être ce qui entraîne une production de faux négatifs plus importante. Par conséquent, les questions de recherche rejetées et les endroits où des différence non signinifcative a été observée ne devraient pas

être interprétés de façon conclusive (l'absence de lien ou de différence n'a pas été démontrée). Ces situations devraient réétudiées avec de nouveaux jeux de données pour augmenter la puissance des résultats. Également, la calibration de l'algorithme SL, seuil d'arrêt de sélection de nouvelles classes candidates, s'est faite par un processus empirique d'essai-erreur. Une réplique de cette étude avec une méthodologie de recherche de seuil pourrait améliorer encore les résultats et offrir une approche plus rigoureuse.

Seules vingt versions provenant de cinq systèmes différents ont été utilisées pour établir les résultats. Bien qu'ils s'agissent de systèmes utilisés dans plusieurs études reconnues dans le génie logiciel empirique, un échantillonnage plus important est nécessaire pour généraliser les résultats. Dans la collecte, nous avons également rejeté certains types de métriques et de mesures de centralité ce qui limite la portée de nos résultats. De même, nous avons obtenu les valeurs par analyse statique du code source. L'utilisation de données provenant d'une analyse dynamique pourraient bonifier les résultats obtenus. Nous avons également identifié lors de la description des métriques d'évaluation et de la collecte des données, en particulier les mesures de centralité, certains biais possibles (sur ou sous-estimation).

Également, la comparaison des résultats obtenus est limitée par le fait que nous adaptons le problème à un nouveau contexte, soit un plus large que le test de régression. Par conséquent, la comparaison des valeurs obtenues dans ce mémoire avec d'autres approches issues de la littérature ne serait pas rigoureuse. La collecte de données et la manière de calculer les métriques d'évaluation diffèrent et cela biaiserait la comparaison. C'est pourquoi nous avons adapté et implémenté des approches de priorisation de la littérature (présentées comme les approches de contrôle) afin de nous définir une base pour la comparaison. Il serait intéressant de répliquer l'exercice de ce mémoire à un plus grand ensemble de données afin d'augmenter la validité des conclusions obtenues.

4.8 Conclusion du chapitre 4

Dans ce long chapitre ont été présenté les résultats des expérimentations, tant en contexte intra-système, qu'inter-système. Nous avons établit les résultats pour toutes

les approches de contrôle et l'approche SL. Nous y avons analysé les résultats afin de répondre aux questions de recherche établies dans l'introduction. Des procédures statistiques ont été appliquées afin de valider le caractère significatif de nos résultats. J'ai également présenté les limites du travail accomplies ainsi que certaines menaces à la validité des résultats. Le prochain et dernier chapitre concluera ce mémoire en revenant sur les principaux résultats.

Chapitre 5

Conclusion

Au cours de ce mémoire, nous avons étudié la sélection et l'ordonnancement de classes candidates au test dans les systèmes orientés objets. Nous avons utilisé les données de vingt versions de cinq systèmes orientés objet, avec code source public, écrits dans le langage Java. Afin d'évaluer notre approche de sélection et priorisation, l'approche score local (SL), nous avons comparé les résultats obtenus avec sept approches de contrôle issues de la littérature. Ces approches ont été adaptées pour être utilisables dans le contexte de sélection et d'ordonnancement des classes candidates aux tests. Nous avons également proposé une évaluation des résultats selon cinq critères clés du processus de test à l'aide de métriques d'évaluation soit proposée par nous, soit éprouvées dans la littérature. Nous avons constaté que l'approche SL permettait d'obtenir de meilleurs résultats sur plusieurs critères que les approches de contrôle étudiées. Nous avons testé les approches dans un contexte de prédiction intra-système et inter-systèmes.

La plus grande contribution de ce mémoire est de proposer l'approche SL qui permet la sélection et la priorisation des classes candidates aux tests hors du contexte des tests de régression. Cette contribution a demandé la combinaison de travaux en détection des fautes logiciels et en priorisation des tests logiciels. Au meilleur de notre connaissance, peu de travaux ont abordé cette problématique de recherche, assez cruciale, dans un aspect aussi général que nous l'avons fait. Sortir du contexte des tests de régression nous a permis de lever l'hypothèse selon laquelle les tests devaient d'abord être écrits avant de pouvoir procéder à la priorisation.

Au niveau des travaux futurs, comme mentionné précédemment, nous trouvons intéressant de répliquer les expérimentations avec de nouveaux systèmes et de nouveaux algorithmes comme le recuit simulé, la recherche taboo ou des algorithmes évolutifs. Cela permettrait de diversifier davantage les jeux de données utilisés dans l'évaluation de l'approche inter-SL et ainsi de confirmer nos résultats. De même, l'approche SL mériterait d'être peaufinée au niveau du seuil critique d'inclusion des classes candidates aux tests. De plus, de nouvelles dimensions comme les ressources disponibles ou la diversité de l'ensemble des tests pourraient être inclues dans l'approche SL afin d'augmenter sa portée et la qualité de ses résultats. L'évolution des techniques de prédiction de fautes dans les classes des systèmes OO est également à considérer comme SL repose sur ces techniques pour proposer un ensemble de classes candidates. D'autres aspects, comme la prédiction du nombre de fautes ou de leur sévérité, pourraient être inclus dans le modèle SL afin d'augmenter la pertinence des classes sélectionnées pour être candidates aux tests.

Bibliographie

[Abran, 2010] Abran, A. (2010). *Software Metrics and Software Metrology*. Wiley.

[Anwer *et al.*, 2017] Anwer, S., Adbellatif, A., Alshayeb, M. et Anjum, M. S. (2017). Effect of coupling on software faults : An empirical study. *In 2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, pages 211–215. IEEE.

[Bansiya et Davis, 2002] Bansiya, J. et Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering*, 28(1):4–17.

[Basili *et al.*, 1996] Basili, V. R., Briand, L. C. et Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, 22(10):751–761.

[Carlson *et al.*, 2011] Carlson, R., Do, H. et Denton, A. (2011). A clustering approach to improving test case prioritization : An industrial case study. *In 2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 382–391.

[Catal, 2012] Catal, C. (2012). On the application of genetic algorithms for test case prioritization : a systematic literature review. *In Proceedings of the 2nd international workshop on Evidential assessment of software technologies*, pages 9–14.

[Charrad *et al.*, 2014] Charrad, M., Ghazzali, N., Boiteau, V. et Niknafs, A. (2014). Determining the best number of clusters in a data set. *J Stat Softw*.

[Chaurasia *et al.*, 2015] Chaurasia, G., Agarwal, S. et Gautam, S. S. (2015). Clustering based novel test case prioritization technique. *In 2015 IEEE Students Conference on Engineering and Systems (SCES)*, pages 1–5.

[Chen *et al.*, 2012] Chen, D., Lü, L., Shang, M.-S., Zhang, Y.-C. et Zhou, T. (2012). Identifying influential nodes in complex networks. *Physica a : Statistical mechanics and its applications*, 391(4):1777–1787.

[Chen *et al.*, 2018] Chen, J., Zhu, L., Chen, T. Y., Towey, D., Kuo, F.-C., Huang, R. et Guo, Y. (2018). Test case prioritization for object-oriented software : An adaptive random sequence approach based on clustering. *Journal of Systems and Software*, 135:107–125.

[Chen *et al.*, 1997] Chen, J.-L., Wang, F.-J. et Chen, Y.-L. (1997). An object-oriented dependency graph for program slicing. *In Proceedings. Technology of Object-Oriented Languages. TOOLS 24 (Cat. No. 97TB100240)*, pages 121–130. IEEE.

[Chen *et al.*, 2016] Chen, L., Ma, W., Zhou, Y., Xu, L., Wang, Z., Chen, Z. et Xu, B. (2016). Empirical analysis of network measures for predicting high severity software faults. *Science China Information Sciences*, 59(12):122901.

[Chidamber et Kemerer, 1994] Chidamber, S. R. et Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493.

[Choudhary *et al.*, 2018] Choudhary, G. R., Kumar, S., Kumar, K., Mishra, A. et Catal, C. (2018). Empirical analysis of change metrics for software fault prediction. *Computers& Electrical Engineering*, 67:15–24.

[Dalrymple-Alford, 1970] Dalrymple-Alford, E. (1970). Measurement of clustering in free recall. *Psychological Bulletin*, 74(1):32.

[Dangalchev, 2006] Dangalchev, C. (2006). Residual closeness in networks. *Physica A : Statistical Mechanics and its Applications*, 365(2):556–564.

[Do et Rothermel, 2006] Do, H. et Rothermel, G. (2006). On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Transactions on Software Engineering*, 32(9):733–752.

[Elbaum *et al.*, 2002] Elbaum, S., Malishevsky, A. G. et Rothermel, G. (2002). Test case prioritization : A family of empirical studies. *IEEE transactions on software engineering*, 28(2):159–182.

[Elish, 2014] Elish, M. O. (2014). A comparative study of fault density prediction in aspect-oriented systems using mlp, rbf, knn, rt, denfis and svr models. *Artificial Intelligence Review*, 42(4):695–703.

[Gyimothy *et al.*, 2005] Gyimothy, T., Ferenc, R. et Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering*, 31(10):897–910.

[He et Garcia, 2009] He, H. et Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284.

[He *et al.*, 2015] He, P., Li, B., Liu, X., Chen, J. et Ma, Y. (2015). An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, 59:170–190.

[Hechenbichler et Schliep, 2004] Hechenbichler, K. et Schliep, K. (2004). Weighted k-nearest-neighbor techniques and ordinal classification.

[Hemmati, 2019] Hemmati, H. (2019). *Advances in techniques for test prioritization*, volume 112, pages 185–221. Elsevier.

[Horwitz et Reps, 1992] Horwitz, S. et Reps, T. (1992). The use of program dependence graphs in software engineering. *In Proceedings of the 14th international conference on Software engineering*, pages 392–411.

[Jureczko et Madeyski, 2010] Jureczko, M. et Madeyski, L. (2010). Towards identifying software project clusters with regard to defect prediction. *In Proceedings of the 6th international conference on predictive models in software engineering*, pages 1–10.

[Kandil *et al.*, 2017] Kandil, P., Moussa, S. et Badr, N. (2017). Cluster-based test cases prioritization and selection technique for agile regression testing. *Journal of Software : Evolution& Process*, 29(6).

[Kaur et Malhotra, 2008] Kaur, A. et Malhotra, R. (2008). Application of random forest in predicting fault-prone classes. *In 2008 International Conference on Advanced Computer Theory and Engineering*, pages 37–43. IEEE.

[Larsen et Harrold, 1996] Larsen, L. et Harrold, M. J. (1996). Slicing object-oriented software. *In Proceedings of IEEE 18th international conference on software engineering*, pages 495–505. IEEE.

[Li *et al.*, 2007] Li, Z., Harman, M. et Hierons, R. M. (2007). Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering*, 33(4):225–237.

[Liaw et Wiener, 2002] Liaw, A. et Wiener, M. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

[MacWilliams et Sloane, 1977] MacWilliams, F. J. et Sloane, N. J. A. (1977). *The theory of error correcting codes*, volume 16. Elsevier.

[Martin, 1994] Martin, R. (1994). Oo design quality metrics. *An analysis of dependencies*, 12(1):151–170.

[Masri et El-Ghali, 2009] Masri, W. et El-Ghali, M. (2009). Test case filtering and prioritization based on coverage of combinations of program elements. *In Proceedings of the Seventh International Workshop on Dynamic Analysis*, pages 29–34.

[McCabe, 1976] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320.

[Moeyersoms *et al.*, 2015] Moeyersoms, J., de Fortuny, E. J., Dejaeger, K., Baesens, B. et Martens, D. (2015). Comprehensible software fault and effort prediction : A data mining approach. *Journal of Systems and Software*, 100:80–90.

[Ouellet et Badri, 2019] Ouellet, A. et Badri, M. (2019). Empirical analysis of object-oriented metrics and centrality measures for predicting fault-prone classes in object-oriented software. *In International Conference on the Quality of Information and Communications Technology*, pages 129–143. Springer.

[Ouellet et Badri, 2021] Ouellet, A. et Badri, M. (2021). Prediction of faults in object-oriented software using object-oriented metrics and centrality measures. page 51.

[Pradhan *et al.*, 2017] Pradhan, D., Wang, S., Ali, S., Yue, T. et Liaaen, M. (2017). Cbga-es : A cluster-based genetic algorithm with elitist selection for supporting multi-objective test optimization. *In 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 367–378. IEEE.

[Rahman *et al.*, 2012] Rahman, F., Posnett, D. et Devanbu, P. (2012). Recalling the" imprecision" of cross-project defect prediction. *In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11.

[Rothermel *et al.*, 1999] Rothermel, G., Untch, R. H., Chu, C. et Harrold, M. J. (1999). Test case prioritization : An empirical study. *In Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*, pages 179–188. IEEE.

[Rothermel *et al.*, 2001] Rothermel, G., Untch, R. H., Chu, C. et Harrold, M. J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on software engineering*, 27(10):929–948.

[Russell et Norvig, 2016] Russell, S. et Norvig, P. (2016). *Artificial Intelligence : A Modern Approach*. Pearson.

[Sammut et Webb, 2011] Sammut, C. et Webb, G. I. (2011). *Encyclopedia of machine learning*. Springer Science& Business Media.

[Saporta, 2011] SAPORTA, G. (2011). *Probabilités, analyse des données et statistique.* Technip.

[Shatnawi, 2012] SHATNAWI, R. (2012). Improving software fault-prediction for imbalanced data. *In 2012 international conference on innovations in information technology (IIT)*, pages 54–59. IEEE.

[Spinellis, 2005] SPINELLIS, D. (2005). Tool writing : a forgotten art ?(software tools). *IEEE Software*, 22(4):9–11.

[Team, 2020] TEAM, R. C. (2020). R : A language and environment for statistical computing.

[Tosun *et al.*, 2009] TOSUN, A., TURHAN, B. et BENER, A. (2009). Validation of network measures as indicators of defective modules in software systems. *In Proceedings of the 5th international conference on predictor models in software engineering*, pages 1–9.

[Wang *et al.*, 2016] WANG, S., ALI, S., YUE, T., BAKKELI, y. et LIAAEN, M. (2016). Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search. *In Proceedings of the 38th International Conference on Software Engineering Companion*, pages 182–191.

[Zhao *et al.*, 2015] ZHAO, X., WANG, Z., FAN, X. et WANG, Z. (2015). A clustering-bayesian network based approach for test case prioritization. *In 2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 542–547.

[Zimmermann *et al.*, 2009] ZIMMERMANN, T., NAGAPPAN, N., GALL, H., GIGER, E. et MURPHY, B. (2009). Cross-project defect prediction : a large scale experiment on data vs. domain vs. process. *In Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100.

Annexe A

Démonstration de la moyenne de l'APFD lorsque les fautes sont distribuées uniformément

La moyenne de l'APFD lorsque les fautes sont distribuées uniformément est asymptotiquement de 50 % lorsque le nombre de classes est élevé. Soit un ensemble de classes candidates au test de taille n contenant m fautes. La variable aléatoire F indique le nombre de fautes dans la $i^{ème}$ classe candidate au test et E est l'opérateur d'espérance mathématique. On suppose que F suit une loi uniforme.

$$E\{APFD\} = E\left\{1 - \sum_{i=1}^{n} \frac{if_i}{nm} + \frac{1}{2n}\right\} \tag{A.1}$$

$$= 1 - \sum_{i=1}^{n} \left(\frac{i}{nm} \times E\{f_i\}\right) + \frac{1}{2n} \tag{A.2}$$

$$= 1 - \sum_{i=1}^{n} \left(\frac{i}{nm} \times \frac{m}{n}\right) + \frac{1}{2n} \tag{A.3}$$

$$= 1 - \frac{1}{n^2} \sum_{i=1}^{n} i + \frac{1}{2n} \tag{A.4}$$

$$= 1 - \frac{n(n+1)}{2n^2} + \frac{1}{2n} \tag{A.5}$$

$$= 1 - \frac{n(n+1)+n}{2n^2} \tag{A.6}$$

$$= 1 - \frac{n^2}{2n^2} + \frac{2n}{2n^2} \tag{A.7}$$

$$= \frac{1}{2} + \frac{1}{n} \tag{A.8}$$

$$\tag{A.9}$$

D'où, on observe le comportement asymptomatique suivant :

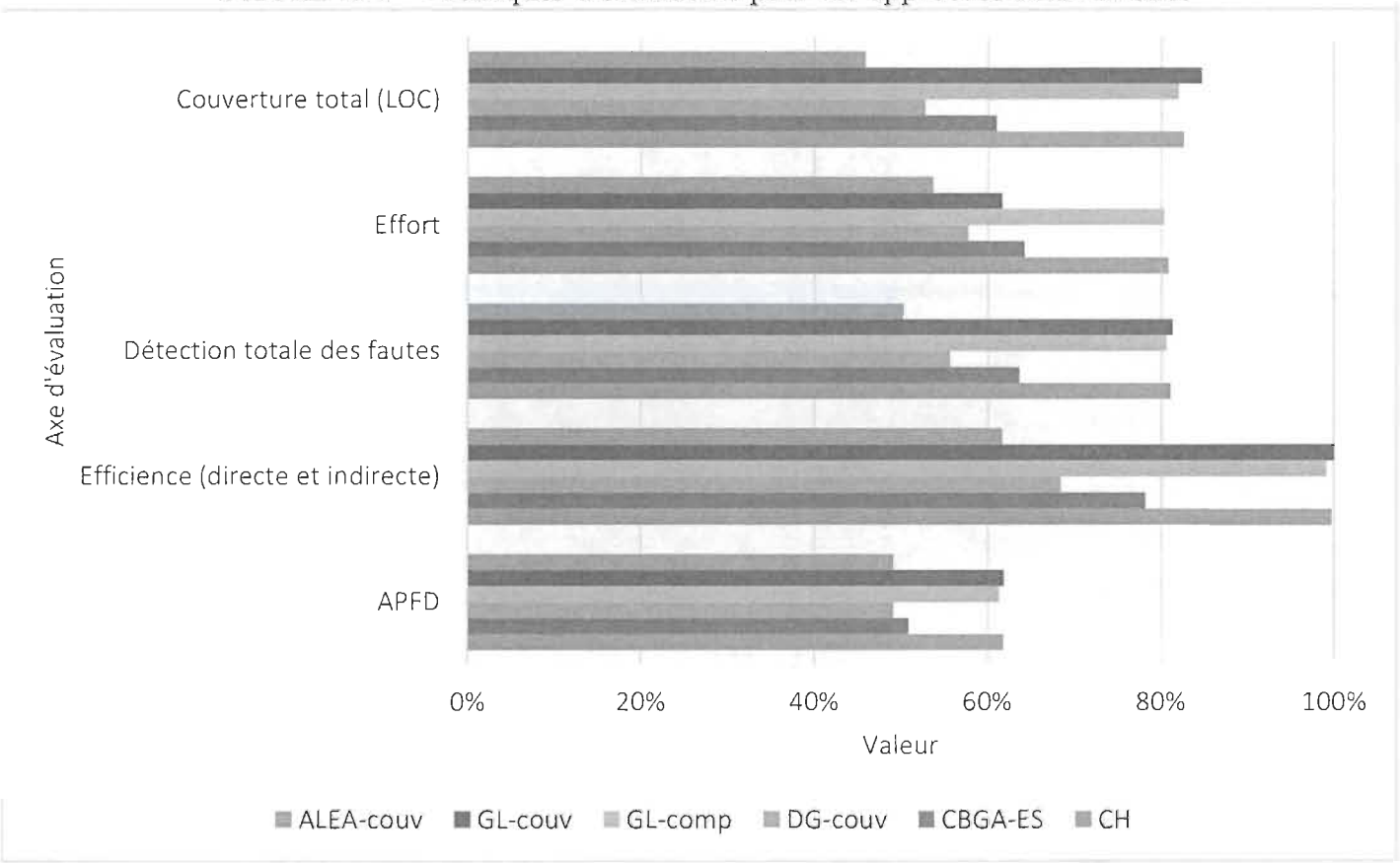
$$\lim_{n\to\infty} E\{APFD\} = \lim_{n\to\infty} \left(\frac{1}{2} + \frac{1}{n}\right) = \frac{1}{2} \tag{A.10}$$

Annexe B

Graphiques

Sur tout les graphiques de cet annexe les valeurs d'efficience ont été normalisée sur en divisant par la plus grande valeur disponible, afin d'obtenir une échelle commune avec les autres métriques d'évaluation. Le lecteur est prié de se référer aux tableaux correspondants pour obtenir les valeurs.

FIGURE B.1 – Métriques d'évaluation pour les approches sans variante



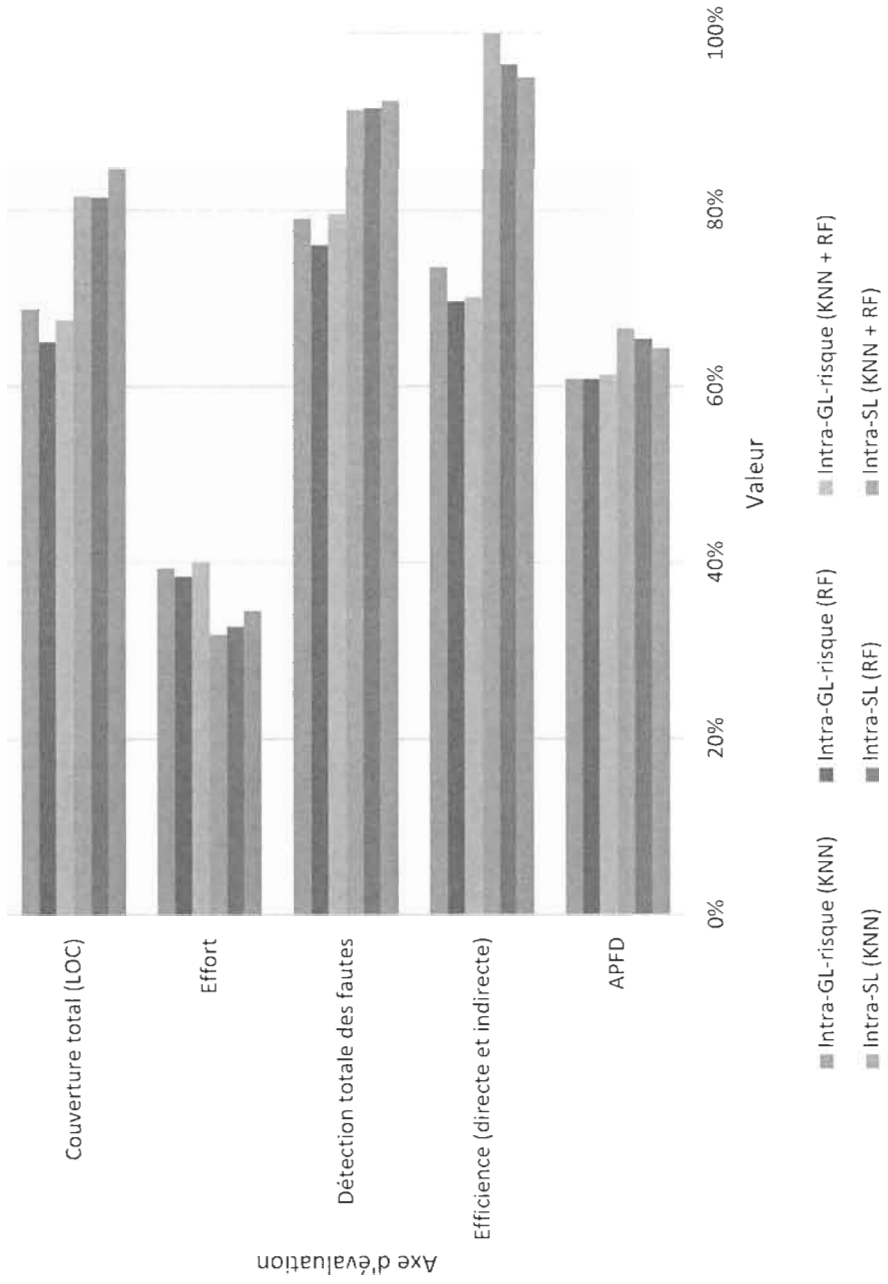


FIGURE B.2 – Métriques d'évaluation pour les approches intra-SL

FIGURE B.3 – Métriques d'évaluation pour les approches inter-SL