Université de Montréal

Amélioration de la prédiction de la qualité du logiciel par combinaison et adaptation de modèles

par

Salah Bouktif

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophie Doctor (Ph.D.)
en informatique

mai, 2005

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée :

Amélioration de la prédiction de la qualité du logiciel par combinaison et adaptation de modèles

présentée par :

Salah Bouktif

a été évaluée par un jury composé des personnes suivantes :

Jacques A. Ferland
(président-rapporteur)

Houari A. Sahraoui
(directeur de recherche)

Balázs Kégl
(codirecteur)

Yann-Gaël Guéhéneuc
(membre du jury)

Coral Calero
(examinateur externe)

John Mullins
(représentant du doyen de la FES)

Résumé

Dans ce travail, nous proposons une approche d'amélioration de la prédiction de la qualité du logiciel en utilisant la combinaison et l'adaptation des modèles prédictifs (CAMP). C'est une approche inspirée de la technique de mélange d'experts qui, à partir d'un ensemble de modèles existants et d'un échantillon reflétant les spécificités d'un environnement particulier (un contexte logiciel dans une organisation particulière), dérive un modèle le plus adapté possible à ce contexte. Concrètement, notre approche vise, comme les autres alternatives d'amélioration de la prédiction, à augmenter l'exactitude des modèles prédictifs. Mais elle se distingue par la prise en considération de deux autres propriétés importantes de modèles, à savoir, la facilité d'interprétation et la capacité, ou le potentiel de généralisation.

L'approche CAMP est fondée sur une idée reposant sur trois principes qui se chargent respectivement de la promotion des trois propriétés étudiées. Ces principes peuvent être résumés de la façon suivante :

- décomposer les modèles en expertises pour faciliter leur interprétation et leur manipulation ;
- combiner les expertises pour enrichir et généraliser les modèles ;
- adapter les expertises pour mieux convenir à un contexte spécifique.

Pour appliquer ces principes, trois principaux mécanismes sont définis pour se charger respectivement de l'identification (éventuellement la décomposition), de la combinaison et de l'adaptation des expertises contenues dans les modèles. Afin de bien adhérer à ces principes et de contourner la complexité de ces mécanismes, nous avons eu recours aux techniques métaheuristiques. Deux familles des ces techniques sont utilisées pour mettre en oeuvre notre approche, à savoir, les méthodes basées sur la population et représentées par les algorithmes génétiques (AG), et les méthodes basées sur la recherche locale et représentées par le recuit simulé (RS) et la recherche avec tabous (RT). Ainsi, deux appli-

cations particulières de notre approche sont également conçues, l'une s'intéresse aux modèles de type arbres de décisions et l'autre s'occupe des classificateurs bayésiens. Dans le but d'effectuer l'évaluation de l'approche CAMP, nous avons construit un environnement « semi-réel » dans lequel le contexte d'organisation est une évolution d'un vrai système logiciel (API Java), mais les modèles sont « simulés » (c'est-à-dire, obtenus par apprentissage sur des données réelles). La construction de cet environnement est à double apport. En effet, nous avons défini d'une part, un facteur de la qualité crucial pour la maintenance de logiciel qui est la stabilité de classes dans un logiciel orienté objets, d'autre part, nous avons construit des modèles de stabilité de type arbres de décision et de type classificateurs bayésiens. Les résultats obtenus dans le cadre des différentes expériences montrent que les objectifs fixés pour améliorer la prédiction sont bien atteints, à savoir, la bonne capacité prédictive (exactitude), la bonne capacité de généralisation et la facilité d'interprétation du modèle dérivé par l'approche CAMP.

Mots Clés

Qualité du logiciel, arbres de décision, classificateurs bayésiens, algorithmes génétiques, recherche avec tabous, recuit simulé.

Abstract

In this work, we propose an approach to improve software quality prediction by using combination and adaptation of models. It's a new approach that we call CAMP, inspired from the technique of mixture of experts. By using a set of existing models and a sample of data reflecting specificities of a particular environment (a software context in a particular organization), the approach derives a more suitable model to this context.

Concretely, CAMP approach aims, like the other alternatives of prediction improvement, to increase the accuracy of models. Furthermore, it is characterized by taking into account two other significant properties of software quality predictive models, namely, the interpretability and the generalisability.

The CAMP approach is founded on three principles that consider the promotion of the three studied model properties (predictive accuracy, interpretability and generalisability). These principles can be summarized as follows :

- decompose the models into expertises to facilitate their interpretation and their handling;
- combine the expertises to enrich and generalize the models;
- adapt the expertises to be more appropriate for a specific context.

To apply these principles, three main mechanisms are defined to be given the responsibilities, respectively, of identification (possibly decomposition), combination and adaptation of expertises contained in the models. In order to adhere well to these principles and to overcome complexity of these mechanisms, we had recourse to metaheuristic techniques. Two families of these techniques are used to implement our approach, namely, the methods based on the population and represented by the genetic algorithms, and the methods based on local search and represented by simulated annealing and tabu search.

Two particular applications of our approach are also conceived. The first is interested in decision

trees based models and the second deals with bayesian classifiers. To empirically evaluate CAMP, we built a "semi-real" environment in which context of organization is a real software systems (API Java), but the models are "simulated" (i.e., obtained by training on real data). The construction of this environment is with double contribution. On the one hand, we defined, a quality factor that is crucial for the maintenance of software which is the stability of classes in object oriented software, on the other hand, we built stability models based on decision trees and on bayesian classifier.

The results obtained by various experiments show that the objectives fixed to improve software quality prediction are well achieved, namely, the higher predictive accuracy, the good generalisability and the good interpretability of the model derived by the CAMP approach.

Keywords

Software quality, decision trees, Bayesian classifiers, genetic algorithms, tabu search, simulated annealing.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

AD	arbre de décision
AG	algorithme génétique.
CAMP	combinaison et adaptation des modèles prédictifs.
CB	classificateur bayésien
IA	intelligence artificielle
LF	logique floue
OO	orientée objet
PF	points de fonction
PG	programmation génétique
PMO	problème d'optimisation multi-objectif
RB	réseau bayésien
RBC	raisonnement à base de cas
RL	régression linéaire
RN	réseaux de neurones
RR	régression robuste
RS	recuit simulé
RT	recherche avec tabous
TPC	table de probabilités conditionnelles

Remerciements

À la fin de cette thèse, je tiens à exprimer mes vifs remerciements à mon directeur de recherche le professeur Houari Sahraoui d'avoir accepté de diriger mes travaux de recherche. En plus de son support, Ses remarques pertinentes et ses conseils m'ont permis de mieux présenter et structurer mes idées. Je remercie également le professeur Bàlazs kégl d'avoir accepté de codiriger mes recherches et d'avoir participé à mon support financier.

Mes remerciements vont aussi au président de jury le professeur Jacques Ferland et au membre de jury le professeur Yann-Gaël Guéhéneuc d'avoir accepté de participer au jury de cette thèse. Je les remercie pour leurs commentaires qui ont contribué à l'amélioration de ce document.

Je remercie également la professeur Coral Carelo, d'avoir accepté d'examiner ma thèse et d'avoir participé à l'amélioration de ce document par ses commentaires pertinents portant sur l'état de l'art de la qualité de logiciel. Je tiens à remercier le représentant du doyen le professeur John Mullins d'avoir participé à ce jury.

Mes remerciements vont à tous mes amis pour leur soutien moral. Je remercie particulièrement Mohamed Adel serhani pour ses encouragements et ses appels téléphoniques malgré les distances géographiques.

Fawzi Hmissi est un ami que je viens de découvrir pendant des moments difficiles de la rédaction de cette thèse. Je le remercie fortement pour le temps précieux qu'il a dépensé dans la lecture de ce document comme s'il était le sien.

Je remercie mon ancien ami le professeur Mohammed Mejri pour la lecture et l'examen de la première version de ma thèse. Ses impressions et ses encouragements m'ont donné la fierté de produire un tel travail et l'énergie de parfaire ce document.

Je remercie tous les membres du laboratoire GEODES pour leur sympathie et encouragements.

Mes remerciements les plus profonds vont à ma mère Fattoum pour ses expressions inégalement encourageantes (« Mz... hou »), sa patience et toutes les valeurs qu'elle m'a inculquées. Ces remerciements vont à ma femme Abir pour sa compréhension, ses encouragements et son support pendant les moments difficiles de ce travail et à toute ma famille pour leur amour et leur respect.

À la mémoire de mon père

À ma mère

À ma femme

À mon futur bébé

À tous ceux que j'aime

Chapitre 1

Introduction

1.1 Contexte et problématique

Pendant les trois dernières décennies, la communauté de génie logiciel s'intéressait de plus en plus à la qualité, à ses caractéristiques, à ses indicateurs, et aux modèles qui permettent de la prédire. L'histoire des modèles de prédiction de la qualité du logiciel remonte aux années 60. À cette époque, le champ d'application de l'informatique commençait à s'étendre et le besoin de budgétiser, de planifier, de contrôler, d'analyser les risques, et d'estimer les coûts s'est fait cruellement sentir. Ainsi, des travaux de recherche significatifs sur l'estimation du coût débutait par une étude étendue en 1965 sur l'impact de 104 attributs de 169 projets logiciels sur le coût [Nelson, 1966]. Ceci a mené au développement de quelques modèles d'estimation du coût qui se sont avérés fort utiles. Vers la fin des années 70, des modèles plus robustes tels que SLIM, Checkpoint, PRICE-S, SEER SEM et COCOMO sont proposés.

La plupart de ces modèles ont été développés à peu près en même temps et leurs concepteurs ont fait face au même dilemme : l'augmentation de la taille du logiciel rend le développement plus complexe et augmente le risque de l'échec. Il est donc nécessaire de bien planifier ce développement. Or cette même complexité rend toute forme de prédiction ou d'estimation (coût et qualité) extrêmement difficile. Par la suite, de nombreux chercheurs se sont penchés sur le problème de la prédiction de la qualité donnant lieu à de nombreuses contributions (voir section 3.5).

Plus tard, l'avènement de la conception et la programmation par objets n'a fait qu'accentuer la problématique de la prédiction. En effet, durant les premières années de cet avènement, l'industrie a fait

preuve de beaucoup de tolérance vis-à-vis de ce nouveau paradigme. Maintenant, la conception et la programmation par objets ont atteint l'étape de maturité et les produits logiciels OO deviennent de plus en plus complexes. Les besoins en matière de qualité deviennent davantage des facteurs déterminants dans le choix des alternatives de conception et de codage pendant le développement des logiciels. Par conséquent, il est important que la qualité du logiciel soit évaluée pendant les différentes étapes du développement.

Pendant les dernières années, il a été proposé un grand nombre de modèles de prédiction de la qualité dans la littérature. Habituellement, le but de ces modèles est de prédire un facteur de qualité à partir d'un ensemble de mesures directes. En général, il existe deux méthodes pour construire les modèles prédictifs de qualité. La première repose sur les données historiques. Dans cette alternative, diverses techniques sont appliquées pour dériver des régularités à partir de ces données (voir par exemple [Briand et Wüst, 2002]). La deuxième méthode est basée sur les connaissances d'experts qui se présentent souvent sous la forme d'heuristiques spécifiques à un domaine en particulier (voir par exemple [Fenton et Ohlsson, 2000]).

Les données collectées à partir de vrais systèmes logiciels sont cruciales dans les deux méthodes de construction : dans la première, nous en avons besoin pour construire les modèles, et dans la seconde, nous en avons besoin pour la validation empirique des modèles. Dans la plupart des domaines où des modèles prédictifs sont établis (comme la sociologie, la médecine, les finances, et la reconnaissance de la parole), les chercheurs ont le privilège d'utiliser des grands dépôts de données à partir desquels des échantillons représentatifs peuvent être extraits. Cependant, dans le domaine du génie logiciel, de tels dépôts sont rares. Le manque de données rend les modèles existants difficiles à généraliser, à valider et à utiliser. Le choix d'un modèle de prédiction de la qualité approprié à un contexte d'une compagnie en particulier est une décision difficile et non triviale puisque les modèles universels de prédiction de la qualité n'existent pas encore [Fenton et Pfleeger, 1997; Abdel-Ghaly et *al.*, 1986]. La pénurie de données ne fait qu'accentuer ce problème. Plusieurs raisons sont à l'origine de cette pénurie de données. Nous relevons par exemple la rareté, voir l'absence, des collectes systématiques de données dans les compagnies de logiciels ou encore la confidentialité de l'information. Ces deux problèmes ne sont pas purement d'ordre technique et leur résolution n'est pas facilement réalisable, du moins à court terme [Khoshgoftaar et *al.*, 1995a].

Outre l'exactitude (la capacité prédictive) d'un modèle, qui est d'une importance capitale, d'autres propriétés doivent être considérées. La facilité d'interprétation, ou le potentiel d'explication d'un modèle est une propriété importante. En effet, les modèles de prédiction de la qualité s'adressent en premier lieu aux gestionnaires et aux responsables de projets. Toute décision découlant de ces modèles doit être clairement justifiée pour permettre à ces derniers d'entreprendre les actions correctives. Dans beaucoup de travaux, ces propriétés sont rarement atteintes [Jeffery et Low, 1990]. Ceci est dû à de nombreuses difficultés allant des mauvaises caractéristiques des données sur la qualité à l'utilisation inappropriée des techniques de construction de modèles pour sa prédiction [Gray et MacDonell, 1997].

Notre présent travail est motivé par la recherche de solution à la pénurie de données et par la promotion de la qualité des modèles prédictifs en considérant en plus de leur capacité prédictive, leur facilité d'interprétation, ou d'utilisation et leur capacité de généralisation.

1.2 Objectifs

Nous proposons dans notre recherche, une approche d'amélioration de la prédiction de la qualité du logiciel par combinaison et adaptation de modèles prédictifs (CAMP). À partir d'un ensemble de modèles existants qui prédisent un même facteur de qualité et en tenant compte des spécificités d'un environnement logiciel particulier, notre approche construit un nouveau modèle se caractérisant par les propriétés suivantes : une *bonne capacité prédictive*, *une facilité d'interprétation* et *une capacité de généralisation suffisante*. La première propriété permet de garantir une adéquation du modèle à l'environnement considéré. La seconde propriété offre la facilité d'utiliser le modèle comme un outil de décision. Enfin, la troisième permet au modèle de tenir compte de l'évolution future du contexte logiciel où il est utilisé. Ainsi, la performance de ce dernier se maintient longuement, permettant une vie prolongée du modèle.

Inspirée de la technique de mélange d'experts [Jacobs et *al.*, 1991], notre approche consiste à combiner des modèles existants et à les adapter afin de produire (voir section 4.4), un meilleur modèle. Le processus de combinaison et d'adaptation de notre approche est guidé par un petit échantillon de données représentant un environnement logiciel particulier (contexte).

Afin de considérer les propriétés précédemment soulignées des modèles et en tenant compte la

complexité des problèmes de combinaison, nous utilisons deux familles des métaheuristiques comme techniques de notre approche : les méthodes basées sur la population, représentées par les algorithmes génétiques (AG) et les méthodes basées sur la recherche locale, représentées par le recuit simulé (RS) et la recherche avec tabous (RT) [Ferland et Costa, 2001].

Notre approche générale, basée sur les métaheuristiques, peut être appliquée à plusieurs types de modèles de prédiction. Pour illustrer son application, deux « spécialisations » de notre approche sont réalisées respectivement sur deux types de modèles que sont les arbres de décision et les classificateurs bayésiens. Plus particulièrement, un algorithme par métaheuristique (AG, RS et RT) est proposé pour chaque type de modèle.

Une évaluation rigoureuse de notre approche est également réalisée sur la problématique de prédiction de la stabilité des classes dans les logiciels orientés objets (OO).

1.3 Les contributions majeures

Les contributions majeures de cette thèse sont les suivantes :

- **amélioration de la prédiction dans une organisation particulière :** nous avons proposé une approche qui permet de produire un modèle de prédiction de la qualité plus approprié à une organisation particulière. Ce modèle est dérivé en utilisant des modèles existants généralement peu adaptés à cette organisation et une description du contexte (spécificités) de cette dernière.
- **considération de plusieurs propriétés des modèles de prédiction de la qualité :** outre l'exactitude, nous avons considéré d'autres propriétés des modèles de prédiction de la qualité du logiciel. Malgré leur importance, ces propriétés sont rarement atteintes dans beaucoup de travaux [Fenton et Neil, 1999a]. Dans cette thèse nous avons d'abord identifié les plus importantes d'entre elles, par la suite nous leurs avons proposées un ordre de priorité qui considère le modèle comme un outil d'aide à la prise de décision devant être précis, stable et facile à utiliser. En effet, notre approche considère les trois premières propriétés les plus prioritaires selon cet ordre, à savoir, l'exactitude (la bonne capacité prédictive), la facilité d'interprétation et la bonne capacité de généralisation du modèle.
- **proposition d'une méthode de combinaison de modèles basée sur les métaheuristiques :** Pour

combiner des modèles nous avons utilisé deux familles de méthaheuristiques (1) les méthodes basées sur la population représentées par les algorithmes génétiques (AG) et (2) les méthodes basées sur la recherche locale (ou basée sur le voisinage) représentées par la recherche avec tabous (RT) et par le recuit simulé (RS). Nous avons proposé pour chacun des trois algorithmes deux adaptations, une pour combiner des modèles de type arbres de décision et une autre pour combiner des classificateurs bayésiens.

- **Une solution pour l'apprentissage de modèles en cas de pénurie de données :** Afin de construire un modèle de prédiction en circonstance de rareté de données, nous avons proposé une approche qui permet d'apprendre à partir des connaissances contenues dans des modèles existants. Il s'agit d'un apprentissage à haut niveau qui utilise des expertises à la place des données d'entraînement. Ces expertises sont apprises sur des données qui ne nous sont pas disponibles.
- **Modèles de la stabilité de classes :** Nous avons créé un environnement semi réel afin de mener une évaluation empirique rigoureuse de notre approche. L'utilisation du facteur « stabilité de classes » d'un logiciel OO, pour valider les résultats de l'approche CAMP est une contribution à double apport. Le premier consiste en la définition de la stabilité de classe en tant que telle, basée sur la variation entre les versions, le deuxième consiste en la construction des modèles pour la prédire à partir d'un nombre de métriques structurelles convenables.

Cette thèse présente un travail regroupant plus qu'un domaine de recherche notamment le domaine de la qualité du logiciel, le domaine d'optimisation et le domaine d'apprentissage. En effet, les communautés respectives de ces domaines peuvent se servir de cette thèse et en tirer des conclusions.

Enfin, l'originalité et la contribution des travaux de recherche présentés dans cette thèse ont été reconnues par la communauté de la qualité de logiciel à travers les publications suivantes : [Bouktif, 2002; Azar et *al.*, 2002; Bouktif et *al.*, 2002a; Bouktif et *al.*, 2002b; Bouktif et *al.*, 2004]. Les autres aspects de nos travaux sont en cours de publication.

1.4 Organisation de la thèse

Le présent document est organisé comme suit. Le deuxième chapitre introduit deux domaines impliqués dans notre travail sur les modèles de prédiction de la qualité du logiciel. Ainsi, le domaine

d'apprentissage et celùi de l'optimisation combinatoire sont présentés à travers la description de leurs principaux méthodes, concepts et notations.

Le troisième chapitre présente l'état de l'art de la qualité du logiciel, il est inévitablement grand à cause de l'ampleur et la diversité de la littérature sur la qualité de logiciel. Dans ce chapitre nous donnons plus d'intérêt aux modèles de prédiction de la qualité, à leurs problèmes ainsi qu'aux solutions présentées dans la littérature.

Le quatrième chapitre est le coeur de notre thèse, il décrit notre solution pour améliorer la prédiction de la qualité du logiciel. Une approche générale de combinaison et d'adaptation des modèles de prédiction (CAMP) est proposée dans ce chapitre.

Le cinquième chapitre présente une solution spécifique qui se réalise par l'application de notre approche CAMP aux modèles de type arbres de décision. Cette application comporte la proposition et l'adaptation de trois algorithmes basés respectivement sur les algorithmes génétiques, sur la recherche avec tabous et sur le recuit simulé.

Le sixième chapitre est une deuxième application de notre approche CAMP aux modèles probabiliste représentés par les classificateurs bayésiens. Dans ce chapitre nous proposons trois autres adaptations de AG, de RT et de RS, au problème de combinaison classificateurs bayésiens.

Le septième chapitre s'intéresse à l'évaluation empirique de notre approche CAMP. Nous y présentons un processus expérimental rigoureux pour vérifier plusieurs hypothèses portant sur la supériorité des modèles résultant de l'application de CAMP. Notre approche est éprouvée sur des modèles de prédiction du facteur « stabilité de classes » dans un logiciel OO.

Et enfin nous concluons sur les perspectives qu'ouvre le travail présenté dans cette thèse, où nous ferons aussi une synthèse de nos recherches et de nos travaux futurs.

Chapitre 2

L'apprentissage et l'optimisation combinatoire

2.1 Introduction

Dans ce chapitre notre intérêt porte d'abord, sur certaines définitions empruntées au domaine de l'apprentissage qui vont servir à comprendre plusieurs mécanismes utilisés dans la modélisation de la qualité du logiciel. D'autre part, nous nous intéressons à certaines notions et techniques d'optimisation combinatoire qui vont servir d'outils pour la mise en oeuvre des solutions apportées au problème de l'amélioration de la qualité des prédictions traité dans ce travail. Certaines techniques ou définitions (d'apprentissage) sont illustrées par des exemples touchant la qualité du logiciel, d'autres sont décrites d'une manière brute sans aucune adaptation à un problème particulier car leurs applications pour l'amélioration de la prédiction de la qualité du logiciel est un objectif principal de notre présente recherche.

2.2 L'apprentissage

Les résultats des algorithmes d'apprentissage sont de plus en plus utilisés dans la tâche de modélisation de la qualité du logiciel pour remplacer éventuellement des opinions d'experts. Les modèles produits sont utilisés pour prédire le coût et l'effort de développement et d'autres facteurs représentant la qualité du logiciel. Le processus d'apprentissage à partir de données, qui va être décrit dans la

section 2.2.2, se déroule en plusieurs étapes. D'abord, une présentation de données sous forme d'observations (voir section 2.2.1) est nécessaire. Ensuite, il faut spécifier le type de prédiction à savoir, s'il s'agit d'une régression ou d'une classification. Et puis, un processus de généralisation commence à partir de données pour y dériver des régularités. Une fois formé, un modèle de prédiction est évalué pour déterminer son exactitude ainsi que la performance de l'algorithme d'apprentissage utilisé.

2.2.1 Présentation de données

Un exemple de données (appelé également cas ou observation) est une agrégation de propriétés appelées attributs. Ils décrivent une même entité, par exemple, un composant ou un produit logiciel. Supposons qu'il s'agisse de déterminer l'effort de développement pour un projet à partir de certains attributs, comme la *taille* et le *type* du projet. Pour ce faire, on a besoin d'un ensemble d'occurrences de valeurs de ces attributs appelées observations. Une observation serait par exemple ($type = organique; taille = 25\,000$), c'est-à-dire que ce projet est de type organique et sa taille est estimée à 25 000 lignes. On peut l'écrire sous forme d'un vecteur $\mathbf{x}_{Pr1} = (organique, 25\,000)$. D'une façon générale, une observation désignée par $\mathbf{x}$ peut être notée $\mathbf{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(d)})$, où $x^{(j)}$ est la valeur du $j^{ième}$ attribut et d est le nombre d'attributs. Une observation $\mathbf{x}$ est constituée d'un nombre d'*attributs d'entrée* qui sont associés à un autre attribut appelé *attribut de sortie* noté conventionnellement y. Dans l'exemple de calcul de l'effort de développement d'un projet, l'effort est égal à 70, donc $y_{Pr1} = 70$ est associée à $\mathbf{x}_{Pr1}$ pour former le couple $(\mathbf{x}_{Pr1}, y_{Pr1})$. L'espace décrit par toutes les valeurs des attributs de $\mathbf{x}$ est appelé *l'espace d'entrée* et celui décrit par les valeurs de y est appelé *l'espace de sortie*. Les valeurs des attributs d'entrée ou de sortie peuvent être généralement discrètes ou continues.

2.2.2 Tâche d'apprentissage

Le processus de prédiction de la valeur de l'attribut de sortie d'un cas se réalise en utilisant une méthode (ou un modèle) de prédiction. Plusieurs méthodes de prédiction sont utilisées, comme les systèmes de règles, les systèmes de régression, les réseaux de neurones, etc. La construction d'un modèle nécessite une collection d'observations pour lesquelles les valeurs de sortie sont déjà connues, on les appelle, *données d'entraînement*, *données d'apprentissage* ou encore *données de construction*. Un ensemble d'entraînement défini par $D = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ est un échantillon de n observations tirés de

la même distribution inconnue $p(z)$, où $z = (\mathbf{x},y)$ et de manière indépendante (*i.i.d : independant and identically distributed*).

Le but de l'apprentissage d'un modèle est de trouver une fonction f étant donné un ensemble de points de données de la forme $(\mathbf{x},f(\mathbf{x}))$. L'apprentissage supervisé suppose l'existence d'une **fonction de perte** $L : X \times Y \times Y \rightarrow \mathbb{R}^+$, avec X le domaine de $\mathbf{x}$ et Y le co-domaine de y et de f. La quantité $L(z,f)$ est un réel non négatif indiquant le désaccord en un point de donnée entre la valeur de sortie réelle et la valeur de sortie issue de l'apprentissage. La fonction de perte détermine comment l'erreur est pénalisée en un point spécifique de donnée. Pour considérer tous les erreurs au niveau de tous les points de données, on doit trouver une méthode qui combine toutes les pénalités locales. La **fonction de risque** est ainsi définie dans cet objectif. À une fonction f -solution du problème d'apprentissage- est associé un risque $R(f)$ défini comme l'espérance de la fonction de perte :

$$R(f) = E(L(z,f)) = \int L(z,f)p(z)dz.$$

Puisque la distribution $p(z)$ est inconnue, le risque doit être calculé d'une manière empirique en utilisant l'échantillon D de taille n. Ainsi, on définit la fonction de risque empirique par :

$$\hat{R}(f,D) = \frac{1}{n}\sum_{i=1}^{n} L(z_i,f).$$

Le processus de construction d'un modèle à partir d'un ensemble de données d'entraînement s'appelle *apprentissage inductif*. Le but de l'induction est de trouver une approximation $\hat{f}$ de $\mathcal{F}$ à partir de D qui minimise $\hat{R}(f,D)$, avec $\mathcal{F}$ est l'ensemble de solutions possible de f. Le modèle $\hat{f}$ construit par apprentissage inductif peut être utilisé comme abstraction de l'échantillon D ou pour faire des prédictions sur des observations avec des valeurs de sortie inconnues.

Le processus de prédiction peut être une *régression* (voir la section 2.2.3) ou une *classification* (voir la section 2.2.4). L'exactitude ou la capacité prédictive d'un modèle construit pour faire des prédictions est appelée aussi *performance de généralisation*. Elle est habituellement mesurée en utilisant une fonction de mesure d'erreurs et un ensemble d'observations appelées *données de test* prises de la même distribution que les données d'entraînement et pour lesquelles les valeurs de sortie sont également connues. Une estimation précise de l'exactitude (la capacité prédictive) nécessite aussi une méthode permettant de mieux utiliser les données, pour une évaluation plus fiable. Nous présentons en détails ces mesures

et ces méthodes d'évaluation de la capacité prédictive des modèles de prédiction dans les sections 2.2.6 et 2.2.5.

2.2.3 Régression

La régression est la tâche qui permet de prédire la valeur d'un attribut numérique continu, qualifiée de variable (ou attribut) de sortie, en fonction d'autres attributs qualifiés de variables d'entrée :

$$\hat{f}(\mathbf{x}) = \hat{y} \tag{2.1}$$

Exemple tiré de COCOMO I

Dans le but de prédire l'effort nécessaire pour le développement d'un logiciel, un estimateur, ou modèle de prédiction de l'effort E, peut être celui de COCOMO I [Boehm, 1981] :

- Si Type de logiciel = *organique*
  alors $\hat{E} = 2{,}4\, taille^{1.05}$
- Si Type de logiciel = *semi détaché*
  alors $\hat{E} = 3.0\, taille^{1.12}$
- Si Type logiciel = *embarqué*
  alors $\hat{E} = 3.6\, taille^{1.20}$

Soit un projet de gestion de compte bancaire de type *organique* et qui nécessite *a priori* un code de taille estimé à 25 000 lignes de code, l'effort nécessaire de développement est $\hat{E} = 2.4(25)^{1.05} = 70$ $homme-mois$.

Pour une régression, la fonction de perte est définie par :

$$L(z,f) = L((\mathbf{x},y),f) = (f(\mathbf{x}) - y)^2$$

Ainsi, sa fonction de risque empirique est définie par :

$$\hat{R}(f,D) = \frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x}_i) - y_i)^2.$$

Pour estimer l'exactitude des prédictions (la capacité prédictive) d'un modèle de régression $\hat{f}$, une fonction de mesure d'erreur est appliquée sur un ensemble de test de m observations. La fonction la plus

répandue dans le cas de la régression est l'erreur moyenne quadratique (EMQ), donnée par l'équation 2.2.

$$EMQ = \frac{1}{m}\sum_{i=1}^{m}(\hat{f}(\mathbf{x}_i) - y_i)^2. \tag{2.2}$$

2.2.4 Classification

La classification est la tâche qui permet de prédire, en fonction des attributs d'entrée, une valeur d'un attribut de sortie qui prend ses valeurs dans un ensemble fini C d'étiquettes (labels), appelées aussi *classes* (voir l'équation 2.3).

$$\hat{f}(\mathbf{x}) = \hat{y} \in C. \tag{2.3}$$

Considérons la prédiction de la qualité du logiciel en utilisant le nombre de défauts dans un module ou dans un composant. La majorité des composants contiennent zéro ou très peu de défauts, ce qui engendre une distribution du nombre de défauts complètement déséquilibrée. Dans ce cas, au lieu de prédire le nombre de défauts potentiels dans un composant logiciel, on détermine seulement si un composant est susceptible de contenir des défauts ou non. Ainsi, on prédit la classe à laquelle le composant appartient (à la classe des composants à haut risque ou à celles à faible risque de contenir des défauts), de sorte que le modèle de prédiction est réduit à un modèle de classification binaire. L'ensemble des valeurs possibles de sortie $C = \{haut\ risque, faible\ risque\}$. Pour prédire à quelle classe de C appartient un composant logiciel, on a besoin d'un modèle de prédiction, ou un classificateur. Par exemple,

$$\begin{cases} Si\ Complexité > s_c \vee Effort\ de\ test < s_t\ Alors\ y = haut\ risque \\ Sinon\ y = faible\ risque. \end{cases}$$

avec s_c et s_t respectivement un seuil de *Complexité* et un seuil de l'*Effort de test*.

Pour une classification, la fonction de perte est définie par :

$$L(z,f) = L((\mathbf{x},y),f) = I_{\{f(x)\neq y\}}.$$

Ainsi, sa fonction de risque empirique est définie par :

$$\hat{R}(f,D) = \frac{1}{n}\sum_{i=1}^{n} I_{\{f(x)\neq y\}}.$$

avec $I_{\{a\neq b\}}$ est 0 quand a et b sont égaux, et 1 autrement.

Comme pour la régression, il y a des fonctions de mesure d'erreurs associées à la classification $\hat{f}$. La fonction la plus connue de mesure de la capacité prédictive d'un classificateur est calculée sous forme du pourcentage des observations mal classifiées dans ensemble de test de taille m (voir l'équation 2.4).

$$Erreur\ de\ Classification = \frac{1}{m}\sum_{i=1}^{m} I_{\{\hat{f}(x)\neq y\}}. \tag{2.4}$$

2.2.5 Méthodes d'évaluation des modèles

Toutes les méthodes d'évaluation de la performance d'un modèle se basent sur un processus de prélèvement, dans le but de former à partir d'un nombre d'observations disponibles (ayant toutes des valeurs de sortie), deux ensembles de données. Le premier constitue l'ensemble d'entraînement qui va servir à la construction du modèle, le deuxième constitue l'ensemble de test qui va servir à l'évaluation du modèle produit. Par la suite, la capacité prédictive de ce dernier est déterminée en utilisant une fonction de mesure d'erreurs comme celles décrites dans la section 2.2.6. Cette section décrit trois des méthodes les plus connues d'évaluation de modèles, à savoir, la validation croisée de « k-fold » ([Efron, 1979; Stone, 1977]), la validation croisée de Monte Carlo et le *bootstrapping* ([Rao et Tibshirani, 1997; Shao, 1996]). Les deux premières méthodes seront employées dans le présent travail.

2.2.5.1 Validation croisée « *k-fold* »

Dans cette méthode, l'ensemble des observations disponibles est aléatoirement divisé en k partitions [Efron, 1979]. L'ensemble de test est constitué en mettant de côté une des k partitions. Les $k-1$ partitions restantes constituent l'ensemble d'entraînement. De cette façon, on crée k ensembles d'entraînement et k ensembles de test. Chaque fois qu'un modèle est construit sur un ensemble d'entraînement, sa capacité prédictive est mesurée sur l'ensemble de test correspondant. La capacité prédictive moyenne (moyenne des k performances de généralisation) servira comme une estimation de la capacité prédictive du modèle.L'avantage de l'utilisation de cette validation croisée réside dans l'augmentation de la variance de l'estimé de l'erreur (de test) avec le nombre d'échantillons de test (partitions) tout en gardant le plus possible d'observations pour l'entraînement.

2.2.5.2 Bootstrapping

Dans cette méthode, l'ensemble d'entraînement est formé d'un échantillon (sous ensemble d'observations disponibles) sur lequel on fait des remplacements à partir de l'ensemble des observations disponibles [Efron, 1979; Efron et Tibshirani, 1993]. L'ensemble du test est formé de la même façon. Dans des versions du *bootstrapping*, l'ensemble d'entraînement est constitué de toutes les observations disponibles sur lesquelles on fait les remplacements (c'est-à-dire que chaque observation disponible est considérée zéro ou plusieurs fois) et l'ensemble du test est formé de l'ensemble entier des observations disponibles sans remplacements. Des versions récentes préfèrent la formation de l'ensemble du test par des observations disponibles qui ne participent pas à l'ensemble d'entraînement, voir [Rao et Tibshirani, 1997] et [Shao, 1996].

2.2.5.3 Validation croisée de Monte Carlo

La validation croisée de Monte Carlo est un cas particulier de la validation *bootsrapping*. L'ensemble d'entraînement est constitué en général de $2/3$ des observations disponibles choisies aléatoirement et l'ensemble du test est formé du reste. La capacité prédictive du modèle produit est calculée sur $1/3$ des observations disponibles formant l'ensemble du test. Ce processus est généralement répété plusieurs fois et l'évaluation des algorithmes est faite en utilisant la moyenne des capacités prédictives.

2.2.6 Les fonctions d'évaluation

Plusieurs fonctions d'évaluation de la performance des modèles de prédictions sont proposées dans la littérature. Nous trouvons entre autres, l'erreur moyenne quadratique (voir équation 2.2 dans la section 2.2.3). Toutefois, nous nous intéressons plus dans cette section aux mesures utilisées en génie logiciel.

Dans ce domaine, Conte et ses collaborateurs ont proposé *l'erreur relative* (ER) comme mesure de l'inexactitude d'une prédiction [Conte et *al.*, 1986], $ER = \frac{A-E}{A}$, avec, A est la valeur réelle de la sortie du modèle et E est sa valeur estimée. Pour calculer cette erreur sur un ensemble d'observations (du test) de taille m, on calcule *l'erreur relative moyenne*

$$\overline{ER} = \frac{1}{m}\sum_{i=1}^{m} ER_i \tag{2.5}$$

Dans cette formule, certaines petites valeurs de l'erreur relative peuvent annuler (ou compenser) l'effet d'autres grandes valeurs. Autrement dit, les erreurs de sous-estimation peuvent annuler les erreurs de sur-estimation. C'est pourquoi on a défini la valeur absolue de ER comme *la magnitude de l'erreur relative* (MER). Par la suite on a défini *la magnitude moyenne de l'erreur relative* par la formule de la moyenne suivante:

$$\overline{MER} = \frac{1}{m}\sum_{i=1}^{m} MER_i. \tag{2.6}$$

Si $\overline{MER}$ est petite alors les prédictions sont bonnes. Conte et ses collaborateurs suggèrent que $\overline{MER}$ soit inférieure à 25% pour parler d'une prédiction acceptable. Ils définissent ainsi une autre mesure de l*a qualité de prédiction* qui peut exprimer leur suggestion. En effet, soit m le nombre d'observations qui vont servir à l'évaluation de la prédiction d'un modèle et soit k le nombre d'observations dont la magnitude moyenne de l'erreur relative ($\overline{MER}$) est inférieure ou égale à un seuil q, *la qualité de prédiction* notée $PRED$ est alors définie par $PRED(q) = \frac{k}{m}$. Par exemple, $PRED(0{,}25) = 0{,}9$ indique que 90% des valeurs prédites avec un ajustement d'au plus 25% font partie de l'ensemble de leurs valeurs réelles. Ainsi, Conte et ses collaborateurs peuvent exprimer leur suggestion en affirmant qu'un modèle est acceptable si sa qualité de prédiction $PRED(0{,}25)$ est au moins égale à 0,75.

Les précédentes fonctions sont généralement utilisées pour évaluer la qualité des prédictions de type régression. Dans le cas de classification, plusieurs mesures sont proposées dans la littérature. El Emam a présenté la plupart de ses mesures dans [El Emam, 2000], parmi lesquelles nous citons le test **chi-square** (χ^2) [Almeida et *al.*, 1998; Basili et *al.*, 1997], la sensitivité et la spécificité [Almeida et *al.*, 1998], la proportion correcte (appelée aussi exactitude dans [Porter et Selby, 1990b] et [Porter, 1993b]), l'erreur de classification type I et type II [Khoshgoftaar et *al.*, 1995a; Khoshgoftaar et *al.*, 1996b], le taux correct positif [Briand et *al.*, 1998] et Kappa (κ) [Briand et *al.*, 1998; Briand et *al.*, 2000]. Ces mesures sont utilisées surtout dans les cas de classification binaire. Elles sont définies en fonction d'un nombre de variables exprimant des résultats intermédiaires d'évaluation de la classification sous forme d'une notation connue sous le nom de **matrice de confusion**.

		Classes prédites		
		$classe_1$	$classe_2$	
Classes réelles	$classe_1$	n_{11}	n_{12}	N_{1*}
	$classe_2$	n_{21}	n_{22}	N_{2*}
		N_{*1}	N_{*2}	N

TAB. 2.1 – *Matrice de confusion d'une classification binaire à deux classes, $classe_1$ et $classe_2$*

Nous supposons que nous évaluons un modèle de classification binaire sur un ensemble de test contenant N observations. La matrice de confusion correspondante est données par le tableau 2.1 où n_{ij} est le nombre d'observations qui sont réellement classifiées en tant que $classe_i$ mais prédites $classe_j$, N_{i*} est le nombre de toutes les observations réellement classifiées en tant que $classe_i$ et N_{*i} est le nombre de toutes les observations prédites $classe_i$.

Le tableau 2.2 présente une brève description de chacune des mesures précédemment énumérées. Pour permettre une meilleure lecture, nous allons considérer la prédiction de la stabilité des composants logiciels en utilisant une classification à deux classes de composants logiciels respectivement, *stable* ($classe_1$) et *instable* ($classe_2$).

Mesure	Description	Définition	Utilisation
La sensibilité	Proportion des composants instables qui ont été correctement classifiés en tant que composants instables	$s = \frac{n_{22}}{n_{21}+n_{22}}$	[Almeida et *al.*, 1998]
La spécificité	Proportion des composants stables qui ont été correctement classifiés en tant que composants stables	$p = \frac{n_{11}}{n_{11}+n_{12}}$	[Almeida et *al.*, 1998]
Proportion correcte (exactitude)	Proportion des composants correctement classifiés	$C = \frac{n_{11}+n_{22}}{N}$	[Porter, 1993b]
Fausses classifications	Taux des composants mal classifiés de Type I et Type II		
Type I	Proportion des composants stables qui ont été classifiés instables	$(I) = 1 - p$	[Khoshgoftaar et *al.*, 1995a]
Type II	Proportion des composants instables qui ont été classifiés stables	$(II) = 1 - s$	[Khoshgoftaar et *al.*, 1996b]
Taux correct positif	Proportion des composants correctement prédits instables	$VTP = \frac{n_{22}}{N_{\cdot 2}}$	[Briand et *al.*, 1998] [Briand et *al.*, 2000]
Kappa	Généralement employé pour mesurer l'accord entre les proportions (réelles et prédites), voir ([El Emam et *al.*, 2001a])	$\kappa = \frac{\frac{n_{11}+n_{22}}{N} - \sum_{i=1}^{2} \frac{N_{i\cdot}N_{\cdot i}}{N^2}}{1 - \sum_{i=1}^{2} \frac{N_{i\cdot}N_{\cdot i}}{N^2}}$	[El Emam et *al.*, 2001a], [Cohen, 1960], [Briand et *al.*, 2000]
Taux d'erreur équilibré	C'est la moyenne des proportions des composants correctement classifiés de toutes les classes	$TEE = \frac{s+p}{2}$	[El Emam, 2000] , [El Emam et *al.*, 2001b]
Indice J de Youden	appelé aussi **J-index** est une transformation linéaire de TEE	$J = s + p - 1$	[Youden, 1961]

TAB. 2.2 – *Mesures d'évaluation de la classification binaire*

2.3 L'optimisation combinatoire

2.3.1 Introduction

Cette section décrit les techniques d'optimisation combinatoire utilisées dans cette dissertation. La résolution d'un problème d'optimisation combinatoire consiste à explorer un espace de recherche afin d'optimiser une ou plusieurs fonctions données, appelées aussi *critères d'optimisation*, ou *fonctions objectifs*. La complexité de la taille et de la structure de l'espace de recherche et le nombre de fonctions à optimiser conduisent à l'utilisation des méthodes de résolutions différentes. Dans la littérature, nous trouvons deux familles de techniques d'optimisation combinatoire : *les méthodes exactes* et les *méthodes heuristiques* [Talbi, 2001]. Les méthodes exactes garantissent à tout coup de trouver la meilleure solution. Cependant, ces techniques ne sont pas efficaces devant la complexité d'un problème ou devant le nombre de critères d'optimisation. Elles peuvent dans certains cas ne pas ébaucher une solution dans un temps de résolution réaliste. Parmi ces méthodes, nous trouvons principalement la méthode *Branch and Bound* et les algorithmes A^*, voir [Talbi, 2001]. En revanche, les méthodes heuristiques ne garantissent pas de trouver la meilleure des solutions, mais plutôt une solution approchée de la solution optimale dans un temps raisonnable. Elles sont primordiales pour résoudre les problèmes de grandes tailles. Ces méthodes peuvent être divisées en deux classes. La première représente les algorithmes spécifiques à un problème donné qui utilisent les connaissances du domaine [Gandibleux et *al.*, 1994], la deuxième s'intéresse aux métaheuristiques qui sont des algorithmes généraux applicables à une grande variété de problèmes d'optimisation.

Dans cette dissertation, notre intérêt porte sur trois métaheuristiques réputées efficaces et largement utilisées dans la résolution des problèmes combinatoires complexes. Elles représentent à leur tour, deux familles de métaheuristiques: celle basée sur la population, comprenant les algorithmes génétiques, et celle basée sur la recherche locale (ou sur le voisinage), comprenant le recuit simulé et la recherche avec tabous. La justification détaillée du choix de ces techniques est liée aux problèmes de la prédiction de la qualité du logiciel traités dans ce travail et est présentée dans le chapitre 4.

2.3.2 Les algorithmes génétiques

Les algorithmes génétiques (AG) ont été proposés par Holland [1975]. Leur fonctionnement est inspiré de mécanismes biologiques basés sur les critères de sélection naturelle. Dans une population d'une espèce donnée, les individus ont des capacités différentes à s'adapter à un environnement quelconque ; on dit alors que la population n'est pas uniforme. Ce phénomène se produit parce que certains individus sont mieux préparés que d'autres pour survivre et ils ont par conséquent plus de chances de se reproduire. Via la reproduction, les « bonnes » caractéristiques du père et de la mère sont transmises aux enfants et la nouvelle population sera globalement mieux adaptée à l'environnement que la population ancienne. C'est en recombinant les « bonnes » caractéristiques des « bons » individus et avec un peu de hasard que les générations évoluent vers une meilleure adaptation à l'environnement. Ce principe d'évolution des espèces est imité dans la recherche de l'optimum d'une fonction objectif. Cette fonction est considérée analogue à l'aptitude d'un individu à s'adapter.

2.3.2.1 Principe des algorithmes génétiques

Comme tout algorithme d'optimisation, un AG cherche le minimum (ou le maximum) d'une fonction objectif définie dans un espace de solutions. Le processus de l'AG parcourt l'espace de solutions, en partant d'un ensemble de solutions admissibles appelé *population initiale* (G_0) et produit itérativement plusieurs générations de solutions potentielles. À chaque étape, pour passer d'une génération G_t à la génération G_{t+1}, les opérations suivantes sont répétées pour les éléments (solutions) de la génération G_t :

- une évaluation de tous les éléments est effectuée pour connaître leurs niveaux d'aptitude respectifs ;
- une opération de croisement est appliquée avec une probabilité p_c sur des couples parents sélectionnés en fonction de leurs aptitudes, cette opération génère des couples enfants héritiers ;
- une opération de mutation est appliquée avec une probabilité p_m généralement inférieure à p_c, sur des éléments parents, elle génère des enfants mutés ;
- Les enfants issus du croisement et de la mutation sont intégrés dans la nouvelle génération G_{t+1}.

Le processus s'achève lorsqu'un critère d'arrêt est rencontré. Ce critère peut être un nombre maximal de générations que l'on veut exécuter dans un temps raisonnable ou une convergence vers une solution. La convergence vers une solution survient lorsque un nombre d'itérations consécutives est effectué sans amélioration de la solution. Le principe de l'AG est résumé sur la figure 2.1. L'utilisation de l'AG nécessite la définition des mécanismes suivants que nous détaillons par la suite :

1. Un codage des éléments d'une population.
2. Un choix de la population initiale.
3. Une fonction d'évaluation des individus, ou fonction de *fitness*.
4. Un opérateur de croisement.
5. Un opérateur de mutation.
6. Un principe de sélection d'individus.

```
ALGORITHMEGENETIQUE(p_c, p_m, ...)
 1  Initialiser G_0
 2  MEILLEUR ← meilleur chromosome de G_0
 3  MEILLEURSOL ← MEILLEUR                ▷ initialiser la meilleure solution
 4  pour t ← 0 à T
 5    Q ← paires parmi les meilleurs membres de la génération G_t
 6    Q' ← enfants des paires dans Q en utilisant le croisement et la mutation
 7    remplacer les membres les plus faible de G_t par Q' pour créer G_{t+1}
 8    MEILLEUR ← meilleur chromosome in G_{t+1}
 9    si MEILLEURSOL est plus fort que MEILLEURSOL alors
10      MEILLEURSOL ← MEILLEUR
11  retourner MEILLEURSOL
```

FIG. 2.1 – *Résumé de l'algorithme génétique.*

2.3.2.2 Codage des éléments d'une population

L'objectif de cette opération est d'associer à chaque point de l'espace de recherche, une structure de données capable de le décrire et de le distinguer des autres. La qualité du codage des solutions détermine le succès des algorithmes génétiques [Goldberg, 1989]. Ainsi, chaque solution est codée d'une manière qui reflète sa nature. Le codage a pour intérêt de permettre de bien définir ultérieurement, des opérateurs de croisement et de mutation simples. Généralement, chaque solution est représentée par une structure modulaire appelée *chromosome* : un composant dans un chromosome constitue un *gène*.

Traditionnellement, le codage en bits est le plus reconnu, cependant, il n'est pas efficace dans certains cas. Par exemple, deux codes voisins (en terme de distance de Hamming) peuvent correspondre à des éléments qui ne sont pas nécessairement proches. Cet inconvénient peut être évité en utilisant un codage de **Gray** par exemple. Aussi, dans des cas où l'espace de recherche est de grande dimension, chaque variable est représentée par une partie de la chaîne de bits, ce qui complique la structure du problème qui ne sera plus reflétée simplement. Afin de conserver les variables et la structure du problème, on utilise généralement, des vecteurs réels pour coder les solutions [Goldberg, 1991; Wright, 1991]. Toutefois certains problèmes imposent des codages plus ou moins complexes pour bien refléter la structure des solutions.

2.3.2.3 Choix de la population initiale

Ce choix est fait dans l'objectif de produire une population non homogène (non uniforme) d'individus qui servira de base pour les générations futures. Si la solution optimale est complètement inconnue dans l'espace de recherche, on procède à une génération aléatoire des membres de la population initiale. Alors que si certaines informations sur la solution recherchée sont disponibles, la génération de la population initiale va concerner des individus favorisant une convergence plus rapide [Goldberg, 1989].

2.3.2.4 Fonction d'évaluation

Elle évalue les chromosomes en fonction de leurs aptitudes à s'adapter dans un environnement. L'évaluation de l'aptitude d'un chromosome doit être indépendante de celles des autres [Holland, 1975]. La fonction d'évaluation est appelée fonction de *fitness*, car elle donne à chaque chromosome une valeur qui correspond à sa force de résister pour survivre dans un environnement. Elle permet de sélectionner ou de refuser un individu pour ne garder que les individus les plus forts dans la population courante. Par exemple, pour le Problème du Voyageur de Commerce (PVC), la fonction d'évaluation utilisée calcule la distance parcourue par le commis voyageur pour un chemin donné. Plus la distance est courte, plus le parcours est meilleur car moins coûteux.

2.3.2.5 Opérateur de croisement

Cet opérateur est appliqué dans le but de diversifier la population. Une opération de croisement est effectuée entre deux parents, en combinant ensemble les gènes de l'un et de l'autre. Deux types de croisement sont le plus souvent utilisés :

- le croisement par découpage de chromosomes (appelé *slicing crossover* dans [Barnier et Brissct, 1998]). Il consiste à découper en une position aléatoire ou fixe, chacun des chromosomes parents ensuite à échanger les morceaux des deux parents pour produire deux chromosomes enfants. Le découpage peut être fait en plusieurs positions, ou points de coupure. Ce type de croisement est efficace pour les problèmes discrets. La figure 2.2 montre un exemple de croisement à deux points de coupure.

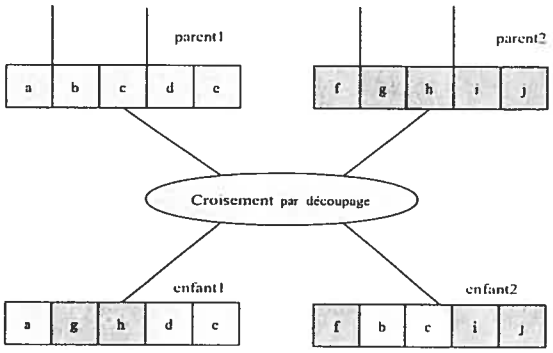


FIG. 2.2 – *Croisement par découpage de chromosomes (slicing crossover).*

- le « croisement barycentrique » est un croisement efficace pour les problèmes continus [Louchet et *al.*, 2001]. Il consiste à combiner linéairement les gènes de deux parents de la façon suivante : deux gènes $G_{P1}(i)$ et $G_{P2}(i)$ sont pris à la même position i chez chacun des parents. Leur croisement donne naissance à deux nouveaux gènes enfants $G_{E1}(i)$ et $G_{E2}(i)$ après une combinaison linéaire définie par : $\begin{cases} G_{E2}(i) = (1-\alpha)G_{P1}(i) + \alpha G_{P2}(i), \\ G_{E1}(i) = \alpha G_{P1}(i) + (1-\alpha)G_{P2}(i), \end{cases}$
où α est une pondération choisie aléatoirement, il est généralement compris entre 0 et 1.

2.3.2.6 Opérateur de mutation

Cet opérateur est appliqué dans le but de garantir l'exploration de l'espace de recherche. En progressant, l'opérateur de croisement seul devient moins efficace, car les individus deviennent similaires et la population est plus uniforme. C'est à ce moment là que l'opérateur de mutation prend la responsabilité de diversifier la population et d'atteindre plus de solutions. Pour un problème discret, la mutation standard consiste à sélectionner aléatoirement un gène dans un chromosome et à le remplacer par un autre qui lui est différent (voir la figure 2.3). Dans le cas d'un problème continu, le gène sélectionné est modifié en lui ajoutant un bruit [Louchet et *al.*, 2001]. Théoriquement, cet opérateur influence énormément la convergence de l'AG.

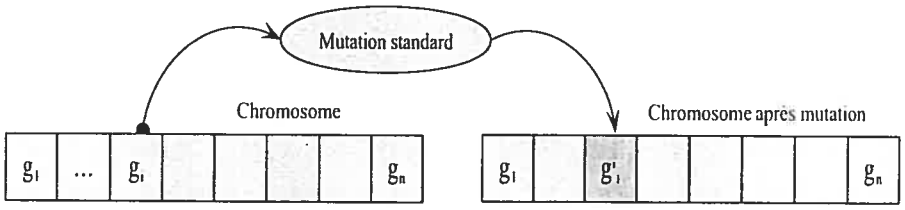


FIG. 2.3 – *Principe de la mutation (g_i est le gène à modifier).*

2.3.2.7 La sélection

La sélection a comme objectif de choisir les individus de la population qui doivent survivre et d'éliminer les mauvais. On trouve plusieurs principes de sélection plus ou moins adaptés aux problèmes qu'ils traitent :

Sélection par roulette de casino Comme son nom l'indique, c'est une sorte de roulette de casino sur laquelle sont placés tous les chromosomes de la population. La place accordée à chacun des chromosomes est proportionnelle à sa force (*fitness*), de cette manière les bons chromosomes occupent les places les plus grandes [Falkenauer, 1998]. La figure 2.4 illustre ce principe avec un exemple d'une population de neuf chromosomes (individus). Ainsi, une bille lancée a plus de chance de tomber sur les bons chromosomes, qui seront sélectionnées pour un croisement ou une mutation que sur les mauvais. Un aspect important de la sélection par roulette de casino est qu'elle permet de choisir les chromosomes d'une manière probabiliste, non pas d'une manière déterministe. En effet le chromosome 5 dans

la figure 2.4 possède la force la plus élevée de la population, mais il n'y a aucune garantie qu'il soit réellement sélectionné à un moment donné. On est cependant sûre qu'en moyenne les chromosomes vont être choisis proportionnellement à leurs forces.

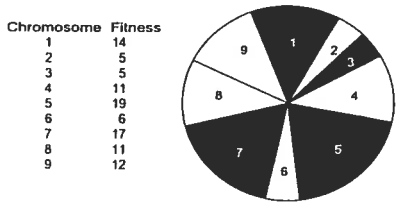


FIG. 2.4 – *Principe de la sélection par roulette du casino.*

Sélection par rang Lorsque la valeur de *fitness* des chromosomes varie énormément, la sélection par roulette de casino pose le problème suivant : le chromosome le plus fort qui occupe par exemple, 80% de la place sur la roulette fait en sorte que les autres chromosomes auront très peu de chance d'être sélectionnés, ce qui a pour effet de stopper l'évolution. La sélection par rang [Falkenauer, 1998] commence par trier la population des N chromosomes selon la *fitness* et se poursuit en attribuant un rang à chaque chromosome égal à sa position après le tri. Ainsi, le plus faible chromosome aura le rang 1 et le plus fort aura le rang N. Par la suite, la sélection par roulette est appliquée, mais le rang est considéré à la place de la *fitness*. La place occupée par un chromosome est proportionnelle à son rang. Avec la sélection par rang, l'écart entre les chances est réduit et tous les chromosomes ont une chance d'être sélectionnés. Cependant, la convergence sera plus lente.

Sélection « steady-state » Cette sélection est plutôt une stratégie de renouvellement de la population [Falkenauer, 1998]. À chaque génération de N individus, les n meilleurs chromosomes, $n < N$, sont sélectionnés, pour créer des chromosomes enfants. Ensuite, ces derniers prennent la place des chromosomes les plus faibles dans la génération suivante. Le reste de la population survit à la nouvelle génération.

Élitisme On utilise la stratégie d'élitisme [Holland, 1975] dans le but de ne pas perdre les meilleurs chromosomes de l'ancienne génération après les opérations de croisement et de mutation. L'élitisme consiste en fait à cloner un ou plusieurs de ces meilleurs chromosomes et à les mettre dans la nouvelle

génération. Le reste de la population subit les opérations usuelles de la reproduction. Cette méthode accélère la convergence de l'AG et peut être utilisée avec d'autres méthodes de sélection.

2.3.2.8 Les paramètres d'un algorithme génétique

Les algorithmes génétiques peuvent être contrôlés à travers différents paramètres dont les plus importants sont : la taille de la population, la probabilité de croisement et la probabilité de mutation. La détermination de ces paramètres dépend étroitement de la nature du problème auquel est appliqué l'AG. On sait que la taille de la population peut être fixe ou variable pendant l'exécution de l'AG et que la probabilité de mutation est faible ($< 0{,}2$) par rapport à celle du croisement ($> 0{,}5$). Les bonnes valeurs de ces paramètres sont fixées empiriquement, après un nombre important d'exécutions.

2.3.3 Le recuit simulé

Le recuit simulé (RS) a été proposé par Kirkpatrick et ses collaborateurs [1983] lorsqu'ils ont montré l'analogie entre la recherche d'une solution optimale en optimisation combinatoire et la recherche de l'état d'équilibre thermique d'un solide (énergie minimale) en thermodynamique. Cette méthode est ainsi inspirée de celle de Métropolis [1953] qui était utilisée pour modéliser l'évolution d'un solide vers son état d'équilibre thermique.

2.3.3.1 Principe du recuit simulé

Dans le domaine de la métallurgie, l'état de la matière dépend de la température : elle est en général à l'état liquide à haute température et à l'état solide à basse température. Si on souhaite obtenir un solide, on procède à un abaissement de la température. Deux façons de diminuer la température sont possibles. La première est une baisse très brusque, on obtient ainsi un « verre », caractéristique de la technique de « trempe ». Il s'agit d'une structure avec un minimum local d'énergie. La deuxième est une baisse progressive de la température, laissant le temps aux atomes d'atteindre l'équilibre, on tendra alors vers une structure de plus en plus régulière avec un minimum global d'énergie, en obtenant ainsi un cristal. Si l'abaissement n'est pas assez progressif, on obtient une structure avec des défauts. Ceci peut être corrigé par un réchauffement léger de la matière de façon à permettre une mobilité aux atomes qui finissent par de retrouver une configuration plus stable. Ce réchauffement est connu sous le nom de *recuit*.

Le recuit simulé cherche donc à imiter le processus précèdent dans la recherche d'un minimum global d'une fonction objectif. Il fait une analogie, d'une part, entre la fonction objectif C et l'énergie E d'un solide et, d'autre part, entre un paramètre global de contrôle Trs (pseudo-température) et la température de recuit T. À partir d'une solution initiale s_0 et d'une température initiale Trs_0, l'algorithme RS commence à explorer toutes les solutions en procédant par étapes. À chaque étape, l'algorithme fait des transitions de la solution courante dans son voisinage $N(s)$ (voir la section 2.3.3.2). Après une transition, on mesure la variation ΔC de la fonction C. Si elle est négative (la fonction C diminue), alors la nouvelle solution est acceptée, sinon (la fonction C augmente), la nouvelle solution est acceptée avec une probabilité d'acceptation p. Cette acceptation est utilisée pour s'échapper des optima locaux, elle est analogue à la phase du réchauffement (recuit) effectué pour corriger les défauts de la matière. La probabilité d'acceptation p est basée sur la fonction de Boltzman ($e^{\frac{-\Delta C}{Trs}}$).

Après un nombre de transitions Nrs jugé suffisant pour atteindre tout le voisinage de la solution courante (toutes les configurations de la structure à cette température), la température est abaissée à une température Trs_{k+1}, selon une stratégie appelée *schéma de décroissance de la température*. Le processus s'arrête lorsque l'on atteint la température de recuit (inférieure à une valeur arbitraire égale à 1) où quand l'évolution de la solution n'est plus possible. Le résumé de l'algorithme de RS est donné par la figure 2.5. L'utilisation de l'algorithme de RS nécessite la définition d'une fonction de transition afin de parcourir l'espace de recherche et la spécification d'un nombre de paramètres, comme la température initiale, le critère d'acceptation de la transition, le schéma de décroissance de la température, la longueur de température et le critère d'arrêt.

2.3.3.2 Fonction de transition

Une transition est une fonction qui associe à une solution s, une solution voisine s' avec $s' \in N(s)$, où $N(s)$ est le voisinage de s. La définition des transitions d'une solution est une tâche spécifique à la nature du problème traité. Dans la littérature, il est conseillé de choisir des transitions qui modifient légèrement la solution [Kirkpatrick et *al.*, 1983].

```
RECUITSIMULE(Trs0,Nrs,...)
 1    Choisir s0                          ▷ choisir une solution initiale
 2    Trs ← Trs0                          ▷ initialiser la température
 3    STOP ← faux
 4    s ← s0
 5    MEILLEURSOL ← s0                    ▷ initialiser la meilleure solution
 6    tant que nonSTOP
 7        pour i ← 1 Nrs                  ▷ début du pas de température
 8            Générer s' ∈ N(s)           ▷ transiter dans le voisinage de s
 9            ΔC ← C(s') − C(s)           ▷ calculer la variation de la fonction C
10            si ΔC ≤ 0
11                alors s ← s'            ▷ accepter la solution
12                sinon Générer un nombre aléatoire r ∈ [0,1]
13                    si r ≤ e^(−ΔC/Trs)
14                        alors s ← s'
                              ▷ accepter la solution en utilisant
                              ▷ le critère de Metropolis
15            si C(s') ≤ C(MEILLEURSOL) alors MEILLEURSOL ← s'
16        Trs ← α.Trs                     ▷ abaisser la température
17        si critère d'arrêt alors STOP ← vrai
18    retourner MEILLEURSOL
```

FIG. 2.5 – *Résumé d'un algorithme du recuit simulé.*

2.3.3.3 Température initiale

C'est le paramètre qui détermine le nombre de solutions qui vont être acceptées initialement. Il doit être suffisamment élevé pour favoriser l'acceptation d'un grand nombre de mauvaises solutions (solutions avec valeur de C élevée). La valeur de la température initiale Trs_0 est généralement choisie de manière à avoir un taux d'acceptation initial compris entre 50% et 80%. Par exemple $Trs_0 = r.\max(\Delta C)$ avec $r \gg 1 (= 10)$.

2.3.3.4 Schéma de décroissance de la température

La température est réduite progressivement au cours du processus selon une méthode de décroissance appelée *schéma de décroissance* ou encore *schéma de recuit*. Parmi les nombreux schémas de recuit qui ont été proposés deux sont fréquemment utilisés. Le premier est de type géométrique. Il consiste à multiplier l'ancienne température par une constante α positive inférieure à 1 (généralement,

α est comprise entre 0,9 et 1, ce schéma est donné par :

$$Trs_{k+1} = \alpha.Trs_k.$$

Le deuxième est de type arithmétique. Il est défini par :

$$Trs_{k+1} = Trs_k - \theta,$$

avec θ une constante positive.

D'autres schémas plus élaborés sont proposés, comme ceux d'Aarts et Van Laarhoven [1985].

2.3.3.5 Longueur de la température

C'est une traduction de *length of the temperature*, ou longueur de pas, (noté *Nrs* dans l'algorithme de la figure 2.5). Ce paramètre représente le nombre de solutions à visiter avant d'atteindre l'équilibre thermique. En optimisation, c'est le nombre de solutions qu'on peut atteindre à partir de la solution courante. La longueur de température est souvent fixée à une valeur déduite d'expériences préalables. Après avoir effectué ce nombre de transitions, on change la température en utilisant le schéma de décroissance de la température.

2.3.3.6 Critère d'acceptation des transitions

Les critères les plus connus sont ceux de Metropolis [1953] et de Glauber [1963]. Chacun des deux critères accepte les transitions avec une probabilité p différente :

- Pour Metropolis, $p = \begin{cases} 1 & \text{si } \Delta C \leq 0. \\ e^{\frac{-\Delta C}{Trs}} & \text{sinon.} \end{cases}$
- Pour Glauber, $p = \frac{e^{\frac{-\Delta C}{Trs}}}{1+e^{\frac{-\Delta C}{Trs}}}$.

  À noter que le critère de Glauber peut ne pas accepter les transitions qui diminuent le coût de la solution.

2.3.3.7 Critère d'arrêt

Comme pour les autres paramètres, il existe plusieurs choix du critère d'arrêt, les plus utilisés sont ceux de Dolan et de ses collaborateurs [1990], qui proposent de signaler la fin du processus soit lorsque

la température sera inférieure à 1 ou après un nombre de décroissances (10 par exemple) de la température sans amélioration de la solution ou encore lorsque la probabilité d'acceptation devienne inférieure à un certain seuil.

2.3.4 La recherche avec tabous

La recherche avec tabous (RT) a été proposée pour la première fois par Glover [1986]. Elle cherche un minimum, ou un maximum, d'une fonction objectif C qui est définie dans un espace de solutions. L'idée de base consiste à introduire la notion de mémoire dans la stratégie d'exploration des solutions. En effet, pour éviter de tomber dans le piège d'optima locaux, le processus RT interdit de reprendre des solutions récemment visitées. Il utilise pour cette raison une structure de mémorisation appelée *liste taboue* dans laquelle on garde les traces, ou les attributs des solutions interdites dites taboues.

2.3.4.1 Principe de la recherche avec tabous

Dans la RT, le parcours de l'espace des solutions se fait comme celui du recuit simulé par le biais des transitions de la solution courante. Une transition permet de passer d'une solution s à une autre s'. On dit que la nouvelle solution est une voisine de la précédente. L'ensemble des solutions que l'on peut atteindre à partir d'une solution s s'appelle le voisinage de s noté $N(s)$. L'intention originale derrière la liste taboue n'est pas d'empêcher une transition passée d'être répétée mais de s'assurer plutôt qu'elle ne soit pas inversée pendant un nombre ℓ d'itérations, où ℓ est appelé *longueur de la liste taboue*. Comme le montre l'algorithme de la figure 2.6, inspiré de [Ferland et Costa, 2001], le processus de la RT part d'une solution initiale et d'une liste taboue vide. À chaque itération, une meilleure voisine s' est choisie dans un voisinage BONVOISINS de la solution courante s. Chaque élément de BONVOISINS est obtenu par une transition qui satisfait au moins l'une des deux conditions suivantes :

- elle ne doit pas être taboue ;
- elle doit minimiser la fonction objectif C.

Par la suite, la meilleure solution et la liste taboue sont mises à jour (voir section 2.3.4.3). Le processus se termine lorsqu'un critère d'arrêt est rencontré. Nous remarquons que cet algorithme utilise un critère d'aspiration qui ne considère pas le caractère tabou pour une solution qui minimise la fonction objectif C.

L'utilisation de la RT nécessite la définition d'une fonction de transition qui permet de parcourir l'espace de recherche et une stratégie de gestion de la liste taboue et une longueur maximale de cette liste. Quant à la fonction de transition, elle est généralement de la même nature que celle utilisée dans le recuit simulé. La gestion de la liste taboue et la spécification de sa longueur seront décrites dans les paragraphes qui vont suivre.

```
RECHERCHEAVECTABOUS(ℓ, ...)
 1   Choisir s_0                                  ▷ choisir une solution initiale
 2   LISTTABOUE ← ∅
 3   STOP ← faux
 4   s ← s_0
 5   MEUILLEURSOL ← s_0                           ▷ initialiser la meilleure solution
 6   tant que nonSTOP
 7       Déterminer un sous-ensemble BONVOISINS ⊂ N(s) de
             solutions obtenues par des transitions de s qui
             minimisent C ou qui ne sont pas taboues.
 8       Déterminer s' ∈ BONVOISINS tel que s' ← arg min_{z∈BONVOISINS} {C(z)}
 9       s ← s'
10       si C(s') ≤ C(MEUILLEURSOL) alors MEUILLEURSOL ← s'
11       Mettre à jour LISTTABOUE
12       si critère d'arrêt alors STOP ← vrai
13   retourner MEUILLEURSOL
```

FIG. 2.6 – *Résumé d'un algorithme de la recherche avec tabous.*

2.3.4.2 Longueur de la liste taboue

La longueur, ou taille ℓ de la liste taboue, est à déterminer empiriquement. Elle dépend étroitement du problème traité par la RT mais c'est une donnée primordiale. En effet, une liste trop petite peut conduire à un cycle de solutions explorées qui va se répéter indéfiniment, alors qu'une liste trop grande peut interdire des transitions intéressantes. En général, la liste taboue doit être maintenue à une longueur minimale permettant d'éviter un cycle.

2.3.4.3 Gestion de la liste taboue

La liste taboue est une structure cyclique qui permet de garder taboue une transition pour un nombre ℓ d'itérations. Sa gestion consiste à lui ajouter, à chaque itération, les attributs associés à une transition que l'on veut rendre taboue et à y supprimer des attributs associés à une transition que l'on veut autoriser

après ℓ itérations d'interdiction. Si nous supposons que chaque transition t est réversible, alors il existe une autre transition t^{-1} telle que $t^{-1}(t(s)) = s$. Dans ce cas, la liste taboue peut, par exemple, inclure les transitions inverses (t^{-1}) des transitions récemment utilisées (t). Ainsi, une transition t reste interdite tant que sa transition inverse (t^{-1}) est dans la liste taboue. Une transition est supprimée de la liste taboue après ℓ itérations de son insertion.

D'autres stratégies de gestion de la liste taboue sont envisageables. Par exemple, dans une stratégie on peut procéder à la mémorisation des meilleures solutions rencontrées afin d'extraire quelques propriétés communes pour définir les régions où la recherche est prometteuse. Dans ce cas, on rend taboues toutes les transitions qui font sortir de ces régions. On appelle cette façon de procéder, *stratégie d'intensification.* Dans une autre stratégie, on mémorise les solutions les plus fréquemment visitées, ainsi en rendant taboues certaines de ces transitions, on favorise les transitions les moins souvent utilisés. Il s'agit d'une *stratégie de diversification.*

2.3.4.4 Critère d'arrêt

Le critère d'arrêt pour la recherche avec tabous est spécifié généralement sous la forme d'un nombre maximale d'itérations ou en fonction d'un nombre maximale *Maxiter* d'itérations consécutives sans amélioration de la solution.

2.4 Conclusion

Afin de résoudre certains problèmes de prédiction de la qualité du logiciel, nous avons introduit dans ce chapitre deux domaines importants qui trouvent beaucoup d'applications dans le génie logiciel en général. Nous considérons que le domaine de l'apprentissage devient de plus en plus utilisé dans la construction de modèles prédictifs de la qualité. Cette utilisation est importante car elle permet aux concepteurs de disposer de diverses alternatives aux méthodes paramétriques de construction de modèles. Dans ce travail, nous explorons les utilisations de l'apprentissage dans le domaine de la qualité du logiciel, en particulier dans le chapitre 3. Le deuxième domaine introduit est celui de l'optimisation que nous allons utiliser pour améliorer la prédiction. Les techniques et les concepts de ce domaine sont redéfinis et adaptés selon les particularités du problème de la prédiction de de la qualité du logiciel que

nous traitons. Enfin, les formalismes et les notations empruntés à ces deux domaines seront utilisés à travers les chapitres de cette dissertation.

Chapitre 3

Évaluation et prédiction de la qualité du logiciel

3.1 Introduction

Dans ce chapitre, nous donnons un aperçu du domaine de la qualité du logiciel. L'étude de ce domaine comprend plusieurs centres d'intérêts. La prédiction de la qualité constitue l'intérêt principal de ce travail, auquel nous accordons une place importante dans ce chapitre. En effet, nous commençons dans la section 3.2 par une présentation de certaines définitions relatives aux différents points de vue de la qualité du logiciel. Par la suite dans la section 3.3, la métrologie est présentée comme la pierre angulaire de la qualité du logiciel, nous l'abordons à travers ses principaux avènements, une classification et une validation des métriques. Une troisième section survole l'évaluation et la modélisation de la qualité par l'intermédiaire d'une description sommaire des modèles les plus connus de la qualité comme, ceux de McCall et de Boehm. Le reste du chapitre est consacré à l'étude de la prédiction de la qualité. Ainsi, nous présentons différentes approches de modélisation de la qualité utilisées dans l'objectif de prédire certains caractéristiques et comportements du logiciel, comme le coût et la fiabilité. Dans cette partie du chapitre (à partir de la section 3.5), le regroupement, ou le mariage, entre le génie logiciel et l'apprentissage constituera également un intérêt de cette dissertation. Une contribution marquant notre étude de la prédiction est présentée dans la section 3.5.4.2. Elle a comme objet la proposition d'un ensemble de critères d'évaluation des modèles de prédiction. La section 3.6 survole les techniques de construc-

tion de modèles prédictifs, un champ de recherche qui semble être la direction la plus empruntée par les chercheurs en prédiction de la qualité du logiciel. Nous dégageons par la suite, dans la section 3.7, une classification des problèmes de la prédiction selon différents points de vue. Enfin, une esquisse d'approches d'amélioration de la prédiction est donnée dans la section 3.8.

3.2 Qualité du logiciel

La qualité est en général une notion ambiguë : cette ambiguïté réside dans la multitude de ses définitions employées dans chaque domaine. Selon Kan [1995], cela peut être attribué à plusieurs facteurs car, la qualité est multidimensionnel. Elle dépend, en fait, de la nature de l'entité étudiée, de son environnement et de ses attributs. De plus, le terme « qualité » fait partie de notre langage quotidien et sa popularité engendre plusieurs connotations (compréhensions) du terme qui peuvent être différentes de son utilisation professionnelle et scientifique. Différentes dimensions de la qualité sont résumées par Garrin [1984] en cinq vues à partir desquelles il décrit la qualité :

- la vue transcendantale : la qualité est quelque chose qu'on peut reconnaître, mais qu'on ne peut pas définir ;
- la vue d'utilisateur : la qualité est la force apparente du produit pour réaliser des fonctions ;
- la vue de fabrication : la qualité est la conformité aux spécifications ;
- la vue du produit : la qualité est attachée aux caractéristiques intrinsèques du produit, c'est la vue la plus évoquée par les experts de la qualité du logiciel ;
- la vue basée sur la valeur : la qualité dépend des coûts du produit.

Dans le domaine des logiciels, cette confusion est limitée. Cependant, la qualité d'un logiciel est souvent définie dans un sens étroit comme l'absence de « bogues » dans le produit logiciel. Cette définition rejoint la notion de conformité, car un logiciel ayant des défauts est non conforme aux spécifications, mais elle ne mesure pas la satisfaction du client vis-à-vis de certains aspects, comme la facilité d'utilisation et de maintenance, la clarté de la documentation, etc. Une définition claire et juste de la qualité demeure encore un souci pour la communauté du génie logiciel. Deux voies de recherche sur la qualité sont généralement empruntées. Dans la première on s'occupe de la qualité du processus de développement du logiciel et on croit que la qualité de ce dernier détermine la qualité du produit logiciel. Dans la

deuxième on juge qu'un bon processus ne garantit pas un bon produit et que la qualité de ce dernier est plutôt déterminée par un ensemble de propriétés intrinsèques contenues dans le produit.

Selon une troisième perspective, Kitchenham et Pfleeger [1996], en analysant la question « qu'est ce qui distingue un bon logiciel d'un mauvais », soulignent que le contexte aide à déterminer la réponse. En effet, les fautes tolérées dans un logiciel de traitement de textes ne sont pas acceptées dans un système de contrôle de sécurité d'un avion. Par conséquent, ils suggèrent que la qualité doit être considérée au moins de trois façons : la qualité du produit, la qualité du processus qui engendre le produit et la qualité du produit dans le contexte d'environnement où il sera utilisé.

3.2.1 Qualité du produit

Si on demande à différentes personnes d'indiquer les caractéristiques d'un produit logiciel qu'elles jugent importantes pour sa qualité, il est très probable que leurs réponses soient différentes. Cela est dû au fait que l'importance des caractéristiques dépend du profil de la personne qui analyse le produit. En effet, l'utilisateur croit que la qualité réside dans la capacité du logiciel de faire ce qu'il veut et d'une manière simple (facilité d'utilisation). Il privilégie ainsi, les caractéristiques externes du produit. Cependant, un produit logiciel est jugé aussi par d'autres acteurs, comme celui qui le conçoit, celui qui l'implémente et celui qui fait sa maintenance après le codage et pendant la mise en opération. Ces acteurs voient la qualité à travers des caractéristiques internes du produit logiciel, avant même son achèvement [Pfleeger, 1998].

Des modèles de qualité sont construits pour relier certaines caractéristiques internes avec des caractéristiques externes qui peuvent représenter la vue de l'utilisateur ou celle du développeur du logiciel. Par exemple, le modèle McCall, construit par McCall et ses collègues en 1977, montre comment les facteurs externes de la qualité sont reliés aux critères de la qualité de produit [McCall et Walters, 1977] (voir section 3.4.2.1).

Le modèle ISO 9126 est un standard de la qualité du produit grandement utilisé depuis sa création au début des années 80. Il est inspiré du modèle de McCall. En plus de l'organisation hiérarchique des différentes caractéristiques de la qualité du produit (voir l'annexe A), le modèle ISO 9126 offre un ensemble de définitions et de termes usuels pour l'évaluation de la qualité du produit logiciel [ISO, 2003]. Plus de détails sur le standard ISO 9126 sont donnés dans la section 3.4.2.3.

3.2.2 Qualité du processus

L'intérêt pour la qualité du processus découle du fait qu'un produit logiciel peut rencontrer des problèmes sérieux, parce que certaines activités lors de son développement, sont négligées ou mal exécutées. Plusieurs chercheurs croient que la qualité du processus de développement et celui de maintenance est aussi importante que la qualité du produit. L'avantage d'améliorer la qualité d'un processus est de pouvoir améliorer en même temps la qualité des logiciels produits.

De la même manière que pour la qualité du produit, des modèles d'amélioration du processus logiciel et d'évaluation de ses capacités sont proposés, comme CMM (*Capability Maturity Model*) et SPICE (*Software Process Improvement and Capabilty dEtermination*) [Pressman, 1996].

3.2.3 Qualité dans un contexte d'environnement commercial

Ce niveau de la qualité du logiciel est proposé par Kitchenham et Pfleeger [1996]. Il essaie de montrer un autre coté du logiciel rarement analysé, à savoir, la qualité des services que le logiciel fournit dans le milieu économique où il est exploité. Il s'agit de la valeur économique d'un logiciel [Simmons, 1996]. Cette proposition vient de l'idée qui suppose que l'amélioration technique (la qualité technique du processus ou du produit) est traduite en valeurs économiques, plus de détails sont disponibles dans [Pfleeger, 1998].

3.3 Métrologie logicielle

L'utilisation des mesures est fondamentale dans toutes les disciplines de l'ingénierie et de la science. En particulier, le fait d'assigner un descripteur ou une représentation numérique aux quantités et aux qualités relatives à un processus, à une ressource de développement ou à un produit logiciel offre trois avantages :

1. **Le contrôle** : gérer un processus revient en grande partie à le contrôler au travers des indications issues d'une activité de mesure. La métrologie fournit une base pour mesurer et comparer les solutions alternatives [DeMarco, 1982].
2. **L'évaluation** : les mesures permettent d'acquérir une aptitude pour évaluer l'état d'un processus ou d'un produit. La métrologie offre la possibilité d'identifier les tendances, les règles et la

capacité de formuler des jugements.

3. **La prédiction** : la métrologie peut être utilisée pour prédire des valeurs futures d'attributs qui sont *a priori* inconnues, ce qui permet d'aider à la prise de décisions futures. C'est dans cette perspective que la modélisation de la qualité s'insère (voir section 3.5).

Plusieurs sujets sont reliés à la métrologie logicielle. Dans cette section, nous allons aborder les plus importants d'entre eux d'une manière sommaire, en indiquant les références correspondantes. Ainsi, après avoir donné quelques définitions utiles en métrologie, nous présentons les principaux avènements de l'histoire de la métrologie logicielle, une classification des métriques, le paradigme GQM pour la dérivation de métriques et enfin nous survolons la validation des métriques.

3.3.1 Définitions

Définition 3.3.1 *Mesurage :*

Selon ISO/IEC 9126 [2003], c'est l'opération qui consiste à affecter une valeur à un attribut d'une entité dans le monde réel, de sorte que cette dernière soit bien représentée.

Selon Zuse [1998], le mesurage est le processus d'attribution empirique et objective de nombres aux propriétés d'objets et d'événements du monde réel qui permet leurs descriptions.

Définition 3.3.2 *Mesure :*

Selon ISO/IEC 9126 [2003], une mesure est le nombre ou la catégorie assignée à un attribut d'une entité après une opération de mesurage (voir définition 3.3.4).

Définition 3.3.3 *Métrique :*

Selon ISO/IEC 9126 [2003] on définit une métrique de qualité du logiciel comme une échelle quantitative et une méthode qui peuvent être employées pour déterminer la valeur que prend un attribut d'une entité d'un produit logiciel spécifique.

Définition 3.3.4 *Métrologie :*

C'est la science des mesures. Elle s'occupe de la préparation et de l'exécution des opérations de mesurage afin de garantir des résultats fiables. La métrologie permet notamment de choisir la méthode de mesurage appropriée en ce qui concerne la précision recherchée et la pratique des opérations de mesurage de toute organisation.

3.3.2 Principaux avènements de la métrologie logicielle

L'histoire des métriques logicielles a commencé dans les années soixante, quand les modèles Delphi d'estimation du coût sont apparus [Nelson, 1966; Helmer, 1966]. Cette histoire est parcourue à travers des avènement que nous jugeons importants. Nous commençons cette histoire à partir du début des années 70 et nous attribuons un titre à chaque avènement.

3.3.2.1 Métrique de « nombre de lignes de code » (LOC)

La métrique la plus ancienne et utilisée jusqu'à nos jours est **LOC** (nombre de ligne de code). En 1974, Wolverton était l'auteur de l'un des premiers essais de mesure de la productivité [Wolverton, 1974] utilisant LOC. Cette mesure était aussi utilisée comme un « prédicteur » de la fiabilité et de la facilité de la maintenance [Sheppard, 1993]. Probablement à cause de sa simplicité et de sa large utilisation (selon Zuse [1998], il y a plus de dix milles articles scientifiques qui mentionnent cette métrique), elle fait l'objet de nombreuses critiques, voir par exemple [Fenton et Neil, 1999a].

3.3.2.2 Métriques connues en physique versus métriques logicielles

Au milieu des années 70, certains termes comme *la physique du logiciel* (par Kolence [1975]) et *la science du logiciel* (par Halstead [1977]) étaient crées pour introduire des méthodes scientifiques pour manipuler les propriétés et les structures des programmes. La théorie de Kolence a tenté de faire des rapprochements entre certaines métriques connues en physique, comme celles de performance, celles de disponibilité d'un système, etc. et certaines métriques de gestion du développement de logiciels, comme la productivité, le coût par unité de service, et les budgets.

3.3.2.3 Métriques de McCabe et de Halstead

Dans la deuxième moitié des années 70, on a vécu l'apparition des métriques de McCabe [1979] et de Halstead [1977]. McCabe définit la *complexité cyclomatique* d'un programme comme une métrique dérivée de la théorie des graphes qui soit égale au nombre minimal des chemins dans le graphe de flots de contrôle. Halstead montrait que l'effort estimé, la durée de développement, la difficulté et la longueur des programmes, peuvent être calculés en fonction du nombre d'opérateurs, du nombre d'opérandes et du nombres d'utilisations de ces derniers [Halstead, 1977].

3.3.2.4 Points de fonction d'Albrecht (PF)

Albrecht [1979] proposait sa méthode de mesurage de la production de logiciel, basée sur les fonctions « utilisateurs ». Cette méthode n'avait pas les inconvénients du comptage du nombre de lignes de code. Plus tard en 1986, l'IFPUG (*International Function Point Users Group*) a été fondé pour veiller sur la plus large diffusion de la méthode de comptage de points de fonction, tout en garantissant son adéquation aux besoins des utilisateurs.

3.3.2.5 Techniques de mesurage et unités des mesures logicielles

Vers la fin des années 70 et au début des années 80, la communauté des métriques a commencé à chercher plus de rigueur dans les techniques de mesurage logiciel. Curtis [1980] a écrit : « Plus nos techniques de mesurage sont rigoureuses, plus un modèle théorique peut être complètement testé et calibré. Ainsi, le progrès sur une base scientifique en génie logiciel est tributaire d'un mesurage amélioré ». En 1981, Harrison a proposé une nouvelle technique de mesure de la complexité, basée sur la composition du graphe de flots. La même année, Troy et ses collègues proposèrent un ensemble de 24 mesures évaluant plusieurs aspects comme la modularité, la taille, la complexité, la cohésion et le couplage d'un logiciel. À la même époque, Jones [1978], l'un des pionniers de la métrologie logicielle, s'est intéressé aux unités de mesures logicielles.

3.3.2.6 Métriques de conception

Henry et Kafura ont proposé, en 1981, une mesure dans le but d'analyser la complexité des conceptions de grands systèmes logiciels appelée « métrique d'interconnectivité ». Cette métrique est basée sur la multiplication des métriques fan-in et fan-out [1].

L'importance de la cohésion et du couplage dans la compréhension d'un logiciel a poussé plusieurs chercheurs comme Emerson [1984] et Weiser [1982; 1984] à leur trouver des métriques. Dans cette même optique plusieurs travaux ont eu la cohésion pour objet principal [Dhama, 1994; Selby et Porter, 1988; Ottenstein et Thuss, 1989; Ottenstein et Thuss, 1991; Ottenstein, 1992; Briand et *al.*, 2000].

Genero, Piattini et Calero ont distingué une insuffisance dans les métriques qui mesures les aspects

1. fan-in et fan-out sont déterminées à partir du graphe d'appel de modules d'un programme. Elles désignent respectivement le nombre d'arcs entrants et le nombre d'arcs sortants d'un module (voir [Zuse, 1998]).

relationnelles dans la conception comme les agrégations, les associations et les dépendances. Ils ont proposé un ensemble de métriques couvrant ces aspects à différents niveau de granularité : (au niveau classe et au niveau *package*). Ces métriques utilisent les concepts de UML et sont jugées utiles dans le choix entre les alternatives des diagrammes de classes [Genero et *al.*, 2000].

3.3.2.7 Modèle COCOMO

Boehm [1981] a proposé le fameux modèle COCOMO (*COnstructive COst MOdel*) d'estimation de l'effort, du coût et de l'échéancier du développement d'un projet logiciel à partir de sa taille estimée en nombre de lignes de code.

3.3.2.8 Métriques orientées objet

À la fin des années 80, des métriques analysant la programmation orientée objet (OO) sont créées [Rocacher, 1988]. Les métriques OO sont basées généralement sur l'arbre d'héritage. En 1989, Morris [1989] a défini neuf métriques permettant de décrire différentes propriétés dans un programme OO : le nombre de méthodes de classe, le nombre de dépendances d'héritage, le degré de couplage entre objets, le degré de cohésion d'objets, l'effectivité d'un objet, le degré de réutilisation, la complexité moyenne des méthodes, la granularité d'une application et l'effectivité de factorisation.

Au début des années 90, Chidamber et Kemerer [1994] ont proposé un ensemble de métriques OO, qui sont maintenant très utilisées. Cet ensemble comprend : WMC (*Weighted Method Per Class*), DIT (*Depth Of Interitence Tree*), NOC (*Number of Class*), CBO (*Complying Between Classes*), NOMCMC (*Number of Other Methods Called by a Method of Class*) et LCOM (*Lack of cohesion among Method*). Ces métriques sont évaluées sur des programmes écrits en Smalltalk. Plusieurs autres travaux effectués durant les années 90 ont proposé différentes métriques OO, comme les métriques de Briand [1997], celles d'Abreu [1993] et d'autres [Lake, 1994; Taylor, 1993].

3.3.2.9 Métrique de base de données

Calero et ses collaborateurs ont proposé en 2001 un ensemble de métriques structurelles pour mesurer différents aspects d'une base de données relationnelles à objets. Parmi ces métriques nous citons TS (*Table Size*), DRT (*Depth of Referential Tree*) et NSC (*Number of Shared Classes*). Ces métriques

sont jugées pertinentes pour l'estimation de facilité de compréhension et de facilité de maintenance de la base de données de données relationnelles à objets [Calero et *al.*, 2001].

3.3.3 Classification des métriques

Plusieurs classifications des métriques sont proposées dans la littérature [Pfleeger, 1998]. La plus adoptée est celle de Fenton [1997]. En effet dans cette dernière, les métriques du logiciel se classent parallèlement au classement des entités qu'elles mesurent. Par conséquent on en distingue trois classes : les métriques de produit, les métriques de processus et les métriques de ressource. Pour chaque type d'entités (produit, processus et ressource), on distingue deux types d'attributs : *les attributs internes* et *les attributs externes*. Les attributs internes d'un processus, d'un produit ou d'une ressource sont ceux qui peuvent être mesurés sur l'entité proprement dite, indépendamment de son environnement, alors que les attributs externes sont ceux qui peuvent être mesurés en considérant les relations entre l'entité et son environnement (son comportement vis-à-vis de l'environnement). Par exemple, selon ISO/IEC 14598 [2000] partie 3, un attribut interne d'un produit logiciel est une propriété mesurable d'une entité qui peut être dérivée purement en terme de l'entité elle-même. Selon le même standard, un attribut externe est une propriété mesurable d'une entité qui peut être seulement dérivée en considérant comment l'entité est reliée à son environnement.

3.3.3.1 Métriques de processus

Ce sont les métriques qui sont généralement reliées à une activité ou au processus. Elles comprennent la durée d'une ou de plusieurs activités du processus, les efforts correspondant aux activités, le nombre et les types d'incidents survenus au cours des activités du processus, etc.

3.3.3.2 Métriques de produit

Ce sont les métriques qui mesurent les attributs d'un produit. Ce dernier peut être toute entité produite au cours du processus par une ou plusieurs activités : document, prototypes, spécification, conception, test, code, produit final, etc. Ces métriques comprennent d'une part, les métriques d'attributs internes comme la taille, la complexité, le couplage, la cohésion, la structure et, d'autre part, les métriques

d'attributs externes comme la fiabilité, la facilité de maintenance, la facilité de réutilisation, l'intégrité, la stabilité, la facilité de compréhension.

3.3.3.3 Métriques de ressource

Ce sont des métriques qui mesurent les attributs des ressources employées dans la production du logiciel. Elles comprennent les attributs relatifs au personnel, aux matériaux, aux outils et aux méthodes, etc. comme la taille de l'équipe de projet, le nombre d'individus impliqués dans le test, le coût de l'outil AGL[2] de test, l'expérience du personnel, etc.

3.3.4 Dérivation des métriques : le paradigme GQM

Goal Questions Metrics (GQM) est un paradigme qui fournit un cadre de dérivation des mesures. Basili a proposé ce paradigme avec ses collègues après avoir remarqué que beaucoup de métriques sont créées parce qu'elles sont faciles à mesurer et non pas parce qu'on en a besoin [Basili et Perricone, 1984]. L'approche GQM, comprend trois étapes :

1. Spécifier les objectifs majeurs du développement ou de la maintenance.
2. Dériver de chaque objectif les questions qui doivent être répondues de manière à savoir à quel degré l'objectif est atteint.
3. Décider qu'est-ce qu'on doit mesurer pour répondre aux questions de l'étape 2.

3.3.5 Validation des métriques

Dans le passé, il n'y avait pas de validation rigoureuse des métriques logicielles, on comptait sur la crédibilité de celui qui proposait la métrique. Avec l'effort de certains chercheurs, comme Zuse [1998] et Fenton [1997], la validation a pris d'autres formes plus rigoureuses et scientifiques. En général, la validation d'une métrique logicielle est le processus qui permet de s'assurer que celle-ci est une caractérisation numérique propre de l'attribut voulu en montrant que certaines conditions sont satisfaites. Par exemple, la longueur d'un programme est mesurée d'une manière valide si certaines notions intuitives ne sont pas contre-dites. En effet, si on concatène deux programmes P_1 et P_2, une notion intuitive se-

2. Atelier de Génie Logiciel.

rait d'avoir une métrique ℓ qui satisfait $\ell(P_1 : P_2) = \ell(P_1) + \ell(P_2)$, et si P_1 est plus long que P_2, alors ℓ satisfait $\ell(P_1) > \ell(P_2)$. On dit aussi que la condition de représentation de ℓ doit être satisfaite.

Ce type de validation est basé sur la théorie de la mesure qui n'est pas spécifique au domaine du logiciel. Son objectif est de s'assurer que la métrique utilisée reflète bien le comportement et les propriétés de l'entité dans le monde réel.

Certains chercheurs jugent que la validation fondée sur la théorie de la mesure est insuffisante, elle est plutôt **interne**. Ils s'attendent à une validation qui vérifie que la métrique fait partie d'un système de prédiction. Fenton dit qu'une métrique est valide dans le sens large, lorsqu'elle remplit, à la fois, les deux conditions suivantes :

1. Avoir une validité interne qui reflète fidèlement l'attribut à mesurer.
2. Être un composant d'un système valide de prédiction.

Cette manière de valider une mesure est aussi supportée par d'autre chercheurs comme Zuse [1998], Briand [2000] et El Emam [2000]. Cependant, elle reste problématique, faute d'un accord de la communauté des métriques logicielles sur ce que c'est qu'une métrique valide, cohérente et complète. C'est ce qu'écrivait Kitchenham [1995] et qui a été repris par El Emam [2000] lorsqu'il a proposé une méthodologie de validation comportant trois phases : la planification, la modélisation statistique et la *« post modélisation »*.

Compte tenu de ce qui précède, la validation des métriques peut être théorique ou empirique. Dans ce qui suit nous allons donné une description sommaire des deux formes de validation.

3.3.5.1 Validation théorique

En général, pour s'assurer de la validité des métriques, plusieurs chercheurs proposent des ensembles de propriétés que les métriques doivent satisfaire. Ces propriétés sont aussi appelées axiomes [Cherniavsky et Smith, 1991]. Par exemple, la complexité doit être reliée à taille : est un axiome que la métrique de la complexité doit satisfaire. Deux inconvénients majeurs de cette approche axiomatique sont notés dans la littérature [Poels et Dedene, 1997], il s'agit de l'incohérence dans certains ensembles de propriétés et de l'insuffisance des propriétés proposées. Par conséquence, une métrique peut être valide vis-à-vis d'un ensemble de propriétés mais invalide en considérant un autre ensemble.

Briand, Morasca et Basili [1996] ont proposé un ensemble de propriétés qui couvrent une large

gamme de métriques comprenant la taille, la longueur, la complexité, la cohésion et le couplage. Briand et ses collègues ne prétendent pas que cet ensemble de propriétés est suffisant pour valider les métriques, en revanche, toutes les propriétés proposées sont nécessaires et peuvent être utilisées pour montrer qu'une métrique est invalide. Ils affirment par ailleurs que l'ensemble de propriétés proposées est générique et non contradictoire avec des ensembles d'axiomes formulés par des recherches précédentes. Briand, Morasca et Basili ont relié explicitement leurs propriétés avec les concepts existants de la théorie de mesure et ont invité les chercheurs à étendre et à raffiner les propriétés proposées pour converger vers une théorie de mesure plus complète.

Poels et Dedene [1997] voient que le framework mathématique proposé par Briand et ses collègues [1996] n'est pas assez précis pour garantir une interprétation non ambiguë des définitions des propriétés. Particulièrement pour les propriétés reliées à l'additivité[3], il existe un nombre d'incohérences qui pourraient avoir comme conséquences la confusion et l'empêchement de l'évolution du framework. Selon Poels et Denene, ces incohérences peuvent être regelées en enlevant l'ambiguïté dans les définitions de certaines propriétés présentées dans [Briand et *al.*, 1996]. Poels et Dedene croient que pour atteindre un consensus sur une théorie universelle de mesure de logiciel, la relation entre les métriques et leurs propriétés d'additivité doit être encore clarifiée. Ils proposent une solution pour enlever l'ambiguïté dans les définitions en introduisant une métrique dont l'échelle est ordinale qui évalue la force de connexion entre les modules de logiciel et en reliant cette échelle aux propriétés d'additivité.

3.3.5.2 Validation empirique

Un autre volet de la validation des métriques est celui de la validation empirique. Cette validation est critique pour le succès de toute activité de mesure [Kitchenham et *al.*, 1995; Fenton et Pfleeger, 1997]. Quand on valide empiriquement une métrique, on essaye de répondre à la question suivante : est ce que la métrique est utile dans un environnement particulier de développement étant donné un but derrière la définition de la métrique. Selon Briand et ses collègues [1995], dans un contexte donné, la métrique d'un attribut interne particulier est utile si elle est reliée à une métrique d'un certain attribut externe de l'objet étudié. Par exemple, une métrique de la complexité est toujours définie par rapport à un certain

3. Considérons par exemple les propriétés de complexité liées à l'additivité. L'additivité revient à poser la question suivante : est ce que la complexité d'un système composé de deux entités de logiciel (par exemple, segments de programme, modules, objets de classe) est égale à la somme des complexités des différentes entités

attribut externe comme la propension à engendrer des erreurs ou l'effort.

La validation empirique nécessite les trois étapes suivantes :

1. Collecte des données sur la métrique externe de la qualité et les métriques internes du produit concerné. Ceci nous permettra de déterminer une certaine relation statistique entre les deux types de métriques.
2. Identification des échelles des métriques internes et externes d'attributs. Ceci détermine jusqu'à un certain niveau les types d'analyse de données à effectuer.
3. Sélection des mécanismes adéquats pour l'analyse et la modélisation des relations qui existent entre les métriques.

3.4 Évaluation et modélisation de la qualité

L'évaluation de la qualité d'un produit logiciel revient à savoir mesurer la présence de certains attributs dans ce produit. Ces attributs qualifiés d'externes, sont ceux qui informent sur le comportement du produit logiciel dans son environnement. Par exemple, s'il s'agit du code source, alors la fiabilité est un attribut externe qui décrit un certain comportement vis-à-vis de la machine et de l'utilisateur, formant ensemble, l'environnement du code. Un attribut externe n'est autre qu'une vue de la qualité du logiciel [Fenton, 1991]. S'intéresser séparément à chacune des vues de la qualité est aussi un moyen de contourner la subjectivité de sa définition. Étudier un seul attribut externe permet de rendre la notion de la qualité moins abstraite.

Habituellement, on mesure et on analyse certains attributs internes, parce qu'ils sont des « prédicteurs » (bons indicateurs) d'attributs externes. Cette façon de mesurer la qualité apporte deux avantages : premièrement les attributs internes sont disponibles dans les phases précoces du cycle de vie d'un logiciel, alors que les attributs externes ne sont mesurables que si le développement du logiciel est au moins, achevé. Deuxièmement, les attributs internes sont plus faciles à mesurer grâce à leurs définitions précises qui impliquent uniquement le produit.

Dans le processus d'évaluation de la qualité, on commence par choisir des attributs particuliers de la qualité qui reflètent un intérêt donné, par exemple, fonctionner sans avoir de défaillances. Il s'agit d'un comportement définie par un attribut externe qui est la fiabilité. Par la suite, on cherche à mesurer

cet attribut après avoir trouvé des relations qui l'expriment en fonction des attributs internes plus faciles à mesurer et disponibles *a priori* (tôt dans le cycle de vie du logiciel). Le produit de ce processus est capturé dans un modèle qui regroupe ces attributs (internes et externes) et ces relations. Ce processus s'appelle *la modélisation de la qualité*.

3.4.1 Approches de modélisation de la qualité

Un modèle de qualité du logiciel exprime généralement une relation entre des attributs internes et des attributs externes d'un produit logiciel. Il est un moyen de vulgariser la ou les vues de la qualité qu'il exprime, et ceci par la combinaison d'attributs plus simples à mesurer. Un modèle est d'autant plus accepté que la relation qu'il exprime est compréhensible. Le type de cette relation dépend primordialement de l'approche selon laquelle le modèle est construit et par la suite présenté. Fenton [1997] voit que les modèles de qualité d'un produit logiciel sont construits selon l'une des deux approches suivantes :

- **l'approche basée sur les modèles connus** : pour évaluer la qualité, on accepte la décomposition offerte par le modèle prédéfini ainsi que la façon avec laquelle on combine les métriques élémentaires pour calculer les valeurs des critères et des facteurs désirés.
- **l'approche basée sur les modèles *adhoc*** : dans cette approche, l'utilisateur adopte la philosophie générale de la décomposition de la qualité en plusieurs attributs. Mais il définit sa propre façon de décomposer les attributs et de combiner les métriques. Cette façon permet plus de flexibilité et plus d'adéquation entre les modèles et les spécificités des produits.

3.4.2 Approche basée sur les modèles connus

Dans cette section, nous nous intéressons aux modèles de la qualité qui ont connu une acceptation dans la communauté du génie logiciel et qui prétendent représenter les différents aspects de la qualité. Parmi ces modèles, nous présentons le modèle de McCall [1977], le modèle de Boehm [1978] et le modèle standard de la qualité ISO/IEC 9126 [2003]. Par la suite, nous donnons une brève description d'une approche de modélisation de la qualité proposée par Dromey qui tente d'apporter des solutions à certaines limitations des modèles connus.

3.4.2.1 Modèle McCall

Ce modèle présente trois vues de l'utilisateur concernant un produit appelées respectivement, *l'opération du produit*, *la révision du produit* et *la transition du produit*. Les trois vues sont décomposées en onze facteurs de qualité. En général, ces facteurs sont des attributs externes. Ces facteurs sont décomposés à leur tour en un ensemble de critères (23 au total) et chaque critère est associé à un ensemble de métriques (41 au total). Ces décompositions permet de présenter les caractéristiques de la qualité d'une manières hiérarchique. La figure A.1 dans l'annexe A donne une vue de ce modèle. Pour illustrer l'aspect opérationnel du modèle de McCall, nous allons nous intéresser à un critère en particulier : Ainsi, pour mesurer la complétude (*completeness*), on doit vérifier une liste prédéfinie de conditions et compter combien sont applicables par phase de cycle de vie d'un logiciel : Spécification (S), Conception (C) et Implémentation (I). La liste des conditions pour la complétude est la suivante.

1. Des références non ambiguës (entrée, sortie, fonction) [S, C, I].
2. Toutes les références de données (variables ou références directes à des adresses au moyen de pointeurs) sont définies, calculées ou lues de l'extérieur [S, C, I].
3. Toutes les fonctions définies sont utilisées [S, C, I].
4. Toutes les fonctions utilisées sont définies [S, C, I].
5. Toutes les conditions et tous les calculs utilisés à chaque point de décision sont définis [S, C, I].
6. Tous les paramètres d'appels formels et effectifs sont conformes [C, I].
7. Tous les problèmes de report sont résolus [S, C, I].
8. La conception est conforme aux besoins [C].
9. L'implémentation est conforme à la conception [I].

Il y a six conditions qui s'appliquent à la spécification, huit à la conception et huit à l'implémentation. La mesure de la complétude est égale à :

$$\frac{1}{3}\left(\frac{\textit{le nombre de réponses vrai pour S}}{6} + \frac{\textit{le nombre de réponses vrai pour C}}{8} + \frac{\textit{le nombre de réponses vrai pour I}}{8}\right).$$

Les autres critères sont calculés de la même manière. Des pondérations pourraient éventuellement être associées aux réponses.

3.4.2.2 Modèle de Boehm

Le modèle de Boehm est similaire à celui de McCall du point de vue de la représentation hiérarchique des caractéristiques de la qualité. Boehm intègre dans son modèle non seulement les besoins et les attentes de l'utilisateur mais également les caractéristiques de la performance du matériel. En effet, à travers son modèle, Boehm affirme que la qualité d'un logiciel réside dans le fait qu'il satisfasse les utilisateurs et les programmeurs impliqués dans le cycle de vie de ce logiciel. Le modèle insiste sur l'utilité générale du logiciel en s'appuyant sur les affirmations de Boehm et de ses collègues qui considèrent qu'un logiciel doit être avant tout et pour tout utile. Cette utilité générale est décomposée en trois caractéristiques (vues de la qualité). Chacune est associée à un type d'utilisateurs : l'**utilité** associée à l'utilisateur original pour lequel le logiciel est initialement développé, la **portabilité** concerne l'utilisateur intéressé sous réserve d'une adaptation à son environnement et la **facilité de maintenance** recherchée par le programmeur qui fait les modifications et les adaptations du logiciel. Les trois vues de la qualité sont à leur tour décomposées en un nombre de caractéristiques comme le montre la figure A.2 dans l'annexe A.

3.4.2.3 Modèle standard de la qualité ISO 9126

Au début des années 80, la communauté de la qualité du logiciel a manifesté une volonté de créer un seul modèle de qualité qui regrouperait les différentes vues de la qualité et qui deviendrait un standard universel. L'avantage d'un tel modèle est de faciliter la comparaison des produits. Le fruit de cette tentative a été le standard ISO 9126. C'est un modèle hiérarchique dérivé de celui de McCall et composé de six facteurs de qualité : *la fonctionnalité*, *la fiabilité*, *l'efficacité*, *la facilité d'utilisation*, *la facilité de maintenance* et *la portabilité*. Le standard prétend que ces six facteurs couvrent tous les aspects de la qualité, c'est-à-dire qu'on peut décrire n'importe quels composants de la qualité d'une manière ou d'une autre, en utilisant un ou plusieurs de ces facteurs. Chaque facteur (ou caractéristique) peut être décrit par un nombre de sous caractéristiques (voir la figure A.3 dans l'annexe A). Par exemple, la fiabilité est définie comme « l'ensemble des attributs portant sur l'aptitude du logiciel à maintenir son niveau de service dans des conditions précises et pendant une période déterminée ». Les sous caractéristiques de la fiabilité sont : la maturité, la tolérance aux fautes et la facilité de récupération.

3.4.2.4 Limitations des modèles connus et framework de Dromey

Les modèles connus présentés dans les sections précédentes fournissent une aide précieuse pour la définition des attributs et pour la simplification des aspects abstraits de la qualité. Cependant, aucun de ces modèles ne fournit une raison de la présence ou de l'absence d'un attribut et de son emplacement dans l'hiérarchie du modèle [Pfleeger, 1998]. Par exemple, pourquoi la portabilité se situe-t-elle au niveau le plus élevé de la hiérarchie de ISO 9126 et pourquoi ne constitue-t-elle pas une sous caractéristique ? De plus, aucune directive n'est donnée pour la composition des caractéristiques de bas niveau qui permettrait de définir les caractéristiques de niveau supérieur. Ces problèmes rendent difficile l'évaluation de la complétude et la cohérence[4] d'un modèle.

Dromey a considéré quelques problèmes associés aux modèles connus comme le manque de directives de la composition des caractéristiques de bas niveau. Il a alors proposé un cadre générique de construction de modèles de qualité en se basant sur le fait que « les *propriétés internes tangibles* d'un produit déterminent les *attributs externes de sa qualité* » [Dromey, 1996]. En effet, Dromey souligne que la qualité d'un produit est déterminée essentiellement par le choix des composants représentant le produit (comprenant les documents de spécification, les guides d'utilisateur, les conceptions et le code source), des propriétés tangibles de ces composants et des propriétés tangibles de leurs compositions. Dromey classifie ces propriétés tangibles en quatre classes : propriétés d'exactitude, propriétés internes, propriétés contextuelles et propriétés descriptives.

La construction d'un modèle utilisant le *framework* de Dromey comporte cinq étapes :

1. Identification de l'ensemble d'attributs de la qualité de haut niveau et significatifs (qui varient d'un projet à un autre).
2. Identification des composants du produit.
3. Identification et classification des propriétés tangibles les plus significatives et qui portent sur la qualité de chaque composant.
4. Proposition d'un ensemble d'axiomes pour lier les propriétés du produit aux attributs de la qualité.
5. Évaluation et raffinement du modèle.

4. À noter que la complétude et la cohérence portent sur les aspects de la qualité. Elles ont un sens différent de celui traité dans le chapitre 4.

Dromey illustre l'utilisation de son *framework* en construisant un modèle pour évaluer la spécification, un autre pour la conception et un dernier pour l'implémentation. Plus de détails sur les modèles de Dromey se trouvent dans l'ouvrage [Dromey, 1996]. Malgré que le modèle de Dromey tienne compte de certains problèmes présents dans les modèles précédents, il n'a pas eu une utilisation significative dans l'industrie.

3.4.3 Approche basée sur les modèles *adhoc*

Les modèles de McCall, de Boehm et le standard ISO9126 constituent des références intéressantes pour comprendre la qualité d'un produit logiciel. Cependant, les limitations des uns et le manque de maturité des autres (Modèle de Dromey) ont poussé plusieurs chercheurs et organisations à modéliser la qualité d'une manière *adhoc*. Cette approche est beaucoup utilisée pour la prédiction de la qualité (voir section 3.5. Dans cette section nous ne présentons pas des exemples de modèles issus de cette catégorie d'approche de modélisation, en revanche, nous les abordons dans différentes sections de cette dissertation (voir les sections 3.5.2.1, 3.5.2.2, 3.6 et 3.5.2.3). D'ailleurs, l'intérêt fondateur de notre travail vise l'amélioration de la prédiction des modèles *adhoc*.

3.5 Prédiction de la qualité du logiciel

Le mesurage de la qualité du logiciel est généralement utilisé pour évaluer une entité déjà existante. Le résultat du mesurage est utile car elle permet de comprendre l'existant. Mais dans plusieurs situations, le besoin de prédire la qualité d'une entité qui n'existe pas encore est crucial. Dans la catégorie des logiciels embarqués dans les avions par exemple, on a besoin de construire des programmes avec un haut niveau de fiabilité. En général, le développement de ce type de logiciels prend beaucoup de temps. Néanmoins, if faut s'assurer de la fiabilité désirée le plus tôt possible et certainement avant la mise en opération. Paradoxalement, la fiabilité d'un logiciel est définie à partir de sa performance opérationnelle qui ne peut être mesurée qu'après une période de fonctionnement du logiciel. Ainsi, un recours à la prédiction s'impose pour évaluer la fiabilité du logiciel encours de développement. Dans ce cas, La fiabilité est prédite à partir des indicateurs qui représentent notre compréhension du logiciel et qui sont disponibles dès les premières étapes du développement.

Plusieurs besoins de prédiction portant sur d'autres aspects de la qualité, du coût et de la durée se manifestent dans les premiers moments du développement d'un logiciel. Dans la littérature, différentes motivations justifient les efforts investis dans la prédiction. Fenton [1999a] et Khoshgoftaar [1995c] croient que la prédiction permet de ménager l'effort et d'épargner temps et argent car la correction retardée des fautes (après déploiement du logiciel) est une tâche très souvent coûteuse. Par conséquent, l'identification des modules susceptibles d'avoir un nombre élevé de défauts durant le développement permet d'économiser l'effort du développement et de la maintenance, tout en se concentrant sur le perfectionnement de la qualité de ces modules à « haut risque ».

L'importance de la prédiction a attiré l'attention de plusieurs chercheurs qui ont essayé de proposer un formalisme qui définit d'une manière rigoureuse les concepts de la prédiction. Nous tirons de la littérature quelques définitions utiles pour comprendre la prédiction. Ces définitions seront illustrées par un exemple de la vie courante utilisé dans la littérature pour simplifier les concepts de la prédiction (voir section 3.5.1.4.

3.5.1 Définitions

3.5.1.1 Système de prédiction

Un système de prédiction consiste en un modèle mathématique accompagné d'un ensemble de procédures de prédictions qui permettent de déterminer des paramètres inconnus et d'interpréter le résultat [Fenton et Pfleeger, 1997; Zuse, 1998]. Le but de la prédiction est de fournir aux développeurs une prévision de la qualité dès les premières phases du développement du logiciel. Ainsi, à l'aide de certaines mesures, les praticiens peuvent entreprendre les actions correctives convenables.

3.5.1.2 Fonction de prédiction

Selon Zuse [1998], la fonction de prédiction est de la forme $V(P) = f(u(P))$, avec V la variable à prédire, P un programme (produit logiciel) et u une mesure de P. Un petit changement de la variable u doit causer un petit changement dans la variable externe V. Il en est de même pour un grand changement.

3.5.1.3 Variable externe

La variable externe est une quantification d'un attribut externe (un facteur de qualité) [Zuse, 1998]. Elle est généralement exprimée par une mesure indirecte dérivée de certaines autres mesures de logiciel [ISO, 2003]. Des exemples de variables externes peuvent être le coût de la maintenance, le temps de réparation d'un module, le temps nécessaire pour la compréhension d'un module, etc.

3.5.1.4 Exemple de système de prédiction

Nous présentons ci-après un exemple de système de prédiction inspiré de celui de Fenton [1997]. Le but de cet exemple est d'éclaircir, à travers un problème familier de prédiction, certains éléments utilisés dans les définitions précédentes. Il s'agit de la prédiction du coût d'un voyage en voiture de Montréal à Toronto. Les éléments de ce système sont les suivants :

- L'entité sujet de prédiction est le voyage.
- L'attribut de « qualité » à prédire est le coût du voyage.
- Le modèle mathématique est la formule : $coût = \frac{ab}{c}$, où *coût* est la variable externe, a (la distance entre Montréal et Toronto), b (le coût par litre du carburant) et c (la distance moyenne parcourue par le type de voiture utilisée par litre de carburant) sont des paramètres qui définissent les mesures.
- L'ensemble de procédures utilisées pour déterminer les paramètres des modèles (a, b et c) peut, par exemple, être constitué par des consultations d'une association automobile, questionnements des amis ou lecture du manuel de description de la voiture.

3.5.2 La prédiction de la qualité du logiciel dans la littérature

On trouve dans la littérature plusieurs travaux qui présentent des résultats et des expériences de prédiction de variables, associées à un facteur donné de la qualité du logiciel. Les facteurs qui ont attitré l'intérêt de nombreux groupes de recherches sont l'effort[5] et la fiabilité.

5. On désigne par effort, non seulement l'effort du développement du logiciel mais aussi son coût et sa durée.

3.5.2.1 Prédiction de l'effort

Le gestionnaire et le chef du projet ont de nombreuses motivations pour prédire le coût d'un projet ainsi que son effort et sa durée, dès les premières étapes du cycle de vie d'un logiciel. Ils veulent planifier leurs tâches et allouer des ressources dans le but de diriger la gestion vers l'optimisation du coût. Par conséquent, plusieurs modèles d'estimation du coût et de l'effort ont été proposés comme SLIM de Putman, COCOMO de Boehm et les points de fonction d'Albrecht. Ces modèles utilisent la taille estimée, ou le nombre de points de fonctions comme entrées.

Putman [1978] a proposé le modèle SLIM parmi les premiers modèles de prédiction du coût et de l'effort, pour les projets de grande envergure : Effort $= 0{,}4 * (\frac{LOC}{C})^3 * \frac{1}{t_d^4}$ où C est un facteur technologique qui dépend de la compétence de développement et la complexité du projet, et t_d est la durée du développement. Les deux mesures LOC (voir section 3.3.2.1) et t_d sont estimées.

Dans un registre légèrement différent, Boehm [1981] a proposé les modèles COCOMO pour prédire l'effort de développement d'un projet logiciel. Dans ces modèles, il distingue entre trois types de projets: organique, semi-détaché et embarqué. Il propose par conséquent, un modèle par type de projet:

- Modèle simple: Effort $= 3{,}2LOC^{1.05}$
- Modèle intermédiaire: Effort $= 3{,}0LOC^{1.12}$
- Modèle complexe: Effort $= 2.8LOC^{1.20}$.

Par la suite, Boehm a étendu son modèle pour tenir compte de plusieurs facteurs influençant la qualité appelés *facteurs du coût*. Il multiplie le second membre de son équation de l'effort par le produit de ces facteurs. Ce modèle de Boehm peut être vu comme un seul modèle dont la variable de sortie (variable externe) dépend de la taille estimée et de deux paramètres (a et b) qui informent sur le type de projet (voir l'équation 3.1).

$$\text{Effort} = aLOC^b, \textit{ avec } a \textit{ et } b > 0. \tag{3.1}$$

Plusieurs autres modèles de l'effort ou de coût proposés par la suite sont considérés comme très proches de l'équation de COCOMO :

- Walston Félix propose une nouvelle version de COCOMO et définit son modèle : Effort $= 5{,}2LOC^{0.91}$ [Walston et Felix, 1977].
- Bailey et Basili [Conte et *al.*, 1986], ajoutent une constante à COCOMO : Effort $= 5{,}5 + 0{,}73LOC^{1.16}$

– Doty propose un autre modèle similaire à celui de Walson Felix avec a et b modifiés :

Effort $= 5{,}288 LOC^{1{,}047}$.

3.5.2.2 Prédiction de la fiabilité

La fiabilité est un facteur de qualité présent dans la plupart des modèles comme ISO 9126, McCall, Boehm, etc. La nécessité de prédire la fiabilité est à tel point cruciale que sa modélisation a fait l'objet d'une discipline séparée. Musa, Littlewood et d'autres ont montré, par des exemples, que l'importance donnée à la fiabilité a permis d'améliorer la compréhension et le contrôle du produit logiciel [Fenton et Pfleeger, 1997]. Tout cela a fait de la fiabilité l'aspect de la qualité le plus étudié (le plus présent dans la littérature) et dont les travaux sur la prédiction remontent aux années 70.

L'objectif de la prédiction de la fiabilité est de connaître quand le logiciel aura une défaillance. Étant donné que ce phénomène est probabiliste, on exprime l'incertitude sur la défaillance par une fonction appelée *la fonction de densité de probabilité*, ou *pdf*. Cette fonction est choisie, dans la plupart des modèles, sous une forme exponentielle du temps t, $f(t) = \lambda e^{-\lambda t}$, où λ est appelé le taux de détection des erreurs, ou le taux de défaillance instantanée, ou également le taux de hasard [Kan, 1995].
La probabilité de défaillance entre l'instant 0 et t est définie par: $F(t) = \int_0^t f(t)dt$. On dit que le logiciel survit jusqu'à sa première défaillance. Donc, la fonction de fiabilité n'est autre que la fonction de survie définie par: $R(t) = 1 - F(t)$. En effet, la survie d'un logiciel (ou d'un composant logiciel) à l'instant t_i de la $i^{ième}$ défaillance est définie par : $R_i(t_i) = 1 - F(t_i)$.
La fiabilité d'un logiciel est supposée croissante d'une défaillance à une autre. En effet, après une modification (une correction), il est très probable d'avoir minimisé la récurrence des fautes qui ont causé les défaillances déjà survenues : c'est la notion de *la croissance de fiabilité*.
Jelinski et Moranda ont développé, en 1972, un modèle (JM) de croissance fiabilité avec une *pdf* définie par :

$$f(t_i) = \phi(N - i + 1)e^{-\phi(N-i+1)t_i},$$

où N est le nombre initial de fautes et ϕ est la contribution de chaque faute au taux de défaillance. Ainsi, le temps entre deux défaillances croît avec chaque défaillance qui survient [Jelinski et Moranda, 1972].
Plusieurs modèles apparentés à JM sont apparus comme celui de Schick et Wolverton [1978] dont la *pdf*

est définie par :

$$f(t_i) = \phi(N - i + 1)t_i e^{-\phi(N-i+1)t_i^2/2}.$$

Le modèle de croissance de la fiabilité MO de Musa et Okumoto [1983] vient insister sur la considération de la durée de correction d'une défaillance, contrairement au modèle JM qui suppose que celle-ci est négligeable, et qu'une telle correction n'engendre pas d'autres défauts. Goel et Okumoto [1979] traitent aussi le même problème en introduisant la probabilité de débogage imparfait p, dans la fonction *pdf* de leur modèle GO :

$$f(t_i) = \phi(N - p(i-1))e^{-\phi(N-p(i-1))t_i}.$$

D'autres modèles de prédiction de la croissance de la fiabilité ont été proposés. Ils sont généralement basés sur une fonction *pdf* de forme exponentielle. Nous mentionnons dans ce sens, le modèle LW de Littlewood [1973] et les modèles NHPPM (*Non Homogeneous Poisson Process Model*) [Cox et Isham, 1980].

3.5.2.3 Autres travaux et résultats de prédiction

Les résultats présentés ci-après sont le fruit de plusieurs travaux sur la prédiction d'un facteur de la qualité ou d'une relation reliant deux attributs d'un produit logiciel. Avec leur présentation, nous voulons souligner deux constats. Le premier concerne la diversité des prédictions et le deuxième a trait à la forme dans laquelle sont exprimées les relations découvertes. En effet, plusieurs de ces relations sont exprimées dans un langage naturel sans aucun formalisme ou technique de présentations de connaissance.

- Hetzel [1993], en utilisant la métrique KLOC, a fait des investigations pour conclure qu'il y a une corrélation de 0,82 entre KLOC et l'effort réel de développement, et que seulement 12% des prédictions de l'effort sont à 20% de leurs valeurs réelles. Autrement dit, près de 9 projets sur 10 ont un effort correctement prédit.
- Basili et Perricone [1984] ont trouvé une relation évidente entre la mesure de la complexité du logiciel et les erreurs découverte durant le développement et la phase opérationnelle.
- Khoshgoftaar et ses collègues [1990] ont rapporté qu'il y a une relation étroite entre la complexité et le nombre de changements causés par les erreurs trouvées plus tard dans les phases de test et de validation.

- Basili et Perricone [1984] et Kafura et son équipe [1988] ont conclu à l'existence d'une relation linéaire entre mesures de complexité et nombre d'erreurs. La même conclusion à été plus tard confirmée par Khoshgaftaar et ses collaborateurs [1990] et par Henry et ses collaborateurs [1990].
- Schach [1993] a soutenu que le nombre de lignes de codes est une bonne mesure pour prédire le nombre de défauts. Il a aussi ajouté que si un programmeur croît qu'une ligne de code a 2% de chance de contenir une faute, alors un module de 100 lignes de longueur à tester est sensé contenir deux fautes.
- Takahashi [1996] affirme l'existence d'une relation linéaire entre d'une part la fréquence de changement dans les spécifications du programme, l'expérience du programmeur, le volume du document de conception et d'autre part, le taux d'erreurs.
- Akiyama [1971] a développé des équations de régression pour prédire le nombre de défauts, le nombre de décisions C, le nombre de sous-routines J et une variable composée $C+J$. La première équation, par exemple, prédit le nombre de défauts à partir du nombre de lignes de code (LOC). De cette équation, on peut déduire qu'un module de 1 $KLOC$ (1 000 lignes de code) est sensé avoir approximativement 23 défauts. Une des équation de Akiyama est de la forme $D = 4{,}86 + 0{,}018LOC$, où D représentant le nombre de défauts.
- Halstead [1977] a proposé de prédire le nombre de défauts d'un programme à partir d'une métrique de volume V. Cette dernière est définie, comme les autres métriques de Halstead, en fonction du nombre d'opérateurs et du nombre d'opérandes dans un programme. Le nombre de défauts est prédit par l'équation suivante : $D = \frac{V}{3\,000}$, avec 3 000 indiquant le nombre moyen de discriminations mentales entre des décisions faites par le programmeur.
- Lipow [1982] utilise la théorie de Halstead et intègre le calcul de V dans une équation de prédiction de la densité de défauts à partir du nombre de lignes de code $\frac{D}{LOC} = A_0 + A_1 LogLOC + A_2 Log2LOC$, où les coefficients A_i sont dépendants de la moyenne de nombre d'opérateurs et d'opérandes par lignes de code pour un langage particulier. Par exemple, pour Fortran, $A_0 = 0.0047$, $A_1 = 0.0023$ et $A_2 = 0{,}000043$.
- Lounis, Sahraoui et Melo [1998] ont vérifié l'hypothèse selon laquelle certaines formes de couplage (mesures de couplage) ont un impact direct sur la propension des classes OO à engendrer des erreurs (*error-proneness*). Ils ont défini une relation entre le couplage et un attribut important

de la qualité du logiciel qui est l'*error-proneness*.

- Mao, Sahraoui et Lounis [1998] ont utilisé l'apprentissage et particulièrement les arbres de décision pour prédire la réutilisabilité des programmes OO.

Ces travaux sont présentés à titre d'exemples. Ils ne couvrent ni toutes les activités de la prédiction ni ses différentes techniques. Par ailleurs, une autre liste non exhaustive accompagnera, dans la section 3.6.2, la présentation des techniques utilisées pour différentes activités de la prédiction. Toutefois, nous notons qu'un modèle de prédiction peut être disponibles sous forme de relations, de règles ou d'affirmations exprimées dans un langage naturel comme le montre certains des exemples présentés ci-dessus.

3.5.3 Validation de la prédiction

Un système de prédiction est valide, s'il produit des prédictions exactes [Fenton et Pfleeger, 1997]. En effet, valider un système de prédiction dans un environnement donné consiste à évaluer la performance du modèle sur des observations dont la valeur de sortie est connue (données de test ou d'évaluation). La validation d'un système de prédiction est identique à celle d'un résultat d'un apprentissage telle que nous la décrivons dans le chapitre 2.

3.5.4 Critères d'évaluation d'un modèle de prédiction

3.5.4.1 État de l'art

Iannino et ses collègues [1984] ont proposé un ensemble de critères pour évaluer les modèles de fiabilité. Ces critères sont aussi applicables aux autres modèles, sauf que leur importance varie d'une caractéristique de qualité à une autre (fiabilité, facilité de la maintenance, effort, stabilité). Les critères d'évaluation proposés sont :

- *la capacité prédictive* : c'est la capacité du modèle à prédire la valeur de leur sortie d'une manière exacte (avec le minimum d'erreurs).
- *la possibilité prédictive* : c'est la possibilité du modèle à prédire avec une exactitude satisfaisante les quantités (les valeurs de variables externes) nécessaires pour bien contrôler le développement et les phases opérationnelles, c'est le nombre de variables que le modèle peut prédire.

- *la qualité des hypothèses* : c'est le degré de concordance et de consistance des hypothèses avec la réalité logique et à partir d'un point de vue du génie logiciel.
- *l'applicabilité* : c'est le degré d'applicabilité du modèle à différents produits logiciels ayant différentes propriétés (taille, structure, domaine d'application, fonction, etc.).
- *la simplicité* : c'est le degré de simplicité du modèle qui doit être simple en tenant compte de trois aspects : (1) avoir une collecte de données (pour déterminer les paramètres des modèles) simple et peu coûteuse ; (2) avoir des concepts simples et ne pas exiger une connaissance mathématique profonde de la part des développeurs afin de simplifier la compréhension ; (3) être prêt à l'implémentation sous forme d'outil d'aide à la décision.

Les critères d'évaluation des modèles de fiabilité décrits sont présentés dans l'ordre d'importance proposé par le groupe d'Iannino. Cet ordre reflète un point de vue purement académique. Ceci a poussé Kan [1995] à proposer un ordre basé plutôt sur un point de vue de praticien en génie logiciel. Le nouvel ordre de Kan donne plus d'importance à la simplicité et à la qualité des hypothèses après la capacité prédictive et moins d'importance à l'applicabilité et à la possibilité prédictive (nombre de variables externes prédites).

La simplicité et la transparence d'un modèle, à la différence de sa complexité et de la difficulté de son interprétation, sont des critères importantes pour l'acceptation des modèles. En effet, les modèles de prédiction s'adressent en premier lieu aux gestionnaires et aux responsables de projets. Toute décision découlant des modèles doit être clairement justifiée pour permettre à ces derniers d'entreprendre les actions correctives nécessaires. Andrew et ses collègues [1997] jugent que la simplicité des formules mathématiques et leurs facilités d'être comprises sont des facteurs de réussite des modèles de prédiction. Ils affirment qu'un modèle sans sémantique qui lui soit attaché est très susceptible d'être refusé.

Plusieurs efforts sont investis en ce qui a trait au problème de calibrage et d'adaptation des modèles. L'objectif de ces efforts est d'élargir leurs applicabilités. En particulier, les versions de COCOMO intermédiaire et de COCOMO II ne sont autres que des améliorations pour permettre leur utilisation par une large gamme de clients. Aussi, l'applicabilité des points de fonction d'Albrecht est l'une des raisons d'être d'IFPUG (*International Function Point User Group*).

3.5.4.2 Proposition d'un ordre de critères d'évaluation des modèles

À la lumière de notre précédente analyse, nous proposons dans cette dissertation un ordre d'évaluation des modèles qui favorise, successivement, la capacité prédictive, la simplicité, et l'applicabilité. La capacité prédictive est considérée avec sa définition commune utilisée dans la littérature [Fenton et Pfleeger, 1997]. Cependant, nous considérons dans la définition de la simplicité d'un modèle principalement sa capacité d'expliquer les relations de causalité entre caractéristiques de la qualité et métriques. En d'autres termes, nous nous intéressons à la capacité d'aider dans le processus de la prise de décision. Nous regroupons dans la simplicité d'un modèle, la facilité de son interprétation, la facilité de son utilisation et encore sa transparence (visibilité des connexions entre les entrées et les sorties des modèles). Dans la suite nous utiliserons *« la facilité de son interprétation »* pour désigner la simplicité.

Le critère d'applicabilité, également désigné dans notre dissertation par l'expression *« capacité de généralisation »*, est la possibilité d'appliquer le modèle dans diverses circonstances en maintenant une capacité prédictive satisfaisante. Ceci peut être réalisé par le biais d'un calibrage, par exemple, ou par divers moyens d'adaptation. L'applicabilité est alors la capacité du modèle à s'adapter et à résister, en gardant une capacité prédictive satisfaisante, au changement de l'environnement.

3.6 Techniques de construction des modèles de prédiction

La construction de modèles prédictifs de qualité du logiciel est une tâche complexe qui peut impliquer plusieurs techniques dans le but de produire des modèles ayant une bonne prédiction. Ces techniques peuvent être basées sur l'opinion d'experts, sur l'analogie, sur la décomposition, sur les statistiques ou sur des méthodes d'apprentissage. Nous nous intéressons, dans ce présent travail, aux deux dernières familles de techniques, vu leurs utilisations répandues.

Nous rappelons que nous abordons de nouveau le sujet de la modélisation de la qualité, mais dans un objectif de prédiction. Et comme nous l'avons mentionné dans la section 3.4, nous nous intéressons, dans cette section, à la modélisation basée sur l'approche *adhoc* identifiée par Fenton et Pfleeger [1997]. En effet, les techniques de construction sont empruntées généralement au domaine de la statistique ou au domaine de l'apprentissage.

3.6.1 Techniques statistiques

Les techniques statistiques essayent, dans l'ensemble, de trouver une formule mathématique explicite qui exprime la relation entre les variables d'entrée d'un modèle et ses variables de sortie. Les modèles qui en résultent sont basés généralement sur des équations de régression.

Ces techniques sont très utilisées dans la prédiction en général et dans celle visant des facteurs économiques et financiers en particulier. Ce type de prédiction aide à comprendre le marché et par conséquent à prendre des décisions de type « À quel moment dois-je me retirer du marché ». Cette tendance a influencé la prédiction de la qualité du logiciel. En effet, les premiers travaux de prédiction ont concerné les facteurs économiques du logiciel tel que l'estimation du coût.

Parmi ce type de techniques, on trouve la régression linéaire (RL), avec ses variétés, l'analyse discriminante (ADis), la régression logistique (RLog). À cette liste non exhaustive, nous pouvons ajouter, l'analyse en composantes principales (ACP) qui a été impliquée souvent comme un prétraitement permettant un meilleur choix des variables d'entrée. Une description sommaire des principales techniques de régression est donnée dans l'annexe B.1.

Ces techniques ont une popularité considérable comme en témoignent de nombreux travaux de prédiction de la qualité du logiciel. En effet, la régression linéaire a été employée pour prédire le nombre de changements correctifs [Khoshgoftaar et *al.*, 1992; Khoshgoftaar et *al.*, 1993], l'analyse discriminante a été appliquée pour détecter les modules contenant des défauts [Khoshgoftaar et Szabo, 1994a; Munson et Khoshgoftaar, 1992] et la régression logistique a été employée pour identifier les composants à haut risque [Briand et *al.*, 1993b; Briand et *al.*, 1993a].

3.6.2 Techniques d'apprentissage

Les techniques d'apprentissage sont utilisées dans la prédiction de la qualité du logiciel dans le but de surmonter certaines limitations des méthodes statistiques qui concernent surtout les caractéristiques irrégulières de données servant à la construction ou à la validation des modèles prédictifs. Ces techniques essayent de trouver des régularités dans les données pour pouvoir exprimer des règles, des expertises, des connaissances, des jugements d'experts, etc.

Activité de prédiction	Techniques utilisées	Références
Identification des modules à haut risque	PG	[Evett et *al.*, 1998]
	RN	[Hong et Wu, 1997]
		[Khoshgoftaar et *al.*, 1995b]
		[Khoshgoftaar et *al.*, 1997]
		[Lanubile et Visaggio, 1997]
	RBC	[El Emam et *al.*, 2001a]
		[Ganesan et *al.*, 2000]
	AD	[Briand et *al.*, 1992]
		[Porter et Selby, 1990a]
	AC	[de Almeida et Matwin, 1999]
	PLI	[Cohen et Devanbu, 1997]
Prédiction de la taille	RN,PG	[Dolado, 2000]
Prédiction du coût	AD	[Briand et *al.*, 1992]
	RBC	[Briand et *al.*, 1999]
	RB	[Chulani et *al.*, 1999]
Prédiction de l'effort de développement	RBC	[Shepperd et Schofield, 1997]
		[Vicinanza et *al.*, 1990]
	AD,RN	[Srinivasan et Fisher, 1995]
	AG avec RN	[Shukla, 2000]
	RN,RBC	[Fenton et Pfleeger, 1997]
Prédiction de l'effort de maintenance	RN,AD	[Jorgensen, 1995]
Prédiction et analyse de ressources	AD	[Selby et Porter, 1988]
Prédiction du coût de correction	AD,PL	[de Almeida et *al.*, 1998]
Prédiction de la fiabilité	RN	[Karunanithi et *al.*, 1992]
Prédiction de défauts	RB	[Fenton et Neil, 1999a]
Prédiction de la facilité d'utilisation	AD	[Mao et *al.*, 1998]
Prédiction de la date de la prochaine version	RN	[Dohi et Osaki, 1999]
Prédiction de la testabilité	RN	[Khoshgoftaar et *al.*, 2000]

TAB. 3.1 – *Activités de prédiction utilisant les techniques d'apprentissage*

Elles tiennent compte de l'intelligence de l'humain, par exemple, de la façon d'exprimer des opinions ou de la capacité à comprendre les relations entre les attributs de la qualité d'un logiciel. Parmi ces techniques nous citons les arbres de décision (AD), le raisonnement à base de cas (RBC), les réseaux et les classificateurs bayésiens (RB et CB), la programmation et les algorithmes génétiques (PG et AG), les réseaux de neurones (RN) et la programmation logique (PL). Une description sommaire de certaines de ces techniques est donnée dans l'annexe B.2.

Le tableau 3.1 extrait d'une étude sur l'état de la pratique des techniques d'apprentissage en génie logiciel [Zhang et Tsai, 1986] présente un ensemble d'utilisations des techniques d'apprentissage pour construire des modèles de prédiction. Ces utilisations montrent les tendances de la modélisation de la qualité de logiciel.

3.7 Problèmes de la prédiction de la qualité du logiciel

Dans cette section, nous abordons ce que nous considérons comme les principaux problèmes de la prédiction de la qualité du logiciel. En analysant ces problèmes, nous remarquons qu'ils sont liés généralement à l'un des facteurs suivants :

- l'environnement de construction et d'application du modèle de prédiction ;
- les hypothèses considérées lors de la construction du modèle ;
- les caractéristiques de la technique de modélisation (construction) ;
- la capacité prédictive du modèle ;
- les données de construction du modèle.

3.7.1 Problèmes liés à l'environnement

Un des problèmes fondamentaux de la prédiction de la qualité du logiciel est dû à la difficulté de la prise en compte de la variation de l'environnement. Cette variation prend plusieurs formes, elle peut être causée par des changements au niveau du processus de développement, du style et des concepts de programmation, des domaines d'applications des logiciels, etc.

Par exemple, les conditions sous lesquelles Akiyama a développé son modèle de régression pour prédire le nombre total de fautes n'ont aucune relation avec les environnements de programmation

contemporains [Fenton et Neil, 1999a]. Au sein d'un même environnement, on peut avoir des variations qui sont causées par l'évolution à court terme des organisations, pour faire face aux pressions technologiques dans le milieu du logiciel. À titre d'exemple, Humphrey [1989] rapporte que dans la même organisation, la densité des fautes peut varier de 1 à 10 fois. Cette variation est attribuable, en grande partie, à l'évolution de l'environnement, se produisant par exemple par l'amélioration des procédures de test.

En conclusion, le fait qu'un environnement logiciel est en évolution constante doit être pris en compte lors de la construction des modèles de qualité du logiciel.

3.7.2 Problèmes liés aux hypothèses de la prédiction

L'objectif des modèles de prédiction est de trouver une relation entre le présent (attributs internes disponibles *a priori*) et le futur (facteur de qualité mesurable *a posteriori*). Pour simplifier la réalisation de cet objectif, les chercheurs formulent des hypothèses, en supposant par exemple, que le modèle est une relation linéaire entre les variables et que les attributs « prédicteurs » sont indépendants les uns des autres, ou encore, en supposant que le temps de réparation des erreurs est négligeable : hypothèse utilisée dans la plupart des modèles de fiabilité. Plusieurs hypothèses sont valables dans certaines circonstances, mais elles ne le sont pas dans d'autres. Certaines hypothèses sont rarement réalistes comme l'indépendance de tous les attributs d'entrées [Kan, 1995]. Par exemple, l'examen de certains modèles, par Breslawski et Subramanian [1989] et par Pfleeger [1989], a montré que les facteurs de coût du modèle COCOMO ne sont pas indépendants comme ils sont supposés l'être.

3.7.3 Problèmes liés aux techniques de modélisation

Des études ont été effectuées sur les structures des modèles de prédiction et sur les techniques de leur construction ont révélé plusieurs raisons derrière l'échec des modèles à atteindre leurs objectifs. Ces raisons sont directement liées aux limitations des techniques de construction des modèles [Fenton et Neil, 1999a; Fenton et Ohlsson, 2000; Gray et MacDonell, 1997]. En effet, plusieurs problèmes de la prédiction sont hérités directement des techniques utilisées pour construire les modèles. La plupart des techniques présentent une portée limitée décrite sous forme d'hypothèses restrictives et non réalistes. Nous présentons dans ce qui suit quelques exemples de limitations des techniques de construction.

La régression linéaire est encore parmi les techniques les plus utilisées. Pourtant elle présente plusieurs limitations. Une étude critique des approches basées sur la régression a été menée par Fenton [1999a], dans laquelle il a montré plusieurs problèmes comme la multi-colinéarité, la suppression non justifiée des points de données, l'utilisation des données transformées à la place des données originales, la distribution uniforme exigée de données, etc. Ces problèmes sont détaillés dans [Fenton et Neil, 1999a].

Les réseaux de neurones se comportent comme des boîtes noires car ils ne fournissent pas une explication concernant la manière dont les sorties du modèle sont obtenues. Ils souffrent aussi d'un problème appelé *l'oubli catastrophique*: quand il est entraîné sur de nouvelles données, le RN perd ses connaissances existantes [Robins, 1995].

Avec les systèmes flous, il est difficile de garantir une haute capacité prédictive sans perdre la facilité d'interprétation du modèle. Ce problème apparaît quand on construit un modèle avec un très grand nombre de règles [Gray et MacDonell, 1997].

Les modèles basés sur le raisonnement à base de cas sont connus par leur intolérance au bruit de données [Breiman et *al.*, 1993]. Aha [1991] rapporte que la création de ce type de modèle n'est pas une tâche triviale à cause du choix difficile de la fonction de similarité qui influence fortement la performance du modèle.

Les arbres de décision présentent aussi un problème connu sous le terme d'instabilité [Takahashi et *al.*, 1997]. En effet un petit changement dans l'ensemble d'entraînement peut provoquer un grand changement dans la structure de l'arbre.

3.7.4 Problèmes liés à la difficulté d'interprétation des modèles

Ces problèmes concernent l'incapacité d'un modèle à expliquer ses prédictions. Les modèles exprimés par des formules mathématiques sont souvent mal compris par les utilisateurs, surtout lorsqu'ils mettent en jeu plusieurs variables. Parfois, on perd les traces des attributs originels lorsque ces derniers subissent des transformations (logarithmique par exemple). La capacité d'explication d'un modèle est un facteur principal qui influence grandement son acceptation.

Avec l'établissement d'une liaison claire et explicite entre les attributs d'entrée et le facteur prédit de qualité, l'utilisateur peut être aidé à prendre une décision en dirigeant, éventuellement, ses efforts vers le

perfectionnement d'un attribut interne en particulier. Andrew et ses collègues [1997] affirment que sans sémantique claire attachée à un modèle, on ne peut pas atteindre un niveau satisfaisant de validité de ce dernier. Dans cette perspective, nous pouvons donner l'exemple des réseaux bayésiens qui sont basés sur l'équation de la sémantique globale (voir l'équation B.6) qui garantit cette interprétation explicite. En effet, la structure et la procédure de construction des RB sont basées sur cette équation qui préserve les relations de causalité (voir [Fenton et Neil, 1999a; Fenton et *al.*, 2002]) entre les attributs de la qualité. Ceci dote un modèle basé sur les probabilités bayesiènnes (RB ou CB), d'une capacité d'explication précise, explicite et en même temps réaliste.

3.7.5 Problèmes liés à la capacité prédictive

Bredero et ses collègues [1989] ont effectué plus d'une vingtaine d'études sur l'évaluation de la capacité prédictive des modèles de prédiction de l'effort et de la durée. Ces études ont amené aux deux résultats suivants :

1. Les valeurs réelles sont très différentes des valeurs prédites, même avec des valeurs connues et non estimées du nombre de lignes de code.
2. Différents modèles donnent des prédictions très différentes pour les mêmes valeurs d'entrée.

Pareillement, dans le cas de la fiabilité, nous citons l'étude effectuée par Abdel-Ghaly [1986] qui a montré une grande variation des comportements de différents modèles sur le même ensemble de données. Plus de détails sur cette étude seront donnés dans le chapitre 4.

Ces études sur la capacité prédictive ont abouti à deux conclusions communes très importantes : La première concerne la relation étroite entre la capacité prédictive d'un modèle et les données. En effet, Abdel-Ghaly et ses collègues concluent que la capacité prédictive d'un modèle varie d'un ensemble de données à un autre. Fenton [1997] ajoute qu'un modèle qui produisait de bonnes prédictions dans le passé n'est pas garanti de garder les mêmes performances dans le futur. Ceci est attribuable à l'évolution des données qui suivent l'évolution de l'environnement (voir section 3.7.1).

La deuxième concerne le choix d'un modèle. En effet, plusieurs techniques de mesure de la capacité prédictive peuvent aider dans la décision du choix d'un modèle, mais elles ne peuvent pas indiquer définitivement le meilleur modèle pour des circonstances futures *a priori* inconnues.

3.7.6 Problèmes liés aux données

La prédiction de la qualité est une problématique essentiellement empirique. Lorsqu'on parle d'un modèle de la qualité du logiciel, plusieurs problèmes sont à résoudre par des expérimentations et par des observations de la pratique réelle. On a besoin de données pour représenter, abstraire, analyser et valider les relations et les phénomènes observés. Les données sont cruciales, au moins pour construire et valider les modèles de prédiction. Malheureusement, la pénurie de données sur de vrais systèmes logiciels constitue un obstacle qui empêche d'atteindre un niveau de maturité acceptable pour la discipline de la prédiction de la qualité du logiciel.

Même si la prédiction de la fiabilité nécessite une collecte de données plus simple que les autres aspects de la qualité et qu'elle a déjà concentré sur elle l'intérêt de plusieurs chercheurs, Littlewood montre que seulement une poignée de données est publiée sur le problème de la croissance de fiabilité [Littlewood, 1991]. Le problème est plus sérieux dans la prédiction des autres facteurs de la qualité.

Parfois, la pénurie de données est due à un problème de confidentialité. Certaines organisations sont réticentes envers l'accès à leurs données, par crainte de perdre la bonne réputation de leurs produits. Une autre raison très valable est le manque d'expertises et d'outils nécessaires pour la collecte de données, ce qui rend la tâche complexe et coûteuse.

Les données sur la qualité du logiciel, lorsqu'elles sont disponibles, ont des caractéristiques qui rendent leurs analyses difficiles. Ces caractéristiques comprennent les données manquantes, le grand nombre de variables (dimensions), les valeurs extrêmes, la petite taille d'échantillons et la forte colinéarité entre les variables. Elles rendent le processus de modélisation de la qualité beaucoup plus difficile et les modèle dérivés moins performants. Certains de ces problèmes peuvent être traités par l'application de techniques, comme les transformations de variables ou l'analyse en composantes principales, alors que d'autres sont difficiles à résoudre, comme les données manquantes et la taille réduites des ensembles de données [Gray et MacDonell, 1997].

Un autre problème qui concerne la représentativité des données est associé généralement à la taille réduite des ensembles de données disponibles sur la qualité et techniques d'échantillonnage utilisées. Ce problème est principalement à l'origine de l'inadéquation des modèles. Ceci est dû au fait que les modèles sont construits à partir d'échantillons non représentatifs de l'environnement. L'application d'un

modèle à un ensemble de données issues d'un nouvel environnement différent de celui où le modèle a été construit, suppose que les nouvelles données sont assez bien représentées par l'ensemble des données de construction (ou d'entraînement). Cette hypothèse est très loin d'être réaliste. Kemerer [1993] conclut dans son étude sur les modèles d'estimation du coût que le fossé entre les valeurs prédites et les valeurs réelles varie de 85% à 610% lorsque le modèle est appliqué dans un environnement différent de celui de son développement. Ce problème nous amène à parler de la capacité de généralisation du modèle car propriété du modèle dépend étroitement de la représentativité des données de construction. En soulignant que la représentation de toutes les circonstances est un objectif irréaliste, nous comprenons évidement qu'il n'y ait pas de modèles universels, et nous pouvons viser des compromis de généralisation d'un modèle (voir la section 4.4).

3.8 Approches d'amélioration de la prédiction

Face aux problèmes de la prédiction présentés dans la section précédente, plusieurs directions sont suivies dans le but de surmonter certaines de ces limitations. Des efforts sont fournis pour améliorer la capacité prédictive des modèles, augmenter leurs capacités d'explication, palier au manque de données de construction, etc. Ces efforts ont donné naissance à des approches d'amélioration de la prédiction que nous pouvons classer en trois catégories : (1) des approches basées sur le choix de la technique de modélisation ; (2) des approches fondées sur le calibrage et l'optimisation de paramètres du modèle ; (3) des approches basées sur la combinaison des prédictions de modèles.

3.8.1 Approche basée sur le choix de la technique de modélisation

L'objectif de cette approche est de proposer des technique alternatives qui permettent généralement à résoudre plusieurs problèmes à la fois, comme la faible capacité prédictive et la difficulté d'interprétation des modèles. Cette approche est très critique envers les modèles statistiques traditionnels basés sur la régression linéaire. Fenton est l'un des premiers supporteurs de cette voie d'amélioration de la prédiction. Elle considère que la cause de la faiblesse des prédictions est l'utilisation inappropriée de techniques de construction des modèles. Les supporteurs de cette approche suggèrent qu'on tient compte du comportement de la technique choisie vis-à-vis des caractéristiques restrictives des données de construc-

tion du modèle. Ils recommandent que le choix d'une technique soit basé sur sa capacité d'exprimer, d'une manière logique et appropriée au domaine du génie logiciel, les connaissances extraites à partir des données. Cette manière comprend la garantie d'une sémantique claire des relations exprimées par le modèle, la prise en compte de la notion d'incertitude nécessaire pour mener un discours réfléchi et réaliste sur la qualité, etc. (voir [Fenton et *al.*, 2002; Gray et MacDonell, 1997; Sahraoui et *al.*, 2002]). Nous rappelons qu'une meilleure capacité prédictive est un objectif commun de toutes les tentatives d'amélioration de la prédiction, entre autres, l'approche que nous développons dans ce travail. En effet, plusieurs chercheurs travaillent sur l'amélioration de la prédiction en adoptant cette approche, nous mentionnons à titre d'exemple, Fenton, Khoshgooftar, Sahraoui. Dans ce qui suit, nous allons présenter de brèves descriptions de certains de ces travaux.

3.8.1.1 Travaux de Fenton et son équipe

Fenton constate qu'un problème majeur de la prédiction est l'utilisation inappropriée des méthodes de construction des modèles. Dans plusieurs de ses écrits, Fenton [1997; 1999a; 2000; 2002] critique fortement les mesurages employant des opérations statistiques. Il qualifie ces modèles de *naïfs* parce qu'ils ignorent les relations causales entre des attributs internes et les attributs externes de la qualité. Ces modèles utilisent naïvement la taille et de la complexité, par exemple, sans une validation rigoureuse qui montre que ces mesures sont appropriées à la prédiction d'un facteur particulier de la qualité. Les développeurs de ces modèles donnent peu d'attention à la rigueur et au réalisme des hypothèses utilisées avec la technique de construction. Fenton ajoute que la faiblesse de ces modèles découle, entre autres, de la confusion entre la capacité du modèle à représenter les données historiques et sa capacité de prédire l'inconnu, de la multi-colinéarité, de la suppression non justifiée des points de données, de l'utilisation des données transformées à la place des données originales, etc.

Ainsi, Fenton résume la plupart des problèmes de la prédiction dans la naïveté des techniques et de leur utilisation. Le raisonnement de Fenton permet de développer une grande partie de son approche. Il reconnaît que les problèmes liés aux données et à la complexité d'évaluation de la qualité du logiciel ne seront pas résolus facilement. Cependant, il croit que l'utilisation d'une technique probabiliste est une piste vers la résolution des problèmes de la prédiction. Ces méthodes sont basées sur les réseaux

bayésiens, décrits dans la section B.2.4. Les avantages des RB comprennent, selon Fenton [1999a] :

- la spécification de relations complexes en utilisant les expressions basées sur les probabilités conditionnelles ;
- l'utilisation de l'analyse de type « quoi-si » ;
- la facilité de comprendre les enchaînements des raisonnements apparemment contradictoires et complexes, par l'intermédiaire d'un format graphique ;
- la modélisation explicite de l'incertitude et de l'ignorance accompagnant les estimations ;
- la possibilité de la prédiction malgré les données incomplètes, ou manquantes.

Fenton [1999a] présente un prototype d'un RB développé dans le but de montrer le potentiel des RB. Avec son exemple de RB, Fenton ne prétend pas résoudre tous les problèmes des modèles traditionnels naïfs, mais il montre la possibilité de combiner différentes écoles de pensées sur la *prédiction des défauts*, dans un seul modèle. Ce réseau contient plusieurs noeuds représentant des attributs couvrant la phase de spécification et la phase du test du cycle de vie d'un logiciel. Les noeuds représentant les efforts et la complexité prennent les états suivants : « très haut », « haut », « moyen », « bas » et « très bas », les noeuds représentant les nombres de défauts prennent des valeurs entières et celles représentant les densités de défaut prennent des valeurs réelles. Les relations entre ces noeuds sont exprimées sous forme de probabilités qui peuvent être déterminées suite à une analyse de la littérature ou à partir d'hypothèses logiques sur la direction et la force de la relation entre les attributs. Ces probabilité sont organisées dans des tables de probabilités conditionnelles attachées aux noeuds. La figure 3.1 montre l'allure du modèle RB. La prédiction des défauts en utilisant ce modèle consiste à déterminer les valeurs des noeuds de défauts à partir des états respectives de l'effort de conception, de la complexité du problème et de l'effort de test (noeuds racines) et en utilisant les tables de probabilités attachées aux noeuds non racines (ombrés dans la figure 3.1). La description détaillée de ce prototype peut être trouvée dans [Fenton et Neil, 1999a]. Fenton [2000] rapporte qu'il continue de travailler sur de nombreux projets, en utilisant les réseaux bayésiens, comme méthode pour créer des modèles de qualité plus sophistiqués.

3.8.1.2 Travaux de Khoshgoftaar et son équipe

Khoshgoftaar et son équipe ne s'intéressent pas de la même façon que Fenton à la sémantique et aux relations de causalité exprimées dans les modèles. Leur objectif est plutôt d'améliorer la capacité

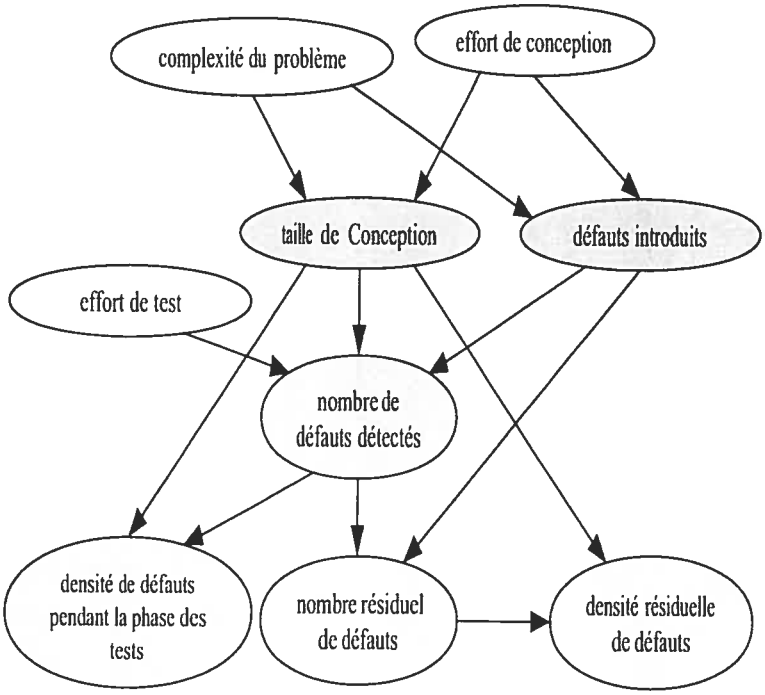


FIG. 3.1 – *Un exemple prototype d'un réseau bayésien développé par Fenton.*

prédictive. Leurs travaux contribuent essentiellement à l'introduction de certaines techniques dans la modélisation de la prédiction de la qualité du logiciel, comme le raisonnement à base de cas, la programmation génétique, les arbres de régression et de décision, les réseaux de neurones et la régression logistique. Un échantillon de ces travaux est donné dans le tableau 3.1. Avec chacune de ces techniques, Khoshgoftaar et son équipe essayent d'améliorer un aspect ou de résoudre un problème pour maximiser la capacité prédictive par rapport aux techniques statistiques traditionnelles. Parmi ces tentatives nous relevons ce qui suit.

Pour classifier les modules par rapport à leur propension à générer des fautes, Khoshgoftaar et ses collègues ont utilisé, entre autre, les arbres de régression [Liu et Khoshgoftaar, 2004] avec lesquels ils ont pu trouver des résultats satisfaisants.

L'analyse discriminante a été employée pour détecter les modules ayant une faible testabilité. Un programme a une faible testabilité, si étant donné un ensemble de tests, ces derniers ne sont pas susceptibles de détecter des fautes. Cette technique a été jugée excellente. Elle a permis de comprendre la relation entre les attributs d'entrée et la mesure de testabilité [Khoshgoftaar et Szabo, 1994b].

Les algorithmes génétiques sont employés afin de construire les réseaux de neurones optimisés pour la détection des modules ayant une grande propension à générer des fautes. Ce regroupement des deux techniques (AG et RN) a produit des RN plus performants que le meilleur RN construit manuellement et en un temps plus court [Khoshgoftaar et *al.*, 1997].

Ces contributions sont d'autant plus intéressantes qu'elles permettent d'enrichir l'activité de la prédiction par de nouvelles méthodes.

Dans le même but d'utiliser méthodes performantes, Khoshgoftaar et ses collègues combinent souvent plusieurs techniques empruntées à l'IA pour résoudre des problèmes qui ne peuvent pas être contournés avec une seule technique. En particulier pour estimer l'effort, la logique floue a été combinée avec le raisonnement à base de cas (RBC). Ainsi, l'utilisation de la logique floue a permis de surmonter la limitation du RBC qui se manifeste lorsque les attributs d'un projet sont décrits par des mesures catégorielles (« petit », « moyen » et « grand »). L'approche obtenue permet d'estimer l'effort, quelles que soient les mesures utilisées, numériques ou catégorielles [Khoshgoftaar et *al.*, 2002].

3.8.1.3 Travaux de Sahraoui et son équipe

Sahraoui et son équipe s'intéressent à l'amélioration de la prédiction en général, en travaillant sur la prédiction de plusieurs facteurs de la qualité (maintenabilité, stabilité, changeabilité, etc.) et en employant diverses techniques empruntées de l'IA. À mi-chemin entre les travaux de Fenton et ceux de Khoshgoftaar, Sahraoui et son équipe s'intéressent d'une part à l'amélioration de la facilité de compréhension et de la facilité d'interprétation des modèles et d'autre part à l'augmentation de leurs capacités prédictives. Ils utilisent des techniques comme la logique floue, les arbres de décision, les algorithmes génétiques et les classificateurs bayésiens pour améliorer les prédictions.

Pour promouvoir l'interprétabilité, les arbres de décision sont employés pour prédire la *facilité de réutilisation* des composants logiciels. Par la suite, la capacité prédictive du modèle ainsi construit est améliorée par un algorithme génétique [Sahraoui et Azar, 1999].

Sahraoui et son équipe [2001] distinguent deux types de techniques de construction de modèles de prédiction. Le premier type utilise des données historiques et se base généralement sur des analyses statistiques : les modèles issus de ce type de techniques sont le plus souvent naïfs (inefficaces à la prise de décision) à cause de la grande distance cognitive entre les mesures internes et les caractéristiques à

prédire de la qualité. Le deuxième type utilise les connaissances extraites des heuristiques spécifiques du domaine : les modèles ainsi produits comprennent les opinions d'experts sur les liens entre le facteur de qualité à prédire et les mesures internes sous forme de relations causales. Mais, malgré le fait que ces modèles soient enrichis pour supporter la prise de décision, ils restent difficiles à généraliser. Dans l'objectif d'améliorer la qualité de la prédiction en ciblant, en même temps, le problème de la naïveté et celui de la généralisabilité des modèles, Sahraoui et son équipe ont proposé une approche d'extension des modèles naïfs obtenus par des techniques statistiques. L'idée de cette approche est d'intégrer, dans ces modèles naïfs, des heuristiques spécifiques du domaine. Ces heuristiques sont des affirmations basées sur des prépositions floues, comme « éviter les méthodes longues » au lieu « d'éviter les méthodes dont la longueur est supérieur à 50 lignes » pour éviter un engagement sur la valeur du seuil (50) qui dépend de plusieurs facteurs liés à un contexte particulier. L'intégration des deux raisonnements (flou et classique) dans un seul modèle est réalisée par la transformation du modèle naïf (statistique) en un modèle *naïf flou*, pour enfin étendre ce dernier en utilisant les heuristiques du domaine. La figure 3.2 récapitule l'approche globale de cette transformation d'un modèle naïf en un modèle flou causal [Sahraoui et *al.*, 2001].

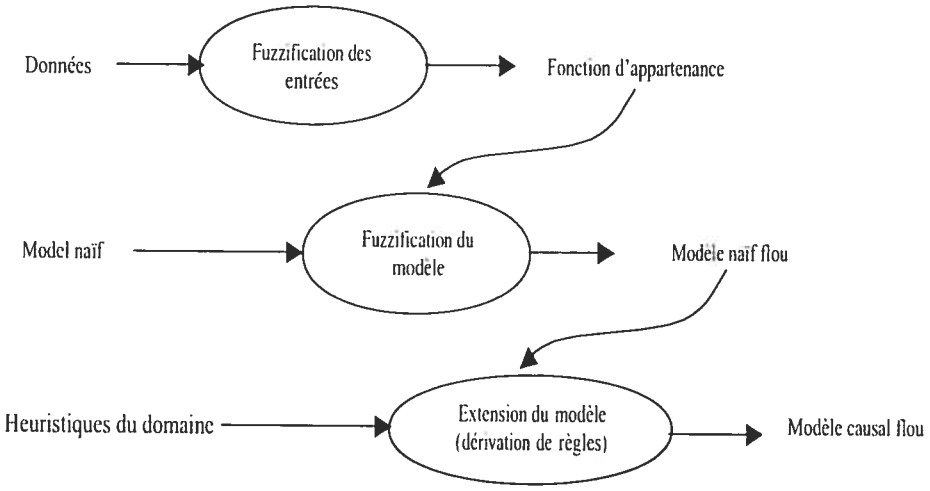


FIG. 3.2 – *Approche de transformation du modèle naïf au modèle causal.*

Sahraoui et son équipe [2002] présentent un cadre de construction et d'utilisation de modèles de prédiction de la qualité basés sur les règles. Pour contourner le problème des valeurs seuils connu dans les modèles de règles classiques, ce groupe utilise une technique basée sur la logique floue. Les modèles

dérivés par ce cadre ont une meilleure performance et ils sont plus facile à interpréter. Ces modèles sont étendus par des heuristiques du domaine afin de les utiliser dans la prise des décisions.

Dans un autre travail [Grosser et *al.*, 2003] au sein de la même équipe on propose une approche de construction des modèles prédictifs basée sur le raisonnement à base des cas. Ce travail traite le cas la stabilité des classes OO. L'approche proposée permet de contourner de l'insuffisance des connaissances théoriques sur la stabilité. Elle cherche les similarités structurelles entre les classes exprimées par des métriques pour déterminer leurs degrés de stabilité.

3.8.1.4 Conclusion sur l'approche basée sur le choix des techniques

Nous avons présenté trois types de travaux illustrant l'amélioration de la prédiction, qui se basent sur le choix d'une ou de plusieurs techniques de modélisation. Les exemples brièvement abordés utilisent les réseaux bayésiens, le raisonnement à base de cas et la logique floue. Cependant, l'amélioration de la prédiction employant des techniques d'IA ne se limite pas à ces travaux présentés ci-dessus. D'autres travaux utilisant la programmation génétique, la programmation logique, les arbres de décisions, les réseaux de neurones, etc. (voir [Zhang et Tsai, 1986]) peuvent s'inscrire sous la portée de cette approche.

3.8.2 Approche basée sur le calibrage

Comme nous l'avons présenté dans la section 3.5.4.1, l'étude de Kemerer a relevé que les estimations du coût variaient de 85% à 610% entre les valeurs prédites et les valeurs réelles. En revanche, par la suite d'un calibrage, le modèle d'estimation est significativement amélioré et son erreur varie seulement entre 50% et 100%. Le calibrage à pour objectif d'améliorer la capacité prédictive d'un modèle, surtout lorsque ce dernier est sur ajusté[6]. Le calibrage est effectué lorsque un modèle est appliqué dans un nouvel environnement, autre que celui où il a été développé. Il consiste ainsi à déterminer les valeurs d'un ensemble de paramètres d'un modèle existant dans le but d'améliorer sa performance (capacité prédictive). Ces paramètres peuvent être des facteurs multiplicatifs des coefficients d'une régression, des seuils d'attributs[7], etc. Certains définissent le calibrage comme une action de configuration ou de spécialisation d'un modèle générique. Cette technique a été initialement utilisée pour les modèles d'es-

6. Un modèle sur ajusté (*overfitted*) signifie que la capacité prédictive du modèle est bonne sur les données d'entraînement, mais faible sur les données de test.

7. Comme les constantes utilisées dans les noeuds internes d'un arbre de décision.

timation du coût et de l'effort, puis pour les modèles de la fiabilité afin de réduire le bruit dans les prédictions. Dans la littérature, Fenton présente des techniques sophistiquées de calibrage pour le cas de la fiabilité [Fenton et Pfleeger, 1997]. Par ailleurs, Pfleeger [1998] souligne que ces techniques sont utilisables pour les autres tâches de prédiction de la qualité du logiciel.

Zuse [1998] considère le calibrage comme une tâche de mesurage. Il définit le calibrage d'une mesure m comme la modification d'un système relationnel numérique à l'aide d'une fonction f tel que $m' = f(m)$, sans modifier le système relationnel empirique.

3.8.2.1 Calibrage basé sur l'ajustement de paramètres

Ce type de calibrage est utilisé dans le cas des modèles statistiques comme COCOMO. Il consiste à ajuster un ensemble de paramètres du modèle pour l'adapter à un environnement particulier. Plusieurs modèles commerciaux d'estimation du coût de développement comme FAST, PRICE et SEER sont conçus pour servir d'outils d'estimation pour une large gamme d'organisations. Le calibrage, dans ce cas, est nécessaire pour augmenter la capacité prédictive du modèle générique en le « spécialisant » pour une nouvelle situation (un contexte d'organisation particulier, voir la section 7.5).

Dans le cas de COCOMO le calibrage ajuste les paramètres a et b de l'équation 3.1 et un ensemble de facteurs de coût. Malgré un désaccord concernant le nombre de facteurs d'ajustement d'un modèle du coût ou de l'effort, les chercheurs s'entendent sur la nécessité du calibrage [Goodman, 1993].

3.8.2.2 Calibrage basé sur l'apprentissage par renforcement

Ce type de calibrage est utilisé pour améliorer la prédiction des modèles la fiabilité. Durant un certain temps du fonctionnement du logiciel, ces modèles sont utilisés de la même manière, sans aucun ajustement. Pendant ce même temps, des défaillances surviennent et des données avec lesquelles on peut juger la capacité prédictive des modèles sont disponibles. Ces deux sources d'informations constituent un moyen pour améliorer la capacité prédictive d'un modèle de fiabilité. Cette idée vient du fait qu'on peut apprendre à partir des défaillances qui surviennent et à partir des mauvaises prédictions déjà effectuées, de leurs natures et de leurs causes. Ceci peut être fait très tôt, après un nombre d'observations, afin d'améliorer les prédictions futures. Ce processus est appelé re-calibrage [Fenton et Pfleeger, 1997], il se base sur le diagramme u-plot. Les détails sur cette technique peuvent être trouvés dans [Fenton et

Pfleeger, 1997], [Brocklehurst et *al.*, 1990] et [Denton, 1999].

3.8.2.3 Calibrage basé sur les algorithmes génétiques

Ce calibrage est utilisé par Azar [2004] pour améliorer la capacité prédictive des modèles de stabilité de composants logiciels (classes).

Un modèle est un ensemble de règles de classification et chaque règle est un ensemble de conditions associées à une réponse de la forme, « *SI DIT* < 6 *OU coh* $> 0{,}2$ ALORS *instable* (composant instable)». Azar utilise une fonction de calibrage fondée sur les opérateurs d'un algorithme génétique pour dériver un modèle adapté un environnement particulier. Ce processus de calibrage consiste à recombiner les conditions des règles, à modifier les valeurs seuils des conditions ou à modifier la réponse d'une règle. Ceci sert à améliorer la capacité prédictive du modèle calibré.

3.8.3 Approche basée sur la combinaison

La combinaison des modèles est une approche très utilisée dans l'apprentissage, précisément dans le but d'améliorer les prédictions des modèles déjà entraînés (existants). Cette approche consiste dans la plupart des cas à faire la combinaison linéaire des prédictions (sorties) d'un ensemble de modèles. On utilise souvent des techniques comme celles présentées dans la section 4.6. Le résultat de cette méthode de combinaison est une prédiction moins biaisée.

Lyu et Nikora [1992] ont examiné quatre méthodes pour combiner différents types de modèles de prédiction de la fiabilité, notamment les modèles logarithmique, exponentiel et de Littlewood et Verrall. Ils ont comparé la performance de la combinaison avec celles des modèles composants[8]. Ils ont trouvé que la combinaison a permis une meilleure capacité prédictive. Quatre méthodes ont été utilisées pour combiner trois modèles composants afin d'obtenir un modèle composite[9]. Ces méthodes sont les suivantes :

La première est la combinaison linéaire à coefficients égaux (CLE), où chaque modèle est pondéré par $\frac{1}{N}$, avec N qui est le nombre de modèles composants (3 dans ce cas).

La deuxième est la combinaison linéaire à coefficients inégaux (CLI). Pour chaque observation, la

8. Un modèle composant est un modèle qui participe à la combinaison.
9. Un modèle composite est le modèle formé par la combinaison des modèles composants.

moyenne pondérée des trois prédictions composantes est calculée en favorisant la médiane des prédictions. En effet, pour la combinaison de trois modèles, la plus faible prédiction ainsi que la plus forte contribuent par le un sixième à la prédiction finale et la prédiction médiane contribue par les quatre sixièmes. Les coefficients sont alors déterminés dynamiquement en se basant sur la prédiction des trois modèles.

La troisième est la combinaison linéaire orientée médiane (CLM). À chaque observation d'entrée, la prédiction composante est égale à la médiane des prédictions et les autres prédictions sont ignorées. La détermination de la médiane est dynamique (à chaque observation d'entrée).

La quatrième est la combinaison linéaire à coefficient dynamique (CLD). C'est la méthode la plus performante selon Lyu et Nikora [1992]. Cette combinaison utilise la tecnnique de la variation de la vraisemblance « préquentielle » [10] pour déterminer les valeurs des coefficients de pondération des modèles composants [Denton, 1999].

3.8.4 Conclusion sur les approches d'amélioration

En analysant la littérature, nous constatons que la majorité des approches d'amélioration de la prédiction ont pour but la maximisation de la capacité prédictive. Ceci est les cas des approches de calibrage et de combinaison. Nous trouvons par ailleurs certains travaux, comme ceux décrits dans les sections 3.8.1.1 et 3.8.1.3 qui donnent importance à la facilité d'interprétation des modèles. On souligne dans ces travaux que le modèle doit servir d'outil d'aide à la prise des décisions dans le processus d'amélioration de la qualité du logiciel. On conclut que des modèles causaux, basés par exemple sur les réseaux bayésiens ou sur la logique floue, sont recommandés pour l'amélioration la prédiction tout en garantissant la traçabilité des relations exprimées par le modèle.

Les approches d'amélioration de la prédiction que nous avons présentées visent généralement l'amélioration explicite d'un seul critère d'évaluation du modèle. Cette limitation est plus distingue lorsque il s'agit d'un critère important comme la facilité d'interprétation. Par exemple, une approche basée sur la combinaison a pour objectif principal l'amélioration de la capacité prédictive, cependant, elle néglige le critère de facilité d'interprétation du modèle. Par consequent, après des combinaisons comme celles présentées dans la section 3.8.3, on ne peut pas savoir quel modèle est responsable de la valeur prédite.

10. Un aperçu sur cette technique est présenté dans [Fenton et Pfleeger, 1997].

Ce manque de traçabilité rend l'interprétation du modèle composite difficile.

Afin de contourner la limitation de ces approches il est question de considérer simultanément plusieurs propriétés des modèles prédictifs. Considérer toutes les propriétés peut s'avérer une tâche très difficile, cependant, viser les plus importantes est plus admissible.

L'avantage des approches existantes de combinaison réside dans la réutilisation des connaissances offertes par les modèles déjà construits. Profiter de plusieurs expertises est avantageux pour la capacité prédictive et la capacité de généralisation du modèle, sauf que les méthodes de combinaisons utilisées risquent de dérager le critère d'interprétabilité. Une solution envisageable consiste à trouver une méthode de combinaison qui préserve la facilité d'interprétation du modèle composite.

3.9 Conclusion

Ni cette présentation de l'état de l'art, ni cette dissertation n'aborde la question de mesurer ou de ne pas mesurer la qualité et ses attributs, car nous considérons que cette question est déjà répondue par plusieurs chercheurs. Durant les dernières années, plus d'un millier de métriques sont proposées et plus de 5 000 papiers concernant le mesurage et la qualité ont été publiés [Zuse, 1998]. Pour cette même raison, la présente dissertation ne peut pas couvrir toute l'histoire ni tout l'état de l'art de la qualité du logiciel, avec son mesurage et sa prédiction. Nous avons préféré orienter ce chapitre vers la réalisation de trois objectifs que nous jugions nécessaires pour le travail présenté.

Ainsi, nous avons abordé l'historique de la qualité de son mesurage et de sa prédiction dans l'objectif de présenter leur importance et leurs concepts de base. Par la suite, nous avons consacré une grande partie à la présentation de la modélisation de la qualité et de l'état de l'art du mariage entre la prédiction de la qualité et les techniques empruntées de l'IA. La dernière partie de ce chapitre a discuté les problèmes de la prédiction et les solutions proposées dans la littérature pour l'améliorer.

Chapitre 4

CAMP : Approche de combinaison et d'adaptation des modèles prédictifs

4.1 Introduction

Ce chapitre présente une approche générale de combinaison et d'adaptation des modèles prédictifs (que nous appelons CAMP) pour améliorer la prédiction de la qualité du logiciel. Nous entendons par amélioration de la prédiction, la prise en compte d'une ou plusieurs propriétés de modèles prédictifs de qualité afin de les promouvoir. Pendant que les tentatives d'améliorations existantes (voir section 3.8) considèrent habituellement une seule propriété de modèles, nous travaillons dans notre approche sur la considération de trois propriétés de modèles. Nous nous intéressons explicitement à la capacité prédictive, à la facilité d'interprétation et à la capacité de généralisation [1]. Notre choix de ces trois propriétés est justifié dans la section 3.5.4 où nous proposons un ordre de priorités des propriétés de modèles. Afin de justifier certains choix d'idées et orientations, nous commençons par présenter les motivations de notre approche. La première section contient une analyse des principales causes du problème de la faible capacité prédictive des modèles de qualité. Par la suite, nous précisons, dans la deuxième section, les objectifs visés par notre approche CAMP. Dans la troisième section, nous fournissons une description formelle du problème ciblé de combinaison de modèles, tout en tenant compte des objec-

1. Nous rappelons que nous ne prétendons pas à la généralisation absolue, ou à l'universalité du modèle. Nous visons la capacité du modèle à supporter au moins l'évolution du contexte d'une organisation en particulier.

tifs fixés. Dans la quatrième section, nous passons en revue et discutons les méthodes habituellement utilisées pour combiner des modèles. Nous consacrons la cinquième et la sixième section à la description de l'approche CAMP, en présentant ses principes et ses mécanismes. Les techniques utilisées pour définir les mécanismes et appliquer les principes de CAMP sont présentées dans la septième section. Les concepts de ces techniques ont été décrits dans le chapitre 2, leurs adaptations respectives à notre problème dépendraient essentiellement du type des modèles prédictifs de la qualité mis en jeux. Ces adaptations seront décrites sommairement dans ce chapitre, elles seront détaillée dans les chapitres 5 et 6 portant respectivement sur deux applications de CAMP à deux différents types de modèles.

4.2 Analyses et motivations

Les modèles de qualité du logiciel souffrent de plusieurs problèmes qui ont été décrits dans le chapitre 3. Dans cette section, nous n'allons pas nous intéresser à tous les problèmes mais plutôt nous concentrer sur le problème de la faible capacité prédictive des modèles. Notre analyse consiste à chercher progressivement les origines de ce problème pour lui trouver une cause principale que nous pouvons tenter de résoudre ou contourner. Ainsi, nous choisissons en connaissance de cause, les alternatives les plus sûres qui amènent à la solution la plus appropriée.

En analysant la faible capacité prédictive des modèles (voir la section 4.2.1), deux problèmes sont mis en cause successivement : la faible représentativité de données (voir la section 4.2.2) et la pénurie des données (voir la section 4.2.3). Par la suite, une alternative de contourner ce dernier problème (la pénurie des données) est montrée dans la section 4.2.4 par une comparaison théorique entre la combinaison des données et la combinaison des modèles. Le résultat de cette comparaison est une prémisse à l'idée principale de l'approche CAMP.

4.2.1 Analyse de la faible capacité prédictive

Les expériences montrent que les modèles de prédiction présentent une grande variation vis-à-vis de leur capacité prédictive [Fenton et Pfleeger, 1997], [Pfleeger, 1998]. En effet, la capacité prédictive varie d'un modèle à un autre et d'un ensemble de données [2] à un autre. Une expérience menée par Abdel

2. Ensemble de données sur lequel on applique le modèle, que nous allons appeler souvent dans cette dissertation « ensemble d'application ».

Ghaly dont le but était de comparer plusieurs modèles de fiabilité a montré des écarts importants entre les prédictions des différents modèles utilisés sur un même ensemble de données (données de Musa [Fenton et Pfleeger, 1997]). La figure 4.1 illustre les résultats trouvés [Abdel-Ghaly et *al.*, 1986] avec les modèles de fiabilité JM, GO, LNHPP, DU et LV (voir section 3.5.2.2 pour plus d'informations sur certains de ces modèles).

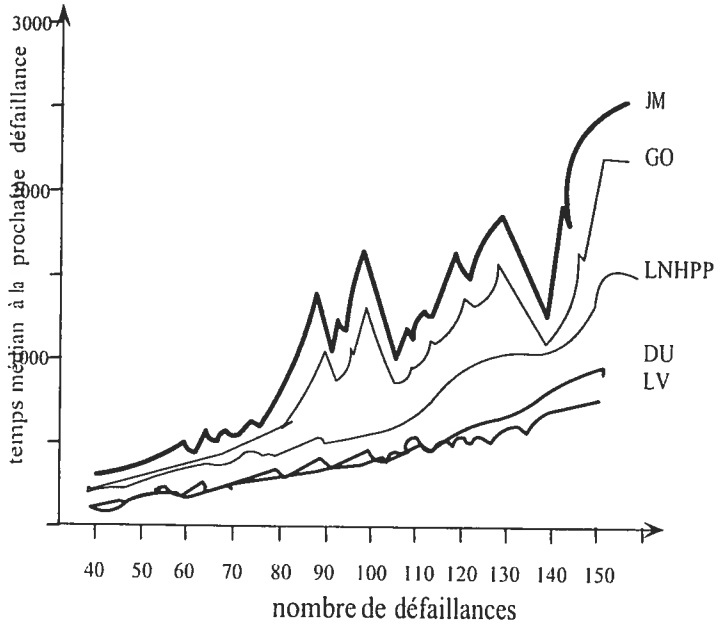


FIG. 4.1 – *Problème de la capacité prédictive : cas des modèles de fiabilité*

De tels résultats sont typiquement décourageants pour les utilisateurs potentiels des modèles de prédiction de la qualité, car la capacité prédictive d'un modèle est un critère important pour sa crédibilité et son acceptation. Une faible capacité prédictive se manifeste de deux manières. D'une part, les prédictions sont biaisées car les valeurs prédites sont considérablement différentes des valeurs réelles, D'autre part, les prédictions sont bruitées parce que les valeurs prédites de cas similaires varient d'une façon nettement différente de la variation des valeurs réelles correspondantes. Pour une fin comparative, certaines techniques notamment statistiques, sont utilisées pour analyser la capacité prédictive d'un modèle. Dans le cas des modèles de la fiabilité, Abdel Ghaly suggère le diagramme « u-plot » pour analyser le biais et la vraisemblance « préquentielle » afin d'analyser le bruit [Pfleeger, 1998].

Ces techniques, comme elles sont décrites par Fenton [1997], aident à comparer les modèles entre

eux et à décider si un modèle est meilleur qu'un autre pour un ensemble de données particulier. Par contre, elles ne peuvent pas identifier d'une manière définitive le meilleur modèle. D'ailleurs, il est impossible de connaître, *a priori*, quel modèle utiliser car le comportement d'un modèle dépend d'une part de l'ensemble des données sur lesquelles ce dernier va être appliqué et qui est *a priori* inconnu et d'une autre part de la représentativité des données qui ont été utilisées pour entraîner ce modèle [Abdel-Ghaly et *al.*, 1986; Fenton et Pfleeger, 1997; Denton, 1999].

4.2.2 Représentativité des données

La plupart des travaux analysant les modèles de qualité reconnaissent que la principale cause de l'inexactitude de ces derniers est le grand écart qui existe entre le milieu où le modèle est construit et celui où le modèle est utilisé. En effet, lorsqu'un modèle est construit à partir d'un ensemble particulier de données, sa capacité prédictive est à son niveau maximal sur cet ensemble. Lorsqu'il est appliqué sur un deuxième ensemble de données, sa capacité prédictive baisse selon l'écart entre l'ensemble de construction et celui d'application. Il s'agit du problème de la représentativité des données utilisées pour construire le modèle. C'est en fait la similarité entre les données de construction et celles d'application qui fait augmenter ou diminuer cette représentativité. La figure 4.2 montre le cas le plus habituel de l'écart entre les données de construction et celles d'application. Cet écart est exprimé par une faible intersection entre l'ensemble D_c de construction et D_a d'application. $\mathcal{V}$ représente tout l'espace d'entrée (données possibles).

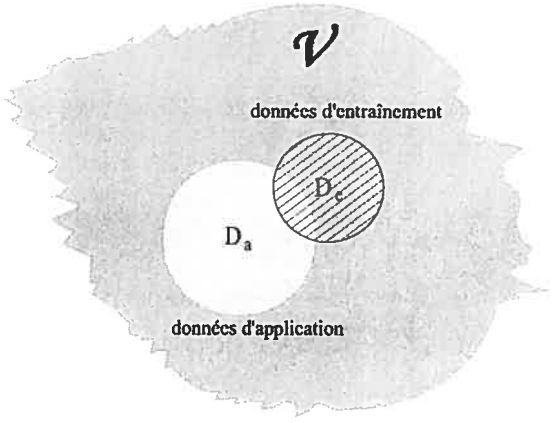


FIG. 4.2 – *L'écart entre l'ensemble de construction et celui d'application*

Concrètement, les ensembles de données peuvent se ressembler essentiellement dans certains critères comme la répartition des données dans l'espace, la nature et la taille des systèmes logiciels à partir desquels les données sont collectées, la densité de certaines données, etc. Intuitivement, plus la taille des données utilisées pour construire le modèle est grande, plus on peut favoriser l'existence de certains de ces critères de ressemblance. Ainsi, on augmente la chance d'avoir une grande intersection entre l'ensemble de construction et d'autres ensembles d'application.

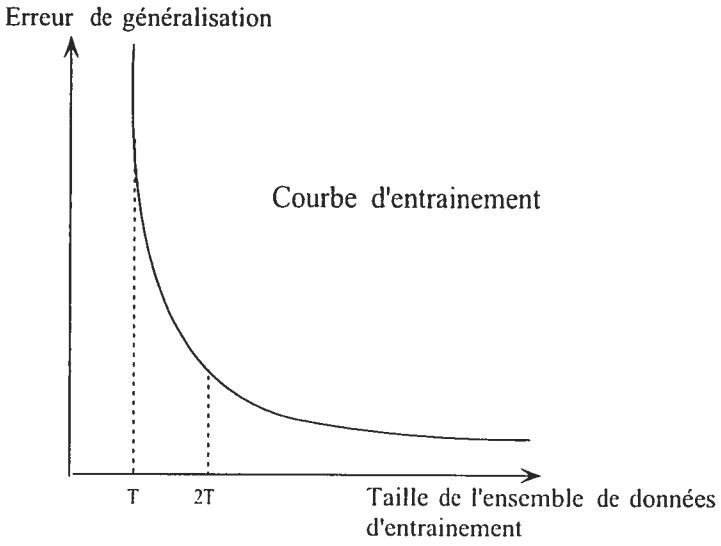


FIG. 4.3 – *Influence de la taille d'échantillon de données d'entraînement sur la performance du modèle*

La figure 4.3 montre l'influence de la taille des données de construction sur la performance de modèle (résultats des travaux présentés dans [Kai et Boon, 1997]). Dans cet exemple, si on double la taille de l'ensemble des données d'entraînement, l'erreur de généralisation diminue et la capacité prédictive du modèle augmente, surtout au début du processus d'apprentissage quand la taille des données est petite (cas habituel en génie logiciel, comme on peut le voir dans section 4.2.3). Avoir un grand ensemble de données permet une meilleure représentativité grâce à une diversité et à une plus grande couverture de l'espace des données. Par conséquent, la combinaison des ensembles de données est un moyen conventionnel et encourageant[3] pour produire un meilleur modèle. Cependant, cette solution est tributaire de la disponibilité de données, un « cauchemar » dans la prédiction de la qualité du logiciel.

3. « Plus il y a de données, mieux c'est ».

Ceci nous ramène à analyser le problème de la pénurie de données dans l'espoir de résoudre celui de la représentativité de données.

4.2.3 Pénurie de données

Malgré la nécessité d'établir des bases scientifiques appropriées en génie logiciel (lois, mesures, modèles, etc.) et de mener des études empiriques rigoureuses, ce domaine souffre de la pénurie de données publiées sur des systèmes logiciels industriels [Fenton et *al.*, 2002]. En particulier, la modélisation de la qualité du logiciel en souffre davantage, en dépit de certains efforts précieux d'un nombre réduit de groupes de recherches qui ont réussi à travailler sur des données provenant de vrais systèmes industriels (voir par exemple [Shen et *al.*, 1985], [Levendel, 1990], [Daskalantonakis, 1992], [Kenney et Vouk, 1992], [Carman et *al.*, 1995], [Kaaniche et Kanoun, 1996], [Khoshgoftaar et *al.*, 1996a], [Ohlsson et Alberg, 1996] et [Mao et *al.*, 1998]).

Ce problème de pénurie de données affecte directement la représentativité des ensembles utilisés pour la construction de modèles de la qualité. En effet, certains chercheurs construisent et valident leurs modèles sur des petits ensembles de données qui représentent des cas étroitement liés aux particularités de l'environnement où ces données sont collectées. Dans des situations pareilles, les modèles produits sont imprécis, instables et peu généralisables. La pénurie de données est causée par le manque de programmes de contrôle de la qualité et de méthodologies de collections automatiques de données au sein des organisations, lui-même dû en grande partie au coût élevé des collectes de données. Par exemple, le laboratoire de génie logiciel de la NASA dépense 15% du coût du développement de ses projets sur la collecte et le traitement des données [Chooman, 1983]. Une autre cause de cette pénurie est la réticence des organisations à divulguer des données qui peuvent nuire à la confidentialité professionnelle ou à la réputation de ses produits.

La résolution du problème de pénurie de données nécessite un effort considérable sans pour autant garantir un bon résultat. Cette difficulté réside dans la nature des tâches que l'on peut entreprendre pour vaincre la pénurie de données: on doit sensibiliser les organisations à intégrer un programme de qualité, à développer ou à adopter une méthodologie de collecte, de traitement et de validation de données, inviter les chercheurs à publier les données qu'ils utilisent pour construire leurs modèles, etc. À court terme, la réalisation de telles tâches est très difficile.

4.2.4 Combinaison de données versus combinaison de modèles

Supposons qu'un ensemble d'échantillons de données soit disponible, une première approche intuitive est de combiner tous ces échantillons pour entraîner un seul modèle de prédiction plus performant en utilisant un algorithme d'apprentissage. Cette approche est fondée sur la règle « plus il y a de données, mieux c'est » [Kai et Boon, 1997]. Une autre approche, qui diffère de la première par la manière avec laquelle les données sont utilisées, entraîne un modèle de prédiction sur chaque échantillon de données puis combine les modèles produits. La première approche est appelée « combinaison de données », la deuxième est appelée « combinaison de modèles ».

La disponibilité de différents ensembles de données fournit une certaine variation de la représentation des données dans un espace d'observations et les modèles construits séparément, à partir de ces ensembles indépendants de données, deviennent des « spécialistes » dans différentes parties de l'espace. Par conséquent, la combinaison de ces modèles permet une coopération entre ces spécialistes. Chaque modèle a une participation plus ou moins importante, selon la partie de l'espace où l'on se trouve et selon la façon avec laquelle la combinaison a été effectuée (voir section 4.6). Cependant, la combinaison des données réduit la variation de ces dernières et en forme une représentation globale.

La figure 4.4 est une version modifiée d'une figure proposée dans [Kai et Boon, 1997] qui explique cette dernière constatation en utilisant la densité des données dans un espace à une seule dimension. Un algorithme d'apprentissage utilisant des ensembles combinés de données (voir la figure 4.4(b)) produit toujours un modèle avec une bonne performance dans la région de densité élevée (r_1) alors que les régions de faible densité (r_2 et r_3) peuvent être négligées, c'est-à-dire que la performance du modèle est faible à l'extérieur de r_1. En revanche, les modèles construits à partir des ensembles séparés des données A et B (voir la figure 4.4(a)) fonctionnent mieux sur leurs régions respectives et particulièrement sur r_2 et r_3. Cette performance est expliquée par la vue plus limitée (concentrée) sur les données que possède chacun des modèles.

À titre illustratif, les travaux publiés par Meir [1994] et par Sollich et Krogh [1996] ont comparé la performance de la combinaison des modèles de type régression linéaire à celle du modèle résultant de la combinaison des ensembles séparés de données. Ils ont attribué la supériorité de la combinaison des modèles à la réduction de la variance des données causée par la combinaison des ensembles.

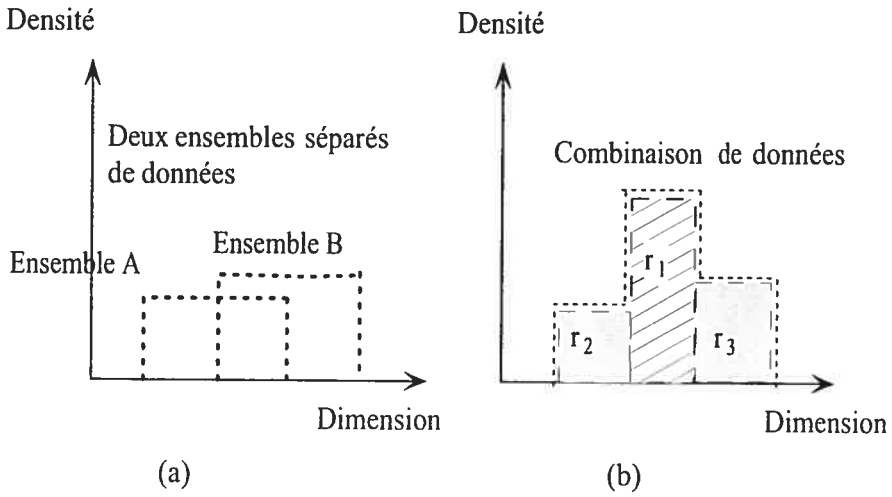


FIG. 4.4 – *Différentes vues sur les données : Ensembles combinés Versus ensembles séparés*

En dépit de la disponibilité des données, le choix entre les deux alternatives de combinaison est en faveur de la combinaison des modèles [Kai et Boon, 1997]. En situation de pénurie de données sur de vrais systèmes logiciels, la combinaison des modèles s'impose davantage.

4.3 Hypothèses de l'approche

Nous présentons dans ce chapitre une approche générale de combinaison et d'adaptation de modèles. Par ailleurs, une spécialisation de cette approche est nécessaire en vue de son application à différents types de modèles. La diversité des types de modèles et de leurs techniques de construction constitue une entrave à l'application de notre approche à tous les types de modèles. Dès lors, nous limitons notre spécialisation, dans le présent travail, aux modèles non traditionnels basés essentiellement sur les techniques d'intelligence artificielle qui constituent une tendance de la modélisation de la qualité (voir section 4.3.1) et plus précisément, à ceux de types classificateurs (voir section 4.3.2). Par conséquent, dans la présentation de notre approche générale, ces types de modèles sont les plus considérés.

4.3.1 Tendance de la typologie des modèles

Les solutions qui sont apportées aux problèmes de la prédiction de la qualité sont fondées principalement sur l'utilisation des techniques d'apprentissage et surtout sur celles qui considèrent certaines propriétés de modèles comme la facilité d'interprétation de modèles. Par exemple, face à l'importance de cette propriété, certains chercheurs comme Fenton [1999b; 2000; 2000b; 2002], Neil [2000a; 2001] et Boehm [1999] ont proposé ces dernières années, l'utilisation des réseaux bayésiens comme solution potentielle pour modéliser la causalité dans le processus de prédiction de la qualité du logiciel.

Les arbres décisions ont été grandement utilisés pour permettre une meilleure capacité d'explication des modèles produits [Porter et Selby, 1990a; Esteva et Reynolds, 1991; Briand et *al.*, 1993b; Briand et *al.*, 1993a; Porter, 1993a; Briand et *al.*, 1993a; Mao et *al.*, 1998; Lounis et *al.*, 1998]. Aussi, parallèlement à ces tendances, nos priorités dans cette dissertation, portent sur les modèles non traditionnels. Ces modèles tiennent compte des aspects de la prédiction en génie logiciel, comme la subjectivité, l'incertitude. Ils possèdent une structure transparente qui permet au praticien de comprendre les relations causales entre les attributs prédicteurs et le facteur de qualité prédit. Nous croyons aussi que ces modèles basés essentiellement sur les techniques d'apprentissage constituent la tendance future de la modélisation de la qualité de logiciel.

4.3.2 Prédictions versus classifications

Plusieurs problèmes de prédiction sont réduits à des problèmes de classification. Cette réduction est surtout utilisée lorsque la variable à prédire possède une distribution déséquilibrée. Considérons la prédiction de la qualité du logiciel, exprimée en nombre de défauts. La majorité des composants contiennent zéro ou très peu de défauts, ce qui engendre une distribution complètement déséquilibrée du nombre de défauts. Dans ce cas, au lieu de prédire le nombre de défauts potentiels dans un composant logiciel, on détermine la susceptibilité du composant à contenir des défauts. Ainsi, on prédit la classe à laquelle le composant appartient (*à haut risque* ou *à faible risque*). Par conséquent, le modèle de prédiction est réduit à un modèle de classification binaire. Une autre raison pour laquelle on convertit la variable à prédire en une variable dans l'échelle ordinale ou nominale est de mener un discours facilement compréhensible sur la qualité du logiciel. Pour ces raisons, les mécanismes de notre approche

sont illustrés par des exemples de modèles de classification.

4.4 Objectif et idée de l'approche

4.4.1 Constats

Suite à l'analyse des problèmes majeurs dont souffre la prédiction de la qualité du logiciel, nous sommes convaincus que l'inexactitude et la difficulté de choix des modèles sont causées principalement par la non représentativité des données qui est engendrée avant tout par la pénurie de données.

La résolution du problème de pénurie de données nécessite un effort considérable sans pour autant garantir un bon résultat. En fait, à part quelques initiatives très rares et coûteuses, comme celle du laboratoire de génie logiciel de la NASA qui vient de publier une base de données collectées sur leurs systèmes industriels, la pénurie de données demeure un problème difficile à résoudre.

Parallèlement à la pénurie de données publiées, plusieurs modèles de prédiction sont publiés dans la littérature [Fenton et Ohlsson, 2000; Fenton et Neil, 2000b; Fenton et Neil, 1999a; Denton, 1999; Aroui, 1996]. Ces modèles peuvent être de type connu, utilisant une technique bien identifiée comme la régression linéaire, les réseaux de neurones, les arbres de décision, le réseaux bayésiens, le raisonnement à base de cas (voir la section 3.6.2), ou peuvent être également exprimés dans un langage naturel sans avoir utilisé une formalisme déterminé : sous forme d'affirmations, opinions d'experts, résultats de vérification d'une hypothèse, etc. Ce dernier type est très fréquent (voir des exemples dans la section 3.5.2.3). Même si le nombre de modèles varie considérablement d'un facteur de qualité à un autre, il est toujours en croissance [Fenton et Pfleeger, 1997; Pfleeger, 1998; Fenton et Ohlsson, 2000; Denton, 1999]. La fiabilité est un exemple de facteurs qui attirent le plus d'efforts de modélisation et dont le nombre de modèles est de plus en plus croissant [Fenton et Pfleeger, 1997; Denton, 1999].

Le dernier constat est fondé sur un raisonnement qui considère que les modèles disponibles sont des abstractions faites à partir de données qui ne sont pas ou ne sont plus disponibles. Ils sont d'ailleurs les seules traces disponibles sur les données. Par conséquent, la pénurie de données peut être contournée en utilisant les connaissances représentées dans ces modèles.

4.4.2 Objectif

L'objectif de notre recherche est de permettre aux praticiens d'avoir un modèle de prédiction possédant les propriétés suivantes :

1. Une meilleure capacité prédictive : les prédictions doivent être précises lorsqu'on utilise le modèle dans un environnement logiciel propre à une organisation (dans [Aroui, 1996], cette propriété est appelée *la capacité réplicative*).
2. Une bonne capacité de généralisation : le modèle doit avoir une aptitude à tenir compte au moins de certaines variations futures dans le contexte de l'organisation (dans [Aroui, 1996], cette propriété est appelée la *capacité prévisionnelle*). Le modèle doit alors garder un certain niveau de performance à l'extérieur du contexte actuel.
3. Une interprétation facile : le modèle doit aider dans l'explication des prédictions et dans la prise de décision quelques soient les phases du processus de développement du logiciel, en exprimant les relations de causalité entre les entrées et les sorties du modèle (cette propriété est appelée aussi la *transparence*) [Gray et MacDonell, 1997; Fenton et *al.*, 2002; Aroui, 1996]. C'est aussi par la connaissance des valeurs des métriques responsables de chaque décision qu'on favorise la facilité d'interprétation du modèle.

4.4.3 Idée maîtresse de l'approche

Nous proposons une approche d'amélioration de la prédiction de la qualité. Cette approche réutilise les modèles existants qui prédisent un même facteur de qualité pour produire un modèle plus adéquat à un environnement logiciel particulier. Cet environnement est représenté par un petit ensemble de données collectées à partir de certains systèmes logiciels développés par une organisation particulière ou dans un contexte logiciel particulier. Le modèle résultant doit vérifier les propriétés citées ci-dessus.

Inspirée du fameux mélange d'experts, notre idée consiste à combiner ces modèles et à les adapter pour obtenir une meilleure prédiction de la qualité. Nous appelons notre approche CAMP (approche de Combinaison et d'Adaptation de Modèles Prédictifs).

4.5 Description formelle du problème

Dans cette section, nous présentons les définitions et le formalisme utilisés dans la suite de ce chapitre, afin de bien décrire notre approche. La notation et les concepts utilisés proviennent des formalismes habituellement utilisés en apprentissage. Pour décrire notre problème d'une manière claire et transparente, nous le relions le plus possible aux notations et aux concepts appropriés au génie logiciel.

4.5.1 Définitions

Définition 4.5.1 *Échantillon de données :*

Un échantillon de données est un ensemble $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ *de n observations ou* points de données *où* $\mathbf{x}_i \in \mathbb{R}^d$ *est un vecteur de d attributs et* $y_i \in C$ *est une valeur de sortie (un label dans le cas de modèles de classification).*

Dans le domaine de la prédiction de qualité du logiciel, une observation $\mathbf{x}_i$ représente un composant bien défini d'un système logiciel (par exemple, une classe dans le cas d'un logiciel OO). La valeur de sortie y_i représente le facteur de qualité à prédire du composant logiciel $\mathbf{x}_i$. Les attributs de $\mathbf{x}_i$ (notés par $x_i^{(1)}, \ldots, x_i^{(d)}$) sont des métriques logicielles (comme le nombre de méthodes, la profondeur dans l'arbre d'héritage). Ces attributs sont considérés comme appropriés au facteur particulier de qualité du logiciel à prédire. $x_i^{(j)}$ prend ses valeurs dans $A_j \subset \mathbb{R}$, où A_j est le domaine de définition du $j^{ième}$ attribut[4]. Le facteur de la qualité y_i prend ses valeurs dans l'ensemble C. Dans le cas d'une classification où y_i peut prendre seulement un nombre fini de valeurs, C est un ensemble fini de ces valeurs possibles. Dans la prédiction de la qualité du logiciel l'espace de sortie C est habituellement un ensemble ordonné $c_1, c_2, \ldots, c_q$ d'étiquettes, ou labels (voir la section 4.3.2).

Définition 4.5.2 *Un modèle :*

Un modèle de qualité est une représentation qui exprime une relation établie entre, d'une part, les attributs d'un composant logiciel $\mathbf{x}$ *et d'autre part, un facteur de qualité y. Cette relation peut être définie par une fonction* $f : \mathcal{V} \mapsto C$ *qui prédit la valeur* $y_i \in C$ *d'un facteur de qualité y de n'importe quelle observation* $\mathbf{x}_i \in \mathcal{V}$. $\mathcal{V}$ *est l'espace d'entrée de f, il est défini par le produit cartésien* $A_1 \times \ldots \times A_d$, *où*

4. Nous notons que la valeur d'un attribut peut être un réel, un entier, une étiquette, etc. Mais, nous la supposons réel pour alléger la notation.

A_j est le domaine de définition du $j^{ième}$ attribut et d est le nombre d'attributs d'entrée.

Un modèle est construit ou validé sur un échantillon de données comme D. La relation que le modèle exprime peut être représentée de plusieurs manières, selon la méthode employée pour sa construction (méthode d'apprentissage, analogie, jugement d'experts ou autres). Par conséquent, la fonction f qui définit le modèle peut prendre plusieurs formes (formules mathématiques, système de règles).

Définition 4.5.3 *Contexte d'organisation :*

Nous appelons contexte d'organisation, *un ensemble de données $D_c = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{nc}, y_{nc})\}$ de nc observations tel que pour tout composant $\mathbf{x}_i$, la valeur du facteur de qualité y_i est connue.*

Le contexte d'organisation est en fait un ensemble de données qui sont collectées sur des systèmes logiciels opérationnels, produits par une organisation donnée. D'un coté, ces données représentent une description détaillée des composants des logiciels produits. Pour chaque composant, cette description comporte un ensemble d'attributs internes (généralement structurels et relationnels) et un attribut externe représentant son comportement dans son environnement. Ce comportement est le facteur de la qualité désiré par l'organisation.

D'un autre coté, ces données reflètent un style de développement de logiciels, une expérience de développeurs, de bonnes et de mauvaises pratiques dans le processus de développement, une expérience antérieure d'organisation en matière de développement d'un type particulier de logiciels, etc. Ces données informent alors sur plusieurs conditions qui peuvent influencer l'atteinte de certains objectifs de la qualité dans l'organisation. C'est le contexte l'activité de développement de logiciels au sein d'une organisation.

4.5.2 Formulation du problème

Nous considérons le problème de combiner et d'adapter N modèles existants $f_1, \ldots, f_N$, afin d'obtenir un modèle de prédiction adéquat pour un contexte d'organisation D_c disponible. Ces modèles existants sont supposées être construits, respectivement, à partir des échantillons $D_1, \ldots, D_N$ et prédisent le même facteur de qualité.

La combinaison et l'adaptation doivent permettre la réalisation des objectifs mentionnés dans la section précédente 4.4: produire un modèle compromis ayant une meilleure capacité prédictive dans le

contexte d'organisation D_c et ayant une bonne capacité de généralisation, tout en étant le plus interprétable possible.

4.6 Alternatives de combinaison

La première façon de combiner les N modèles est de trouver parmi eux le modèle qui prédit mieux que les autres dans le contexte D_c. L'avantage de cette méthode réside dans sa simplicité et dans sa capacité à conserver les propriétés provenant des modèles originaux. Son inconvénient est qu'elle n'utilise que les connaissances d'un seul expert et ne combine pas les expertises.

Une deuxième façon de combiner les modèles, quoique plus compliquée, est de construire une somme pondérée de leurs sorties [Perrone et Cooper, 1993]. Formellement,

$$f(\mathbf{x}) = \sum_{j=1}^{N} w_j f_j(\mathbf{x}),$$

où $w_j \geq 0$, $j = 1, \ldots, N$. Pour trouver les poids par apprentissage, on peut utiliser, par exemple, un algorithme de la famille *ensemble de vote* comme *AdaBoost* [Freund et Schapire, 1997]. Si nous considérons le cas des modèles de classification binaires, avec 1 et -1 comme valeurs de sortie possibles(labels), la combinaison des classificateurs est donnée par :

$$f(\mathbf{x}) = \begin{cases} 1 & \text{si } \sum_{j=1}^{N} w_j f_j(\mathbf{x}) \geq 0, \\ -1 & \text{sinon.} \end{cases} \tag{4.1}$$

Les poids, ou les coefficients de pondération w_j des modèles, ont une interprétation naturelle du fait qu'ils mesurent la confiance en le $j^{ième}$ modèle, sur l'ensemble de données du contexte D_c. Cependant, certaines bonnes propriétés provenant des modèles originaux sont en quelque sorte perdues. Par exemple, l'interpretabilité est dégradée par le fait qu'il y ait plus d'un modèle responsable de chaque décision.

D'autres méthodes de combinaison à coefficients variables sont utilisées pour combiner un ensemble de modèles, afin de produire un modèle plus performant. La plus répandue de ces méthodes est celle utilisée par Jacob et ses collègues, connue sous le nom de « *mélange d'experts* » [Jacobs et *al.*, 1991]. Cette méthode utilise un réseau de pondération qui décide comment déterminer les poids des experts en fonction de chaque point de données. Ce réseau de pondération sélectionne un ou un petit nombre

d'experts qui apparaissent les plus appropriés pour fonctionner sur un exemple de données. Durant leur apprentissage, les poids changent seulement pour les experts choisis. Les experts qui classifient bien l'exemple se voient attribuer plus de poids pour cet exemple alors que les experts qui le classifient mal verront leurs poids diminués. Les autres experts sont considérés spécialisés dans d'autres cas de données suffisamment différents. Durant le processus d'entraînement des poids, les experts se découplent les uns des autres et se spécialisent dans des petites portions de l'espace d'entrées.

Ces méthodes sont plus générales que les méthodes des ensembles sur deux aspects. D'abord, les poids w_j des experts ne sont plus constants mais plutôt évalués en fonction des entrées $w_j(\mathbf{x})$. Deuxièmement, après l'apprentissage des poids, les experts f_j sont entraînés une seconde fois et les deux étapes sont réitérées jusqu'à convergence. En raison de cette deuxième propriété, nous ne pouvons pas utiliser les algorithmes généraux de mélange d'experts. En effet, notre but est de réutiliser des experts existants. De plus, nous ne voulons pas perdre les bonnes propriétés des modèles existants comme la facilité d'interprétation.

Toutefois, il faut mentionner qu'il existe des algorithmes à mi-chemin entre les méthodes d'ensembles et l'approche par mélange d'experts qui apprennent les poids en fonction des données sans avoir changé les experts originaux (voir [Moerland et Mayoraz, 1999; Meir et *al.*, 2000]). Ces méthodes peuvent être appliquées à notre problème mais la complexité croissante des fonctions des poids $w_j(\mathbf{x})$ rend l'interprétation des modèles difficile.

Les méthodes existantes de combinaison ne considèrent pas certaines propriétés de modèles de prédiction comme la facilité d'interprétation, appelée aussi transparence [Merz, 1998]. Cette limitation nous impose la recherche d'autres manières de combiner les modèles de qualité du logiciel. Les méthodes recherchées doivent préserver les propriétés ciblées par les objectifs de notre approche.

4.7 Principes de l'approche CAMP

Trois préoccupations conditionnent notre approche de combinaison et d'adaptation de modèles prédictifs (CAMP). La première concerne la **capacité prédictive** du modèle résultant (ou composite), la deuxième est l'adaptabilité du modèle au changement de l'environnement (de sa construction et de son application), il s'agit de la **capacité de généralisation** du modèle, la troisième est l'aptitude de l'ap-

proche à améliorer, ou du moins à conserver le niveau d'**interprétabilité** des modèles originaux (ou modèles composants). Considérons, par exemple, la combinaison de modèles de type arbres de décision, réputés avoir un bon niveau d'interprétabilité. Nous voulons que notre approche puisse produire un modèle transparent, aussi interprétable qu'un arbre de décision et dont la capacité prédictive et la capacité de généralisation sont meilleures que celles des arbres existants.

Afin de répondre à ces trois préoccupations, nous adoptons les trois principes suivants :

1. **Premier principe** : adapter pour améliorer la capacité prédictive des modèles.
2. **Deuxième principe** : combiner pour généraliser les modèles.
3. **Troisième principe** : décomposer pour faciliter l'interprétation des modèles.

4.7.1 Premier et deuxième principe : adapter et combiner

Soient $f_1, \ldots, f_N$, N modèles qui prédisent un même facteur y de qualité. Ces modèles sont construits respectivement à partir des échantillons $D_1, \ldots, D_N$, qui peuvent éventuellement se chevaucher (couvrir un même sous espace de données). Nous voulons prédire le facteur y de qualité pour les produits logiciels développés par une organisation *DZERO* (représentable par un ensemble virtuel de données D_0). Soit $D_t \subset D_0$, un ensemble de données disponibles formant le contexte de l'organisation *DZERO*. Trois solutions sont envisageables. La première consiste à évaluer tous les modèles sur D_t et à choisir le meilleur d'entre eux. La deuxième consiste à utiliser seulement D_t pour entraîner un nouveau modèle f_t pour prédire tous les cas de D_0. La troisième solution propose de combiner et d'adapter les modèles $f_1, \ldots, f_N$ en utilisant D_t.

Le modèle résultant désiré est un modèle qui doit à la fois être spécialisé dans son contexte D_t et être général par ailleurs, au moins sur tout l'ensemble D_0. C'est-à-dire, il doit bien prédire à l'extérieur du contexte et garder, lorsque ce dernier change ou évolue, une capacité prédictive satisfaisante.

Lorsque le contexte évolue, deux types d'observations lui sont généralement ajoutés. Le premier type représente une nouvelle[5] gamme de composants logiciels et reflète une nouvelle pratique dans le processus de développement. L'ajout de ce type d'observations est très occasionnel. Le second type est le plus fréquent, représente des composants comparables à ceux produits par d'autres organisations, et

5. Nouvelle pour toutes les organisations de logiciels.

reflète des pratiques communes à plusieurs environnements de développement de logiciels.

Une grande partie du deuxième type d'observations est fort probablement représentée par la combinaison des ensembles de données D_j, $j \in \{1,\ldots,N\}$. Plusieurs nouveaux cas du contexte évolué sont ainsi bien prédits pas la combinaison des modèles f_j.

En conséquence, nous constatons que le modèle désiré à la fin doit acquérir (ou avoir une vision sur) deux types de connaissances : (1) des connaissances spécifiques au contexte particulier d'organisation D_t et (2) des connaissances communes (générales) à plusieurs contextes du domaine de développement de logiciel.

Le premier type de connaissances peut être acquis, soit en choisissant un modèle (première alternative), soit en utilisant un algorithme d'apprentissage (deuxième alternative), soit en adaptant certains modèles (troisième alternative). Le deuxième type de connaissances (connaissances communes du domaine) n'est acquis dans les conditions décrites ci-dessus[6] que par la combinaison des « expertises » contenues dans les modèles existants.

Suite au précédent raisonnement, nous retenons donc la troisième alternative qui permet de réunir les deux types de connaissances. Ainsi, en réutilisant des modèles existants, nous acquérons les connaissances communes du domaine (c'est la combinaison). Par ailleurs, en guidant la combinaison par les données spécifiques du contexte de l'organisation, nous considérons les connaissances spécifiques (c'est l'adaptation). Les connaissances des modèles existants sont éventuellement ajustées pour s'adapter et mieux convenir au contexte particulier de l'organisation.

4.7.2 Troisième principe : décomposer

Concernant l'interprétation du modèle, nous proposons une décomposition de chacun des modèles existants en un ensemble d'expertises. Un critère de décomposition est la variation de la performance du modèle d'une région à une autre dans l'espace de données, ou l'espace d'entrée. En effet, un modèle est généralement construit à partir d'un échantillon de données qui ne représente pas tous les cas possibles. Ces mêmes données sont, le plus souvent, réparties d'une manière non équilibrée dans l'espace ce qui rend la capacité prédictive du modèle construit variable d'une région de l'espace à l'autre. La prédiction est plus fiable dans les régions où l'échantillon d'entraînement a une bonne couverture des données.

6. Conditions caractérisées, entre autres, par la disponibilité des modèles et la rareté des données $(D_1 \ldots D_N)$.

Cette représentation inéquitable des régions de l'espace de données nous suggère l'idée d'étudier le comportement ou l'expertise d'un modèle par région de l'espace que nous appelons **partition**.

Concrètement, une partition de l'espace des données représente une gamme de composants logiciels qui partagent un certain nombre de propriétés, comme l'appartenance à une catégorie de taille, de complexité ou de couplage. Par exemple, un modèle peut être plus performant dans la ou les partitions qui correspondent aux composants ayant une faible cohésion, alors que dans les cas de forte cohésion, un autre peut donner des meilleurs résultats.

La figure 4.5 montre l'allure d'une décomposition d'un modèle en expertises.

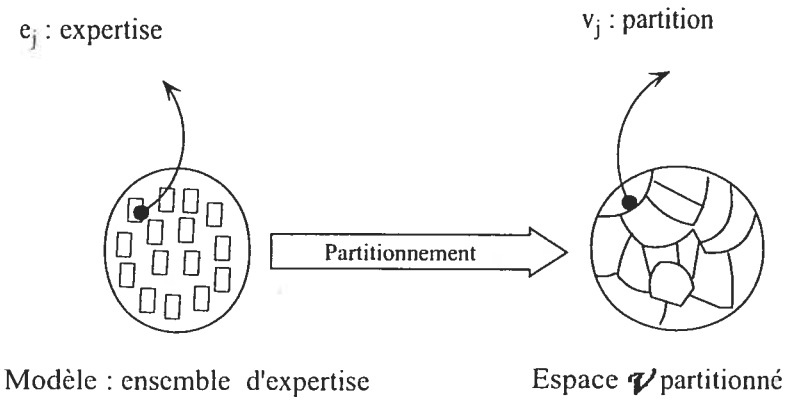


FIG. 4.5 – *Décomposition d'un modèle en un ensemble d'expertises*

Selon le type des modèles, l'espace d'entrée $\mathcal{V}$ peut être traité de deux manières. Dans la première, il est subdivisé en un nombre de partitions prédéterminées par l'algorithme d'apprentissage utilisé pour entraîner le modèle et dans ce cas, une fonction de prédiction est définie pour chaque partition. Par exemple, un modèle de type arbre de décision est décomposable en un ensemble de règles qui couvrent chacune une région de décision. Plus de détails sur la décomposition d'un arbre de décision sont donnés dans le chapitre 5.

Pour la deuxième manière, l'espace d'entrée est vue comme une seule entité. Dans ce cas, la même fonction de prédiction est définie pour tout l'espace. Par exemple, un modèle de type régression linéaire est représenté par la même fonction dans tout l'espace d'entrée. Pour ce second type de modèles, nous procédons à une subdivision de l'espace d'entrée $\mathcal{V}$ en un ensemble de partitions dont le nombre et les dimensions seront des paramètres de notre décomposition. La restriction du modèle à chaque partition

constitue donc la fonction de prédiction en local. Une expertise est dans ce cas définie comme la restriction du modèle sur une partition v de l'espace d'entrée $\mathcal{V}$. Ainsi le modèle est un ensemble d'expertises locales.

Définition 4.7.1 *Expertise :*

Une expertise est la restriction d'un modèle f à un sous-espace, ou partition, $v \subset \mathcal{V}$, définie par une fonction d'expertise $e : v \mapsto C$ qui prédit la valeur du facteur de qualité $y \in C$ de n'importe quelle observation $\mathbf{x}_i \in v$, avec $e(x_i) = f(x_i)$. Nous appelons aussi cette expertise, un expert local.

Définition 4.7.2 *Modèle comme un ensemble d'expertises :*

Un modèle de qualité peut être perçu comme un ensemble d'expertises ou d'experts locaux. Il peut être formellement, défini par

$$f(\mathbf{x}) = \begin{cases} e_1(\mathbf{x}) & si\ \mathbf{x} \in v_1, \\ e_2(\mathbf{x}) & si\ \mathbf{x} \in v_2, \\ \vdots & \\ e_R(\mathbf{x}) & si\ \mathbf{x} \in v_R. \end{cases} \tag{4.2}$$

où chaque e_j, $j = 1,\ldots,R$ est la fonction d'expertise dans la partition $v_j \subset \mathcal{V}$. La décomposition d'un modèle en un nombre d'expertises correspond à un partitionnement de l'espace d'entrée $\mathcal{V}$ en R partitions $v_1,\ldots,v_R$, avec $f(\mathbf{x}) = e_j(\mathbf{x})$ et $\mathbf{x} \in v_j$ et $j = 1,\ldots,R$.

4.7.3 Description sommaire de CAMP

Une récapitulation de notre approche est donnée par la figure 4.6, dans laquelle un ensemble de modèles qui prédisent le même facteur de qualité sont réutilisés et adaptés à un contexte d'organisation représenté par un ensemble de données D_c. Chaque modèle correspond à un partitionnement de l'espace de données en un ensemble de régions (partitions). Sur chaque partition est définie une expertise. Les mécanismes de l'approche CAMP consistent à combiner et à adapter les expertises pour mieux convenir au contexte D_c. Le résultat de CAMP est un modèle composé de deux types d'expertises : (1) des expertises originelles provenant des modèles existants et (2) des expertises adaptées obtenues par modifications d'un nombre d'expertises originelles.

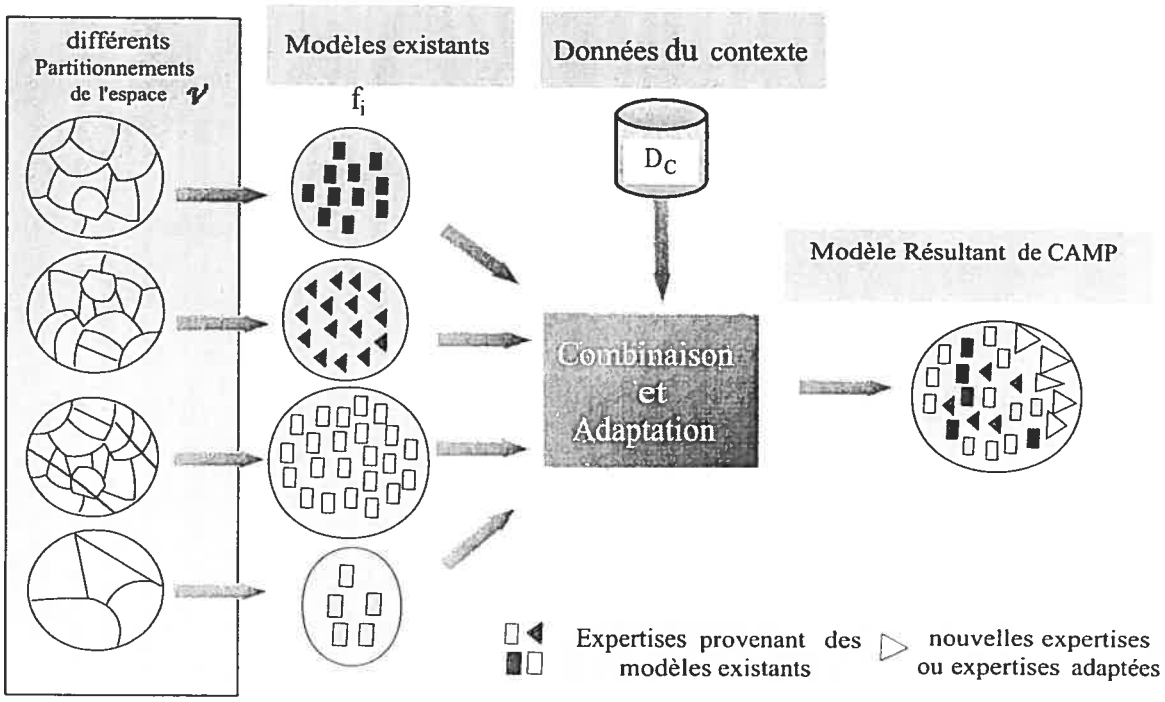


FIG. 4.6 – *Vue d'ensemble de l'approche CAMP*

4.8 Mécanismes de l'approche CAMP

La tâche de combinaison et d'adaptation peut être traitée comme un problème d'apprentissage à haut niveau [7] utilisant les expertises existantes et ayant comme entrées les données d'un contexte D_c. L'induction ou l'apprentissage d'un modèle consiste à construire, progressivement, de meilleures combinaisons d'expertises. Ce processus nécessite plusieurs itérations qui comportent évidement des combinaisons et éventuellement des adaptations de ces expertises. En effet, le processus part d'un ensemble de modèles initiaux qui subissent d'abord, une opération de prétraitement afin de fournir une structure modulaire et unifiée pour tous les modèles (section 4.8.1). Par la suite, l'étape itérative s'amorce, elle alterne des opérations de combinaison (section 4.8.4) et d'adaptation (section 4.8.5) afin de produire de nouveaux modèles. La progression des opérations est guidée par une évaluation des modèles (section 4.8.2) dans le but de choisir les meilleurs pour les itérations suivantes. Le processus se termine lorsque le meilleur

7. Il s'agit d'un apprentissage non pas à partir des données mais plutôt à partir d'expertises entraînées sur des données.

modèle est produit. Les modèles initiaux, ainsi que les modèles intermédiaires, doivent vérifier deux propriétés fondamentales qui sont : *la complétude* et *la cohérence* (voir section 4.8.3).

4.8.1 Prétraitement des modèles

Cette opération consiste en premier lieu à identifier tous les attributs (les métriques) utilisés dans les différents modèles $f_1, \ldots, f_N$. En second lieu, un espace $\mathcal{V}$ sera défini par $\mathcal{V} = A_1 \times, \ldots, \times A_d$, commun pour tous les modèles, où A_j est le domaine de définition unifié de la métrique $x^{(j)}$ et d est le nombre total de métriques utilisées dans tous les modèles. Par la suite, selon le type de modèle, un partitionnement P_i de l'espace $\mathcal{V}$ est associé à chaque modèle f_i. En dernier lieu, chaque modèle f_i est décomposé en un ensemble d'expertises $e_{i1}, \ldots, e_{iR}$.

Cette opération de prétraitement s'effectue au début du processus pour préparer une structure du modèle flexible et manipulable. Cette structure (en morceaux) nous permet de varier aisément la composition en expertise du modèle. En effet, nous pouvons effectuer des modifications de différentes ampleurs, au niveau du modèle et au niveau des expertises (section 4.8.5). C'est grâce à cette structure que nous favorisons plus de transparence qui aide à mieux interpréter les valeurs de la sortie du modèle.

4.8.2 Évaluation d'un modèle : mesure de la capacité prédictive

Les opérations de notre processus sont effectuées dans le but de créer, progressivement, de nouveaux modèles plus performants. Afin d'évaluer ces derniers, de les comparer entre eux et de choisir le meilleur, nous utilisons d'une part un contexte d'organisation D_t comme ensemble d'évaluation et d'autre part une fonction de mesure de performance du modèle qui dépendra de la nature de la prédiction (une régression ou une classification). Cette évaluation est un moyen de guider la progression du processus itératif de combinaison et d'adaptation. Par ailleurs, l'échantillon D_t peut également être vu comme un ensemble d'entraînement au travers de la tâche d'apprentissage que l'approche CAMP réalise. Les fonctions de mesure de performance de modèles sont présentées dans le chapitre 2. Nous nous limitons dans ce travail à l'utilisation des mesures de la performance des classificateurs. Nous supposons que la taille de notre ensemble d'entraînement D_t est n et que des résultats intermédiaires d'évaluation (n_{ij} dans le tableau 4.1) sont rassemblés sous forme d'une représentation matricielle appelée *matrice de confusion* [El Emam, 2000]. Cette matrice a été présentée dans le chapitre 2 pour le cas de la classifica-

tion binaire, sa version généralisée pour la classification à q classes est donnée par le tableau 4.1, où n_{ij} est le nombre d'observations avec label réel c_i mais classifiées c_j, avec $i,j = 1,\ldots,q$. Une revue de la

		Label prédit			
		c_1	c_2	...	c_q
Label réel	c_1	n_{11}	n_{12}	...	n_{1q}
	c_2	n_{21}	n_{22}	...	n_{2q}
	⋮	⋮	⋮	⋱	⋮
	c_q	n_{q1}	n_{q2}	...	n_{qq}

TAB. 4.1 – *Matrice de confusion d'une fonction de décision à q classes.*

littérature nous propose plusieurs mesures d'évaluation de la capacité prédictive des classificateurs (voir la section 2.2.6). Cependant, nous avons choisi deux mesures qui nous semble être les plus utilisées dans la littérature. Ce sont *la proportion correcte*, ou *l'exactitude* , et *l'indice J de Youden*.

4.8.2.1 La proportion correcte

Cette mesure est aussi appelée en anglais *correctness*. La plupart des travaux d'évaluation ou de validation de modèles de classification de la qualité du logiciel utilisent la valeur de la proportion correcte comme mesure de la capacité prédictive, par exemple [Schneidewind, 1994], [Almeida et *al.*, 1998], [Fenton et Kitchenham, 1990] et [Harrell et *al.*, 1996]. Elle est également appelée *exactitude* dans [Glasberg et *al.*, 2000] et [Hanley, 1982]. Cette mesure est employée intuitivement afin de donner le taux de réussite globale de la tâche de classification du modèle. Une bonne raison de son utilisation est la facilité de son interprétation. La proportion correcte est souvent utilisée pour évaluer les classificateurs binaires (voir chapitre 2). Toutefois, elle est facilement généralisable aux cas de la classification non binaire, sa définition est donnée par l'équation 4.3.

$$C(f) = \frac{\sum_{i=1}^{q} n_{ii}}{\sum_{i=1}^{q}\sum_{j=1}^{q} n_{ij}}. \tag{4.3}$$

où f est un classificateur et n_{ij} est le nombre de cas d'évaluation avec label réel c_i et classifiés comme c_j (voir tableau 4.1).

4.8.2.2 L'indice *J*

Cette mesure est aussi appelée en anglais *Youden Jindex* [Youden, 1961]. L'indice *J* est utilisé pour surmonter le problème de la répartition non équilibrée des exemples sur les classes, propre à certaines mesures d'évaluation des classificateurs binaires couramment utilisées dans la littérature de la qualité du logiciel, telles que la proportion correcte, les fausses classifications de type I et de type II, le coefficient kappa, etc. Dans le cas des classifications binaires, l'indice *J* est défini par l'équation 4.4.

$$J = \frac{1}{2}\left(\frac{n_{11} - n_{12}}{n_{11} + n_{12}} + \frac{n_{22} - n_{21}}{n_{22} + n_{21}}\right) \tag{4.4}$$

La mesure de Youden consiste à considérer que la capacité prédictive d'un classificateur binaire est la moyenne des deux termes suivants :

$$\frac{n_{11} - n_{12}}{n_{11} + n_{12}}.$$

qui mesure la réussite du classificateur dans l'identification des cas de la première classe et :

$$\frac{n_{22} - n_{21}}{n_{22} + n_{21}}.$$

qui mesure la réussite du classificateur dans l'identification des cas de la deuxième classe.
On déduit ainsi, une autre forme de l'indice *J*, donnée par l'équation 4.5, où *p* et *s* sont respectivement, la spécificité et la sensitivité du classificateur, décrites dans le tableau 2.2 du chapitre 2.

$$J = \frac{n_{11}}{n_{11} + n_{12}} + \frac{n_{22}}{n_{22} + n_{21}} - 1 = p + s - 1. \tag{4.5}$$

Dans la littérature, l'indice *J* est habituellement utilisé pour évaluer les classificateurs binaires, mais il peut être généralisé pour servir aux classifications à *q* classes. Sa forme généralisée est définie par l'équation 4.6.

$$J = \frac{1}{q}\sum_{i=1}^{q}\frac{n_{ii} - \sum_{j \neq i}^{q} n_{ij}}{\sum_{j=1}^{q} n_{ij}}. \tag{4.6}$$

où le terme $\frac{n_{ii} - \sum_{j \neq i}^{q} n_{ij}}{\sum_{j=1}^{q} n_{ij}}$ mesure la réussite du classificateur dans l'identification des cas de la $i^{ième}$ classe. L'indice *J* est la moyenne des réussites du classificateur dans l'identification des cas des différentes classes. Il varie de -1 (pour la plus faible capacité prédictive) à 1 (lorsque la capacité prédictive est parfaite). Toutefois, nous proposons de relativiser cette mesure, afin de faciliter son interprétation et sa comparaison avec celle de la proportion correcte. Nous effectuons donc une transformation linéaire de

J, et nous l'appelons *indice J transformé* (JT), il prend ses valeurs dans l'intervalle $[0,1]$. JT est donné par l'équation 4.7.

$$JT = \frac{J+1}{2} = \frac{1}{q}\sum_{i=1}^{q}\frac{n_{ii}}{\sum_{j=1}^{q} n_{ij}}. \tag{4.7}$$

4.8.2.3 Justification de l'utilisation d'une mesure d'évaluation

Si nous avons le même nombre d'observations pour chaque classe, alors $JT(f) = C(f)$, où f est un classificateur. Cependant, si l'ensemble de données d'évaluation n'est pas équilibré[8], $JT(f)$ donne un poids (une considération) plus bas aux observations dont les classes sont minoritaires. C'est le cas pour les données de prédiction de la qualité du logiciel qui sont souvent non équilibrées. Les composants logiciels tendent à être classifiés d'une manière non équilibrée par rapport à un facteur de qualité. Par exemple, dans nos expériences (voir chapitre 7), nous avons eu beaucoup plus de classes stables que de classes instables. Par ailleurs, sur un ensemble de données non équilibrées, une faible erreur d'entraînement (une grande proportion correcte) peut être atteinte par le classificateur constant f_{const}, qui assigne la classe de la majorité à chaque observation d'entrée. Avec JT, le classificateur constant f_{const} et le classificateur aléatoire f_{alea} (qui assigne des classes d'une manière aléatoire et uniforme aux observations d'entrée), auraient une capacité prédictive proche de 0.5, tandis qu'un classificateur parfait devrait avoir un $JT = 1$.

De point de vue statistique, $JT(f)$ mesure l'exactitude moyenne par classe : c'est l'exactitude $C(f)$ en supposant que la distribution est *a priori* uniforme pour toutes les classes, c'est-à-dire lorsque le nombre d'observations par classes est presque constant.

En conclusion, la justification de l'utilisation de l'indice JT se résume dans son adéquation aux ensembles de données non équilibrés (voir aussi le chapitre 7).

4.8.3 Propriétés d'un modèle

Définition 4.8.1 *Complétude d'un modèle :*

Un modèle f est dit complet lorsque chaque observation représentant un composant logiciel $\mathbf{x}$ *a, au moins, une valeur de sortie y indiquant la valeur de son facteur de qualité. La complétude est vérifiée si, pour chaque $\mathbf{x} \in \mathcal{V}$, il existe au moins une expertise e_j qui est définie au point* $\mathbf{x}$ *tel que $e_j(\mathbf{x}) = y$.*

8. Le nombre d'observations par classe est très variable.

On déduit que l'*incomplétude* se manifeste lorsque aucune expertise du modèle f n'est applicable à au moins un composant $\mathbf{x} \in \mathcal{V}$.

Définition 4.8.2 *Cohérence d'un modèle :*

Un modèle f est dit cohérent lorsqu'à chaque observation $\mathbf{x}$*, il doit correspondre au plus, une seule valeur de sortie y. Ceci revient à dire que chaque composant logiciel doit avoir une seule décision indiquant la valeur de son facteur de qualité. La cohérence est vérifiée si, pour chaque* $\mathbf{x} \in \mathcal{V}$*, il existe au plus une seule expertise e_j qui est définie au point* $\mathbf{x}$ *tel que* $e_j(\mathbf{x}) = y$.

On déduit que l'*incohérence* se manifeste lorsque, plusieurs expertises du modèle f sont applicables au même composant logiciel $\mathbf{x} \in \mathcal{V}$, dans ce cas f n'est même plus une fonction.

4.8.4 Mécanisme de combinaison

C'est le mécanisme le plus sollicité de notre approche. Il s'appuie sur des opérations de combinaison. Chacune de ces opérations fait intervenir des modèles existants et est fondée sur la réutilisation des expertises. Il s'agit de regrouper plusieurs expertises provenant de modèles différents, pour former à chaque fois, un nouveau modèle.

L'objectif de la combinaison est de rechercher les bonnes expertises (« poches d'expertises »). Ce sont les expertises qui ont servi dans des environnements (où elles ont été construites) différents de D_t et elles ont montré leurs performances dans le contexte D_t. Nous qualifions ces expertises de *réutilisables*. Elles sont susceptibles de représenter les connaissances communes du domaine. Ce type de connaissances favorise une bonne capacité de généralisation des modèles, ce qui répond à l'une des nos préoccupations. Recombiner cette catégorie d'expertises est susceptible de produire un modèle plus généralisable. Afin de guider cette recherche, les combinaisons produites sont évaluées et celles qui contiennent des bonnes expertises sont sélectionnées. Concrètement, la combinaison consiste en un nombre d'opérations successives d'ajout ou de suppression d'une expertise dans un modèle. Nous appelons ces opérations *opérations de base* (voir section 4.8.6). Ces dernières ont comme mission de changer légèrement le comportement du modèle en y intégrant des expertises empruntées des autres modèles existants.

4.8.5 Mécanisme d'adaptation

C'est le mécanisme qui donne à un petit nombre d'expertises d'un modèle, la chance de changer. Cette opération est effectuée dans l'objectif de donner une flexibilité aux expertises pour mieux s'adapter au contexte particulier D_c. Les expertises qui sont mauvaises (ou faibles) auront moins de responsabilités et ceci est réalisé par un rétrécissement de leurs « territoires » (ou partitions v_j). Celles qui sont bonnes (ou fortes) auront plus de responsabilités, ce qui se réalise par un élargissement de leurs partitions v_j. Lorsqu'une partition v_j est rétrécie, plus de responsabilités sont données aux expertises des partitions voisines pour combler l'incomplétude du modèle (voir le paragraphe 4.8.6.2 sur la suppression).

Une adaptation d'une expertise peut aussi se faire par la modification de sa fonction d'expertise e_j. En effet, en la changeant, la fonction d'expertise e_j peut prendre la valeur d'une autre fonction f_i parmi les fonctions représentant les modèles originaux. Une opération d'adaptation est accomplie en effectuant plusieurs opérations de base d'ajout et de suppression.

Après une adaptation, un nouveau modèle contenant une proportion de nouvelles expertises est produit. Une fois validées sur D_c, les nouvelles expertises enrichissent le modèle par des connaissances spécifiques du contexte. Ainsi, l'adaptation supporte notre objectif qui concerne l'amélioration de la capacité prédictive du modèle.

4.8.6 Opérations de base

4.8.6.1 Ajouter une expertise à un modèle

Cette opération consiste en l'ajout d'une expertise à un modèle, tout en préservant sa propriété de cohérence (voir définition 4.8.2).

Soit e_a une expertise définie dans une partition v_a de l'espace $\mathcal{V}$ et soit f un modèle composé de R expertises $e_1,\ldots,e_R$, définies respectivement dans les partitions $v_1,\ldots,v_R$ de $\mathcal{V}$, l'ajout de e_a à f entraîne un nouveau partitionnement de l'espace $\mathcal{V}$. Ce partitionnement est différent du précèdent dans les régions qui se chevauchent avec la partition v_a et se manifeste par le rétrécissement des partitions intersectées par v_a. Formellement, soit e_i une expertise définie dans v_i, tel que $v_i \cap v_a$ est non vide, alors v_i^*, la nouvelle partition de e_i, serait égale à $v_i - (v_i \cap v_a)$ (voir la figure 4.7). Une opération d'élargissement est en fait une suppression suivie d'une opération d'ajout, par conséquent, la propriété de cohérence doit

être préservée de la même manière qu'après un ajout.

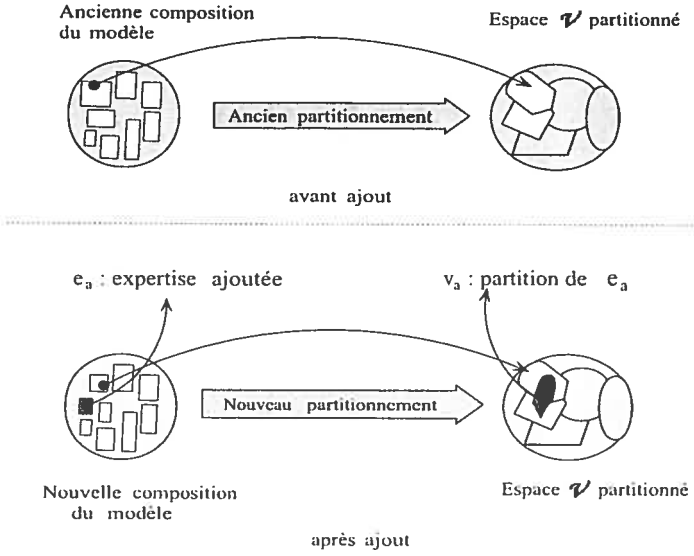


FIG. 4.7 – *Ajout d'une expertise à un modèle*

4.8.6.2 Supprimer une expertise d'un modèle

Cette opération consiste en la suppression d'une expertise d'un modèle tout en préservant la propriété de complétude du modèle (définition 4.8.1).

Soit e_s une expertise définie dans la partition v_s de l'espace $\mathcal{V}$ et soit f un modèle composé de R expertises $e_1, \ldots, e_R$, définies respectivement dans les partitions $v_1, \ldots, v_R$ de $\mathcal{V}$, la suppression de e_s de f entraîne la suppression de v_s, donc un nouveau partitionnement touchant les régions voisines de la partition v_s est nécessaire pour préserver la complétude du modèle. Il se traduit par l'élargissement de certaines partitions voisines de e_s. Cet élargissement peut être effectué selon différentes stratégies. Un choix simple de stratégie consiste à élargir une partition voisine choisie au hasard.

Formellement, soit e_i une expertise définie dans v_i voisine de v_s choisie au hasard, pour combler le vide crée par la suppression de v_s il suffit d'élargir v_i et d'étendre par conséquent le domaine de définition de e_i. La nouvelle partition v_i^* de e_i serait égale à $v_i + v_s$. La figure 4.8 montre une autre façon de combler ce vide qui consiste à élargir plusieurs partitions voisines de v_s.

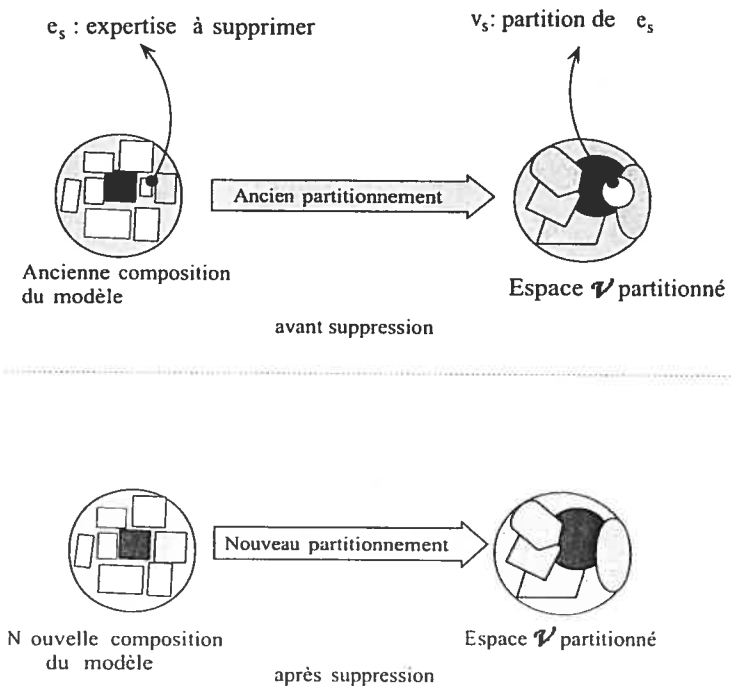


FIG. 4.8 – *suppression d'une expertise d'un modèle*

4.9 Techniques utilisées dans CAMP pour la combinaison et l'adaptation

L'un des objectifs principaux de l'approche CAMP est de trouver le modèle qui commettrait le minimum d'erreurs une fois utilisé dans un contexte particulier $D_{c'}$. La combinaison et l'adaptation produisent à chaque itération de nouveaux modèles qui forment un espace grandissant de solutions potentielles à notre problème de recherche du meilleur modèle. Trouver la meilleure solution revient à explorer cet espace de recherche afin de maximiser une fonction de mesure de performance du modèle dans le contexte $D_{c'}$. Ainsi, nous sommes en présence d'un problème d'optimisation combinatoire. Vu la taille de l'espace de recherche qui ne cesse de grandir à cause du nombre de combinaisons possibles d'expertises, les méthodes de recherche exactes ou déterministes nécessitent une puissance calculatoire qui croît exponentiellement avec la taille des instances du problème (nombre d'expertises dans notre cas).

Afin d'avoir un ordre de la complexité de notre problème, supposons que nous voulons combiner trois modèles qui utilisent deux métriques en entrée et que chaque modèle est composé en moyenne,

de 9 expertises (c'est l'équivalent de trois intervalles par attribut ou un espace d'entrée de 9 partitions). Si nous considérons seulement les modèles produits par les combinaisons possibles sans compter les adaptations, la taille de l'espace de recherche serait nettement supérieure à 18!.
C'est approximativement la somme des arrangements, respectivement à 1, à 2,..., et à N dans N qui est égale à $(N! + N! + N!/2! + \ldots + N!/N!)$. C'est un problème plus complexe que celui du voyageur de commerce $(N!/2N)$, où N est le nombre des villes, et qui correspond par analogie, au nombre total d'expertises dans les modèles existants. Il s'agit alors d'un problème NP-complet [Garey, 1979]. Par conséquent, un recours aux méthodes métaheuristiques ou approchées s'impose pour limiter l'explosion combinatoire [Talbi, 2001].

Ces méthodes sont prometteuses pour ce genre de problèmes d'optimisation. Elles conduisent à des solutions plus que satisfaisantes appelées aussi *solution approchées optimales* en un temps raisonnable. Parmi ces méthodes, nous mentionnons celles qui sont basées sur la population tels que les algorithmes génétiques (AGs), les colonies de fourmis, etc. et celles qui sont basées sur la recherche locale, ou le voisinage tels que le recuit simulé (RS), la recherche avec tabous (RT), etc. [Ferland et Costa, 2001] (voir chapitre 2). Les plus utilisées de ces méthodes sont les AG, le RS et la RT qui sont souvent employées ensemble afin de comparer leurs adéquations respectives à un même problème particulier. Ces trois méthodes font l'objet des techniques de combinaison et d'adaptation de notre approche CAMP.

4.9.1 Technique basée sur la population : algorithme génétique

Comme il est détaillé dans la section 2.3.2 du chapitre 2, l'AG est une métaheuristique basée sur la population. L'idée fondamentale de l'AG est de commencer à partir d'un ensemble de solutions initiales et d'utiliser des mécanismes inspirés de l'évolution biologique, pour dériver des nouvelles solutions probablement meilleures [Holland, 1975]. Cette dérivation débute dans notre problème par un ensemble initial de modèles existants G_0 (appelé la population initiale de modèles) qui produit progressivement une séquence de générations de modèles $G_1, \ldots, G_T$, chacune obtenue par transmutation de la précédente. Chaque modèle composé d'un ensemble d'expertises constitue un chromosome et chaque expertise correspond à un gène. La force de chaque chromosome (modèle) est mesurée par une fonction objectif appelée *fonction de fitness* qui correspond ici à la capacité prédictive du modèle (voir chapitre 3). À chaque génération, l'algorithme choisit un nombre de modèles en utilisant une méthode de sélection qui

accorde une priorité aux modèles les plus performants. Sur les modèles choisis, l'algorithme applique un des deux opérateurs : le croisement ou la mutation, avec des probabilités respectives p_c et p_m, où p_c et p_m sont des paramètres d'entrées de l'algorithme. L'opérateur de croisement mélange des expertises (gènes) provenant de deux modèles (chromosomes) différents, ce qui permet d'accomplir l'opération de combinaison visée par notre approche, alors que l'opérateur de mutation change aléatoirement certaines expertises dans un modèle, ce qui permet d'accomplir l'opération d'adaptation. Les nouveaux modèles produits par les croisements et les mutations constituent la prochaine génération. L'algorithme s'arrête lorsqu'un critère de convergence est satisfait ou si un nombre fixe de générations est atteint. Afin d'accélérer la convergence de l'algorithme, certaines techniques d'optimisation sont utilisées. La plus courante est l'élitisme qui consiste à garder un nombre de meilleurs modèles d'une génération à la génération suivante.

En plus des correspondances établies ci-après, entre les mécanismes de notre approche et ceux de l'AG (voir le tableau 4.2), le codage détaillé des chromosomes, le croisement, la mutation et la fonction de *fitness*, doivent être définis en détail. CAMP est une approche générale développée pour plusieurs types de modèles et son application en utilisant un AG dépend de la structure et du type de sortie des modèles. Par conséquent nous détaillons des adaptations de l'AG, aux aspects de certains types particuliers de modèles dans les chapitres 5 et 6, consacrés à l'application de l'approche CAMP.

Approche CAMP	**Algorithme génétique AG**
Modèles	Chromosomes
Modèles existants	Population initiale
Expertise	Gène
Combinaison	Croisement
Adaptation	Mutation
Mesure de performance du modèle	Fonction de *fitness*

TAB. 4.2 – *Tableau de Correspondance CAMP - Méthode basée sur la population (GA).*

4.9.2 Technique basée sur la recherche locale

4.9.2.1 Recherche avec tabous

La méthode de recherche avec tabous [Glover, 1986] est une métaheuristique fondée sur le principe de la recherche locale. Partant d'une solution initiale qui est l'un des modèles existants dans notre cas , le processus consiste à chaque itération, à choisir le meilleur modèle dans le voisinage du modèle courant, même si ce choix n'entraîne pas une amélioration. Ce voisinage est constitué de tous les modèles obtenus en effectuant des transitions ou mouvements, à partir du modèle courant. Une transition est une opération effectuée sur une expertise du modèle. Elle peut être un ajout, une suppression, un changement de la fonction de l'expertise, etc. Afin d'éviter le problème des optima locaux dans lequel ce processus peut facilement tomber, la recherche avec tabous utilise une structure de mémorisation temporaire dans laquelle elle conserve les caractéristiques des dernières transitions ou de leurs transitions inverses : cette structure est appelée *liste taboue*. Une transition reste interdite pendant un nombre d'itérations égal à la taille de cette liste. Le meilleur modèle du voisinage est choisi pour la prochaine itération. Ce choix doit améliorer la meilleure solution ou du moins, il ne doit pas être une transition taboue. C'est grâce aux transitions que nous pouvons accomplir les opérations de combinaison et d'adaptation de notre approche.

Tout comme l'AG, l'application de la recherche avec tabous dépend du type de modèles ainsi que de leur type sortie. En effet, l'adaptation de cette technique à notre problème nécessite de définir en détail, les éléments et les paramètres de la RT, comme la fonction de transition, la taille et la gestion de la liste de tabous. Une récapitulation de la correspondance entre les mécanismes de l'approche CAMP et les concepts de la recherche avec tabous est donnée dans le tableau 4.3.

4.9.2.2 Recuit simulé

La méthode du recuit simulé RS est une métaheuristique fondée aussi sur le principe de la recherche locale. Elle est inspirée de la méthode du recuit utilisée en métallurgie, qui consiste à simuler l'évolution d'un solide vers son équilibre thermique [Kirkpatrick et *al.*, 1983]. Le RS utilise une analogie entre la minimisation d'un critère en optimisation combinatoire et la recherche de l'équilibre thermique d'un solide. Cette analogie est expliquée en détail dans la section 2.3.3 du chapitre 2. En partant d'un modèle

Approche CAMP	**Recherche avec tabous ou recuit simulé**
Modèles	Solutions
Modèles existants	Différentes solutions initiales
Combinaison	Transitions
Adaptation	Une transition
Mesure de performance du modèle	Fonction objectif

TAB. 4.3 – *Tableau de Correspondance CAMP - Méthodes de recherche locale (RT et RS).*

existant (une solution initiale), le processus consiste à chaque itération, à choisir arbitrairement un modèle dans le voisinage du modèle courant. Le voisinage d'un modèle est constitué de la même manière que celui de la RT. Le nouveau modèle obtenu après transition du modèle courant est accepté ou refusé selon un critère d'acceptation probabiliste. Ce critère dépend de la variation de la performance du modèle causée par l'effet de transition et d'un autre paramètre de contrôle qu'on appelle *la température de recuit*. Ce critère basé sur la fonction de Boltzmann (voir section 2.3.3.6), permet d'accepter certaines transitions même si celles-ci dégradent la performance du modèle (« transitions dégradantes »). Grâce à ce phénomène d'acceptation probabiliste, l'algorithme de RS échappe au piège des optima locaux. Lorsqu'un nombre suffisant de transitions est effectué (déterminé en fonction de la taille de l'espace de recherche), on réduit légèrement la température de recuit selon une méthode appelée *schéma de recuit* (voir détails dans le chapitre 2). Le processus débute avec une température de recuit initiale relativement élevée permettant d'accepter un grand nombre de transitions dégradantes. Par la suite, la température est progressivement diminuée, ceci baisse par conséquent le nombre de transitions acceptées. Ainsi, en se dirigeant vers la fin du processus, les modèles obtenus auront des performances très proches les unes des autres et on s'approche fortement du meilleur modèle.

Une première adaptation de la méthode RS à notre approche CAMP est résumée dans le tableau 4.3. Un deuxième niveau d'adaptation va dépendre du type de modèles manipulés. Ce sujet est détaillé dans les chapitres 5 et 6. Pour son application à un type donné de modèles, le RS doit avoir comme la RT, une fonction de transition, une stratégie d'acceptation, un schéma de recuit et certaines valeurs de paramètres à spécifier comme la température initiale et le nombre de transitions avant de baisser la température de

recuit, appelé aussi *palier de température*.

4.10 Conclusion

Dans ce chapitre, nous avons fixé trois objectifs pour améliorer la prédiction de la qualité du logiciel. Nos objectifs proposent la considération de trois propriétés, ou *facteurs de qualité de modèle* qui sont : la capacité prédictive, la capacité de généralisation et la facilité d'interprétation d'un modèle.

À l'égard du sérieux problème de pénurie de données, nous avons opté pour une amélioration de la prédiction basée sur la combinaison de modèles. Habituellement, cette solution est utilisée en apprentissage pour combiner, d'une manière ou une autre, les sorties des modèles. Une telle solution permet d'augmenter la capacité prédictive et d'améliorer la capacité de généralisation mais néglige et souvent détériore « l'interprétabilité » ou la transparence des modèles.

Notre approche de combinaison et d'adaptation de modèles CAMP est une approche basée sur la combinaison inspirée du mélange d'experts, avec l'avantage de garantir un niveau « d'interprétabilité » de la prédiction, supérieur ou égal à celui des modèles originaux.

L'approche CAMP est fondée sur une vision microscopique (fine) qui considère le modèle ou les modèles comme un ensemble d'expertises qui peuvent être selon des circonstances, faibles, fortes, moyennes, etc. Ces expertises peuvent collaborer par combinaison et évoluer par adaptation, pour former les meilleures prédictions dans des circonstances particulières. Cette manière de réutiliser les modèles existants repose sur les principes suivants :

1. Décomposer pour mieux interpréter.
2. Combiner pour augmenter le potentiel de généralisation.
3. Adapter pour améliorer la capacité prédictive et mieux convenir.

Nous avons eu recours aux métaheuristiques pour mettre en oeuvre les mécanismes de l'approche CAMP en respectant les trois principes précédents. Nous proposons, l'utilisation de deux familles de métaheuristiques : celle basée sur la population et représentée par les algorithmes génétiques et celle basée sur la recherche locale et représentée par le recuit simulé et par la recherche avec tabous. Ces trois techniques sont souvent utilisées ensemble dans un but comparatif. Afin de les appliquer à notre approche CAMP, ces techniques nécessitent une adaptation qui dépend étroitement du type des modèles

et du type de la tâche de prédiction (classification ou régression).

Dans ce chapitre, nous avons commencé par décrire un premier niveau d'adaptation de ces méthodes (AG, RT et RS) en identifiant la correspondance entre les principaux mécanismes de l'approche CAMP et ceux des méthodes. Une adaptation complète sera détaillée dans les chapitres 5 et 6, relativement aux applications de CAMP, respectivement, aux arbres de décision et aux classificateurs probabilistes (classificateurs bayésiens).

Chapitre 5

Application de l'approche CAMP aux modèles arbres de décision

5.1 Introduction

Ce chapitre présente une application de notre approche de combinaison et d'adaptation de modèles prédictifs aux modèles de qualité du logiciel de type arbres de décision. Nous appelons cette application *spécialisation de l'approche CAMP*. Pour ce faire, nous allons utiliser les trois techniques métaheuristiques jugées adéquates pour la mise en oeuvre des mécanismes de CAMP, à savoir, les algorithmes génétiques (AG), le recuit simulé (RS) et la recherche avec tabous (RT). Ces techniques sont présentées dans le chapitre 2, sans aucune adaptation à un problème quelconque. Dans le chapitre 4, nous avons établi une correspondance entre les mécanismes de CAMP d'une part et les opérations offertes par chacune des trois techniques d'une autre part. Dans le présent chapitre, nous détaillons l'adaptation des trois algorithmes pour la spécialisation de CAMP en vue de son application aux modèles de type arbres de décision.

Après avoir présenté une description des arbres de décision (section 5.2), nous commençons la mise en oeuvre de l'approche CAMP par la définition d'un concept commun aux trois techniques qui est une représentation de modèle de type arbre de décision (section 5.3). Dans la section 5.4, nous décrivons l'opération de prétraitement des arbres de décision candidats à l'application de l'approche CAMP. Par la suite, nous décrivons successivement les adaptations à notre problème des concepts spécifiques de l'AG

(section 5.5), puis de la RT et du RS (section 5.6). Dans la section 5.8, nous présentons une solution particulière aux limitations (voir section 5.7) des modèles résultant de l'application de CAMP aux arbres de décision.

5.2 Description d'un modèle de type arbre de décision

Un arbre de décision (ou de classification) est composé d'un ensemble de noeuds et d'arcs étiquetés. Les noeuds incluent les noeuds internes, les feuilles et la racine. Les noeuds internes et la racine de l'arbre sont appelés noeuds de décision, chacun est étiqueté par un test. Dans un modèle de prédiction de la qualité du logiciel de type arbre de décision, chaque noeud interne est un test qui compare la valeur d'un attribut (une métrique) à un seuil. Ce test peut être appliqué à toute observation d'entrée. Les réponses possibles correspondent aux étiquettes des arcs. Dans le cas de noeuds de décision binaires, l'arc gauche correspond par convention, à une réponse positive au test et l'arc droit correspond à une réponse négative au test. Les feuilles sont étiquetées par des labels appelés *classes*. À toute observation est associée une seule feuille de l'arbre de décision. On définit cette association en commençant à la racine et en descendant dans l'arbre selon les réponses aux tests qui étiquettent les noeuds internes. Le label de la feuille atteinte présente la valeur à prédire (facteur de qualité) associée à une observation.

Cette structure de classification a une traduction immédiate en terme de règles de décision mutuellement exclusives. Une règle représente une branche formée d'une séquence de noeuds et d'arcs, commençant de la racine jusqu'à une feuille. Elle représente géométriquement une région de l'espace dont les coordonnées des points vérifient les inéquations exprimées dans les conditions de la règle.

La figure 5.1 montre un exemple d'arbre de décision, qui classifie les composants logiciels en deux classes indiquant leurs niveaux de stabilité (*stable* ou *instable*), en utilisant deux attributs, ou métriques *LCOMB* et *NPPM* [1].

1. Ces deux métriques sont décrites dans le chapitre 7.

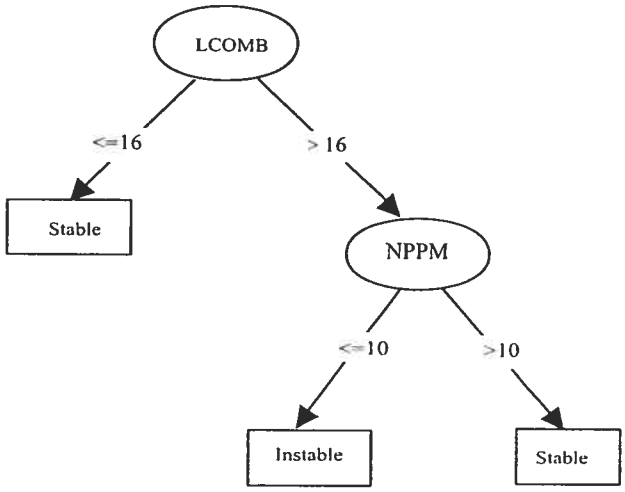


FIG. 5.1 – *Arbre de décision qui classifie les composants selon leurs niveaux de stabilité.*

5.3 Représentation d'un modèle de type arbre de décision sous forme d'expertises

5.3.1 Notations et formalismes

Nous utilisons les notations de la section 4.5, mais afin d'adapter l'approche CAMP au cas des arbres de classification, nous introduirons certaines notations empruntées de la géométrie dans un espace à plusieurs dimensions.

Soit T un arbre de classification utilisant d attributs pour classifier des observations d'entrée $\mathbf{x} \in \mathbb{R}^d$ en un nombre de classes $c_1,\ldots,c_q$. Chaque noeud interne de T est un test de type $x^{(j)} < \alpha_j$, avec $j = 1,\ldots,d$, $x^{(j)}$ un attribut de l'observation $\mathbf{x}$ et α_j une constante appartenant au domaine de définition de l'attribut $x^{(j)}$. Chaque attribut $x^{(j)}$ prend ses valeurs dans un domaine de définition limité par deux bornes L_j et U_j, respectivement, inférieure et supérieure. Cet arbre T peut être représenté par un partitionnement binaire S d'un espace $\mathcal{V} \subset \mathbb{R}^d$ défini par le produit cartésien des intervalles $[L_j,U_j]$, $j = 1,\ldots,d$:

$$\mathcal{V} = [L_1,U_1] \times ,\ldots, \times [L_d,U_d].$$

Le partitionnement S revient à subdiviser récursivement l'espace $\mathcal{V}$ au moyen d'un hyperplan h_j d'équation $x^{(j)} = \alpha_j$ en deux sous-espaces. Un hyperplan h_j est associé à chaque noeud interne de test $(x^{(j)} < \alpha_j)$. La subdivision se poursuit jusqu'à ce que chaque sous-espace ne contienne que les points

ayant la même classe.

Le partitionnement S nous permet de voir un arbre de décision comme un ensemble *d'hyperrectangles isothétiques* (c'est-à-dire, ayant des faces parallèles aux axes des attributs). Ainsi, un modèle de type arbre de décision peut être codé par un vecteur de paires (*hyperrectangle*,*label*). Un hyperrectangle (aussi appelé *d-rectangle*) R_k est défini par un produit cartésien d'intervalles $[\ell_j,u_j]$,

$$R_k = [\ell_1,u_1] \times ,\ldots, \times [\ell_d,u_d],$$

avec ℓ_j et u_j sont les coordonnées des arrêts du d-rectangle, sur l'axe de l'attribut $x^{(j)}$. De plus amples détails sur la manipulation et l'utilisation des hyperrectangles peuvent être trouvés dans [Petty et Mukherjee, 1998; Berchtold et *al.*, 1996; D´ Amore et *al.*, 1995].

La figure 5.2 illustre le partitionnement binaire de l'espace correspondant à l'arbre de la figure 5.1.

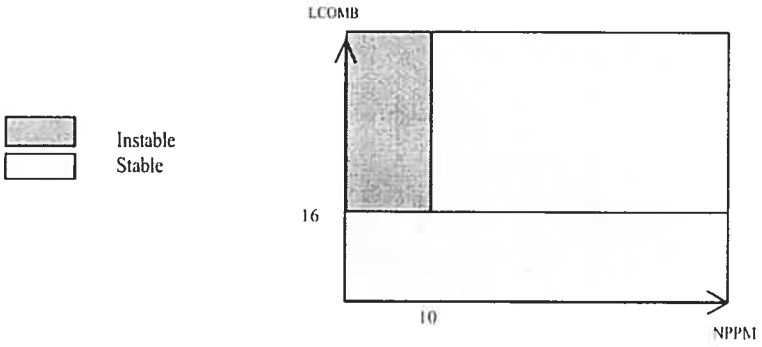


FIG. 5.2 – *Un arbre de décision représenté par un partitionnement binaire d'un espace bidimensionnel.*

5.3.2 Identification des expertises dans un arbre de décision

Les expertises qui composent un modèle de type arbre de décision sont les paires $(R_k,Label_k)$. Une fonction d'expertise est définie par :

$$e_k : R_k \mapsto C = \{c_1,\ldots,c_q\},$$

avec $e_k(\mathbf{x}) = Label_k$ et $Label_k$ prenant ses valeurs dans C. Nous remarquons que la fonction d'expertise est constante sur toute la partition R_k et, par conséquent, un modèle de type arbre de décision est donc redéfini conformément à la forme générique exprimée dans l'équation 4.2, par la fonction de décision

suivante :

$$f : \mathcal{V} \mapsto C = \{c_1, \ldots, c_q\} \text{ et } f(\mathbf{x}) = \begin{cases} Label_1 & \text{si } \mathbf{x} \in R_1, \\ Label_2 & \text{si } \mathbf{x} \in R_2, \\ \vdots & \\ Label_R & \text{si } \mathbf{x} \in R_R, \end{cases} \tag{5.1}$$

Pour simplifier la notation, nous représentons un d-rectangle par le vecteur $\big((\ell_1,u_1),\ldots,(\ell_d,u_d)\big)$.

Supposons, pour l'arbre de la figure 5.1, que $0 < \text{NPPM} \leq 100$ et $0 < \text{LCOMB} \leq 50$, cet arbre est donc représenté par:

$$\begin{pmatrix} \big((0,10),(16,50)\big) : Instable), \\ \big((10,100),(16,50)\big) : Stable), \\ \big((0,100),(0,16)\big) : Stable) \end{pmatrix}.$$

5.4 Prétraitement des modèles de type arbre de décision

Comme il est indiqué dans le chapitre 4, le prétraitement des modèles (arbres de décision dans notre cas) consiste à « unifier » l'espace d'entrée $\mathcal{V}$ pour tous les arbres candidats à la combinaison. Chaque arbre aura après prétraitement comme attributs d'entrée toutes les métriques de tous les arbres. Chaque métrique aura les mêmes bornes, respectivement, supérieure et inférieure dans tous les arbres. Ainsi, seul le partitionnement de l'espace $\mathcal{V}$ va distinguer un arbre d'un autre.

Pour illustrer cette opération, nous considérons deux arbres de décision qui prédisent le même facteur de qualité. Le premier utilise les deux métriques *NPPM* et *LCOMB* et le deuxième utilise les deux métriques *DIT* et *LCOMB*. Après le prétraitement désiré, les deux arbres auront les trois métriques *NPPM*, *LCOMB* et *DIT* comme attributs d'entrée et un espace d'entrée $\mathcal{V}$ défini par le produit cartésien :

$$\prod_{NPPM.LCOMB.DIT} [L_i,U_i].$$

Cette illustration est représentée dans la figure 5.3. L'opération de prétraitement des modèles de type arbres de décision est indépendante de la technique de mise en oeuvre de l'approche CAMP. Par conséquent, les arbres de décision sont prétraités de la même manière préalablement à l'application de cha-

cune des trois techniques GA, RS et RT.

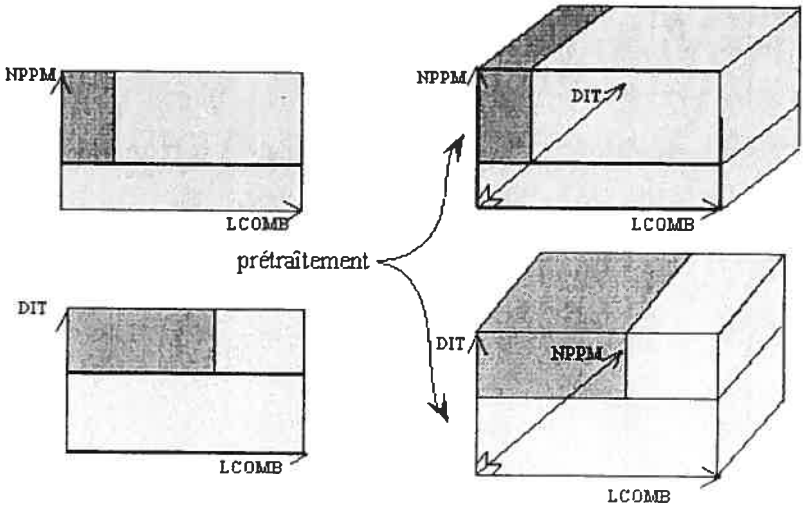


FIG. 5.3 – *Prétraitement des modèles de type arbre de décision.*

5.5 Algorithme génétique

Dans cette section, nous allons utiliser plutôt les concepts empruntés des algorithmes génétiques. En effet, nous déduirons le codage des chromosomes (section 5.5.1) à partir de la représentation d'un arbre de décision (section 5.2) et la fonction de *fitness* à partir de la section 4.8.2 sur l'évaluation des modèles. Par la suite, nous proposons dans les sections 5.5.3 et 5.5.4 une adaptation des opérateurs de l'AG à l'approche CAMP afin de l'appliquer aux arbres de décision.

5.5.1 Codage d'un arbre de décision en chromosome

Étant donnée la représentation d'un arbre de décision fournie dans la section 5.3 et en tenant compte de la correspondance établie dans la section 4.9.1, chaque modèle, après avoir subit l'opération de prétraitement, doit être représenté par un chromosome. Ce dernier est codé sous forme d'un ensemble de couples $(d\_rectangle.Label)$. Chacun de ces couples (une expertise) constitue alors un gène[2] du chromosome, avec $d\_rectangle = ((\ell_1,u_1),\ldots,(\ell_d,u_d))$.

2. Pour des raisons de simplification, nous allons confondre gène et d-rectangle.

Par exemple, dans la représentation de l'arbre de la figure 5.2, les gènes sont de simples rectangles colorés où la couleur représente la valeur du label (classe), ils sont au nombre de trois. Donc, le chromosome correspondant à cet arbre est un partitionnement de l'espace
$\mathcal{V} = [L_{LCOMB}, U_{LCOMB}] \times [L_{NPPM}, U_{NPPM}]$.

5.5.2 Fonction de *fitness*

Une façon de mesurer la fonction de *fitness* d'un chromosome est d'utiliser, selon la nature de la distribution des labels des observations, la mesure de la proportion correcte, ou l'exactitude (section 4.8.2.1) ou l'indice J transformé (JT) (section 4.8.2.2). Le choix retenu de la fonction de *fitness* est abordé et justifié empiriquement dans le chapitre 7.

5.5.3 Opérateur de croisement

L'opérateur du croisement est lié étroitement au codage des chromosomes. Dans le cas de structures simples comme les chaînes de bits, un croisement par découpage est convenable. Cependant, lorsque la structure d'un chromosome présente une certaine complexité comme dans le cas des arbres de décision, d'autres formes de croisement sont nécessaires [Falkenauer, 1998]. Le recours à ces formes vise à réduire la complexité du calcul des chromosomes enfants et à respecter les contraintes imposées par certaines propriétés de la solution (propriétés d'un modèle). Nous discutons dans la suite de cette section, trois tentatives de croisement.

5.5.3.1 Croisement intuitif

Une manière intuitive d'effectuer l'opération de croisement entre les chromosomes est celle du croisement par découpage ou *slicing crossover* (voir section 2.3.2.5). Elle consiste à découper chacun des deux chromosomes parents en deux sous-ensembles de gènes (d-rectangles dans notre cas). Ce découpage est aléatoire et n'est pas unique. Deux nouveaux chromosomes enfants sont créés en intercalant les sous-ensembles de d-rectangles. Si nous appliquons une telle opération à notre problème, il est possible que les chromosomes résultants ne puissent plus vérifier les propriétés d'un modèle, en l'occurrence, la complétude et la cohérence, définies dans la section 4.8.3. Cette première tentative de croisement, ainsi

que les deux problèmes (d'incomplétude et d'incohérence) qui peuvent se produire dans le cas d'arbres de décision sont décrits dans la figure 5.4.

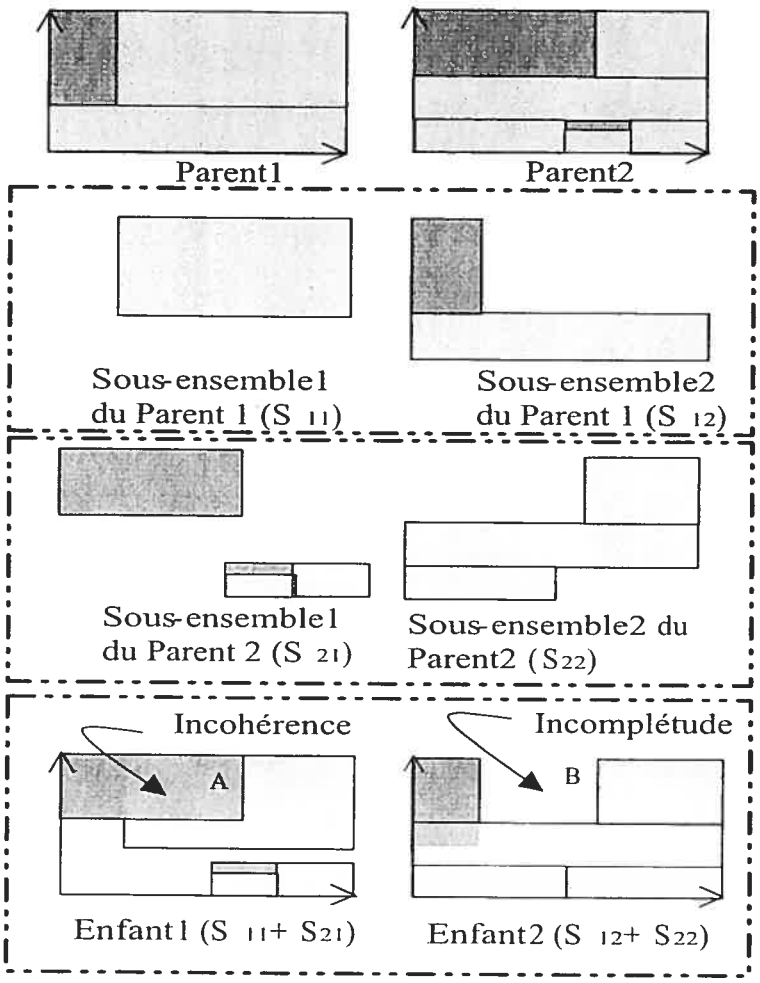


FIG. 5.4 – *Problèmes de l'utilisation du croisement standard.*

Chacun des deux chromosomes parents (*Parent*1 et *Parent*2) est divisé en deux sous ensembles de d-rectangles (S_{11} et S_{12}, issus du *Parent*1 et S_{21} et S_{22}, issus du *Parent*2). Les chromosomes enfants (*Enfant*1 et *Enfant*2) sont formés respectivement par les combinaisons de S_{11} avec S_{21} et de S_{22} avec S_{12}. Chaque d-rectangle dans les chromosomes enfants garde ses mêmes coordonnées dans l'espace $\mathcal{V}$. Inévitablement, les deux situations d'incomplétude et de l'incohérence, sont rencontrées. En effet, la région B de l'espace partitionné par le chromosome *Enfant*2 n'est pas couverte par un d-rectangle et dans ce cas, le modèle représenté par *Enfant*2 peut être une fonction mais non définie dans la région B :

c'est le problème d'incomplétude. La région A de l'espace partitionné par le chromosome $Enfant1$ est couverte en même temps par deux d-rectangles ayant deux labels (décisions) différents et dans ce cas-ci, le modèle représenté par le chromosome $Enfant1$ n'est même pas une fonction : c'est le problème d'incohérence.

5.5.3.2 Croisement par greffe de d-rectangles

Dans le but de préserver la cohérence et la complétude des modèles enfants, nous proposons un nouvel opérateur de croisement inspiré des opérateurs définis pour les problèmes de groupement [Falkenauer, 1998]. Afin d'obtenir un enfant avec le nouvel opérateur, nous choisissons un sous-ensemble aléatoire de d-rectangles à partir d'un parent et nous l'ajoutons à l'ensemble entier de d-rectangles du deuxième parent. La taille du sous-ensemble aléatoire des d-rectangles ajoutés vaut a fois le nombre de d-rectangles du parent, où $a > 0$ est un paramètre de l'opérateur du croisement.

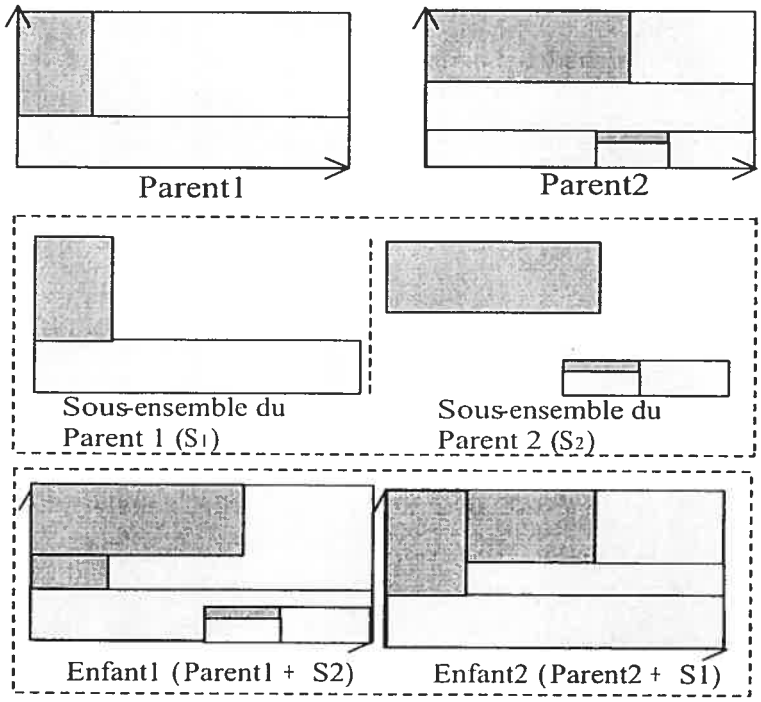


FIG. 5.5 – *Croisement qui préserve la cohérence et la complétude.*

En gardant tous les d-rectangles de l'un des parents, nous préservons la complétude du chromosome enfant qui n'était pas automatiquement assurée. Et pour garantir la cohérence, nous décidons que

les d-rectangles ajoutés seront prédominants (les d-rectangles ajoutés l'emportent sur les d-rectangles originels dans le processus de décision). La figure 5.5 illustre le nouvel opérateur de croisement.

Après croisement, il est possible que parmi les régions de décision de l'espace $\mathcal{V}$ nouvellement partitionné, il y ait certaines régions qui ne soient pas des d-rectangles. Cette situation résulte du fait que certains d-rectangles sont partiellement « couverts » par d'autres (voir figure 5.6). Pour maintenir la conformité des nouveaux chromosomes enfants au codage de modèle (décrit dans la section 5.5.1), nous transformons toutes les régions résiduelles en d-rectangles, comme indiqué par la figure 5.6.

La transformation des résidus en d-rectangles crée un nouveau partitionnement de l'espace $\mathcal{V}$. Cette opération est extrêmement consommatrice du temps et de l'espace. Elle constitue un problème de recherche à part entière en géométrie computationnelle [D´ Amore et *al.*, 1995; Berchtold et *al.*, 1996; Petty et Mukherjee, 1998]. D'une part, des résidus doivent être calculés pour chaque paire possible des d-rectangle après chaque croisement : ceci est fait pour plusieurs croisements et est répété pour toutes les générations dans le processus de l'AG. D'autre part, dans le pire des cas, un résidu doit être décomposé en $2d$ d-rectangles (où d est la dimension de l'espace d'entrée $\mathcal{V}$). Par conséquent, après plusieurs générations, l'espace peut devenir très fragmenté, la complexité de l'opération de transformation des résidus en d-rectangles croît, ce qui la rend irréaliste dans les conditions de notre problème.

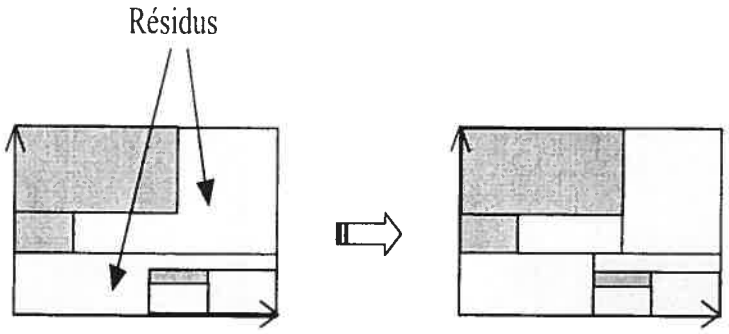


FIG. 5.6 – *La transformation des résidus.*

5.5.3.3 Croisement par superposition des couches de d-rectangles

Pour éviter les problèmes de complexité de la fragmentation de l'espace, nous modifions notre codage du modèle ainsi que l'opérateur de croisement. Nous gardons intact le sous-ensemble de d-

rectangles ajoutés (qui cachent des parties de l'espace) et attribuons un niveau de prédominance comme élément supplémentaire aux d-rectangles.

Chaque gène devient ainsi un triplet (*d_rectangle,label,niveau de prédominance*). Les d-rectangles de la population initiale ont le niveau 1. Chaque fois qu'un d-rectangle est ajouté à un chromosome, son niveau de prédominance sera égal au niveau maximal des d-rectangles du chromosome « d'accueil » incrémenté de 1 (voir la figure 5.7). Pour trouver le label d'une observation d'entrée **x** (un composant logiciel, par exemple), nous devons d'abord trouver tous les d-rectangles qui contiennent le point de coordonnées **x**, puis nous assignons à **x** le label du d-rectangle qui a le niveau de prédominance le plus élevé. Cette stratégie de prédominance est semblable à ce qui est utilisé dans les systèmes de règles où les règles ont des priorités qui doivent être considérer pour résoudre les conflits et les contradictions entre elles. Le modèle enfant dérivé par ce croisement conserve totalement la propriété de transparence des experts originaux (arbres de décision) puisque chaque décision peut être associée à un seul d-rectangle et chacun des d-rectangles provient d'un seul expert qui est identifié.

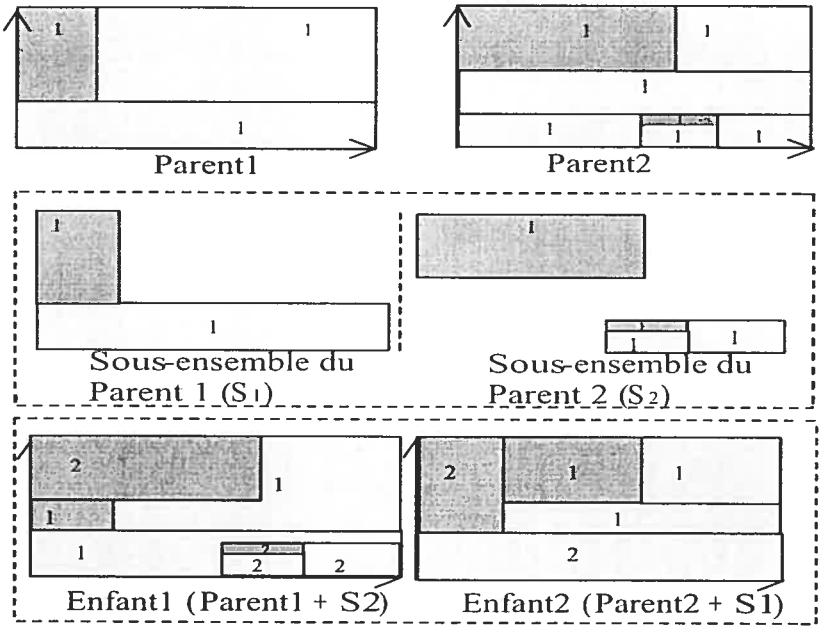


FIG. 5.7 – *Croisement par superposition de couches de d-rectangles.*

5.5.4 Opérateur de mutation

L'opérateur de mutation, comme il est décrit dans le chapitre 2, est un changement aléatoire que subissent les gènes et qui se produit avec une faible probabilité p_m. Dans notre problème, cet opérateur consiste à changer, aléatoirement, le label d'un d-rectangle. Dans la prédiction de qualité du logiciel, particulièrement pour un arbre de décision, l'espace de sortie C est habituellement un ensemble ordonné $c_1, \ldots, c_q$ de labels. La mutation que nous proposons consiste à permettre le label d'un d-rectangle (gène) c_k de changer avec la probabilité p_m en c_{k+1} ou en c_{k-1} si $1 < k < q$, en c_2 si $k = 1$ ou en c_{q-1} si $k = q$. La figure 5.8 illustre l'opérateur de mutation. La raison principale de notre choix de cet opérateur de

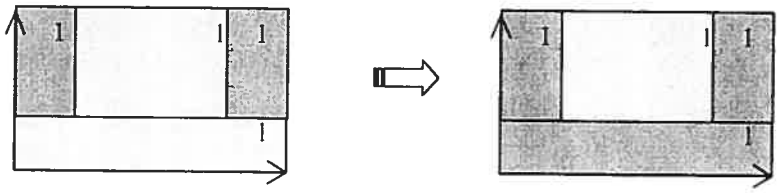


FIG. 5.8 – *Mutation par changement du label d'un d-rectangle.*

mutation est sa simplicité. Toutefois, en tenant compte des objectifs du mécanisme d'adaptation de notre approche CAMP (décrit dans la section 4.8.5), nous pouvons envisager d'autres types de mutations. Par exemple, nous pourrions changer la taille (l'encombrement) d'un d-rectangle ou affecter le label d'un d-rectangle à un d-rectangle adjacent.

5.6 Recuit simulé et recherche avec tabous

L'algorithme du recuit simulé (RS) ainsi que celui de la recherche avec tabous (RT) sont tous deux des métaheuristiques basées sur la recherche locale pour optimiser une fonction appelée *fonction de coût*, ou encore fonction objectif. Dans notre cas, cette fonction n'est autre que la capacité prédictive d'un arbre de décision. Elle est mesurée de la même façon que la fonction de *fitness* dans les AG. Comme il a été mentionné dans le chapitre 4, les algorithmes RS et RT utilisent deux concepts communs qui sont la représentation de solutions et la fonction de transition, appelée aussi *fonction de voisinage*. Concernant la représentation des solutions, nous allons utiliser le même codage que celui retenu pour l'application

de l'AG, à savoir le triplet (*d_rectangle*,*label*,*niveau de prédominance*), tandis que nous proposons une fonction de transition qui permet de parcourir le voisinage d'une solution courante (arbre de décision dans notre cas). Nous rappelons que cette fonction réalisera les tâches des deux mécanismes principaux de l'approche CAMP, à savoir, la combinaison et l'adaptation des solutions. Quant aux autres concepts spécifiques à chacun des algorithmes (RS et RT) et nécessaires pour son application, nous procédons de la manière suivante :

- Pour le RS, nous utilisons les mêmes paramètres et stratégies décrits dans la section 2.3.3, à savoir, la longueur de température, le schéma de décroissance de la température, le critère d'acceptation des transitions et le critère d'arrêt.
- Pour la RT, nous définissons une stratégie de gestion de la liste taboue et nous employons les critères d'arrêt usuels et le paramètre de longueur de la liste taboue.

Nous présentons dans la section 5.6.1 une description de la fonction de transition et dans la section 5.6.2, nous proposons une manière de gérer la liste taboue. Concernant les choix des autres stratégies et les valeurs des paramètres des deux algorithmes, ils sont abordés dans le chapitre 7.

5.6.1 Fonction de transition

Une transition intuitive serait réalisée en effectuant un changement dans la structure de l'arbre de décision courant, afin d'obtenir un arbre voisin. En d'autres termes, étant donné la représentation d'un arbre de décision sous forme d'un ensemble de d-rectangles (expertises), une transition consiste à apporter une modification raisonnable sur sa composition en d-rectangles. Nous entendons par modification raisonnable un petit changement qui touche un nombre réduit de d-rectangles situés dans une même région de l'espace $\mathcal{V}$. Un choix simple est de considérer une transition qui ne modifie qu'un seul d-rectangle. Ainsi, une transition peut être l'une des actions suivantes :

1. Un changement du label (fonction d'expertise) d'un d-rectangle choisi aléatoirement dans le vecteur de d-rectangles qui composent le modèle. Ce type de transition est comparable à l'opérateur de mutation décrit dans la section 5.5.4.
2. Un ajout, au modèle courant, d'un nouveau d-rectangle (nouvelle expertise) choisi aléatoirement dans un dépôt de d-rectangles. Cette action suppose l'existence d'un dépôt de d-rectangles (ou d'expertises) obtenu par la dissociation de tous les d-rectangles composant les modèles existants.

Ce dépôt exclut les d-rectangles composants le modèle initial choisi pour débuter l'algorithme de recherche (RS ou RT). Cette exclusion est une façon simple de préserver la propriété de la complétude du modèle.

La figure 5.9 montre l'opération de formation d'un dépôt de d-rectangles. Une fois ajouté, le

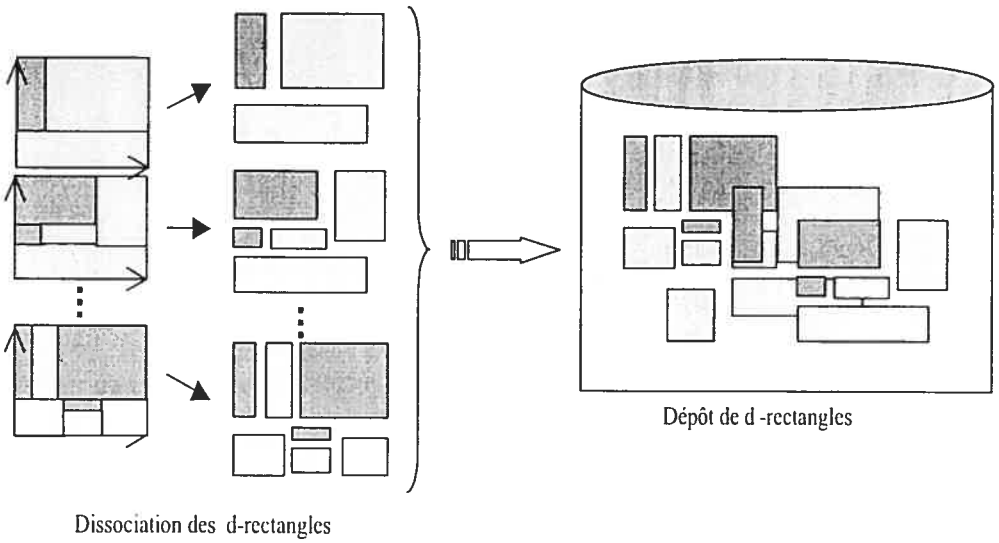


FIG. 5.9 – *Formation d'un dépôt de d-rectangles.*

nouveau d-rectangle domine les d-rectangles originels dans la région qu'il occupe dans l'espace $\mathcal{V}$. Une fragmentation de l'espace $\mathcal{V}$ aurait pû être préférée pour obtenir, après chaque transition, la même structure du modèle, mais cette dernière est problématique à cause de sa complexité (voir section 5.5.3.2). Afin d'éviter les inconvénients de la fragmentation de l'espace, nous adoptons la même méthode utilisée dans le croisement par superposition de d-rectangles. Ainsi, chaque d-rectangle ajouté aura un niveau de prédominance égal au plus haut niveau des d-rectangles originels incrémenté de 1. Le niveau de prédominance des d-rectangles du modèle initial est égal à 1. La figure 5.10 illustre l'opération d'ajout d'un d-rectangle à un modèle de type arbre de décision.

3. Une suppression, à partir du modèle courant, d'un d-rectangle choisi aléatoirement parmi ceux qui ont un niveau de prédominance supérieur à 1. Cette dernière condition est également un moyen simple pour préserver la complétude du modèle.

4. Un élargissement d'un d-rectangle du modèle courant choisi aléatoirement : ceci revient à rajouter ce d-rectangle, au modèle courant, après avoir augmenter son « volume » (en augmentant certaines de ses dimensions).
5. Un rétrécissement (diminution du « volume ») d'un d-rectangle de niveau de prédominance supérieur à 1.
6. Un choix aléatoire d'une action parmi les précédentes.

   C'est cette dernière stratégie que nous utilisons dans nos implémentations.

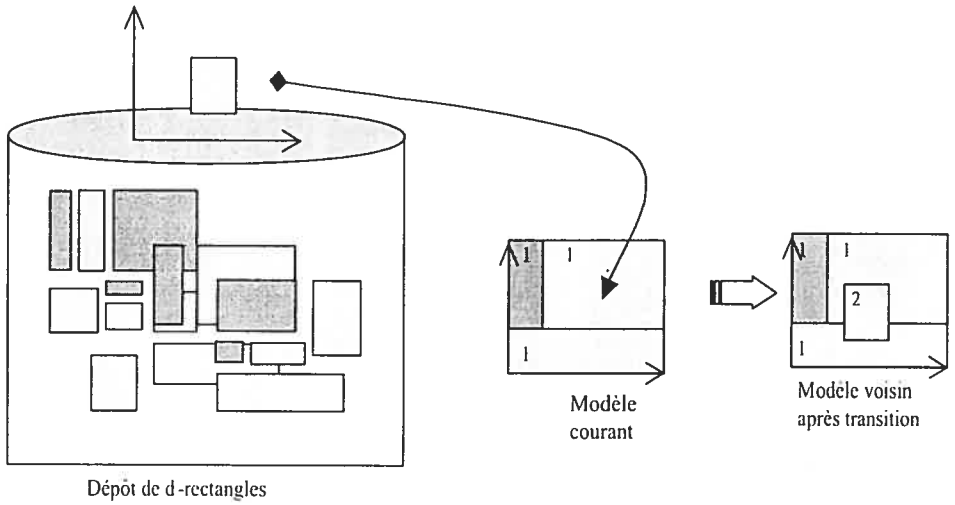


FIG. 5.10 – *Transition par ajout d'un d-rectangle.*

5.6.2 Gestion de la liste taboue dans la RT

La gestion de la liste taboue consiste, après chaque itération de l'algorithme RT, à mettre à jour une liste de transitions qualifiées de taboues. Chacune de ces transitions est interdite pendant un nombre ℓ (longueur de la liste) d'itérations. Dans le cas des modèles de type arbres de décision, nous proposons une stratégie basée sur l'élimination du mouvement inverse [Hansen, 1986]. Elle vise à interdire les transitions inverses des transitions récemment effectuées. Pour ce faire, nous représentons une transition par le couple (*identificateur_d_rectangle.type_transition*), où *identificateur_d_rectangle* est un attribut qui désigne d'une manière unique chaque d-rectangle appartenant au modèle ou au dépôt de d-rectangles (voir section 5.6.1) et *type_transition* est un attribut qui distingue les cinq types de

transition (*changement_label*, *ajout*, *suppression*, *élargissement* ou *rétrécissement*). Une transition est identifiée d'une manière unique par le couple (*identificateur_d_rectangle*,*type_transition*). Notre liste taboue est une file (FIFO) de longueur ℓ de couples représentant les transitions taboues. Après chaque transition, un couple représentant la transition inverse est ajouté en queue de la liste et un autre couple, représentant une transition dont la transition inverse était taboue durant les ℓ dernières itérations, est supprimé de la tête de la liste. Le couple représentant une transition inverse est déduit à partir de celui de la transition effectuée en gardant le même identificateur du d-rectangle (*identificateur_d_rectangle*) et en déduisant le type de transition inverse, à partir de la correspondance entre type de transition et type de transition inverse, établie dans le tableau 5.1.

Type de transition	**Type de transition inverse**
ajout	suppression
suppression	ajout
élargissement	rétrécissement
rétrécissement	élargissement
changement de label (de c_i à c_j)	changement de label (de c_j à c_i)

TAB. 5.1 – *Correspondance entre type de transition et type de transition inverse*

5.7 Limitation des modèles résultant de CAMP

Dans cette section, nous discutons deux propriétés des modèles résultant de l'application des trois techniques d'optimisation AG, RS et RT lors la mise en oeuvre de notre approche CAMP au cas d'arbres de décision. La première propriété est relative à la structure en couche du modèle obtenu et la deuxième concerne sa taille en nombre de d-rectangles.

5.7.1 Structure en couches et aplatissement du modèle

L'introduction du niveau de prédominance des d-rectangles, pendant les opérations de croisement, de mutation et de transition (voir sections 5.5.3, 5.5.3 et 5.6.1), est justifiée par la complexité du calcul des intersections entre d-rectangles. Le modèle résultant de CAMP peut être vu comme un ensemble de

« couches », regroupant chacune les d-rectangles ayant un même niveau de prédominance. Un exemple de modèles en couches composées de simples rectangles est illustré dans la figure 5.11.

Cette structure du modèle est d'une part plus complexe que celle des modèles initiaux et d'autre part difficilement convertible en un simple arbre de décision. Cependant, un modèle sans niveaux de prédominance peut être rétabli en aval du processus de combinaison et d'adaptation. Nous appelons cette opération *aplatissement du modèle* car nous obtenons une seule couche de d-rectangles.

Après l'opération d'aplatissement, le modèle a l'avantage de conserver le même codage que les modèles initiaux et d'être facilement traduit en un arbre de décision.

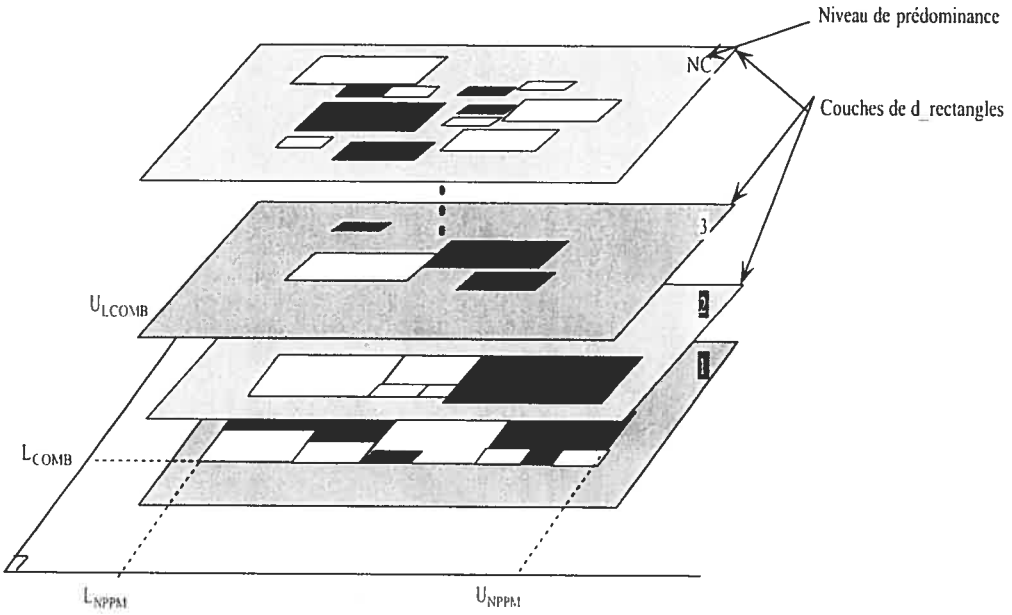


FIG. 5.11 – *Structure en couche des modèles résultants : exemple bidimensionnel (deux métriques).*

Concrètement, l'aplatissement est une série d'opérations de greffe de d-rectangles appartenant aux couches supérieures (dont le niveaux de prédominance supérieur à 1), dans la couche du plus bas niveau de prédominance (couche de niveau 1). Une opération de greffe d'un d-rectangle R_i, dans la couche de niveau 1, comporte les étapes suivantes :

- Étape 1 : identifier un ensemble F des d-rectangles R_j appartenant à la couche de niveau 1 et qui sont intersectés par R_i.
- Étape 2 : supprimer dans F les d-rectangles R_j qui sont complètement couverts par R_i (c'est-à-dire

lorsque $R_j \subseteq R_i$).

– Étape 3 : fragmenter chacun des d-rectangles R_j restant dans F, comme il est montré dans la figure 5.12, de manière à conserver le même codage que celui des modèles initiaux.

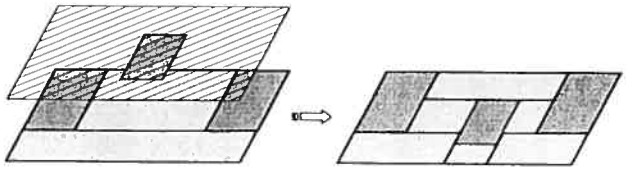


FIG. 5.12 – *Greffe d'un d-rectangle.*

Généralement, le nombre de d-rectangles augmente à chaque opération de greffe. Cette augmentation peut atteindre $2d$, où d est la dimension de l'espace d'entrée du modèle, cependant il peut y avoir des opérations de greffe où le nombre de d-rectangles diminue considérablement. Ce dernier événement est occasionnel durant l'opération d'aplatissement, il survient lorsque le nombre de d-rectangles supprimés à l'Étape 2 est suffisamment grand.

L'aplatissement à la fin demeure une meilleure solution qu'une greffe systématique. Pour l'AG, nous le faisons que sur le dernier modèle et non sur tous les modèles intermédiaires qui peuvent être au nombre de plusieurs milliers. Pour le RS et la RT, les suppressions peuvent réduire le nombre de d-rectangles et éviter des greffes inutiles.

5.7.2 Taille grandissante du modèle

Les modèles résultant de l'application des trois techniques d'optimisation (AG, RS et RT) dans la mise en oeuvre de CAMP au cas des arbres de décision sont constitués d'un grand nombre de d-rectangles qui ont des niveaux de prédominance différents. En effet, plus nous utilisons de manière optimale les capacités de ces précédents algorithmes, en augmentant par exemple leurs nombres d'itérations, plus les modèles obtenus sont de grande taille. Le problème, ainsi engendré est celui de la dégradation de la transparence et de la facilité d'interprétation du modèle. À partir d'un certain nombre de d-rectangles, il n'est pas facile de construire une image mentale claire de la sémantique du modèle.

5.8 Optimisation des modèles résultants

Dans le but de résoudre le problème de la taille, nous proposons un algorithme qui minimise le nombre de d-rectangles dans un modèle. Cet algorithme est fondé sur une idée qui permet de saisir les moments opportuns durant l'opération d'aplatissement, précisément, quand l'événement d'une diminution considérable de la taille du modèle survient. L'algorithme consiste en les étapes suivantes :

- Étape 1 : supprimer tous les d-rectangles complètement couverts par d'autres d-rectangles de niveau de prédominance supérieur.
- Étape 2 : effectuer l'opération d'aplatissement au complet en enregistrant les différentes variations de *la taille du modèle*, respectivement, à la suite de chaque opération de greffe. L'objectif de cette étape est de trouver, pour toute l'opération d'aplatissement, un minimum global d'une fonction que nous appelons « *taille du modèle* ». Cette fonction est calculée après chaque opération de greffe comme la somme du nombre de d-rectangles qui forment la couche de niveau 1 et du nombre de d-rectangles qui restent à greffer. La figure 5.13 montre un exemple de la courbe de la variation de la taille durant la progression de l'opération d'aplatissement.

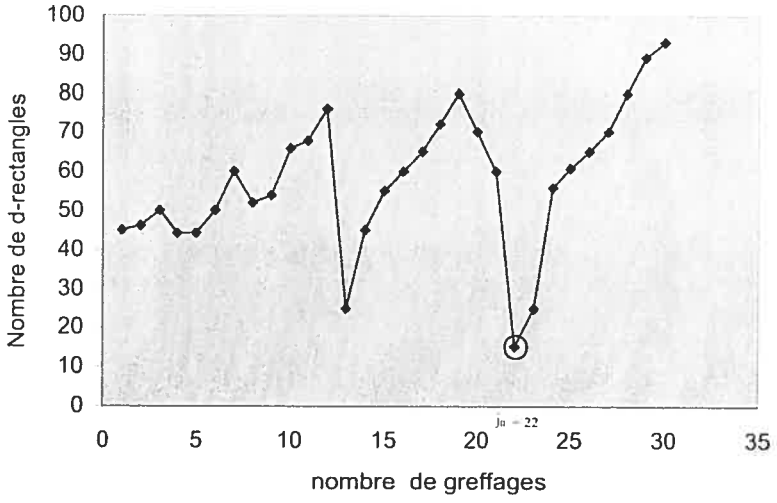


FIG. 5.13 – *Variation de la taille d'un modèle pendant l'opération d'aplatissement.*

Cette opération est effectuée sur un modèle en couches composé de 40 d-rectangles dont 10 constituent la couche de niveau 1 (30 greffes sont nécessaires pour l'aplatissement du modèle). En effectuant la $22^{ième}$ greffe la taille du modèle atteignit son minimum global.

- Étape 3 : soit j_0 le rang de l'opération de greffe qui permet d'atteindre le minimum de la fonction « taille du modèle », cette étape consiste à aplatir partiellement le modèle. En effet, nous greffons les j_0 premiers d-rectangles des couches supérieures, en commençant des plus basses, cette partie du modèle peut être enregistrée à l'étape 2 et le reste des d-rectangles est laissé sous forme de couches.

5.9 Conclusion

Dans ce chapitre nous avons adapté les trois techniques AG, RS et RT empruntées du domaine de l'optimisation au problème de combinaison et d'adaptation de modèles de type arbres de décision. Nous avons défini d'une part, les éléments communs aux trois algorithmes, à savoir, la représentation ou le codage d'un arbre de décision pour tous les algorithmes et la fonction de transition (commune à RS et RT). D'autre part, nous avons spécifié les éléments spécifiques à chacun des algorithmes, à savoir, le croisement et la mutation pour l'AG et la gestion de la liste taboue pour la RT. Les choix respectifs des principaux paramètres et critères de contrôle des différents algorithmes sont discutés dans le chapitre 7. Ces paramètres et critères incluent les conditions d'arrêts pour tous les algorithmes, les probabilités respectives du croisement et de la mutation, la taille de génération pour l'AG, la probabilité d'acceptation des transitions et la longueur et la stratégie de décroissance de la température pour le RS ainsi que la longueur de la liste taboue pour la RT.

La solution apportée au problème de la grande taille du modèle est une solution de compromis qui permet de diminuer le nombre de d-rectangles mais le modèle résultant peut contenir encore un nombre couches. Plusieurs améliorations peuvent être intéressantes pour l'application de l'approche CAMP aux arbres de décision. À titre d'exemple, nous pouvons exploiter la multitude de bons modèles dérivés par les dernières itérations du processus de combinaison et d'adaptation, pour mieux parfaire le modèle solution. En choisissant les meilleurs modèles et en tenant compte d'un autre critère d'évaluation à coté de la capacité prédictive, nous pouvons produire à la fin un modèle respectant les propriétés ciblées

par notre approche CAMP, par exemple, la facilité d'interprétation. Concrètement, cela peut être réalisé par l'introduction de la taille du modèle en d-rectangles comme deuxième fonction d'évaluation parmi les fonctions objectives des algorithmes qui implémentent CAMP. Ainsi, nous serions amenés à traiter un problème d'optimisation à deux objectifs où l'utilisation des métaheuristiques (AG, RS et RT) est tout à fait indiquée. Nous faisons illusion à l'optimisation multi-critère, ou multi-objectif (PMO) [Talbi, 2001], qui cherche à optimiser plusieurs composantes d'un vecteur fonction coût. Dans notre cas ce vecteur comporterait deux composantes : une mesure de la capacité prédictive et une autre de la taille d'un modèle.

Chapitre 6

Application de l'approche CAMP aux classificateurs bayésiens

6.1 Introduction

Ce chapitre est consacré à l'application de l'approche de combinaison et d'adaptation CAMP aux classificateurs bayésien (CB). Outre la représentation d'un autre type de modèles exprimant des relations probabilistes entre attributs (métriques et facteur de qualité), un CB constitue une version simplifiée d'un réseau bayésien, jugé prometteur pour la prédiction de la qualité du logiciel [Fenton et Ohlsson, 2000]. Nous commençons ce chapitre par une présentation détaillée des classificateurs bayésiens (section 6.2). Par la suite, dans la section 6.3, nous donnons une représentation formelle des modèles de type classificateurs bayésiens. Comme étape préalable au processus de combinaison et d'adaptation, les classificateurs bayésiens subissent un prétraitement qui permet d'unifier leurs domaines de définitions (voir section 6.4). Le reste du chapitre sera consacré à la description de la mise en oeuvre des mécanismes de l'approche CAMP, en utilisant les trois techniques métaheuristiques AG, RS et RT. Nous décrivons l'adaptation au cas des CB de certains concepts spécifiques à chacune de ces techniques. Nous commençons par définir les opérateurs génétiques de l'AG (section 6.5), puis nous abordons dans la même section 6.6 l'adaptation de la RT et de RS en commençant par leurs concepts communs.

6.2 Description d'un modèle de type classificateur bayésien

Un classificateur bayésien (CB) est une représentation probabiliste, créée pour résoudre un problème fondamental d'apprentissage qui est la classification. Cette tâche consiste à construire un classificateur capable de prédire le label de classe c d'une observation $\mathbf{x}$ étant donné $a_1,\ldots,a_d$; les valeurs respectives des attributs $x^{(1)},\ldots,x^{(d)}$ de $\mathbf{x}$, avec $c \in C = \{c_1,\ldots,c_q\}$, d le nombre d'attributs et q le nombre de labels de classe. Lorsqu'il s'agit d'un CB, cette tâche consiste à sélectionner le label c le plus probable pour l'observation $\mathbf{x}$, vérifiant l'équation suivante :

$$p(c|a_1,\ldots,a_d) = \arg\max_{c_k} p(c_k|a_1,\ldots,a_d),$$

où $c_k \in C = \{c_1,\ldots,c_q\}$.

En utilisant *la règle de Bayes*, la probabilité $p(c_k|a_1,\ldots,a_d)$ appelée **probabilité *a posteriori***, est équivalente à :

$$\frac{p(a_1,\ldots,a_d|c_k)}{p(a_1,\ldots,a_d)}p(c_k).$$

En réutilisant une seconde fois la règle de Bayes, la probabilité *a posteriori* peut être écrite comme :

$$\frac{p(a_1,\ldots,a_d|c_k)}{\sum_{h=1}^{q} p(a_1,\ldots,a_d|c_h)p(c_h)}p(c_k).$$

Ce calcul est rendu beaucoup plus simple grâce à une hypothèse assez forte qui suppose que tous les attributs sont conditionnellement indépendants [1] étant donnée la valeur du label c_l. Ainsi $p(a_1,\ldots,a_d|c_l)$ avec $l = 1,\ldots,q$ est réduite à

$$p(a_1|c_l) \times \ldots \times p(a_d|c_l)$$

Enfin, on déduit l'équation 6.1 exprimant la forme usuelle de la probabilité *a posteriori* :

$$p(c_k|a_1,\ldots,a_d) = \frac{\prod_{j=1}^{d} p(a_j|c_k)p(c_k)}{\sum_{h=1}^{q}\prod_{j=1}^{d} p(a_j|c_h)p(c_h)}p(c_k). \tag{6.1}$$

La probabilité de type $p(c_l)$ appelée *probabilité marginale* [Fenton et Neil, 1999a], ou encore *probabilité de base* [Elkan, 1997], est facilement estimée à partir des données d'entraînement. Elle est associée inconditionnellement à chaque valeur de label de classe c_l, avec $l = 1,\ldots,q$. Quant à la probabilité de

1. On dit qu'un attribut $x^{(i)}$ est indépendant d'un attribut $x^{(j)}$ étant donné le label c_l, si la probabilité $p(x^{(i)}|x^{(j)},c_l) = p(x^{(i)}|c_l)$, pour toutes les valeurs de $x^{(i)}$, de $x^{(j)}$ et de c_l.

type $p(a_j|c_l)$, appelée *probabilité conditionnelle*, elle est déterminée par apprentissage également à partir de l'ensemble de données d'entraînement. Elle est associée à chaque valeur d'un attribut d'entrée $x^{(j)}$ étant donné le label de sortie c_l, avec $j = 1,\ldots,d$ et $l = 1,\ldots,q$. Ces deux types de probabilités constituent les probabilités *a priori* d'un classificateur bayésien. Ainsi, la construction d'un classificateur à partir d'un ensemble de données d'entraînement, consiste à déterminer ses probabilités *a priori*.

6.2.1 Classificateur bayésien versus réseau bayésien

Un classificateur bayésien peut être représenté sous la forme d'un réseau bayésien (voir section B.2.4). Ce réseau considère l'hypothèse principale derrière la structure simple du CB à savoir que chaque attribut d'entrée $x^{(j)}$ (chaque feuille dans le réseau) est indépendant du reste des attributs étant donné le label de sortie (la racine dans le réseau). La figure 6.1 montre une structure simple d'un réseau bayésien qui représente un CB.

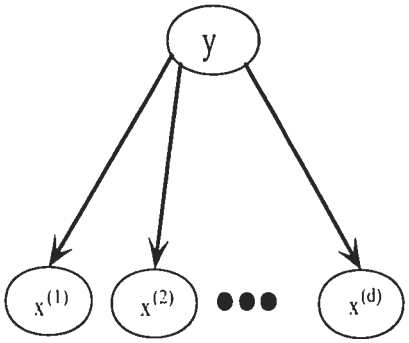


FIG. 6.1 – *Représentation d'un CB sous forme d'un réseau bayésien.*

À chaque attribut $x^{(j)}$ (feuille) est associée une distribution de probabilités conditionnelles de la forme $\{p(a_{jt}|c_k)\}$ étant donné un label de classe c_k, où a_{jt} est l'une des m valeurs que peut prendre l'attribut $x^{(j)}$, avec $j = 1,\ldots,d$ et où $1 \leq t \leq m$. Pour chaque attribut $x^{(j)}$, la somme des probabilités *a priori* est égale à 1 étant donné un label de classe c_k (voir l'équation 6.2). Nous appelons cette propriété triviale *propriété de distribution* :

$$\sum_{t=1}^{m} p(a_{jt}|c_k) = 1. \tag{6.2}$$

Pareillement, une distribution de probabilités marginales $\{p(c_k)\}$, avec $k = 1,\ldots,q$, est associée à l'attribut de sortie (la racine)

6.2.2 Classificateur bayésien et attributs continus

Dans notre présentation du classificateur bayésien, nous avons supposé que chaque attribut d'entrée est représenté par une variable $x^{(j)}$ discrète qui prend ses valeurs dans un domaine fini $Dom_j = \{a_{j1},\ldots,a_{jm}\}$. Cependant, plusieurs attributs (continus) peuvent admettre une infinité de valeurs . Dans ce cas, on utilise une méthode de discrétisation pour avoir un nombre fini d'intervalles à la place d'un nombre infini de valeurs [Ramoni et Sebastiani, 1999]. On parle alors de l'appartenance d'une valeur d'un attribut $x^{(j)}$ à un intervalle I_j inclus dans son domaine de définition. Par conséquent, on exprime par $p(I_j|c_k)$ la probabilité conditionnelle (*a priori*) d'appartenance d'une valeur de l'attribut $x^{(j)}$ à l'intervalle I_j sachant que le label de sortie est c_k. De cette manière, les probabilités *a posteriori* $p(c_k|a_1,\ldots,a_d)$ sont remplacées par $p(c_k|I_1,\ldots,I_d)$, avec $a_1 \in I_1,\ldots,a_d \in I_d$. Ces probabilités sont données par l'équation 6.3

$$p(c_k|I_1,\ldots,I_d) = \frac{\prod_{j=1}^{d} p(I_j|c_k)}{\sum_{h=1}^{q}\prod_{j=1}^{d} p(I_j|c_h)p(c_h)}p(c_k). \tag{6.3}$$

Le tableau 6.1 est un exemple de classificateur bayésien représenté par ses probabilités *a priori* organisées par attribut. Ce CB classifie les composants logiciels selon leurs niveaux de stabilité en deux classes : stable (1) et instable (-1). Il utilise des attributs d'entrée comme *COH*, *LCOMB* et *STRESS*. Les probabilités marginales (inconditionnelles) d'avoir des composants stables et instables sont respectivement égales à 0.777 et à 0.223. Le domaine de chaque attribut est subdivisé en intervalles. À chaque intervalle sont associées deux probabilités conditionnelles *a priori* : la première indique la probabilité d'appartenance d'une valeur de l'attribut à cet intervalle quand le composant est stable, la deuxième indique la même probabilité mais quand le composant est instable. À titre d'exemple, prenons l'attribut *COH* qui est défini dans le domaine [0, 1]. Ce domaine de définition est subdivisé en un nombre d'intervalles (20) dont le premier est [0, 0.05[. À ce premier intervalle sont associées une première probabilité conditionnelle *a priori* (0,094) indiquant la chance d'avoir une valeur de *COH* appartenant à l'intervalle [0, 0,005[sachant que le composant est instable (-1) et une deuxième probabilité conditionnelle *a priori* (0,916) indiquant la chance de la même appartenance mais lorsque le composant est stable (1).

Classe STAB				
-1	1			
0,223	0,777			
Attribut COH				
	(0 , 0,05)	(0,05 , 0,1)	...	(0,9500000000000003 , 1,0)
-1	0,094	0,094	...	0,042
1	0,916	0,015	...	0,051
Attribut LCOMB				
	(0 , 1033/20)	(1033/20 , 1033/10)	...	(19627/20 , 1033)
-1	0,698	0,042	...	0,010
1	0,916	0,015	...	0,006
⋮	⋮	⋮	⋮	⋮
Attribut STRESS				
	(0,0 , 0,04166)	(0,04166 , 0,0833)	...	(0,79166 , 0,833)
-1	0,156	0,073	...	0,021
1	0,928	0,003	...	0,003

TAB. 6.1 – *Un exemple de classificateur bayésien.*

6.3 Représentation d'un classificateur bayésien sous forme d'expertises

6.3.1 Notations et formalismes

Considérons un CB, tel qu'il est décrit dans la section 6.2, utilisant des attributs continus. Chaque attribut $x^{(j)}$ prend ses valeurs dans un domaine $[L_j,U_j]$, avec L_j et U_j, $j = 1,\ldots,d$, les deux bornes respectivement inférieure et supérieure de cet attribut. Supposons que $[L_j,U_j]$ soit divisé en m intervalles $I_{j1},\ldots,I_{jm}$, avec m variant d'un attribut à un autre. Soit une observation $\mathbf{x} = (a_1,\ldots,a_d)$, la première étape d'inférence du CB consiste à trouver pour chacune des valeurs a_j, un intervalle $I_j \subset [L_j,U_j]$, tel que $a_j \in I_j$. Le produit cartésien des intervalles trouvés forme un d-rectangle $R = I_1 \times \ldots \times I_d$. La deuxième étape consiste à calculer les probabilités *a posteriori* pour chaque label de classe c_k sachant que l'observation $\mathbf{x}$ est localisée dans R ($\mathbf{x} \in R$). Ces probabilités *a posteriori* $p(c_k|I_1,\ldots,I_d)$, avec $k =$

$1,\ldots,q$, sont calculées à partir des probabilités *a priori* et en utilisant l'équation 6.3. À la troisième étape, on identifie le label c de l'observation $\mathbf{x}$, tel que $p(c|I_1,\ldots,I_d) = \arg\max_{c_k} p(c_k|I_1,\ldots,I_d)$. Les probabilités *a posteriori* $p(c_k|I_1,\ldots,I_d)$ sont notées par $p(c_k|R)$.

Enfin, un CB peut être représenté, comme tout classificateur, par une fonction f qui est définie dans $\mathcal{V} = [L_1,U_1] \times \ldots \times [L_d,U_d]$ par :

$$f : \mathcal{V} \mapsto C = \{c_1,\ldots,c_q\} \quad et \quad f(\mathbf{x}) = c \text{ telle que } p(c|\mathbf{x}) = \arg\max_{c_k} p(c_k|R) \quad et \quad \mathbf{x} \in R. \tag{6.4}$$

6.3.2 Identification des expertises dans un classificateur bayésien

Afin de décomposer un CB en expertises, deux choix sont envisageables : le premier permet d'exprimer une expertise sous forme de probabilités *a posteriori*, le deuxième considère qu'une expertise peut être représentée par des probabilités *a priori*.

6.3.2.1 Expertise sous forme de probabilités *a posteriori*

Avec cette alternative, une expertise e_i est représentée par le $(q+1)-uplet$: $(R_i,p_{c_1},\ldots,p_{c_q})$, où p_{c_k} est la probabilité *a posteriori* du label c_k dans le d-rectangle R_i, $p_{c_k} = p(c_k|R_i)$. Nous appelons ce type d'expertises *expertises a posteriori*. La fonction de l'expertise *a posteriori* est donnée par l'équation 6.5.

$$e_i : R_i \mapsto C = \{c_1,\ldots,c_q\} \quad et \quad e_i(\mathbf{x}) = c \text{ telle que } p_c = \arg\max_{c_k} p(c_k|R_i). \tag{6.5}$$

Cette décomposition est similaire à celle d'un arbre de décision (voir l'équation 4.2). Son avantage est de combiner facilement un classificateur bayésien avec d'autres types de modèles comme les arbres de décision. L'application de l'approche CAMP, revient alors à manipuler les d-rectangles et la mise en oeuvre des mécanismes de combinaison et d'adaptation en utilisant AG, RS ou RT est la même que dans le cas d'arbres de décision. Cependant, en manipulant des expertises *a posteriori*, la structure d'un modèle résultant de CAMP ne correspond plus à celle d'un classificateur bayésien. Supposons que nous ajoutons (par greffe d'expertises, voir section 5.5.3.2) un sous ensemble d'expertises *a posteriori* à un CB $CB1$. Les probabilités *a posteriori* des expertises ajoutées ne correspondent plus aux distributions des probabilités *a priori* de $CB1$. En effet, pour préserver les bonnes distributions et conserver la structure d'un CB, il est indispensable de manipuler les probabilités *a priori*.

6.3.2.2 Expertise sous forme de probabilités *a priori*

En général, la variation de la probabilité marginale *a priori* de type $p(c_k)$ d'un CB à un autre est faible. Cette affirmation est vérifiée empiriquement dans la section 7.6.3 pour le cas d'un CB de qualité du logiciel. C'est pourquoi ce type de probabilités ne constitue pas une information effective qui permet de distinguer les CB (voir dans [Elkan, 1997]).

Un grand avantage des modèles bayésiens que nous avons évoqué dans cette dissertation est leur transparence. En particulier, un CB exprime grâce aux probabilités conditionnelles *a priori*, les relations de causalité établies entre chaque métrique d'entrée et le facteur de qualité à prédire. Par conséquent, nous cherchons à conserver la structure d'un CB tout au long de notre processus de combinaison et d'adaptation sous forme de probabilités *a priori* (voir le tableau 6.1). Par ailleurs, une expertise *a posteriori* peut être recalculée et retrouvée en utilisant les probabilités *a priori* (voir équation 6.3). Pour ces raisons, nous avons décidé d'utiliser les probabilités conditionnelles *a priori* pour définir une expertise que nous appelons *expertise a priori*. En effet, pour chaque attribut $x^{(j)}$ sont définies m expertises *a priori*. Chaque expertise est représentée par le $(q+1)-uplet$: $(I_{jt}, p_{jc_1}, \ldots, p_{jc_q})$, où p_{jc_k} est la probabilité conditionnelle *a priori* (probabilité *a priori* tout court) $p(I_{jt}|c_k)$ et I_{jt} est l'intervalle de rang t dans le domaine de l'attribut $x^{(j)}$, où $1 \leq t \leq m$. La fonction de l'expertise *a priori* e_{jt} est donnée par l'équation 6.6 :

$$e_{jt} : I_{jt} \mapsto [0.1]^d \quad et \quad e_{jt}(x^{(j)}) = (I_{jt}, p_{jc_1}, \ldots, p_{jc_q}). \tag{6.6}$$

L'utilisation de ce type d'expertise permet de conserver la structure d'un CB (voir figure 6.2). Une façon de surmonter le problème relié à l'utilisation des expertises *a posteriori* est de manipuler ces dernières à travers les expertises *a priori*. En effet, vis-à-vis des mécanismes de combinaison et d'adaptation, nous considérons un CB comme un ensemble d'expertises *a priori*. Par contre, nous utilisons la décomposition en expertises *a posteriori* du CB pour faire l'opération d'évaluation de la performance de ce dernier. Cette représentation d'un classificateur bayésien est utilisée par les trois techniques de combinaison à savoir l'algorithme génétique, le recuit simulé et la recherche avec tabous.

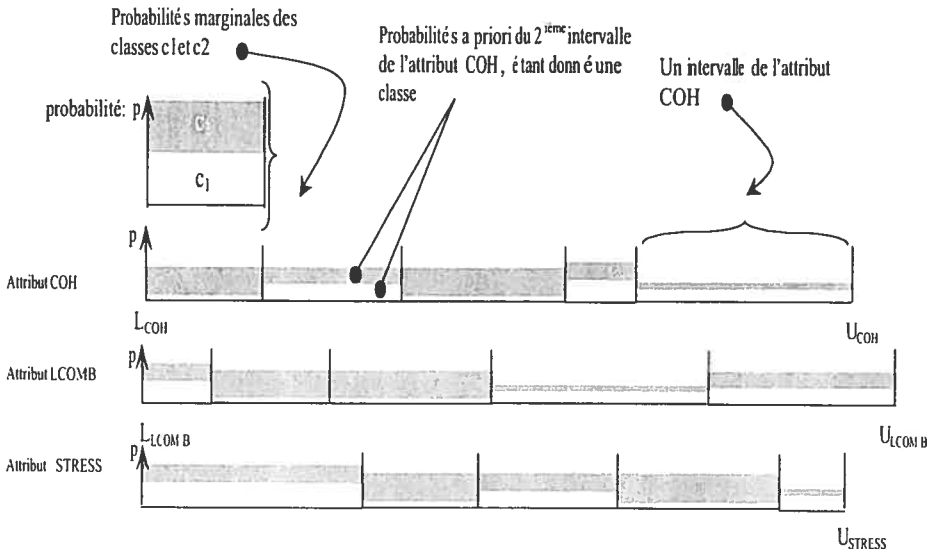


FIG. 6.2 – *Une représentation graphique d'un classificateur bayésien.*

6.4 Prétraitement des modèles de type classificateur bayésien

Conformément au but de l'étape de prétraitement de l'approche CAMP, décrit dans la section 4.8.1, les classificateurs bayésiens candidats à la combinaison doivent avoir le même espace d'entrée $\mathcal{V}$. De la même manière que les arbres de décision, chaque CB aura comme attributs d'entrée toutes les métriques utilisées dans tous les CB. Ainsi, un classificateur peut se voir ajouter zéro ou plusieurs attributs. Lorsque un attribut $x^{(j)}$ est ajouté aux entrées d'un CB, son domaine de définition $[L_j, U_j]$ est gardé sans aucune subdivision et avec une probabilité *a priori* $p([L_j, U_j]|c_k)$ égale à 1 pour tout $k = 1 \ldots .q$. Les bornes L_j et U_j de attribut $x^{(j)}$ prennent, respectivement, le minimum des bornes inférieures et le maximum des bornes supérieures proposés par les différents CB candidats à l'application de CAMP. Dans chaque CB, le domaine de chaque attribut $x^{(j)}$ est élargi par l'ajout d'un intervalle (lorsque on modifie exclusivement la borne inférieure ou supérieure) ou de deux intervalles (lorsque on modifie les deux bornes). Les probabilités *a priori* associées aux intervalles ajoutés prennent des valeurs nulles afin de garder, pour chaque attribut, la somme des probabilités *a priori* égale à 1 étant donné un label de classe c_k (voir l'équation 6.2). Ce choix nous semble le plus justifié car si un attribut $x^{(j)}$ n'est pas défini dans un intervalle donné I_a alors la probabilité d'appartenance des valeurs de $x^{(j)}$ à I_a est nulle, $p(I_a) = 0$ et

par conséquent les probabilités *a priori* $p(I_a|c_k)) = 0$. Ceci peut être interprété comme un manque de représentativité de l'échantillon de données utilisé pour construire le CB de plus petit domaine de $x^{(j)}$. Cependant, d'autres choix peuvent être envisagés pour préserver la distribution, comme étendre le premier et/ou le dernier intervalle(s).

Une illustration de cette opération est présentée par la figure 6.3.

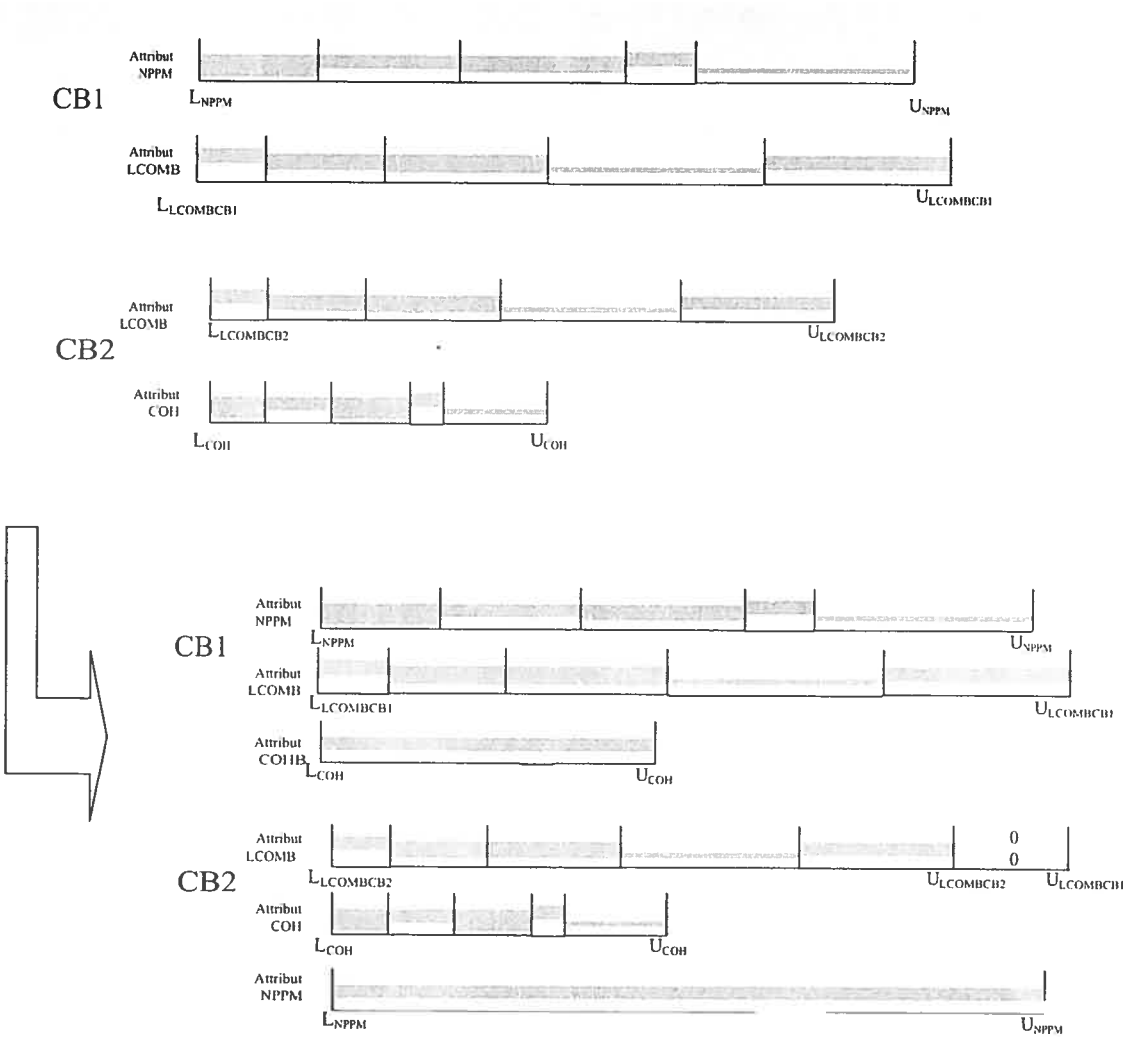


FIG. 6.3 – *Exemple du prétraitement des classificateurs bayésiens.*

Nous considérons deux classificateurs bayésiens $CB1$ et $CB2$ pour prédire le niveau de la stabilité des composants logiciels selon deux classes *stable* et *instable*. Le premier utilise les deux métriques *NPPM*

et *LCOMB*, le deuxième utilise les deux métriques *COH* et *LCOMB*. Dans cet exemple, l'opération de prétraitement comprend une première tâche qui consiste à déterminer les domaines de définition de chaque attribut. Le domaine de *NPPM* est le même que celui proposé par *CB*1, *COH* garde son domaine proposé par *CB*2 et le domaine de *LCOMB* est le même que celui proposé par *CB*1. Par conséquent, dans *CB*1, le domaine de définition de l'attribut *LCOMB* est étendu par l'intervalle $[U_{LCOMB_{CB2}}, U_{LCOMB_{CB1}}]$ dont les probabilités *a priori* associées sont nulles, $U_{LCOMB_{CB2}}$ et $U_{LCOMB_{CB1}}$ sont les deux bornes supérieures de *LCOMB* proposées respectivement par *CB*1 et *CB*2. La seconde tâche consiste à ajouter d'une part, l'attribut *COH* aux entrées du *CB*1, avec $p([L_{COH}, U_{COH}]||stable) = p([L_{COH}, U_{COH}]||instable) = 1$ et d'autre part, l'attribut *NPPM* aux entrées du *CB*2, avec $p([L_{NPPM}, U_{NPPM}]||stable) = p([L_{NPPM}, U_{NPPM}]||instable) = 1$.

Cette opération de prétraitement de modèles de type classificateur bayésien est indépendante de la technique de mise en oeuvre de l'approche CAMP, ainsi, un CB est préalablement « prétraité » de la même manière pour les trois algorithmes GA, RS et RT.

6.5 Algorithme génétique

Dans cette section, nous utilisons les concepts empruntés aux algorithmes génétiques. D'abord, nous identifions à partir de la section 6.2 le codage d'un CB en chromosomes. La fonction *fitness* est brièvement abordée dans la section 6.5.2. Par la suite, nous proposons une adaptation des opérateurs de l'AG au problème de l'application de CAMP aux classificateurs bayésiens.

6.5.1 Codage d'un classificateur bayésien en chromosome

En utilisant la représentation d'un classificateur bayésien sous forme d'expertises fournie dans la section 6.3 et en tenant compte de la correspondance établie dans la section 4.9.1, chaque CB doit être représenté par un chromosome. Ainsi un gène correspondra à une expertise *a priori*. Concrètement, c'est un intervalle auquel sont associées des probabilités *a priori*. Un gène est alors représenté par le $(q+1)-uplet$: $(Intervalle, p_{c_1}, \ldots, p_{c_q})$, où *Intervalle* est un intervalle d'un attribut et p_{c_k} est égale à $p(Intervalle|c_k)$. Pour des raisons de simplification, nous confondons gène et intervalle. Un chromosome est codé sous forme d'un ensemble d'intervalles. La taille d'un chromosome est égale au nombre

de tous les intervalles de tous les attributs. Par exemple, dans la figure 6.2 les gènes sont 15 intervalles. À chacun sont associées deux bandes colorées où les deux couleurs représentent respectivement, les labels de sortie c_1 (*stable*) et c_2 (*instable*) et l'épaisseur d'une bande est proportionnel à la valeur de la probabilité *a priori* représentée.

6.5.2 Fonction de *fitness*

Notre fonction de *fitness* d'un chromosome n'est autre que la capacité prédictive du classificateur bayésien. De la même manière que pour un arbre de décision, une proposition de fonction de *fitness* est alors la fonction d'exactitude (section 4.8.2.1) ou l'indice J transformé JT (section 4.8.2.2) dépendamment de la distribution des labels de classes.

6.5.3 Opérateur de croisement

Étant donnée la structure d'un classificateur bayésien, l'opération de croisement la plus intuitive entre deux chromosomes CB (voir section 2.3.2.5) consiste à effectuer un nombre d'opérations plus élémentaires de « *croisement entre attributs* ». Un croisement entre attributs s'effectue entre deux instances[2] d'un même attribut, issues respectivement des deux chromosomes parents. Pour réaliser un croisement entre deux chromosomes, le nombre de «*croisements entre attributs*» peut atteindre d si nous supposons que tous les attributs participent au croisement. Deux nouvelles instances du même attribut sont formées après chaque croisement élémentaire. Chacune de ces deux nouvelles instances participe à la formation d'un nouveau chromosome enfant. Deux nouveaux CB sont produits au bout d'un nombre de «*croisements entre attributs*». Les instances d'attributs qui n'ont pas participé à un croisement sont insérées directement dans les deux nouveaux CB.

Le problème du croisement est de cette façon réduit à un problème de combinaison d'intervalles qui composent les instances d'attribut. Cette combinaison ressemble à l'opération du croisement utilisée dans le cas d'arbres de décision, sauf que cette dernière manipule des d-rectangles à la place d'intervalles. Par conséquent, plusieurs formes de croisement entre attributs sont envisageables. L'adéquation de ces formes de croisement à notre problème dépend de leur pouvoir de préserver, pour les classifica-

2. Nous appelons instance d'un attribut, une composition particulière de l'attribut (ensemble d'intervalles auxquels sont attachées des valeurs de probabilité *a priori*).

teurs bayésiens produits, d'une part la cohérence et la complétude (voir section 4.8.3), d'autre part la propriété de distribution définie pour chaque attribut (voir l'équation 6.2).

6.5.3.1 Croisement standard

La manière standard d'effectuer l'opération de « *croisement entre attributs* » est celle du croisement par découpage, ou *slicing crossover* (voir section 2.3.2.5). Elle consiste à découper chacune des deux instances parents d'un même attribut (issue chacune d'un chromosome) en deux sous-ensembles de gènes (intervalle dans notre cas). Ce découpage est aléatoire et n'est pas unique. Deux nouvelles instances de l'attribut sont créées en intercalant les sous-ensembles. Si nous appliquions de telles opérations dans notre problème, il est possible que les chromosomes résultants ne puissent plus vérifier les propriétés d'un modèle (la complétude et la cohérence) et que la propriété de distribution ne soit pas respectée pour tous les attributs.

Cette première tentative de croisement ainsi que les deux premiers problèmes (d'incomplétude et d'incohérence) qui peuvent se produire dans le cas de classificateurs bayésiens sont décrits dans la figure 6.4. Dans cette figure, chacune des deux instances de l'attribut *COH* (COH_1 et COH_2) est divisée en deux sous ensembles d'intervalles (S_{11} et S_{12} proviennent de COH_1 et S_{21} et S_{22} proviennent de COH_2). Les instances enfants de l'attribut *COH* ($COH_{Enfant1}$ et $COH_{Enfant2}$) sont formées respectivement par les combinaisons de S_{11} avec S_{21} et S_{22} avec S_{12}. Le problème d'incomplétude et celui d'incohérence rencontrés sont se manifestent comme suit :

Dans l'instance $COH_{Enfant1}$, aucune expertise *a priori* (intervalle avec probabilités) n'est associée à l'intervalle I_1 du domaine de définition de l'attribut *COH*. Il s'agit du problème *d'incomplétude d'attribut* affectant précisément l'instance $COH_{Enfant1}$. Par conséquent, la fonction représentée par le CB utilisant $COH_{Enfant1}$ n'est pas définie pour les observations ayant une valeur de *COH* dans l'intervalle I_1 et pour lesquelles on ne peut calculer de probabilités *a posteriori* $p(c_k|\ldots,I_1,\ldots)$, avec $k = 1,2$. Nous montrons ainsi le problème *d'incomplétude d'un modèle de type CB*.

À l'intervalle I_2 de l'instance $COH_{Enfant2}$ sont associées deux expertises différentes. Ceci est représenté par un chevauchement de deux intervalles auxquels sont associées deux probabilités *a priori* différentes étant donnée la valeur d'un label de classe. Il s'agit du problème *d'incohérence dans un attribut* qui se produit précisément dans l'instance $COH_{Enfant2}$. Le CB contenant $COH_{Enfant2}$ n'est même

pas une fonction car deux probabilités *a posteriori* sont associées aux cas ayant une valeur de *COH* dans l'intervalle I_2. Nous montrons à travers ce cas le problème *d'incohérence d'un modèle de type CB*.

Quant à la propriété de distribution, le croisement standard n'offre aucun moyen de la préserver, la définition d'une distribution n'est pas respectée.

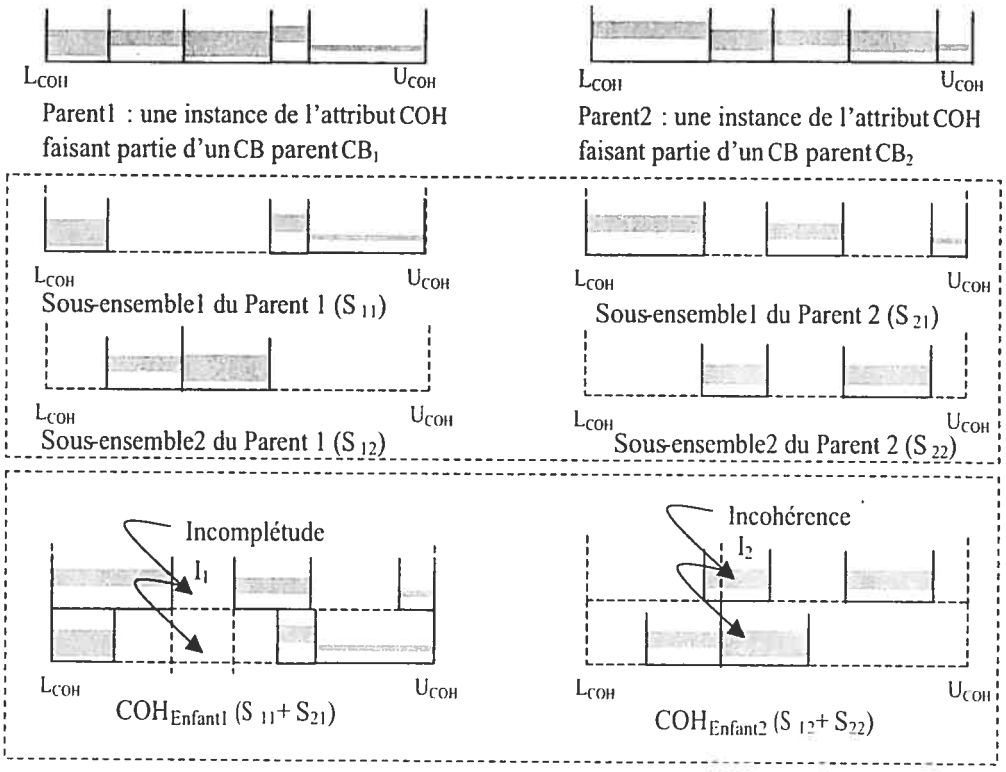


FIG. 6.4 – *Problèmes de l'utilisation du croisement standard.*

6.5.3.2 Croisement par greffe d'intervalles

Comme il est montré dans la section précédente, la propriété de cohérence ainsi que celle de complétude d'un modèle de type classificateur bayésien découlent respectivement de la cohérence et de la complétude des attributs qui composent le CB. Notre souci est de préserver, après chaque opération de croisement, trois propriétés d'attribut qui sont *la cohérence*, *la complétude* et *la distribution*. Pour ce faire, nous proposons un nouvel opérateur de croisement (entre attributs) inspiré des opérateurs définis pour résoudre les problèmes de groupement [Falkenauer, 1998].

Pour obtenir une instance enfant d'un attribut, nous choisissons un sous-ensemble aléatoire d'inter-

valles[3] à partir d'une instance parent de l'attribut et nous l'ajoutons à l'ensemble entier d'intervalles d'une deuxième instance parent du même attribut. En gardant tous les intervalles de l'un des parents, la complétude de l'instance enfant est assurée. Pour préserver la cohérence, nous décidons que les intervalles ajoutés soient prédominants, c'est-à-dire que les probabilités *a priori* attachées aux intervalles ajoutés l'emportent sur celles des intervalles originels dans la nouvelle composition d'un attribut. Ce croisement est à son tour réalisé par une série d'opérations plus élémentaires. Chacune consiste à greffer un seul intervalle de la première instance parent dans la deuxième. Lorsque un intervalle greffé I_g prend place dans la composition de la deuxième instance de l'attribut, les intervalles qui l'intersectent sont supprimés (lorsqu'ils sont complètement couverts par I_g) ou rétrécis (lorsqu'ils sont partiellement couverts par I_g).

Après chaque greffe d'un intervalle I_g, un ajustement de probabilités est nécessaire afin de préserver la propriété de distribution. En effet, seules les probabilités *a priori* attachées à l'intervalle greffé $p(I_g|c_k)$, $k = 1,\ldots,q$ sont conservées alors que celles des intervalles originels restants $I_1,\ldots,I_m$, sont ajustées. Plusieurs choix sont envisageables pour préserver la propriété de distribution. Il suffit, pour chaque classe c_k de distribuer le reliquat de probabilité ($1 - p(I_g|c_k)$) sur les intervalles restants $I_1,\ldots,I_m$. Par exemple, nous pouvons affecter à chaque intervalle restant une probabilité *a priori* proportionnelle à sa longueur. Ce choix suppose que distribution uniforme des valeurs de l'attribut et ignore la distribution originelle des probabilités *a priori*. Une autre alternative consiste à affecter à chaque intervalle restant une probabilité *a priori* proportionnelle à sa probabilité *a priori* originelle. Cette alternative est plus raisonnable car elle ne suppose aucune distribution arbitraire des valeurs de l'attribut et pend en compte les expertises originelles attachées aux intervalles restants $I_1,\ldots,I_m$. Les nouvelles probabilité *a priori* sont affectées conformément à l'équation 6.7 :

$$p(I_t|c_k) = \frac{p(I_t|c_k)}{\sum_{i=1}^{m} p(I_i|c_k)}(1 - p(I_g|c_k)), \tag{6.7}$$

où $k = 1,\ldots,q$, $t = 1,\ldots,m$ et m est le nombre d'intervalles originels restant après greffe. Dans cette équation, nous pondérons le reliquat $1 - p(I_g|c_k)$, au prorata de la probabilité *a priori* originelle $p(I_t|c_k)$.

La taille du sous-ensemble aléatoire des intervalles greffés vaut a fois le nombre d'intervalles de l'attribut parent, où a est un paramètre de l'algorithme génétique. La figure 6.5 illustre le nouvel opérateur

3. Un intervalle est équivalent à une expertise qui est représentée par un intervalle avec ses probabilités *a priori*.

de croisement.

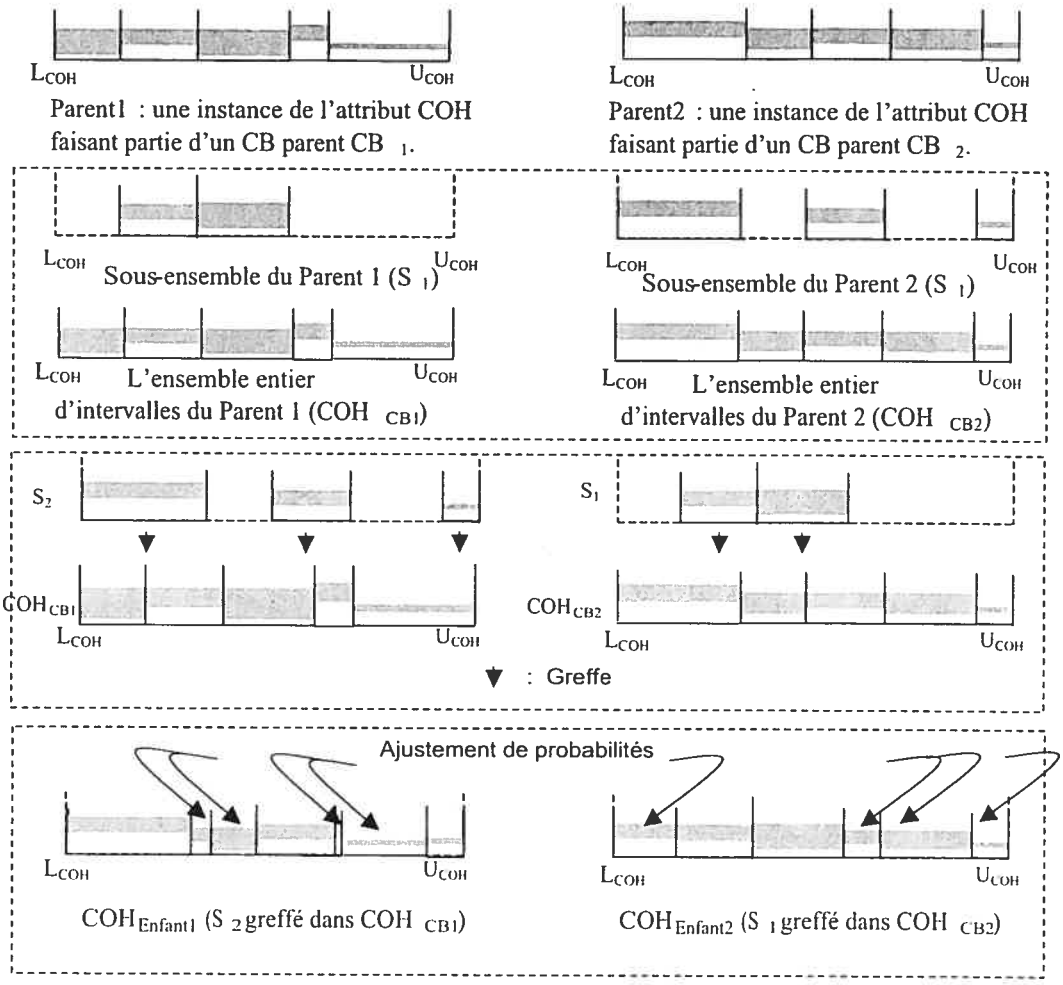


FIG. 6.5 – *Croisement qui préserve la cohérence, la complétude et la distribution.*

6.5.4 Opérateur de mutation

La mutation est un changement aléatoire des gènes qui se produit avec une probabilité faible. Dans notre problème, l'opérateur de mutation consiste à changer aléatoirement certaines probabilités *a priori* attachées à un intervalle dans un attribut. Un tel changement engendre le plus souvent l'ajustement de toutes les probabilités *a priori* de tous les intervalles de l'attribut afin de préserver la propriété de distribution. Afin d'atteindre l'objectif du mécanisme d'adaptation de l'approche CAMP (décrit dans la section 4.8.5), nous envisageons plusieurs formes de mutation dont nous présentons les plus importantes

et les plus simples.

6.5.4.1 Mutation par inter-changement des probabilités *a priori*

Une première forme de mutation consiste à sélectionner d'une manière aléatoire deux intervalles I_1 et I_2 d'un même attribut et d'inter-changer leurs probabilités *a priori* respectives. Après la mutation, les valeurs des probabilités $p(I_1|c_k)$ sont remplacées par les valeurs de $p(I_2|c_k)$ et vice versa, pour tous les labels de classe c_k avec $k = 1,\ldots,q$. L'intérêt de cette tentative de mutation découle de sa préservation automatique de la propriété de distribution et du nombre réduit des intervalles impliqués. La figure 6.6(a) montre un exemple de cette mutation effectuée dans un CB à trois classes.

6.5.4.2 Mutation par permutation cyclique des probabilités *a priori*

Cette tentative de mutation implique tous les intervalles I_t d'un attribut, avec $t = 1,\ldots,m$ où m est le nombre d'intervalles composant l'attribut à « muter ». Elle change les valeurs des probabilités *a priori* $p(I_t|c_k)$, dans tous les intervalles I_t d en :

- $p(I_t|c_{k+1})$ **si** $1 \leq k < q$
- $p(I_t|c_1)$ **si** $k = q$,

ou en :

- $p(I_t|c_{k-1})$ **si** $1 < k \leq q$
- $p(I_t|c_q)$ **si** $k = 1$.

La figure 6.6(b) illustre la mutation d'un CB[4]. L'attribut *COH* est présenté avant et après la mutation, dans cet exemple les probabilités $p(I_t|c_1)$ changent en $p(I_t|c_2)$, $p(I_t|c_2)$ en $p(I_t|c_3)$ et $p(I_t|c_3)$ en $p(I_t|c_1)$. L'avantage de cette forme de mutation est qu'elle préserve également la propriété de distribution.

4. Nous utilisons un CB à trois classes afin de mieux illustrer l'opération de la mutation par permutation cyclique des probabilités.

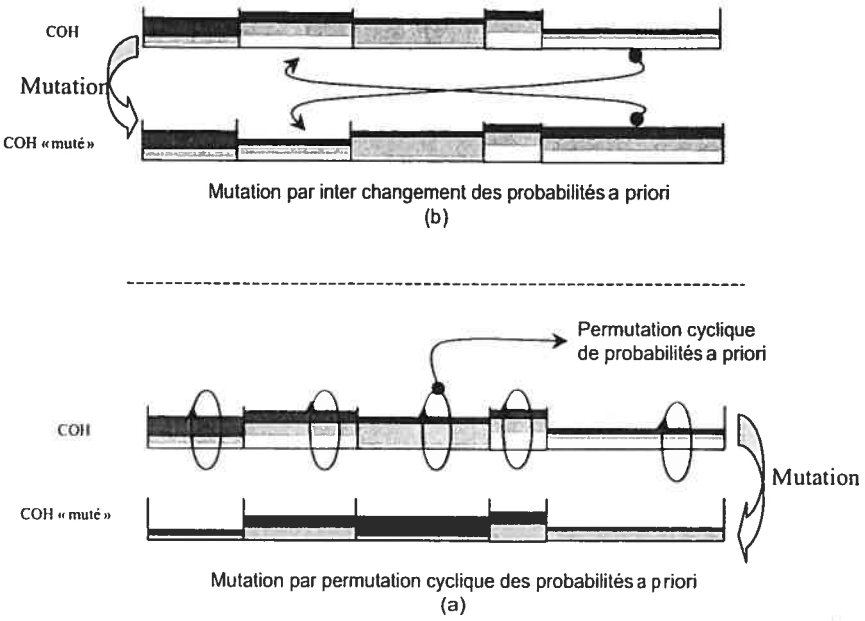


FIG. 6.6 – *Exemples de mutation.*

6.5.4.3 Mutation par changement arbitraire des probabilités *a priori*

Cette tentative de mutation consiste à permettre à une ou plusieurs probabilités *a priori* de changer aléatoirement en prenant des valeurs arbitraires comprises entre 0 et 1. Après ce changement, un ajustement des probabilités *a priori* du reste des intervalles est indispensable pour préserver la propriété de distribution dans l'attribut « muté ». Cet ajustement est effectué d'une manière similaire à celle utilisée dans le croisement par greffe d'intervalles. Après changement aléatoire d'une probabilité *a priori* d'un intervalle, les probabilités *a priori* des intervalles restantes changent au prorata de leurs valeurs originelles (voir section 6.5.3.2).

6.6 Recuit simulé et recherche avec tabous

Comme il a été mentionné dans le chapitre 4, les algorithmes RS et RT optimisent une fonction objectif qui mesure la capacité prédictive d'un classificateur bayésien. Ils utilisent trois concepts communs qui sont la fonction objectif, la représentation des solutions et la fonction de transition. La fonction objectif est mesurée de la même façon que la fonction de *fitness* dans les AG (voir section 6.5.2). Concer-

nant la représentation des solutions, nous utilisons le même codage des classificateurs bayésiens retenu pour l'application de l'AG, tandis que nous proposons, dans la section 6.6.1, une fonction de transition qui permet de parcourir le voisinage d'une solution courante (classificateur bayésien dans notre cas). Quant aux autres concepts spécifiques et nécessaires pour appliquer chacun des deux algorithmes, nous utilisons pour le RS, les mêmes paramètres et stratégies décrits dans la section 2.3.3 à savoir, la longueur de température, le critère d'arrêt, le schéma de décroissance de la température et le critère d'acceptation des transitions. De même, pour la RT, nous définissons dans la section 6.6.2, une stratégie de gestion de la liste taboue. La longueur de cette dernière et les choix des autres stratégies et valeurs des paramètres des deux algorithmes sont présentés dans le chapitre 7.

6.6.1 Fonction de transition

La fonction de transition est l'élément de RS et de RT sur lequel se basent les deux mécanismes principaux de l'approche CAMP: la combinaison et l'adaptation des classificateur bayésiens. Une transition intuitive est réalisée en effectuant un changement dans la structure du classificateur bayésien représentant la solution courante afin d'obtenir un CB voisin. Plus concrètement, à partir d'une représentation d'un CB sous forme d'un ensemble d'expertises (intervalles), une transition consiste à apporter une modification raisonnable sur un nombre réduit d'intervalles. Le grand nombre d'informations attachées à chaque intervalle nous laisse penser à plusieurs formes de transition, en particulier à celles proches des actions utilisées comme alternatives de mutation et qui sont décrites dans la section 6.5.4. Par ailleurs, le mécanisme de combinaison de l'approche CAMP est fondé sur la réutilisation des expertises provenant des différents classificateurs bayésiens existants. En effet, pour implanter ce mécanisme au mieux, nous proposons aussi deux autres formes de transitions basées respectivement sur l'ajout d'expertises (intervalles) à un CB et sur la suppression d'expertises à partir d'un CB.

- **l'ajout d'une expertise** : afin de préserver la cohérence du CB, nous procédons à une greffe d'un nouvel intervalle (nouvelle expertise), choisi aléatoirement à partir d'un dépôt d'intervalles, dans la composition de l'attribut adéquat[5] du CB. Cette action suppose l'existence d'un dépôt d'expertises obtenues par la dissociation de tous les intervalles qui composent les attributs des CB existants.

5. Par exemple, un intervalle de l'attribut *COH* dans le dépôt est ajouté à l'attribut *COH* du modèle.

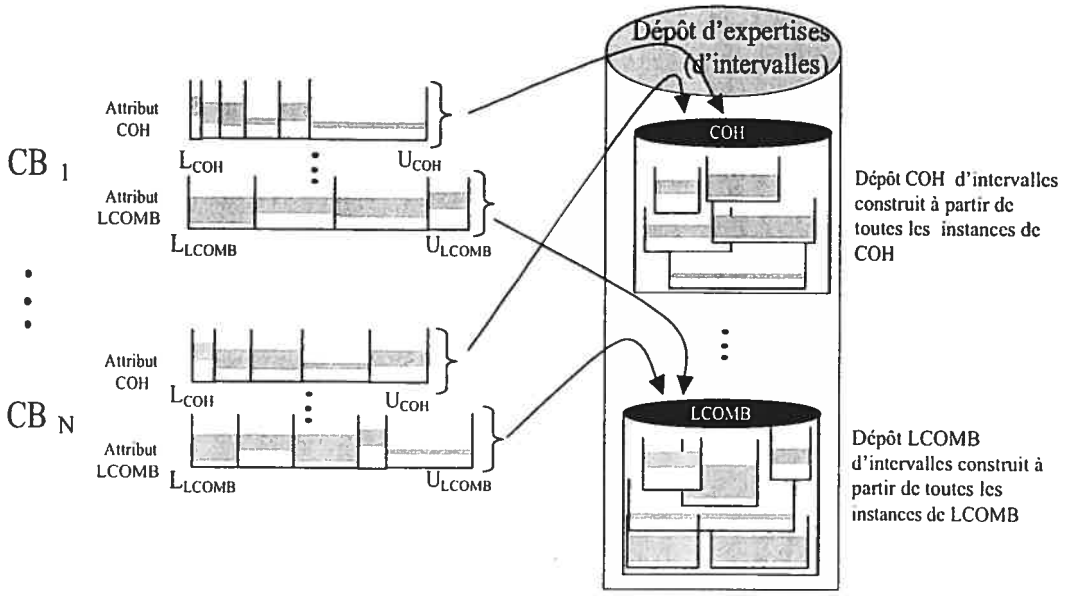


FIG. 6.7 – *Formation d'un dépot d'expertises.*

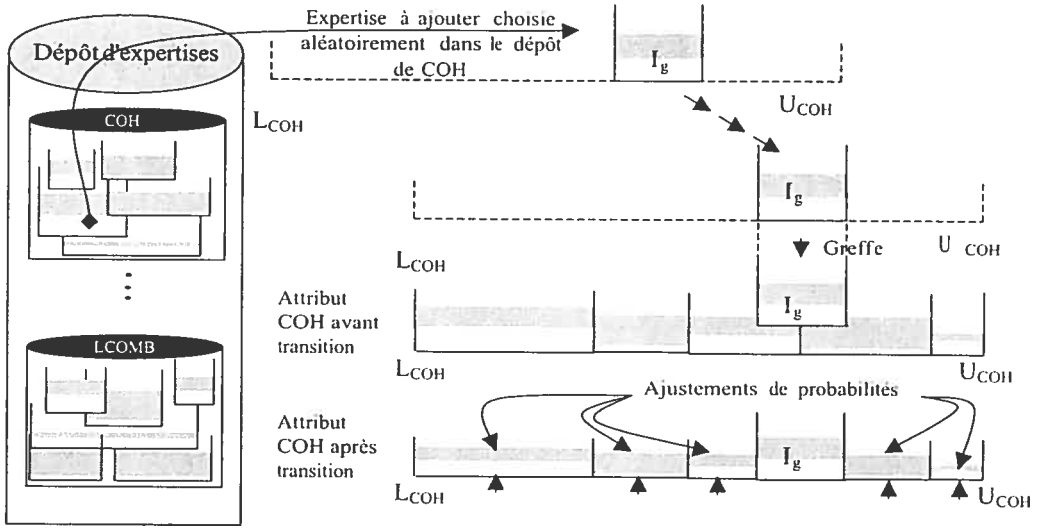


FIG. 6.8 – *Fonction de transition par ajout d'une expertise.*

Ces classificateurs sont différents du CB initial choisi pour débuter l'algorithme de recherche locale (RS ou RT). La figure 6.7 montre le processus de formation du dépôt d'expertises. Une fois

un intervalle I_g greffé, les probabilités *a priori* à l'extérieur de I_g sont ajustées en utilisant l'équation 6.7 pour préserver la propriété de distribution (voir section 6.5.3.2). La figure 6.8 illustre l'opération d'ajout d'une expertise à un modèle de type classificateur bayésien.

- **la suppression d'une expertise**: cette forme de transition consiste à supprimer un intervalle I_s choisi aléatoirement dans la composition d'un attribut. Afin de préserver la complétude, l'un ou les deux intervalles voisins de I_s sont élargis pour combler le vide causé par la suppression. Quant à la propriété de distribution, elle est préservée par un ajustement qui peut s'effectuer de plusieurs manières. Nous pouvons par exemple multiplier chaque probabilité *a priori* $p(I_t|c_k)$ de l'attribut concerné par $(\frac{1}{1-p(I_s|c_k)})$. Ainsi les probabilités sont augmentées proportionnellement à leurs valeurs originelles. Cette manière d'ajustement est raisonnable grâce à sa considération des probabilités originelles (distribution originelle). D'autre choix d'ajustement sont également possibles comme celui qui consiste à remplacer les valeurs probabilités originelles par des valeurs proportionnelles aux longueurs des intervalles. Ce choix suppose une distribution uniforme des valeurs de l'attribut en question.

Un choix aléatoire d'une forme de transition parmi les précédentes est adopté pour l'implémentation de nos algorithmes RT et RS.

6.6.2 Gestion de la liste taboue pour la RT

D'une manière similaire au cas des arbres de décision (voir section 5.6.2), nous proposons une stratégie de gestion de la liste taboue basée sur l'élimination de la transition inverse [Hansen, 1986]. Elle vise à interdire les transitions inverses des transitions récemment effectuées. Pour ce faire, nous représentons une transition par le couple (*identificateur_intervalle*, *type_transition*), où *identificateur_intervalle* est un attribut qui désigne d'une manière unique chaque intervalle appartenant au modèle ou au dépôt et *type_transition* distingue entre cinq types de transition (*ajout*, *suppression*, *inter-changement de probabilités*, *permutation de probabilités* et *changement arbitraire de probabilités*). Notre liste taboue des couples représentant les transitions interdites est une file (FIFO) de longueur ℓ. Après chaque transition, un couple représentant sa transition inverse est ajouté en queue de la liste et un autre représentant une transition dont la transition inverse était taboue durant les ℓ dernières itérations est supprimé de la tête de liste. Le couple représentant une transition inverse est déduit à partir de celui de la transition effectuée

en gardant le même identificateur d'intervalle (*identificateur_intervalle*) et en déduisant le type de la transition inverse à partir de la correspondance établie dans le tableau 6.2 :

Type de transition	**Type de transition inverse**
ajout	suppression
suppression	ajout
inter changement de probabilités(I_i,I_j)	inter changement de probabilités(I_i,I_j)
permutation de probabilités	permutation inverse de probabilités
changement arbitraire de probabilités	changement arbitraire de probabilités

TAB. 6.2 – *Correspondance entre type de transition et type de transition inverse dans le cas d'un CB*

6.7 Conclusion

Une autre application de notre approche CAMP aux classificateurs bayésiens a été décrite dans ce chapitre. L'utilisation les métaheuristiques AG, RS et RT comme techniques de mise en oeuvre de l'approche CAMP nous a permis de profiter de la richesse des classificateurs bayésiens pour mieux implanter les mécanismes de combinaison et d'adaptation. En effet, la définition des principaux concepts des trois algorithmes a été fondée sur la manipulation des probabilités *a priori*. Nous avons procédé de la même manière que dans le chapitre 5, en définissant d'une part les éléments communs aux trois algorithmes, à savoir le codage d'un classificateur bayésien, sa fonction d'évaluation (fonction objectif) et la fonction de transition (commune à RS et à RT) et d'autre part les éléments spécifiques à chacun des algorithmes, à savoir le croisement et la mutation pour l'AG et la gestion de la liste taboue pour la RT. Les choix des principaux paramètres et critères de contrôle des différents algorithmes comme les conditions d'arrêts, les probabilités respectives de croisement et de mutation et la taille de génération dans l'AG, la probabilité d'acceptation des transitions et la stratégie de décroissance de la température dans le RS et la longueur de la liste taboue dans la RT, sont discutés dans le chapitre 7.

Durant l'exécution de l'approche CAMP aux classificateur bayésiens, et afin de préserver les propriétés de ce type de modèles, nous avons choisi certaines stratégies qui concerne essentiellement la distribution des probabilités *a priori* des CB. Ces choix d'ajustement de probabilités nous ont semblé

raisonnables. Cependant, plusieurs autres manières de préserver la propriété de distribution sont également possibles et méritent d'être étudiées.

Les modèles résultant de l'application des trois techniques d'optimisation dans la mise en oeuvre de CAMP au cas des classificateurs bayésiens, sont constitués d'un grand nombre d'intervalles. En effet, plus nous utilisons de manière optimale les capacités de ces précédents algorithmes (en augmentant, par exemple, leurs nombres d'itérations), plus les attributs des CB sont morcelés, c'est-à-dire composés d'un grand nombre d'intervalles. La prise en considération de la taille (nombre d'intervalles) comme un deuxième critère d'évaluation de modèle avec de la capacité prédictive peut réduire la complexité du CB résultant et simplifier son inférence. Il s'agira ainsi de résoudre un problème d'optimisation multi-critères.

Chapitre 7

Évaluation expérimentale de l'approche CAMP

7.1 Introduction

Le but de ce chapitre est l'évaluation empirique de l'approche CAMP. Deux expérimentations sont menées en parallèle afin de valider les résultats de l'application de CAMP, respectivement, sur les arbres de décision et sur les classificateurs bayésiens. Pour chacune des expérimentations, nous avons besoin, d'une part, d'un ensemble de modèles qui prédisent un même facteur de qualité d'un produit logiciel et, d'autre part, d'un ensemble d'observations qui décrivent un contexte d'organisation particulière et pour lesquelles les valeurs du facteur de la qualité en question sont *a priori* connues.

Le facteur de la qualité choisi pour la validation des résultats de notre approche est celui de la stabilité de classes dans un logiciel orienté objets (OO). Nous réduisons la définition de la stabilité d'une classe d'un logiciel à la capacité de cette dernière à évoluer tout en préservant son interface. Ce choix est justifié par le fait que la stabilité ainsi définie est relativement facile à mesurer d'une manière objective et sans avoir recours à une période opérationnelle du logiciel évalué (voir section 7.3).

Un environnement semi-réel est crée afin de fournir les deux entrants de notre approche qui sont les modèles de stabilité de classes OO et les données du contexte (classes OO dont la stabilité est déterminée *a priori*). En tenant compte des spécifications précédentes, ce chapitre est organisé de la manière qui va suivre. D'abord, dans la section 7.2, nous formulons les hypothèses de notre évaluation de CAMP,

en fonction desquelles nous allons concevoir nos expérimentations. Dans la section 7.3, nous proposons notre perception de la prédiction de la stabilité des classes dans un logiciel OO. Par la suite, dans la section 7.4, nous décrivons les étapes de construction des modèles de stabilité de types arbres de décision et classificateurs bayésiens. Dans la section 7.5, nous présentons le contexte d'organisation choisi pour les expérimentations, qui guidera la progression du processus de notre approche CAMP. La conception et le plan détaillé des expérimentations sont décrits dans la section 7.6. Outre la vérification des hypothèses, un autre défi de ce chapitre est la détermination de meilleurs valeurs de paramètres des algorithmes métaheuristiques qui implémentent notre approche. Une bonne configuration de ces algorithmes est primordiale pour renforcer la validité des résultats de CAMP. Pour cette raison, nous consacrons la section 7.7 à la discussion de la configuration des AG, RS et RT utilisés dans CAMP. L'analyse et l'interprétation des résultats sont présentées dans la section 7.8. Avant de conclure cette évaluation empirique de CAMP dans la section 7.11, une discussion portant sur la validité des expériences et des résultats de CAMP est présentée dans la section 7.10.

7.2 Formulation des hypothèses de l'évaluation empirique de CAMP

Dans le but de répondre à des questions qui portent sur l'évaluation de notre approche CAMP, trois catégories d'hypothèses sont à vérifier dans l'environnement expérimental que nous allons créer. Cet environnement est constitué de deux types de modèles de stabilité et un contexte d'organisation.

La première catégorie suppose que les objectifs de l'approche CAMP sont atteignables, à savoir, la meilleure performance des modèles résultants, leur bonne capacité d'explication, ou « interprétabilité », et leur bonne capacité de généralisation. La bonne « interprétabilité », est une propriété garantie par le mécanisme de la décomposition des modèles en expertises, sa considération est d'ailleurs un choix stratégique dans l'approche CAMP. Cependant, il faut vérifier la réalisation des deux autres objectifs. Par conséquent, les deux hypothèses suivantes sont formulées :

- **hypothèse H1** : la capacité prédictive du modèle obtenu en un temps raisonnable par l'approche CAMP est équivalente à celle du meilleur modèle existant.
- **hypothèse H2** : le modèle résultant de l'approche CAMP garde une bonne capacité prédictive lors de l'évolution du contexte d'organisation.

La deuxième catégorie d'hypothèses vise à comparer notre approche à d'autre alternatives d'amélioration de la capacité prédictive des modèles comme la « combinaison des ensembles de données » et la « combinaison de prédictions » [1]. Cette catégorie comporte les hypothèses suivantes :

- **hypothèse H3** : la capacité prédictive du modèle résultant de l'approche CAMP est supérieure à celle du modèle résultant de la combinaison des ensembles de données qui ont servi à la construction des modèles existants.
- **hypothèse H4** : la capacité prédictive du modèle résultant de l'approche CAMP est comparable à celle du modèle résultant de la « combinaison des prédictions » des modèles existants.

La troisième catégorie d'hypothèses se charge d'identifier certains facteurs qui pourraient influencer la performance du modèle résultant de l'approche CAMP. Cette catégorie comporte les hypothèses suivantes :

- **hypothèse H5** : il y a une corrélation assez forte entre le nombre des modèles existants et la capacité prédictive du modèle résultant de l'approche CAMP.
- **hypothèse H6** : un modèle existant de faible capacité prédictive peut participer à l'amélioration du modèle résultant de l'approche CAMP, car il peut renfermer ce que nous appelons des *«poches de bonnes expertises»*.

7.3 La stabilité

Durant sa phase opérationnelle, un logiciel subit divers changements dictés par plusieurs raisons. Nous citons, entre autres, la détection des erreurs, l'évolution des besoins et le changement de l'environnement. Au fur et à mesure que le nombre de changements augmente, le comportement du logiciel se détériore. En particulier sa capacité à absorber de nouveaux changements s'affaiblit et la récession de sa qualité peut aller jusqu'à ce qu'il devient imprévisible [Parnas, 1994]. Il est indispensable de garder, après des modifications vitales et incontournables, un logiciel qui résiste aux effets négatifs du changement. Il s'agit donc de s'occuper de la stabilité du logiciel.

Deux alternatives sont envisageables pour accomplir cette tâche. La première consiste à intégrer des règles garantissant la stabilité dans la phase de la conception, comme proposé dans [Martin, 1997].

1. C'est la combinaison des sorties des modèles.

Cette alternative est supportée par plusieurs chercheurs qui s'intéressent à la stabilité du logiciel (voir section 7.3.1). La deuxième consiste à évaluer et à prédire la stabilité en utilisant des modèles de prédiction pour décider, après un certain nombre de versions, si une restructuration majeure du logiciel est nécessaire pour éviter les effets négatifs de l'évolution des besoins futurs. Sur cette dernière alternative est fondée notre initiative d'améliorer la stabilité. Par conséquent, nous proposons dans la section 7.3.2 une nouvelle manière d'évaluer la stabilité des classes dans un logiciel OO.

7.3.1 Stabilité du logiciel dans la littérature

Le sujet principal traité lors d'un atelier dans une grande conférence sur la programmation OO (OOPSLA)[2], était comment réaliser ou aboutir à la stabilité du logiciel ? Ce sujet a été réduit, après discussion, à la question suivante : « quelles sont les relations entre un logiciel qui a été stable pour une bonne période du temps et son architecture? » La réponse à cette question est venue sous formes de recommandations et de règles sans méthode effective d'évaluation de la stabilité, voici les plus importantes :

- les architectes de logiciels doivent travailler à réduire le nombre d'interfaces entre les composants d'un logiciel afin de maintenir une flexibilité suffisante, dans l'espoir de satisfaire par une telle architecture les besoins futurs [Pekarek, 1999] ;
- trois concepts OO utilisés dans la conception et le développement de logiciels : les classes abstraites, l'héritage et le polymorphisme sont critiques pour la robustesse et la stabilité d'un logiciel [Panthaki et Sahu, 1999] ;
- si un système est assez flexible pour suivre les buts d'une organisation et changer facilement selon les nouveaux besoins qui émergent, alors il peut être considéré stable [Feathers, 1999] ;
- pour réaliser une architecture facilement changeable, les développeurs doivent appliquer les deux principes suivants :
  1. Un composant devrait seulement dépendre des composants qui sont plus stables que lui, c'est *le principe des dépendances stables* (*the Stable Dependencies Principle*).
  2. Les composants qui sont les plus stables devraient être les plus abstraits, c'est *le principe*

2. Object-Oriented Programming, Systems, Languages and Applications.

des abstractions stables (*the Stable Abstractions Principle*).

Les composants instables devraient être concrets et l'abstraction d'un composant devrait être proportionnelle à sa stabilité [Martin, 1997].

La conclusion que nous pouvons tirer de ces réponses suggère que la stabilité du logiciel est affectée par plusieurs attributs liés à la flexibilité, à la complexité, à la visibilité, aux dépendances entre composants, à l'abstraction et à la rigueur d'emploi des concepts OO.

7.3.2 Définition et évaluation de la stabilité

Le paradigme orienté objet est doté de plusieurs principes qui favorisent certains facteurs de la qualité du logiciel, entre autres, le principe de la conception OO recommandant « qu'un module doit être ouvert pour les extensions mais fermé pour les modifications ». Un module peut être une classe, un ensemble de classes, ou un *framework* entier. Pour qu'il soit stable, un module doit être fermé aux modifications, au moins au niveau de son interface, cependant, il doit être ouvert pour l'extension, de sorte qu'il puisse être adapté à de nouveaux besoins émergents. Ce principe est connu sous le nom du principe « ouvert-fermé » de la conception OO (The open-closed principle of object-oriented design) [Panthaki et Sahu, 1999].

Notre évaluation de la stabilité est fondée sur ce principe. Nous nous intéressons à la stabilité d'un logiciel OO à travers l'évaluation de la stabilité de ses classes. Nous considérons qu'une classe est stable si son interface demeure inchangé entre versions. Ainsi, le taux de modification d'un interface indiquera le degré d'instabilité de la classe correspondante.

Définition 7.3.1 *Soit $C\ell_i$ une classe dans un système logiciel OO dans sa version i et soit $I(C\ell_i)$ l'interface de $C\ell_i$ (méthodes publiques locales ou héritées). Nous définissons un niveau de stabilité noté NS de $C\ell_i$ qui peut être mesuré en comparant $I(C\ell_i)$ à $I(C\ell_{i+1})$ (interface la version suivante $i+1$). le niveau de stabilité est donné sous forme d'un pourcentage de conservation des éléments de $I(C\ell_i)$ dans $I(C\ell_{i+1})$. formellement le niveau de stabilité de $C\ell_i$ est défini par :*

$$NS(C\ell_i \longrightarrow C\ell_{i+1}) = \frac{\#(I(C\ell_i) \cap I(C\ell_{i+1}))}{\#I(C\ell_i)}.$$

7.3.3 Prédiction de la stabilité

À partir des règles et des recommandations recensées dans la littérature et résumées dans la section 7.3.1, nous faisons l'hypothèse que la stabilité d'une classe dépend d'une part, de sa conception (sa structure), d'autre part, d'un facteur de *stress* qui est dû aux implantations de nouveaux besoins. Ce stress mesure les effets, à la fois, des modifications directes (ajout de nouvelles propriétés locales) et des modifications indirectes (subies par les autres classes du système), sur la stabilité de la classe considérée. Un modèle de prédiction de la stabilité d'une classe $C\ell_i$ prend la forme d'une fonction f qui reçoit en entrée un ensemble de métriques structurelles $(m_1(C\ell_i),\ldots,m_{d-1}(C\ell_i))$ et une estimation du stress notée $Str(C\ell_i \longrightarrow C\ell_{i+1})$ et produit une prédiction du niveau de stabilité notée $NSP(C\ell_i \longrightarrow C\ell_{i+1})$ de cette classe définie par:

$$NSP(C\ell_i \longrightarrow C\ell_{i+1}) = f\big(m_1(C\ell_i),\ldots,m_n(C\ell_i),Str(C\ell_i \longrightarrow C\ell_{i+1})\big)$$

Tenir compte de tous les types de modifications dont dépend le stress est une tâche difficile. Par conséquent, pour des raisons pratiques, nous nous contentons d'estimer ce dernier en considérant seulement les modifications locales effectuées sous forme d'ajout de méthodes à l'interface de la classe considérée.

Définition 7.3.2 *Une estimation du stress* $Str(C\ell_i \longrightarrow C\ell_{i+1})$ *est approximée par le pourcentage des méthodes ajoutées dans la classe* $C\ell_i$ *entre deux versions. Formellement,*

$$Str(C\ell_i \longrightarrow C\ell_{i+1}) = \frac{\#(I(C\ell_{i+1}) - I(C\ell_i))}{\#I(C\ell_{i+1})}.$$

7.4 Construction des modèles de la stabilité

Dans cette section, nous décrivons les étapes qui amènent à la construction de modèles de stabilité d'une classe d'un logiciel OO. En premier lieu, nous effectuons une sélection d'un ensemble de métriques à partir desquelles la stabilité sera prédite. En second lieu et après avoir développé ces métriques, nous procédons à la collecte de données à partir d'un ensemble hétérogène de systèmes logiciels. Les données collectées servent à l'entraînement d'algorithmes d'apprentissage utilisés pour construire deux types de modèles de stabilité basés sur les arbres de décision et les classificateurs bayésiens.

7.4.1 Attributs d'entrée des modèles

Conformément à notre perception de la prédiction de la stabilité d'une classe OO, nous avons choisi 18 métriques structurelles de classe qui vont, avec une estimation de stress, servir comme variables d'entrées aux modèles. Les métriques choisies appartiennent à une des quatre catégories suivantes : couplage, cohésion, héritage, et complexité et elles constituent une union de métriques utilisées dans différents modèles théoriques [Abreu, 1993; Chidamber et Kemerer, 1994; Briand et *al.*, 1997; Zuse, 1998; Briand et *al.*, 2000]. La liste de ces métriques regroupées par catégorie est donnée par le tableau 7.1.

7.4.2 Collecte de données

Les programmes d'extraction des métriques choisies ont été développés à l'aide de l'outil ACCESS de l'environnement *DISCOVER*©[3]. *DISCOVER* permet d'analyser du code source pour en créer un modèle d'information comprenant une base de données qui renferme les structures, respectivement, physique et logique du code et également les relations entre les différentes entités. À coté des fonctionnalités usuelles de navigation et de requêtes au modèle ainsi créé, *DISCOVER* offre un langage appelé ACCESS, basé sur TCL[4], pour développer des scripts dans l'objectif d'exécuter des opérations complexes, comme le calcul des métriques. C'est ce langage qui nous a servi pour le développement des extracteurs des différentes métriques utilisées.

Les données (valeurs des métriques) sont extraites à partir des 11 systèmes logiciels donnés par le tableau 7.2. Au moins deux versions majeures de chaque système sont disponibles afin d'évaluer la stabilité des classes en utilisant la définition 7.3.1 et le stress en employant la définition 7.3.2. Chaque classe d'un système logiciel est représentée par une observation regroupant ses 18 valeurs des métriques, sa valeur de stress et sa valeur de stabilité.

7.4.3 Entraînement des modèles

Deux algorithmes d'apprentissage supervisé sont utilisés pour entraîner les modèles. L'algorithme C4.5 [Quinlan, 1993] est employé pour entraîner des classificateurs de type arbres de décision et ROC[5]

3. Voir : http://www.mks.com/product/discover/developers.htm.
4. *Tool Command Language*.
5. *The Robust Bayesian Classifier, version 1.0.*, développé dans le cadre de « *The Bayesian Knowledge Discovery project Robust Bayesian Classifier* ».

Nom	Description[a]	Référence
	Métrique de cohésion	
LCOM	*lack of cohesion methods*	[Chidamber et Kemerer, 1994]
COH	*cohesion*	[Zuse, 1998]
COM	*cohesion metric*	[Zuse, 1998]
COM1	*cohesion metric 1*	inspirée de [Zuse, 1998]
	Métrique de couplage	
OCMAIC	*other class method attribute import coupling*	[Briand et *al.*, 1997]
OCMAEC	*other class method attribute export coupling*	[Briand et *al.*, 1997]
CUB	*number of classes used by a class*	[Zuse, 1998]
	Métrique d'héritage	
NOC	*number of children*	[Chidamber et Kemerer, 1994]
NOP	*number of parents*	[Abreu, 1993]
DIT	*depth of inheritance*	[Chidamber et Kemerer, 1994]
MDS	*message domain size*	[Zuse, 1998]
CHM	*class hierarchy metric*	[Zuse, 1998]
	Métrique de taille et de complexité	
NOM	*number of methods*	[Abreu, 1993]
WMC	*weighted methods per class*	[Chidamber et Kemerer, 1994]
WMCLOC	*LOC weighted methods per class*	inspirée de [Chidamber et Kemerer, 1994]
MCC	*McCabe's complexity weighted meth. per cl.*	[Abreu, 1993]
NPPM	*number of public and protected meth. in a cl.*	inspirée de [Zuse, 1998]
NAA	*number of Available attributes*	[Zuse, 1998]

TAB. 7.1 – 18 *métriques structurelles utilisées pour prédire la stabilité d'une classe OO.*

[a] L'utilisation de la désignation en anglais des métriques s'impose à cause de la familiarité de la communauté avec ces appellations.

[Ramoni et Sebastiani, 1999] est utilisé pour entraîner des classificateurs bayésiens. Nous nous conten-

Système	Nombre de versions (major)	Nombre de classes
Bean browser	6(4)	388–392
Ejbvoyager	8(3)	71–78
Free	9(6)	46–93
Javamapper	2(2)	18–19
Jchempaint	2(2)	84
Jedit	2(2)	464–468
Jetty	6(3)	229–285
Jigsaw	4(3)	846–958
Jlex	4(2)	20–23
Lmjs	2(2)	106
Voji	4(4)	16–39

TAB. 7.2 – *Les systèmes logiciels utilisés pour construire les modèles.*

tons dans ce travail d'étudier des classificateurs binaires[6], avec deux classes *stable* (ou 1) quand $NS = 1$ et *instable* (ou -1) quand $NS < 1$.

Pour imiter l'hétérogénéité des experts réels (modèles existants), chaque modèle a été entraîné en considérant un sous-ensemble différent des métriques extraites à partir d'un système logiciel particulier (hypothèses différentes). Ainsi nous avons formé 15 sous-ensembles à partir des 18 métriques, en combinant un, deux, trois, ou quatre des catégories des métriques (voir section 7.4.1). En explorant toutes les possibilités et en utilisant les 11 ensembles de données provenant respectivement des 11 systèmes logiciels, nous avons obtenu $15 \times 11 = 165$ ensembles de données. Chacun des algorithmes C4.5 et ROC est exécuté sur les 165 ensembles entiers de données. Parmi les modèles entraînés, nous avons gardé 40 arbres de décision en éliminant les classificateurs constants et ceux avec une erreur d'entraînement supérieure à 10 %, quant aux classificateurs bayésiens, nous avons conservé les 40 avec le plus petit biais.

6. Parce que la construction de modèles de qualité n'est pas notre objectif principal.

7.5 Description du contexte d'organisation

Nous proposons dans nos expérimentations d'améliorer la prédiction de la stabilité des classes constituant les APIs standards de Java[7]. L'environnement pour lequel nous effectuons la promotion de la prédiction de la stabilité est un milieu au sein *Sun Microsystems* où les API java sont développés. Selon notre approche CAMP, nous devons guider l'amélioration de la prédiction par un ensemble de propriétés qui représentent les produits et qui reflètent les spécificités de *Sun Microsystems* en matière de développement logiciel, comme la rigueur d'utilisation des concepts OO, l'expérience en matière de développement des classes avec le langage Java, la compétence des développeurs, etc. Nous prétendons abstraire ces propriétés dans un ensemble de données que nous appelons contexte d'organisation et auquel nous donnons le nom : contexte *Sun Microsystems_Java*. Ce contexte est un ensemble de données collectées sur quelques 2920 classes Java, représentant l'expérience de Sun Microsystems à travers quatre versions du Jdk (Jdk1.0, Jdk1.1, Jdk1.2. et Jdk1.3). Pour chacune des classes des trois premières versions, sont calculées les valeurs des 18 métriques, une estimation du stress Str et une évaluation NS de la stabilité. Trois évolutions majeures de Java sont ainsi retracées, respectivement, par les trois transitions suivantes : de Jdk1.0 à Jdk1.1, de Jdk1.1 à Jdk1.2 et de Jdk1.2 à Jdk1.3. Le tableau 7.3 donne une description sommaire du contexte *Sun Microsystems_Java*.

Version JDK	Nombre de classes	classes instables
jdk1.0.2 à 1.1.6	187	42
jdk1.1.6 à 1.2.004	583	217
jdk1.2.004 à 1.3.0	2162	180

TAB. 7.3 – *Les transitions Java constituant le contexte de Sun Microsystems_Java*

7.6 Conception des expérimentations

Une étape de préparation est nécessaire pour mener l'exécution des expérimentations d'évaluation de l'approche CAMP. Cette étape comporte essentiellement l'implémentation des différents algorithmes

7. Voir http://java.sun.com.

AG, RS et RT utilisés dans l'approche CAMP et leurs configurations respectives. Vu l'importance de l'étape de configuration pour les métaheuristiques, nous préférons la traiter séparément dans la section 7.7. Dans cette section, nous donnons seulement une brève description des différentes implémentations développées pour expérimenter l'approche CAMP, ainsi que leurs objectifs respectifs (voir section 7.6.1). Par la suite, nous spécifions les choix respectifs de la méthode et de la fonction d'évaluation des modèles employées dans les différentes expérimentations (voir section 7.6.2 et 7.6.3). Dans la section 7.6.4 nous établissons le plan de ces expérimentations et nous introduisons les notations qui sont utilisées plus tard pour interpréter les résultats.

7.6.1 Implémentations de l'approche CAMP

Afin de valider empiriquement les résultats de l'approche CAMP sur les deux types de modèles étudiés dans ce travail (arbres de décision et classificateurs bayésiens) en utilisant les trois techniques métaheuristiques AG, RS et RT, nous développons six algorithmes.

Désignation de l'algorithme	Objectif de l'algorithme
CAMP.AG.AD	Implémenter l'application de l'approche CAMP aux arbres de décision en utilisant les algorithmes génétiques.
CAMP.RS.AD	Implémenter l'application de l'approche CAMP aux arbres de décision en utilisant le recuit simulé.
CAMP.RT.AD	Implémenter l'application de l'approche CAMP aux arbres de décision en utilisant la recherche avec tabous.
CAMP.AG.CB	Implémenter l'application de l'approche CAMP aux classificateurs bayésiens en utilisant les algorithmes génétiques.
CAMP.RS.CB	Implémenter l'application de l'approche CAMP aux classificateurs bayésiens en utilisant le recuit simulé.
CAMP.RT.CB	Implémenter l'application de l'approche CAMP aux classificateurs bayésiens en utilisant la recherche avec tabous.

TAB. 7.4 – *Les différentes implémentations de l'approche CAMP*

La désignation ainsi que l'objectif de chacun des algorithmes sont donnés par le tableau 7.4.

7.6.2 Méthode d'évaluation

Sauf mention contraire, nous utilisons deux méthodes pour estimer avec précision la capacité prédictive des modèles et choisir éventuellement les meilleurs paramètres des algorithmes AG, RS et RT. La méthode de validation croisée de Monte Carlo (voir la section 2.2.5.3) est employée afin de choisir les bonnes configurations des différents algorithmes, la méthode de la validation croisée *10-fold* (voir la section 2.2.5.1) est utilisée dans la vérification des hypothèses principales de notre validation empirique des résultats de CAMP. Nous rappelons que l'ensemble de données d'évaluation est formée par les observations qui constituent le contexte d'organisation *Sun Microsystems_Java*. Dans le cas de la méthode de validation de Monte Carlo, l'ensemble d'entraînement est formé de 1946 observations et celui de test en contient 973, tandis que dans le cas de la validation croisée *10-fold* chaque ensemble d'entraînement est formé de 2628 observations et celui de test contient 292 observations.

7.6.3 Fonction d'évaluation

À partir de la discussion menée dans la section 4.8.2.3 sur le choix de la fonction d'évaluation de la capacité prédictive de modèles, nous avons étudié la distribution des classes stables et instables dans 15 transitions des systèmes logiciels disponibles de tailles allant de 16 à 2337 classes (voir les tableaux 7.2 et 7.3). La figure 7.1 montre les proportions respectives des classes stables et des classes instables dans un ensemble de 15 transitions. Un déséquilibre entre les deux proportions est très visible dans la plupart des systèmes, il est caractérisé par une probabilité moyenne élevée, de 85% de classes stables contre seulement (15%) de classes instables. En conséquence, nous avons décidé d'utiliser la fonction d'évaluation fondée sur l'indice J de Youden et que nous notons JT (voir l'équation 4.7).

7.6.4 Plan des expérimentations

Dans l'objectif de vérifier les hypothèses formulées dans la section 7.2, des exécutions des différents algorithmes qui implémentent l'approche CAMP sont effectuées dans le cas des arbres de décision et dans le cas des classificateurs bayésiens. Chaque hypothèse est le sujet d'une vérification. Tenant compte des objectifs principaux de notre approche (voir section 4.4), les deux premières hypothèses

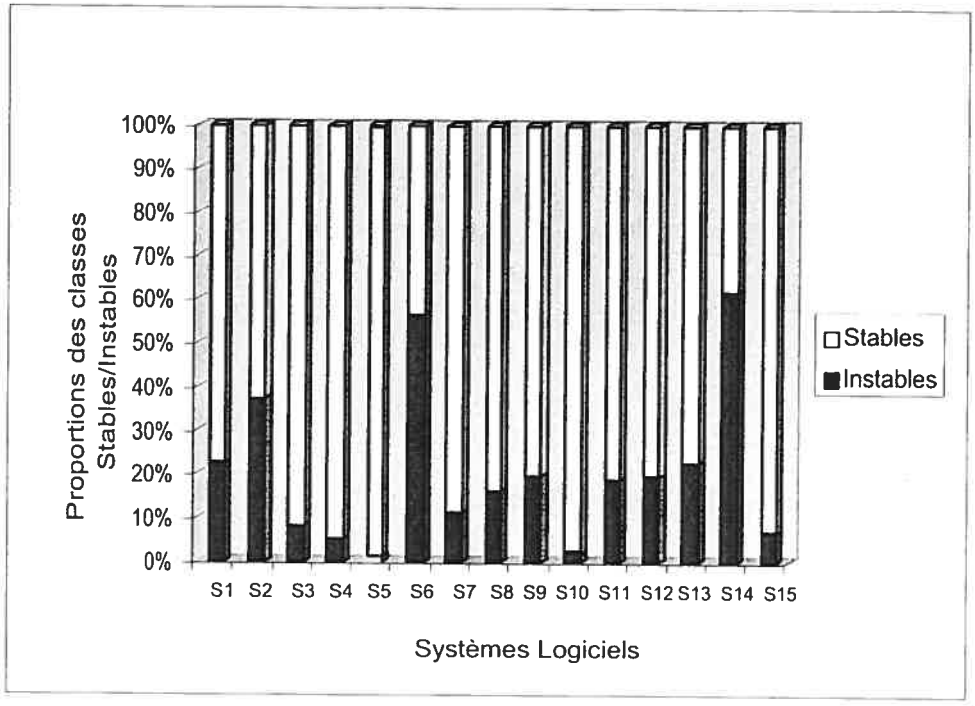


FIG. 7.1 – *Proportions non équilibrées des classes stables et instables.*

principales $H1$ et $H2$ sont vérifiées en utilisant tous les algorithmes, tandis que nous nous contentons, pour la vérification des autres hypothèses, de l'utilisation d'un algorithme par type de modèles. Chaque algorithme, sauf mention particulière, utilise d'une part, 40 modèles construits pour simuler les experts existants et d'autre part, le contexte d'organisation *Sun Microsystems_Java* constitué de quelques 2920 classes. Le tableau 7.5 présente le planning des différentes expériences qui comportent, pour chaque hypothèse à vérifier, les algorithmes utilisés et une description des principales manipulations effectuées.

Hypothèse à vérifier	Algorithmes utilisés	Description de l'expérience et objectif
H1	CAMP.AG.AD CAMP.RT.AD CAMP.RS.AD CAMP.AG.CB CAMP.RT.CB CAMP.RS.CB	Aucune manipulation particulière n'est effectuée sur les entrées ou sur les paramètres de l'algorithme durant son exécution. L'objectif de cette expérience est de comparer, pour chaque algorithme, la capacité prédictive du modèle résultant à celle du meilleur modèle existant.
H2	CAMP.AG.AD CAMP.RT.AD CAMP.RS.AD CAMP.AG.CB CAMP.RT.CB CAMP.RS.CB	Nous allons appliquer le modèle résultant de CAMP sur un ensemble « d'application » (ou test) et sur différentes évolutions de celui-ci. Notre contexte est formé des observations présentant les classes des deux premières versions du Jdk (Jdk1.0 et Jdk1.2), l'ensemble d'application est représente des classes choisies aléatoirement dans la version Jdk1.3 et sa taille vaut $1/3$ de la taille du contexte. L'évolution de ce contexte est simulée par 5 variations de l'ensemble d'application (de 10%, de 20%, de 30%, de 40% et de 50% de sa taille initiale), en ajoutant à chaque fois un nombre de nouvelles observations provenant du Jdk1.3. L'objectif de cette expérience est de retracer la variation de la capacité prédictive du modèle résultant lorsque il est appliqué aux différents ensembles représentant l'évolution du contexte.
H3	CAMP.AG.AD CAMP.AG.CB	Tous les échantillons de données qui ont servi à la construction des modèles « existants » sont combinés dans un même ensemble. Par la suite, deux modèles sont entraînés sur cet ensemble en utilisant les algorithmes d'apprentissage C4.5 et ROC. L'objectif de cette expérience est de comparer pour chaque type de modèles la capacité prédictive du modèle construit à partir de la combinaison de données à celle du modèle résultant de l'approche CAMP.
H4	CAMP.AG.AD CAMP.AG.CB	Les prédictions (les sorties) des modèles existants sont combinées en utilisant un algorithme de la famille *ensembles de vote*, connu sous le nom de *AdaBoost* (voir section 4.6). L'objectif de cette expérience est de comparer la capacité prédictive du modèle dérivé de l'application de l'approche CAMP à celle du modèle produit par *AdaBoost*.
H5 et H6	CAMP.AG.AD CAMP.RS.CB	Nous varions le nombre de modèles existants en entrée des algorithmes. L'objectif de cette expérience est de retracer, en changeant le nombre de modèles existants, la variation de la capacité prédictive du modèle résultant de l'approche CAMP.

TAB. 7.5 – *Plan des expériences.*

7.6.5 Notations

Le tableau 7.6 présente les notations utilisées pour exposer et interpréter les résultats des expériences dans la suite de ce chapitre.

Notation	Signification
$f_{Meilleur.AD}(E)$	Meilleur modèle de type arbre de décision sur l'ensemble de données E
$f_{CAMP.AG.AD}$	Modèle dérivé par l'algorithme CAMP.AG.AD
$f_{CAMP.RS.AD}$	Modèle dérivé par l'algorithme CAMP.RS.AD
$f_{CAMP.RT.AD}$	Modèle dérivé par l'algorithme CAMP.RT.AD
$f_{Meilleur.CB}(E)$	Meilleur modèle de type classificateur bayésien sur l'ensemble de données E
$f_{CAMP.AG.CB}$	Modèle dérivé par l'algorithme CAMP.AG.CB
$f_{CAMP.RS.CB}$	Modèle dérivé par l'algorithme CAMP.RS.CB
$f_{CAMP.RT.CB}$	Modèle dérivé par l'algorithme CAMP.RT.CB
$f_{DON.AD}$	Modèle de type arbre de décision entraîné sur la réunion de tous les ensembles de données
$f_{DON.CB}$	Modèle de type classificateur bayésien entraîné sur la réunion de tous les ensembles de données
$f_{AdaBoost.AD}$	Modèle dérivé par combinaison de prédictions en utilisant *AdaBoost*
$f_{CAMP.*.*}$	tout modèle dérivé par l'approche CAMP
$f_{Meilleur.*}$	$f_{Meilleur.AD}$ ou $f_{Meilleur.CB}$

TAB. 7.6 – *Notations*

7.7 Configuration algorithmique

Étant donné les caractères heuristiques des algorithmes AG, RS et RT, le choix de leurs configurations se fait d'une manière empirique. Pour chaque algorithme, plusieurs exécutions sont effectuées afin d'affecter les meilleures valeurs à ses différents paramètres. Dans cette section nous donnons nos choix de valeurs des paramètres, les résultats importants d'exécutions sont présentées dans l'annexe C.

7.7.1 Configuration de l'algorithme génétique

Dans le cas des arbres de décision (CAMP.AG.AD) ou dans celui des classificateurs bayésiens (CAMP.AG.CB), la population initiale de notre algorithme génétique est formée des modèles existants. Après plusieurs variations des paramètres de l'AG, la stratégie d'élitisme a été utilisée comme moyen de promotion des bons modèles. Ainsi, à chaque itération, la population entière est remplacée à l'exception d'un petit nombre Ne de bons chromosomes. Afin d'obtenir un compromis entre une bonne qualité de la solution (modèle) et un temps d'exécution raisonnable, le nombre de générations T a été fixé à 100 (critère d'arrêt) et le nombre maximum des chromosomes dans une génération a été fixé à 160. Les deux probabilités p_c (probabilité de croisement) et p_m (probabilité de mutation), a (la proportion du sous-ensemble aléatoire de gènes utilisés dans l'opération de croisement) et Ne (nombre d'élites) évoluent en fonction du nombre t de générations déjà complétées. Le tableau 7.7 décrit le changement de ces paramètres de l'AG pour les deux types des modèles. Le paramètre a est noté a_{AD} dans le cas des arbres de décision et il exprime une proportion de d-rectangles ajoutée à un modèle parent lors d'un croisement (voir la section 5.5.3). Il est noté a_{CB} dans le cas des classificateurs bayésiens et il représente deux proportions a_{att} et a_{int} qui indiquent la proportion d'attributs et la proportion des intervalles par attribut utilisés dans une opération de croisement (voir section 6.5.3). La méthode de sélection utilisée est celle de la roulette de casino car la *fitness* des chromosomes ne varie par excessivement pour les deux types de modèles (voir section2.3.2.7).

t	0–10	11–30	31–99
N_e	3	5	10
p_c	0,65	0,65	0,60
p_m	0,02	0,03	0,05
a_{AD}	0,3	0,1	0,05
$a_{CB} : (a_{att}; a_{int})$	(03;0,5)	(0,1;0,3)	(0,05;0,2)

TAB. 7.7 – *Paramètres de l'AG.*

7.7.2 Configuration de la recherche avec Tabous

Comme il a été mentionné dans le chapitre 2, l'algorithme de recherche avec tabous dépend de la longueur ℓ de la liste taboue et du nombre maximal d'itérations sans amélioration de la performance du modèle résultant *Maxiter*.

Par ailleurs, étant donné le cas des classificateurs bayésiens, un modèle peut avoir un grand nombre de bon voisins (une taille par défaut très grande de l'ensemble BONVOISINS, voir l'allure de l'algorithme de RT dans la section 2.3.4.1). Par conséquent, la taille *TBonVoisins* du bon voisinage BONVOISINS doit être gérée et considérée comme paramètre de notre RT. Puisqu'une transition utilise un dépôt d'expertises (d-rectangles ou intervalles), nous allons exprimer *TBonVoisins* en fonction de la taille de ce dépôt, désigné par *TailleDepotDrectnagles* dans le cas des arbres de décision et par *TailleDepotIntervalles* dans le cas des classificateurs bayésiens.

Pour obtenir un compromis entre la bonne capacité prédictive du modèle résultant et un temps raisonnable d'exécution de CAMP, plusieurs exécutions des algorithmes CAMP.RT.AD et de CAMP.RT.CB ont été effectuées pour déterminer les meilleures valeurs de ces trois paramètres. Le résultat de notre choix est résumé dans le tableau 7.8.

Algorithme	ℓ	*Maxiter*	*TBonVoisins*
CAMP.RT.AD	3	20	0.5 *TailleDepotDrectnagles*
CAMP.RT.CB	9	50	0,2 *TailleDepotIntervalles*

TAB. 7.8 – *Paramètres de la RT*

Dans les algorithmes CAMP.RT.AD et CAMP.RT.CB, nous profitons de la disponibilité de plusieurs solutions initiales (modèles existants) pour réitérer la RT autant de fois que nous avons de modèles existants en choisissant à chaque fois un nouveau modèle initial s_0 (voir l'allure de l'algorithme de RS dans la section 2.3.3.1). De la sorte, le modèle résultant est le plus performant des modèles dérivés par toutes les itérations utilisant chacune un modèle initial différent.

Cette même idée d'enrichissement de l'espace de recherche est aussi utilisée dans les algorithmes de recuit simulé, CAMP.RS.AD et de CAMP.RS.CB., pour produire un meilleur modèle.

7.7.3 Configuration du recuit simulé

La performance du recuit simulé est également tributaire du choix de ses paramètres, notamment, le choix de la température initiale Trs_0, de la stratégie de recuit (schéma de recuit), de la longueur de la température Nrs et du critère d'acception des transitions (voir la section 2.3.3). Comme nous l'avons fait pour les deux autres algorithmes (AG et RT), nous effectuons plusieurs exécutions pour de déterminer les meilleures valeurs de ces paramètres. Les exécutions montrent que le comportement du RS utilisé dans notre problème dépend principalement de la stratégie de recuit, du critère d'acception de transitions et de la longueur de la température. Cependant l'algorithme est tout à fait indifférent à partir de certains seuils pour certains paramètres. En effet, la qualité du modèle résultant ne semble pas être beaucoup affectée par le choix de la température initiale si cette dernière est suffisamment élevée. Nous l'avons fixé à $10 \times \Delta JT$, où ΔJT est le maximum de la variation de la capacité prédictive des modèles existants. Au dessus de cette valeur, ce paramètre n'influe que sur le temps d'exécution du RS. Nous avons opté pour un schéma de recuit de type géométrique avec $Trs_{k+1} = \alpha.Trs_k$ (voir la section 2.3.3.4) et pour une stratégie d'acceptation de transition basée sur le critère de Glauber (voir la section 2.3.3.6). Puisque la longueur de la température dépend de la fonction de transition et, par conséquent, de la taille du voisinage qu'on peut atteindre avant d'abaisser la température, sa valeur est liée au type des modèles. Les valeurs de α (appelé aussi *vitesse de recuit*) ainsi que celles de la longueur de température sont présentées par le tableau 7.9. Le recuit simulé se termine lorsque la température Trs passe au dessous de 1.

Algorithme	α	*Longueur de temprature*
CAMP.RS.AD	0.925	300
CAMP.RS.CB	0.915	600

TAB. 7.9 – *Paramètres de la RS*

7.7.4 Conclusion sur la configuration algorithmique

Les choix de configuration des différents algorithmes sont animés par la recherche d'un compromis entre la performance du modèle résultant et un temps d'exécution raisonnable. Ils sont justifiés empi-

riquement pour chacun des deux types de modèles. La méthode d'évaluation employée est celle de la validation croisée de Monte Carlo. Les traces détaillées des exécutions sont données dans l'annexe C. La justification théorique de ces choix est hors de la portée de notre travail. Enfin, nous concluons que le choix de plusieurs valeurs de paramètres des métaheuristiques que nous considérons comme paramètres de l'approche CAMP, est étroitement dépendant du type des modèles manipulés.

7.8 Analyse et interprétation des résultats

Dans cette section, notre objectif est de vérifier les hypothèses pour l'évaluation expérimentale de notre approche. Nous utilisons pour chaque hypothèse l'expérience et les algorithmes spécifiés dans le tableau 7.5. Les configurations des différents algorithmes sont déterminées dans la section 7.7. Dans ce qui suit de cette section, nous analysons et interprétons les résultats obtenus pour chaque hypothèse.

7.8.1 Hypothèse H1: la capacité prédictive

Pour estimer précisément la capacité prédictive JT des modèles résultant de CAMP, nous avons utilisé la validation croisée *10-fold* et nous avons calculé d'une part, les valeurs de la moyenne et de l'écart type de JT sur les données d'entraînement et les données de test, et d'autre part, les valeurs de la moyenne du temps mis pour obtenir un modèle résultant. Cette opération est effectuée pour chacun des algorithmes implémentant l'approche CAMP. Le tableau 7.10 montre les résultats.

Modèles	$JT(\%)$ **du modèle (écart type)**		**Temps d'exécution** (min)
	Entraînement	**Test**	
$f_{Meilleur\_AD}$	66,47 (0,51)	66,39 (4,37)	-
$f_{CAMP\_AG\_AD}$	78,29 (2,44)	77,82 (4,99)	14
$f_{CAMP\_RT\_AD}$	82,16 (0,24)	78,35 (2,66)	9
$f_{CAMP\_RS\_AD}$	82,13 (0,27)	78,51 (1,48)	10
$f_{Meilleur\_CB}$	63,15 (0,54)	63,10 (4,67)	-
$f_{CAMP\_AG\_CB}$	78,56 (1,32)	77,63 (2,95)	12
$f_{CAMP\_RT\_CB}$	79,66 (0,86)	77,67 (2,92)	10
$f_{CAMP\_RS\_CB}$	77,62 (1,14)	77,15 (3,37)	11

TAB. 7.10 – *Capacité prédictive des modèles résultant de CAMP.*

Les résultats obtenus montrent une amélioration de la capacité prédictive car les écarts entre les JT des modèles dérivés par CAMP $f_{CAMP.*.*}$ et le meilleur modèle $f_{Meilleur.*}$ sont plus grands que les écarts types. Par exemple, l'écart entre $JT(f_{CAMP.AG.AD}$ et $JT(f_{Meilleur.*})$ (11,43%) est supérieur aux écarts types de $JT(f_{Meilleur.AD})$ (4,37%) et de $JT(f_{CAMP.AG.AD}$ (4,99%).

Dans le cas des arbres de décision, la recherche avec tabous et le recuit simulé produisent des modèles légèrement supérieurs à celui produit par l'algorithme génétique comme le montre la figure 7.2. Ceci peut être expliqué par le fait que la manière de se déplacer dans l'espace de recherche en utilisant la RT ou le RS est caractérisée par un « pas » plus petit quand il s'agit des arbres de décision. En effet, l'opération de combinaison réalisée en utilisant un AG est basée sur l'opérateur le plus probable qui est celui du croisement ; Toutefois, dans le cas des arbres de décision, ce dernier présente concrètement un déplacement à grand « pas » dans l'espace de recherche, c'est-à-dire après un croisement, la composition d'un modèle en d-rectangles subit un changement significatif. En revanche, la combinaison assurée graduellement par des transitions dans la RT ou le RS permet de parcourir minutieusement (avec de « petits pas ») l'espace de recherche et de ne pas manquer des solutions qui peuvent être meilleures et plus proches de l'optimale.

Par contre, dans le cas des classificateurs bayésiens comme le montre la figure 7.3, les résultats du test ne montrent pratiquement pas d'avantages d'un algorithme sur les autres. Les trois algorithmes utilisent des « pas » assez grands pour parcourir l'espace de recherche. Un croisement, une mutation ou une transition cause un changement significatif dans les probabilités du CB courant et en particulier dans ses probabilités *a posteriori*. Par conséquent, la décision de ce dernier change dans un grand nombre de régions de l'espace d'entrée. Ainsi, le CB obtenu après combinaison ou adaptation est significativement différent de CB courant. Par ailleurs, l'analyse de l'écart type (voir tableau 7.10) montre que la dispersion des capacités prédictives (JT) autour de la moyenne est relativement étroite pour tous les algorithmes. Ce résultat prouve que les modèles produits[8] sont stables (malgré le caractère non déterministe des algorithmes) et que l'estimation de la valeur de la capacité prédictive est plus précise. En particulier, pour les arbres de décision, la dispersion de JT est plus étroite lorsque on utilise la RT et le RS. Ceci est dû à la manière de parcourir l'espace de recherche discutée ci-dessus et qui a tendance à rejeter (ne pas visiter) les modèles avec des valeurs de JT relativement aberrantes.

8. Modèles produits au cours des itérations de la validation croisée.

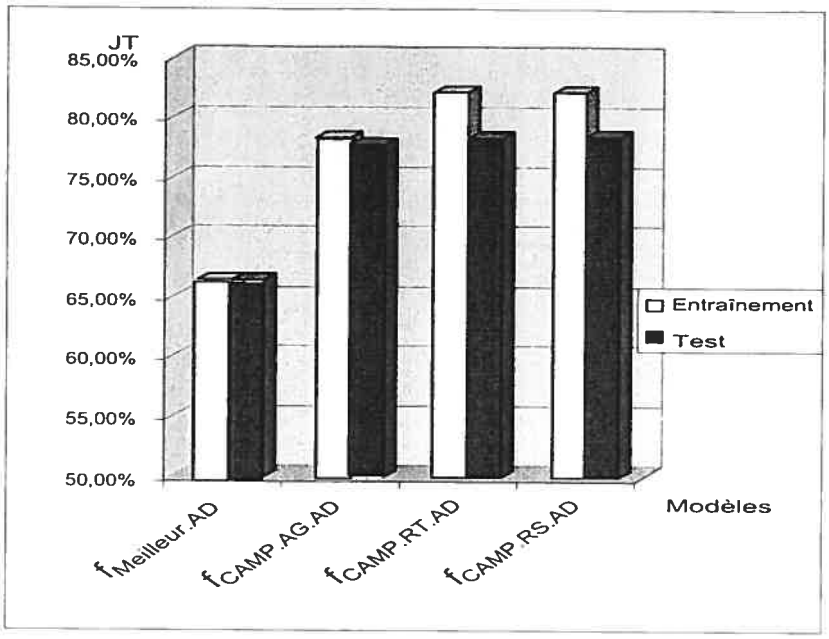


FIG. 7.2 – *Résultats de CAMP sur les arbres de décision*

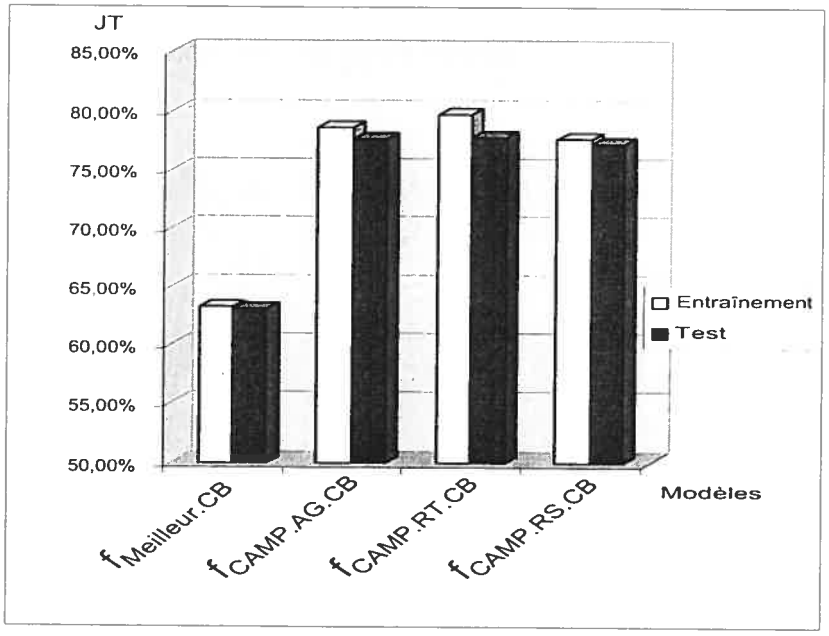


FIG. 7.3 – *Résultats de CAMP sur les classificateurs bayésiens*

Nous notons enfin que la petite différence entre les résultats d'entraînement et ceux du test indiquent qu'il n'y a pas un « *overfitting* » clair et que le temps moyen nécessaire pour l'exécution de l'approche CAMP est effectivement raisonnable pour les différentes algorithmes (une dizaine de minutes pour 19 métriques et 2920 observations comme taille du contexte).

7.8.2 Hypothèse H2: la capacité de généralisation

L'expérience effectuée pour vérifier l'hypothèse H2 est décrite en détail dans le tableau 7.5. Pour mieux présenter les résultats de cette expérience, les ensembles représentant les différentes évolutions du contexte originel (*ContexteOrig*) sont désignés respectivement par *Évolution*0%, *Évolution*10%, *Évolution*20%, *Évolution*30%, *Évolution*40% et *Évolution*50%.

Nous avons appliqué les modèles $f_{Meilleur.AD}(ContexteOrig)$, $f_{CAMP.AG.AD}$, $f_{CAMP.RT.AD}$, $f_{CAMP.RS.AD}$, $f_{Meilleur.CB}(ContexteOrig)$, $f_{CAMP.AG.CB}$, $f_{CAMP.RT.CB}$ et $f_{CAMP.RS.CB}$ sur les ensembles précédents de données. Les résultats sont présentés par le tableau 7.11. Les valeurs de la moyenne et de l'écart type de JT sont calculées pour chaque modèle.

Modèle	**Capacité prédictive $JT(\%)$ du modèle appliqué sur**								
	Contexte Orig	*Évolution 0%*	*Évolution 10%*	*Évolution 20%*	*Évolution 30%*	*Évolution 40%*	*Évolution 50%*	*Écart type JT*	*Moyenne JT*
$f_{Meilleur.AD}$	58,15	60,53	58,41	57,89	61,36	61,22	58,64	1,52	59,67
$f_{CAMP.AG.AD}$	79.65	76,02	72,58	72,65	75,00	74,86	72,02	2.65	74,68
$f_{CAMP.RT.AD}$	83.27	76.02	72,58	72,89	75,23	75,07	74,42	1.37	74,37
$f_{CAMP.RS.AD}$	81.32	82,75	78,74	79,22	80,23	79,94	77,95	1,62	80,02
$f_{Meilleur.CB}$	56.20	74,85	71,32	73,09	70,00	70,75	69,54	6,10	69,39
$f_{CAMP.AG.CB}$	80,75	79,82	80,99	82,25	78,64	79,09	76,95	1,75	79,79
$f_{CAMP.RT.CB}$	80.31	80,41	80,49	81,62	81,14	81,01	77,30	1,41	80,33
$f_{CAMP.RS.CB}$	80.13	82,75	78,74	79,22	80,23	79,94	77,95	1,54	80,16

TAB. 7.11 – *Comportement des modèles résultants face à l'évolution du contexte.*

Deux résultats intéressants peuvent être tirés de ce tableau. Le premier concerne les écarts types faibles qui montrent une dispersion étroite des valeurs de la capacité prédictive JT autour de la moyenne.

Ce résultat est vrai pour tous les modèles produits par CAMP alors que le meilleur modèle $f_{Meilleur.AD}$ présente une dispersion relativement large de ses valeurs de JT. Ceci montre qu'avec une bonne configuration de l'approche CAMP (voir section 7.7), on peut produire des modèles qui ne présentent pas de « *overfitting* » visible et qui sont mieux adaptés à l'évolution du contexte. Le deuxième résultat concerne les bonnes valeurs de la capacité prédictive des modèles produits par CAMP. Ces modèles ont gardé de hautes capacités prédictives. Malgré le taux d'évolution des ensembles de données qui s'élève à 50 %, les modèles de CAMP ont conservé des capacités prédictives nettement supérieures à celles du meilleur modèle existant.

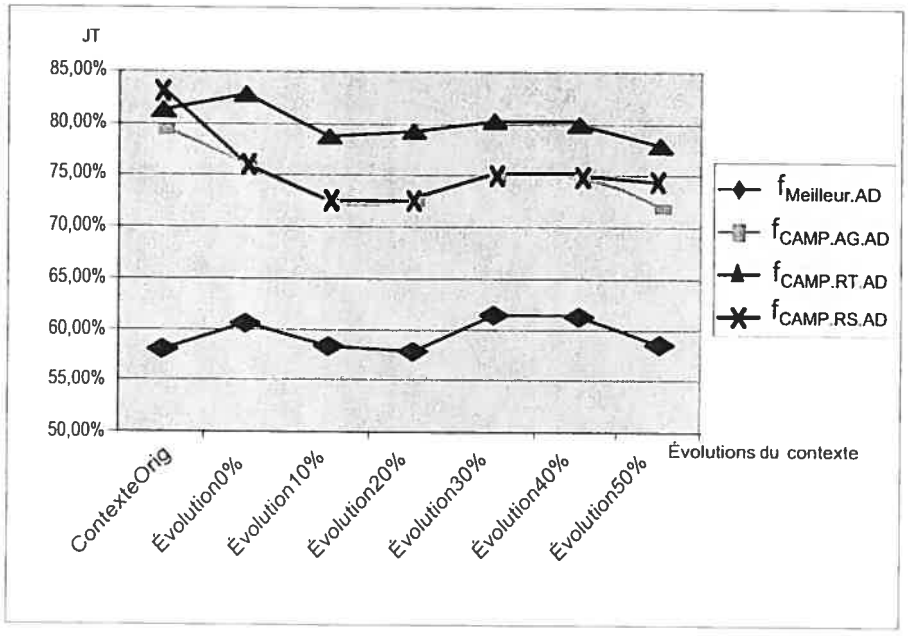


FIG. 7.4 – *Comportement des arbres de décision dérivés par CAMP face à l'évolution du contexte.*

Les figures 7.4 et 7.5 montrent le comportement des modèles résultant vis-à-vis de l'évolution du contexte, respectivement, pour les arbres de décision et les classificateurs bayésiens.

Ces résultats nous permettent de conclure qu'avec l'approche CAMP, on peut produire des modèles ayant des bonnes capacités de généralisation. La bonne qualité des modèles peut être considérée comme le fruit du deuxième principe de notre approche CAMP « Combiner pour généraliser » (voir la section 4.7). Ainsi, la combinaison des expertises qui contiennent des connaissances générales sur le facteur de stabilité est responsable de la bonne capacité de généralisation.

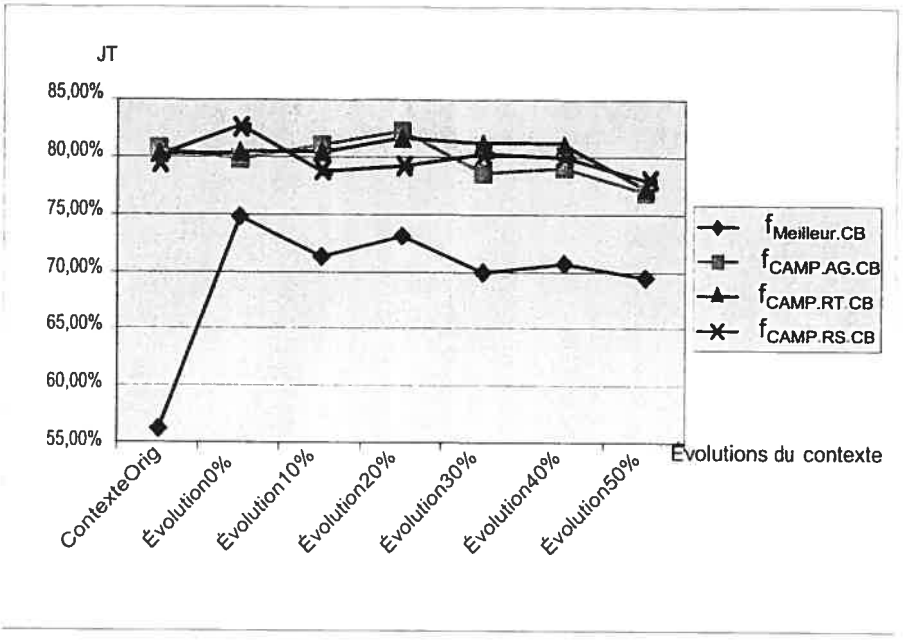


FIG. 7.5 – *Comportement des classificateurs bayésiens dérivés par CAMP face à l'évolution du contexte.*

7.8.3 Hypothèse H3: comparaison entre CAMP et la combinaison des ensembles de données

Afin de mieux comparer les modèles produits par les différentes implémentations de CAMP au modèle entraîné sur l'union de tous les ensembles de données (f_{DON_AD} dans le cas des arbres de décision et f_{DON_CB} dans le cas des classificateurs bayésiens), nous avons utilisé la validation croisée *10-fold* sur le contexte *Sun Microsystems_JAVA* afin d'estimer précisément la capacité prédictive (JT) de chaque modèle. Les valeurs de la moyenne et de l'écart type de JT, sur les données d'entraînement et sur les données de test, sont calculées pour chaque modèle et elles sont présentées dans le tableau 7.12.

Modèles	$JT(\%)$ du modèle (écart type)	
	Entraînement	**Test**
$f_{DON.AD}$	65,24 (0,58)	65,32 (5,10)
$f_{Meilleur.AD}$	66,47 (0,51)	66,39 (4,37)
$f_{CAMP.AG.AD}$	78,29 (2,44)	77,82 (4,99)
$f_{CAMP.RT.AD}$	82,16 (0,24)	78,35 (2,66)
$f_{CAMP.RS.AD}$	82,13 (0,27)	78,51 (1,48)
$f_{DON.CB}$	63,15 (0,54)	63,10 (4,58)
$f_{Meilleur.CB}$	63,15 (0,54)	63,10 (4,67)
$f_{CAMP.AG.CB}$	78,56 (1,32)	77,63 (2,95)
$f_{CAMP.RT.CB}$	79,66 (0,86)	77,67 (2,92)
$f_{CAMP.RS.CB}$	77,62 (1,14)	77,15 (3,37)

TAB. 7.12 – *L'approche CAMP vs. la combinaison de données.*

Ces résultats sont illustrés sur la figure 7.6, qui montre visiblement que tous les algorithmes de l'approche CAMP produisent des modèles nettement plus performant que ceux entraînés sur l'union de tous les ensembles de données qui ont servi à la construction de tous les modèles existants (arbres de décision ou classificateurs bayésiens).

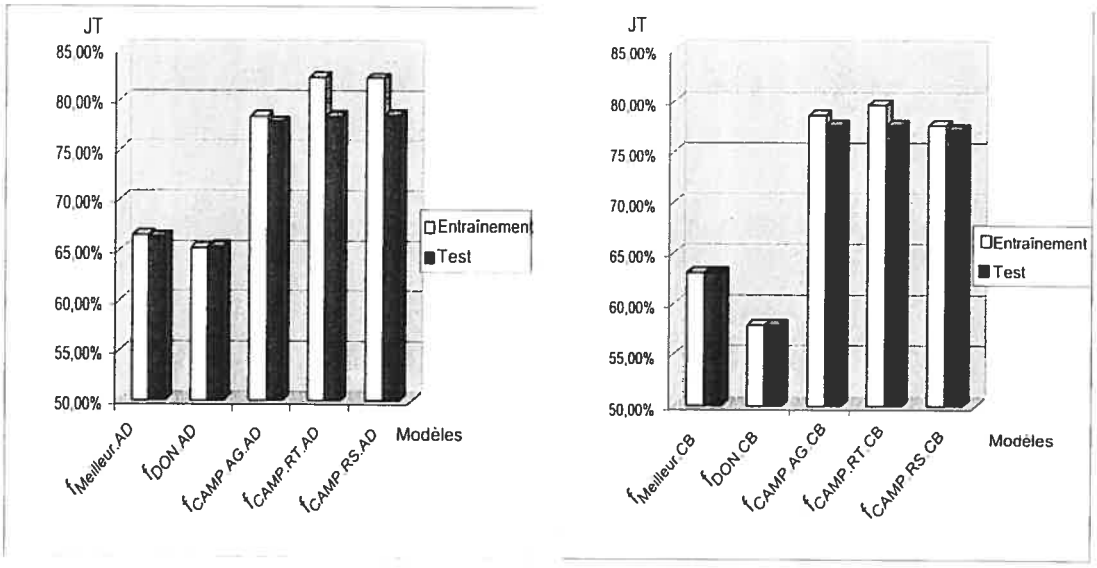


FIG. 7.6 – *Comparaison de l'approche CAMP avec la combinaison de données.*

Cette réalité empirique est expliquée par la réduction de la variance des données après leur combinaison. L'inconvénient de cette manière de combiner les données était une prémisse qui nous a motivé à préférer, dans notre approche, la combinaison des expertises sur celle des données. Plus de détails sur les deux types de combinaison sont discutés dans la section 4.2.4 et dans [Kai et Boon, 1997].

7.8.4 Hypothèse H4: comparaison entre CAMP et la combinaison des prédictions avec *AdaBoost*

Les méthodes de combinaison des prédictions sont les moyens les plus utilisés pour combiner les modèles, cependant, elles ne répondent pas à nos objectifs liés aux exigences du domaine de la qualité de logiciel. Toutefois, nous avons décidé de nous en servir comme repère pour juger la performance de notre approche CAMP. Nous avons effectué une comparaison prématurée entre les résultats d'une première implémentation de l'approche CAMP qui utilise l'AG, et ceux de la combinaison des prédictions avec l'algorithme connu sous le nom *AdaBoost* [Freund et Schapire, 1997].

Dans cette expérience, seulement 23 modèles existants de type arbre de décision ont été combinés et adaptés par la première implémentation CAMP.GA.AD et un contexte d'organisation de 690 observations a été utilisé d'une part pour guider l'approche CAMP, d'autre part pour servir comme un ensemble d'entraînement à *AdaBoost*. Les résultats d'évaluations des deux modèles $f_{AdaBoost.AD}$ et $f_{CAMP.AG.AD}$ résultant de *AdaBoost* et de CAMP sont récapitulés dans le tableau7.13. Ces résultats montrent que $f_{AdaBoost.AD}$ est légèrement plus performant que $f_{MeilleurAD}$ du meilleur expert et que $f_{CAMP.AG.AD}$ est le meilleur des trois.

Modèles	$JT(\%)$ **du modèle (écart type)**	
	Entraînement	**Test**
$f_{MeilleurAD}$	57,97 (0,44)	55,06 (4,50)
$f_{AdaBoost.AD}$	59,41 (0,42)	58,92 (3,84)
$f_{CAMP.AG.AD}$	68,52 (1,13)	65,52 (4,03)

TAB. 7.13 – *Comparaison de l'approche CAMP avec la combinaison des prédictions utilisant AdaBoost.*

7.8.5 Hypothèse H5: la performance de l'approche CAMP et le nombre de modèles existants :

Les résultats montrent une corrélation entre la capacité prédictive d'un modèle résultant de CAMP et le nombre de modèles combinés (existants). Plus précisément, la capacité prédictive du modèle résultant est directement proportionnelle au nombre de modèles existants impliqués dans le processus de l'approche CAMP. Ainsi, les résultats présentés par les courbes 7.7 et 7.8 vérifient l'hypothèse H5 pour les algorithmes CAMP.AG.AD et CAMP.RS.CB. Les valeurs de la corrélation entre la capacité prédictive JT du modèle résultant et le nombre de modèles existants sont calculées pour les deux algorithmes précédents et présentées dans le tableau 7.14.

Modèles	**Corrélation entre JT et le nombre de modèles existants**	
	Entraînement	**Test**
$f_{CAMP.AG.AD}$	0,92	0,78
$f_{CAMP.RS.CB}$	0,86	0,78

TAB. 7.14 – *Corrélation entre la capacité prédictive du modèle résultant et le nombre de modèles existants.*

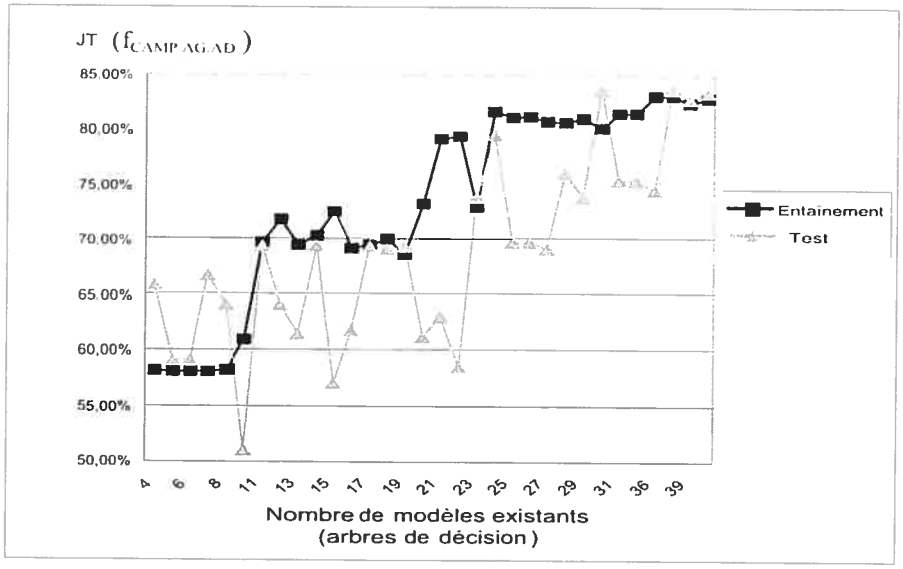
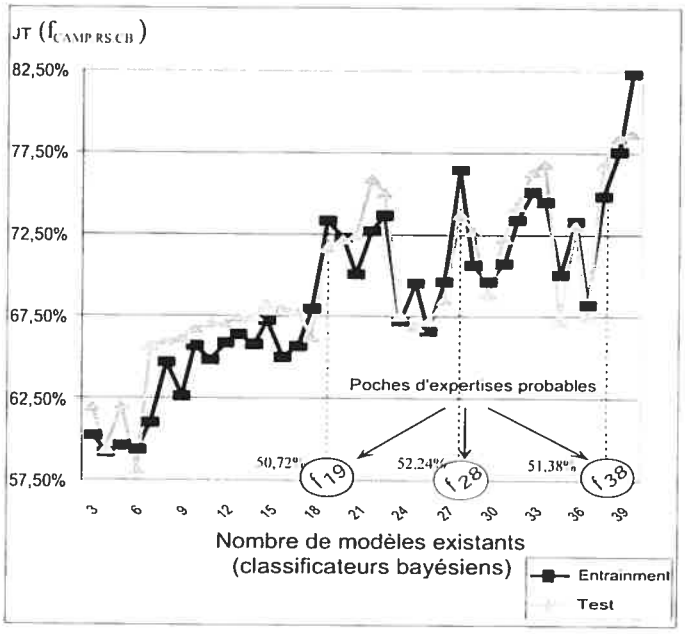


FIG. 7.7 – *Relation entre la capacité prédictive du modèle résultant et le nombre de modèles existants : cas des arbres de décision.*

Par exemple dans le cas de l'algorithme CAMP.AG.AD, les deux variables manifestent une forte corrélation égale à 0,92 lorsque le modèle est évalué sur les données d'entraînement et à 0,78 lorsque il est évalué sur les données de test.

En analysant la variation de JT après chaque implication d'un nouveau modèle dans le processus de l'approche CAMP pour vérifier l'hypothèse H6, nous constatons que le modèle résultant est de plus en plus performant en dépit de la faible capacité prédictive de certains modèles impliqués. Dans la courbe de la figure 7.8, par exemple, même si le $19^{ième}$ modèle impliqué dans le processus CAMP.RS.CB (f_{19}) est d'une faible capacité prédictive égale à 50,72%, nous avons bien distingué l'augmentation de la capacité prédictive du modèle résultant de (6%) comme il le montre la figure 7.8. Le même phénomène se produit aprés l'implication des modèles f_{15}(49,46%), f_{28}(52,24%) et f_{38}(51,38%). Vu la fréquence de ce phénomene nous croyons qu'il est justifié par la possibilité d'avoir de bonnes expertises locales (« poches d'expertises ») enfermées dans des modèles de faibles capacités prédictives.



Modèles existants	f CAMP.RS.CB			
	Entrainement		Entrainement	
f 15	49,46%	47,84%	67,10%	68,11%
f 19	50,72%	55,79%	73,10%	71,51%
f 28	52,24%	48,49%	76,26%	73,53%
f 38	51,38%	49,92%	74,63%	76,63%
f 40	56,84%	57,00%	82,13%	78,51%

FIG. 7.8 – *Variation de la capacité prédictive du modèle résultant en fonction du nombre de modèles existants et détection de poches d'expertises : cas des classificateurs bayésiens.*

7.9 Optimisation de modèles

Dans cette section, nous évaluons l'algorithme d'optimisation des modèles issus de l'application de CAMP aux arbres de décision. Nous rappelons que cet algorithme est proposé et décrit en détails dans la section 5.7.1. Les modèles à aptimiser sont composés de couches de d-rectangles. Étant donné la difficulté des calculs d'intersections de d-rectangles et à cause de la fragmentation excessive de l'espace d'entrée, notre algorithme d'optimisation est complexe. Pour ces raisons et afin d'éviter les effet de la complexité, nous avons choisi d'éprouver notre algorithme sur des modèles produits par la version CAMP.AG.DT de l'approche CAMP effectuant seulement 50 générations. Le tableau 7.15 présentes des résultats très encourageants montrant une diminution significative du nombre de d-rectangles dans les modèles optimisés.

Modèles	**Nombre de d-rectangles**	
	Avant optimisation	**Après optimisation**
*M*1	134	11
*M*2	128	27
*M*3	72	47
*M*4	375	75
*M*5	101	43
*M*6	191	114
*M*7	170	115

TAB. 7.15 – *Résultat de l'algorithme d'optimisation de modèles.*

D'après ce tableau, le taux de réduction du nombre de d-rectangles peut atteindre 90% (*M*1 par exemple, passe de 134 à 11 d-rectangles). Une telle diminution est encourageante bien que notre algorithme peut ne pas avoir d'effet considérable sur certains modèles.

Dans l'objectif de trouver un moyen d'interpréter les modèles de grandes tailles, nous avons recensé les proportions des d-rectangles de différentes classes (*stable* et *instable*). Nous avons trouvé que la proportion de d-rectangles étiquetés *stable* est environ 70%. Ceci nous permet d'exprimer les expertises représentées par les 30% des d-rectangles (*instable*) d'une maniere explicite et le reste (70%) sous forme d'*« expertise par défaut »* (*stable*). Par conséquent, l'interprétation du modèle devient plus simple. Par

exemple, un modèle comportant 100 d-rectangles dont environ 30 étiquetés *instable*, est exprimé par 30 règles explicites (« Si *Condition* Alors *instable*») localisant les régions instables de l'espace d'entrée et une règle par défaut (« Sinon *stable* »). Le remplacement des d-rectangles de la classe majoritaire par une règle (expertise) par défaut permet de mieux simplifier l'expression des expertises du modèle et par conséquent son interprétation.

Avec l'utilisation de notre algorithme d'optimisation suivi d'une simplification en employant « les expertises par défaut », nous allégeons considérablement l'interprétation des modèles.

7.10 Validité des résultats de l'approche CAMP

En vue de nous prononcer sur la validité des résultats de notre approche CAMP, deux volets de la validité inspirés de la littérature sur les études empiriques (comme par exemple [Serrano et *al.*, 2003] et [Perry et Votta, 2000]), sont discutés dans cette section à savoir la validité interne et la validité externe.

7.10.1 Validité interne des résultats de l'approche CAMP

Ce premier volet concerne les menaces internes à la validité qui pourrait empêcher de tirer des conclusions fiables dans le cadre de notre étude spécifique (application de CAMP aux modèles de prédiction de la stabilité de types arbres de décision et classificateur bayésiens). Nous discutons sous ce même volet, les menaces aux validités de conclusion (statistique) et du construit. Notre évaluation de l'approche CAMP à travers cette étude spécifique comporte plusieurs expériences effectuées selon une démarche hypothétiques. La validité de tous les résultats n'est pas exposée à toutes les menaces proposées par Cook [1979]. Certaines menaces, comme l'instrumentation et la fiabilité des mesures, sont liées et concernent toutes les expériences alors que d'autres comme l'ambiguïté ne concernent que certains résultats spécifiques. Par conséquent parmi les menaces de Cook aux validités interne, de conclusion et du construit nous considérons les suivantes.

- **l'instrumentation et la fiabilité des mesures :** cette menace a trait à la validité des mesures utilisées dans nos expériences. Cette validité stipule qu'une mesure est valide si elle mesure ce que l'on cherche à mesurer. Elle concerne précisément toutes les métriques de la qualité, l'évaluation de la stabilité ainsi que la mesure de la capacité prédictive des modèles. Les programmes cal-

culant ces différentes métriques sont conçus d'une manière rigoureuse puis testés sur plusieurs systèmes logiciels de différentes tailles (de 16 à 2700 classes). Le calcul de la capacité prédictive est également testé minutieusement sur une population hétérogène de modèles.

- **le « *fishing* » et le taux de l'erreur :** cette menace concerne notre comportement envers les résultats des expériences. Sous ce même volet de validité s'inscrit l'utilisation de la méthode de la validation croisée pour estimer précisément la capacité prédictive des modèles résultants. À chaque expérience, la moyenne et l'écart types des résultats sont calculés et interprétés. Pour affirmer qu'un résultat de notre approche est supérieur à ceux des autres alternatives, nous vérifions une condition assez sévère sur l'écart type de la capacité prédictive du modèle résultant de CAMP. Cette condition exige que cet écart type doit être inférieur à la différence entre la capacité prédictive du modèle résultant de CAMP et celle du modèle issu de l'alternative étudiée.
- **explication pré-opérationnelle inadéquate de construction :** cette menace concerne la mesure de la capacité prédictive. Cette mesure est bien définie et suffisamment expliquée et discutée (voir section 4.8.2). Avant toute expérience, nous avons décidé l'utilisation de la proportion correcte (*correctness*) comme mesure de la capacité prédictive mais par la suite nous avons découvert que ce choix est inadéquat dans un environnement où les données sont mal distribuées. Un deuxième choix de mesure a été effectué dès les premières expériences après une comparaison entre l'utilisation de la proportion correcte et l'utilisation de l'indice *J* de Youden. Une décision en faveur de ce dernier nous a évité une inadéquation de la mesure de la capacité prédictive. Une justification est donnée dans les sections 4.8.2.3 et 7.6.3.
- **ambiguïté sur la causalité :** cette menace concerne certaines conclusions, en particulier celle qui vérifie l'hypothèse H6. En effet, la question qui peut être posée est : Est-ce que l'amélioration de la capacité prédictive est causée par l'arrivée des nouvelles expertises provenant du modèle nouvellement impliqué dans la combinaison ou grâce à des expertises qui sont déjà impliquées mais qui se manifestent en même temps que l'implication du nouveau modèle ? Cette ambiguïté est résolue par la répétition de l'expérience plusieurs fois et l'observation du même phénomène (le même résultat confirmant la variation significative de la capacité prédictive du modèle résultant suite à l'implication de certains des ces mêmes modèles).

7.10.2 Validité externe des résultats de l'approche CAMP

Le deuxième volet de la validité est celui qui concerne la généralisation des résultats de notre approche. Ce type de validation est une tâche assez délicate car elle comporte plusieurs composantes. Elle concerne la généralisation de l'approche en considérant différents types de modèles de prédiction, différentes tâches de prédiction (stabilité, maintenabilité, fiabilité) et différentes dimensionnalités du problème de prédiction (nombre de métriques utilisées, taille du contexte d'organisation, etc.). Ce volet constitue la validité externe des résultats de l'approche CAMP.

Quoique les modèles traités dans nos expérimentations soient des classificateurs, nous avons appliqué notre approche à deux types différents de modèles. À propos de l'évaluation de l'approche CAMP sur plusieurs tâches de prédiction, cette action est tributaire de la disponibilité et des modèles déjà construits pour différentes tâches de prédiction et des données de contexte d'organisation. Ces deux éléments supposés préalables à l'approche CAMP ne sont pas immédiatement disponibles. Ils nécessitent un travail préparatif comportant la collecte des données du contexte et des modèles publiés souvent sous forme de règles exprimées en langage naturel. Ce travail est relativement ardu pour certains facteurs de la qualité du logiciel qui n'ont pas encore été sujets à un nombre important d'études. En revanche, nous croyons qu'avec la progression de la maturité des programmes de la qualité de logiciel, ce travail deviendra plus facile. Par ailleurs, dans ce chapitre, nous nous sommes plutôt concentrés sur les mécanismes et les résultats de l'approche, c'est pourquoi nous avons conçu un environnement semi-réel sous notre contrôle, qui nous a permis d'expérimenter l'approche CAMP sur la « stabilité de classes ». Nous sommes convaincu que la nature de la tâche de prédiction ne va véritablement pas nuire à la validité des résultats de l'approche CAMP car cet aspect de validité externe peut être aussi vérifié sur d'autres domaines moins contraints que celui de la qualité de logiciel.

En ce qui concerne la dimensionnalité du problème de prédiction, notre environnement nous a permis d'éprouver notre approche sur des modèles utilisant un nombre assez élevé de métriques d'entrée (19) et sur un contexte assez connu (données sur les API standard de Java). Sous ce même volet de la validité externe de l'évaluation de CAMP, s'inscrivent les comparaisons de résultats de ce dernier avec, d'une part, ceux de la combinaison des prédictions et d'autre part, ceux de la combinaison des ensembles de données. Ces comparaisons ont bien montré les résultats meilleurs de l'approche CAMP.

7.11 Conclusion

Deux objectifs sont réalisés dans ce chapitre. Le premier qui est principal, s'intéresse à l'évaluation de l'approche CAMP. Le deuxième, considéré comme secondaire mais important, se rapporte à la bonne configuration des techniques métaheuristiques d'optimisation. Nous nous sommes limités dans la réalisation du deuxième objectif à la détermination des paramètres adéquats à chaque type de modèles. En ce qui concerne l'évaluation empirique de notre approche, nous avons formulé six hypothèses qui ont été par la suite vérifiées avec des résultats très satisfaisants. Cependant, nous ne prétendons pas avoir répondu à toutes les questions sur les caractéristiques et les comportements de notre approche. En effet une question que le lecteur de ces lignes peut se poser est : Quand est ce qu'une technique d'implémentation de CAMP (AG, RS ou RT) est plus adéquate qu'une autre. Il est délicat de donner une réponse finale à cette question. Il nous faudrait utiliser ces techniques afin d'appliquer CAMP à plusieurs types de modèles. Finalement, il est à noter que malgré le peu de place allouée dans ce chapitre à la définition de la stabilité et à la construction des modèles qui la prédisent, ce travail est une contribution en soi au domaine de la qualité.

Chapitre 8

Conclusion et perspectives

Dans ce travail nous avons proposé une approche d'amélioration de la prédiction de la qualité du logiciel en utilisant la combinaison et l'adaptation des modèles prédictifs. C'est une approche qui, à partir d'un ensemble de modèles existants et d'un échantillon reflétant les spécificités d'un environnement particulier (contexte d'organisation), dérive un modèle le plus adapté possible à ce contexte. Cette approche est fondée sur un nombre de principes qui peuvent être résumés de la façon suivante :

- décomposer les modèles en expertises pour faciliter leur interprétation et leur manipulation ;
- combiner les expertises pour enrichir et généraliser les modèles ;
- adapter les expertises pour mieux convenir à un contexte spécifique.

Pour appliquer ces principes, trois principaux mécanismes sont définis pour se charger, respectivement, de l'identification (éventuellement la décomposition), de la combinaison et de l'adaptation des expertises. Afin de bien adhérer à ces principes et de contourner la complexité de ces mécanismes, nous avons eu recours aux techniques métaheuristiques. Ainsi les algorithmes génétiques, le recuit simulé et la recherche avec tabous ont été utilisées pour mettre en oeuvre notre approche de différentes façons, mais aussi pour choisir éventuellement la meilleure d'entre elles, dépendamment des spécificités des modèles. Ainsi, deux applications particulières de notre approche sont conçues, l'une s'intéresse aux modèles de type arbres de décision et l'autre s'occupe des classificateurs bayésiens.

Dans le but d'effectuer l'évaluation expérimentale de l'approche CAMP, nous avons construit un environnement « semi-réel » dans lequel le contexte d'organisation est une évolution d'un vrai système logiciel (API Java), mais les modèles sont « simulés » (c'est-à-dire, obtenus par apprentissage sur des

données réelles). La construction de cet environnement est à double apport. En effet, nous avons défini d'une part, un facteur de la qualité assez crucial pour la maintenance de logiciel qui est la stabilité de classes dans un logiciel OO et, d'autre part, nous avons construit des modèles de stabilité de type arbres de décision et de type classificateurs bayésiens. Les résultats obtenus dans le cadre des différentes expérimentations montrent parfaitement que les objectifs fixés pour améliorer la prédiction sont bien atteints, à savoir, la meilleure capacité prédictive, une bonne capacité de généralisation [1] et une facilité d'interprétation du modèle dérivé par l'approche CAMP.

Bien que nous ayons étudié la validité des résultats de CAMP, il reste à vérifier expérimentalement le comportement de notre approche sur d'autres types de modèles. CAMP peut être éprouvé aussi sur d'autre domaines où des modèles existants sont plus abondants et où des données moins ambiguës sont plus disponibles. Enfin, il nous semble que l'applicabilité de notre approche dans le domaine de la qualité du logiciel croîtrait parallèlement avec l'augmentation du nombre de modèles disponibles.

L'apport de ce travail peut être résumé dans les contributions suivantes :

- **une méthode de combinaison de modèles basée sur les métaheuristiques :** les résultats de notre approche, quand elle est comparée avec d'autres techniques, ont montré que la méthode basée sur la population (AG) et les méthodes basées sur la recherche locale (RT ou RS) sont efficaces pour la combinaison de modèles. Cependant nous ne prétendons pas avoir démontré d'une manière absolue la supériorité de l'une des méthodes sur d'autres.
- **une solution pour l'apprentissage de modèles en cas de pénurie de données :** notre approche peut être vue également comme une méthode d'apprentissage à haut niveau qui utilise des expertises à la place des données d'entraînement. Ces expertises ont été apprises sur des données qui ne sont pas ou ne sont plus disponibles.
- **amélioration de la qualité des prédictions :** dès le chapitre 3, nous avons discuté des critères d'évaluation d'un modèle de prédiction pour lesquels nous avons proposé un ordre de priorité. En effet, outre la capacité prédictive des modèles, l'approche CAMP promeut simultanément la facilité d'interprétation et la capacité de généralisation. Même si ces propriétés sont primordiales pour l'acceptation des modèles, elles sont souvent négligées autant par les travaux sur la prédiction

1. Nous voulons dire la généralisation au sein du contexte d'organisation susceptible d'évoluer.

de la qualité de logiciel que par les méthodes de combinaison [Moerland et Mayoraz, 1999].

- **prédiction de la stabilité :** l'utilisation du facteur « stabilité de classes » d'un logiciel OO pour l'évaluation de l'approche CAMP est une contribution à plus d'un titre. Premièrement, nous avons proposé une définition de la stabilité de classe basée sur les variations entre les versions d'un logiciel. D'un autre côté, nous avons construit des modèles pour la prédire à partir d'un ensemble de métriques structurelles appropriées. De plus, nous avons proposé un meilleur modèle de stabilité grâce à l'approche CAMP.
- **utilisation des métaheuristiques dans le domaine de la qualité du logiciel :** nous avons utilisé des techniques d'optimisation combinatoire pour résoudre un problème en génie logiciel. Les résultats montrent une réussite de l'application de ces techniques dans la prédiction de la qualité du logiciel. En particulier, les techniques métaheuristiques ont montré une applicabilité pour la résolution certains problèmes de prédiction. Ceci constitue une contribution au niveau de l'implication des méthaheuristiques dans le domaine de la qualité du logiciel.

8.1 Limitations et travaux futurs

8.1.1 Facilité d'interprétation et optimisation multi-objectif

Les modèles dérivés par l'approche CAMP peuvent être formés d'un grand nombre d'expertises. En particulier, ces expertises sont des d-rectangles dans le cas des arbres de décision et des intervalles dans le cas des classificateurs bayésiens. Plus nous utilisons de manière optimale les capacités des algorithmes d'optimisation (AG, RS et RT), plus les modèles obtenus sont de grande taille. Le problème engendré est alors celui de la dégradation de la transparence et de la facilité d'interprétation des modèles. En effet, à partir d'un certain nombre d'expertises, il n'est plus facile de construire une image mentale claire de la sémantique du modèle. Un algorithme a été conçu pour palier à ce problème dans le cas des arbres de décision. Bien que cet algorithme ait montré des résultats intéressants, il résout le problème seulement dans le cas d'une décomposition de modèles en d-rectangles. Même si la sévérité de ce problème et son impact sur la facilité d'interprétation du modèle sont discutables, nous le considérons comme une limitation générale de notre approche. C'est pourquoi nous avons pensé à plusieurs manières de le résoudre. En effet, sans s'éloigner des méthodes d'optimisations, nous avons orienté notre réflexion

vers l'optimisation multi-objectif ou multi-critère. C'est ainsi que l'utilisation de cette dernière dans l'approche CAMP serait l'un de nos préoccupations dans les travaux futurs..

En effet, selon Talbi [2001], l'optimisation multi-objectif cherche à optimiser plusieurs composantes d'un vecteur « fonction coût ». Contrairement à l'optimisation mono-objectif, la solution d'un problème multi-objectif (PMO) n'est pas une solution unique, mais un ensemble de solutions, connu comme *l'ensemble des solutions de Pareto optimales (PO)*. Ces solutions sont appelées aussi *solutions admissibles*, *efficaces*, *non dominées*, ou *non inférieures*. Toute solution de cet ensemble est optimale dans le sens qu'aucune amélioration ne peut être faite sur une composante du vecteur sans dégradation d'au moins une autre composante du vecteur. Dans notre cas, les composantes du vecteur « fonction coût » sont : la capacité prédictive du modèle résultant de l'approche CAMP et la taille du modèle exprimée en nombre d'expertises. L'ensemble de solutions Pareto optimales dérivées par CAMP comporterait des modèles ayant des bonnes capacités prédictives et n'ayant pas un grand nombre d'expertises.

8.1.2 Application de l'approche CAMP aux modèles de régression

Pour diverses raisons décrites dans la section 4.3, nous avons commencé par l'application de notre approche ainsi que par son évaluation sur des modèles de classification. Dans nos travaux futurs, nous allons nous occuper des autres types de modèles, particulièrement de ceux de régression. L'application de CAMP aux modèles de régression est un souci pour la généralisation de notre approche. Nous pensons que CAMP produirait dans le cas de la régression des modèles dont la fonction d'expertise est variable d'une région à une autre de l'espace d'entrée. La figure 8.1 montre un exemple de résultat envisageable de l'approche CAMP dans le cas d'une régression linéaire utilisant deux attributs d'entrée x et y et prédisant un facteur de qualité z.

8.1.3 Application de l'approche CAMP aux modèles de types différents

Une fois l'approche CAMP éprouvée sur plusieurs types de modèles, nous procéderons à son application sur une population hétérogène de modèles. Cette application est justifiée, d'une part par la diversité de types des modèles disponibles (voir, section 3.6.2) et d'autre part par les tendances à résoudre des problèmes de la prédiction en impliquant diverses techniques. Cette application de CAMP nécessite, pour chaque type de modèle, une représentation sous formes d'expertises, une unification de

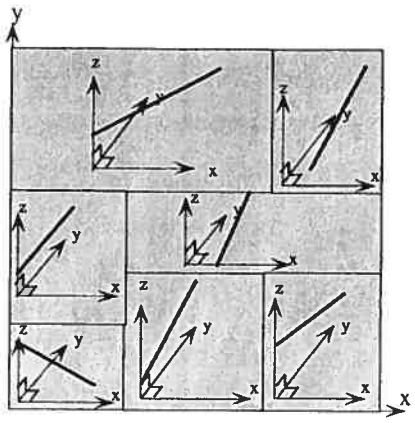


FIG. 8.1 – *Modèle résultant de CAMP dans le cas d'une régression linéaire à deux variables.*

types des entrées et de type la sortie.

Bibliographie

Aarts, E. et Van-Laahonvan, P. (1985). Statistical cooling: a general approach to combinatorial optimization problems. *Philips Journal*, 40:193–226.

Abdel-Ghaly, A., Chan, P., et Littlewood, B. (1986). Evaluation of competing software reliability predictions. *IEEE Transactions on Software Engineering*, 12(9):950–967.

Abreu, F. (1993). Metrics for object-oriented environment. In *Proceedings of the Third International Conference on Software Quality Lake Tahoe Nevada*, pages 55–65.

Aha, D. (1991). Case-based learning algorithms. In *Proceedings of the DARPA Case-Based Reasoning Workshop Morgan Kaufmann, Washington, D.C.*, pages 147–158.

Akiyama, F. (1971). An example of software system debugging. *Information Processing*, 71:353–379.

Albrecht, A. (1979). Measuring applications development productivity. In *Proceedings of IBM Applic. Dev. Joint SHARE/GUIDE Symposium*, pages 83–92.

Almeida, M., Lounis, H., et Melo, W. (1998). An investigation on the use of machine learned models for estimating correction costs. In *Proceedings of the 20th International Conference on Software Engineering*, pages 473–476.

Aroui, M. (1996). *Outils statistiques pour la construction et le choix de modèles en fiabilité des logiciel*. PhD thesis, L'université de Joseph Fourier Grenoble 1, Grenoble.

Azar, D. (2004). *Using genetic algorithms to optimize software quality estimation models*. PhD thesis, School of computer science, McGill university, Montreal.

Azar, D., Bouktif, S., Kégl, B., Sahraoui, S., et Precup, D. (2002). Combining and adapting software quality predictive models by genetic algorithms. In *Proceedings of the 17th International Conference Automated Software Engineering*, pages 283–288.

Barnier, N. et Brissct, P. (1998). Optimizationby hybridation of a genetic algorithm with constraint satisfaction techniques. In *Proceedings of the International conference in evolutionnary computation.*

Basili, V., Condon, S., El-Emam, K., Hendrick, R., et Melo, W. (1997). Characterizing and modeling the cost of rework in a library of reusable software components. In *Proceedings of the 19th International Conference on Software Engineering*, pages 282–291.

Basili, V. et Perricone, B. (1984). Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1):42–52.

Berchtold, S., Keim, D., et Kriegei, H. (1996). The x-tree : An index structure for high-dimensional data. In *Proceedings ot the 22nd VLDB Conference Mumbai(Bombay).*

Boehm, B., Brown, J., et Kaspar, J. (1978). *Characteristics of Software Quality.* TRW Series of Software Technology.

Boehm, B., Steece, B., et Chulani, S. (1999). Bayesian analysis of empirical software engineering cost models. *IEEE Transactions on Software Engineering*, 25(4):573–583.

Boehm, H. (1981). *Software Engineering Economics.* Prentice Hall.

Bouktif, S. (2002). Context driven combination of software quality predictive models. In *proceedings of the 13th International IEEE Symposium on software Reliability*, pages 201–202.

Bouktif, S., Azar, D., Sahraoui, S., Kégl, B., et Precup, D. (2004). Improving rule set based software quality prediction: A genetic algorithm-based approach. *Journal of Object Technology*, 3(4):227–241.

Bouktif, S., Kégl, B., et Sahraoui, S. (2002a). Combining software quality predictive models: An evolutionary approach. In *proceedings of the International Conference on Software Maintenance*, pages 385–392.

Bouktif, S., Kégl, B., et Sahraoui, S. (2002b). Combining and adapting software quality predictive models. In *Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)*, pages 67–81.

Bredero, R.and Caracoglia, G. et Jaggers, C. (1989). Comparative evaluation of existing cost estimation tools. Technical report, MERMAID report D7.1Y.

Breiman, L., Friedman, J., Olshen, R., et Stone, C. (1993). *Classification and Regression Trees*. Chapman and Hall.

Breslawski, S. et Subramanian, G. (1989). Software development effort estimation: Survey and future directions. In *Proceedings of the Decision Sciences Institute Conference, New Orleans, LA,*, pages 602–604.

Briand, L., Basili, V., et Hetmanski, C. (1993a). Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Transactions on Software Engineering*, 19(11):1028–1044.

Briand, L., Basili, V., et Thomas, W. (1992). A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering*, 18(11):931–942.

Briand, L., Devanbu, P., et Melo, W. (1997). An investigation into coupling measures for C++. In *Proceedings of the 19th International Conference on Software Engineering*.

Briand, L., El Emam, K., et Morasca, S. (1995). Theoretical and empirical validation of software product measures. Technical report, International Software Engineering Research Network.

Briand, L., Morasca, S., et Basili, V. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–86.

Briand, L., Thomas, W., et Hetmanski, C. (1993b). Modeling and managing risk early in software development. In *Proceedings of the 15th International Conference of Sofware Engineering*, pages 55–65.

Briand, L., Wuest, J., et Ikonomovski, S. (1999). An assessment and comparison of common software cost estimation modeling techniques. *Procceeding International Conference on Software Engineering*, 1(1):313–322.

Briand, L., Wuest, J., Ikonomovski, S., et Lounis, H. (1998). A comprehensive investigation of quality factors in object-oriented designs: An industrial case study. Technical report, International Software Engineering Research Network.

Briand, L. et Wüst, J. (2002). Empirical studies of quality models in object-oriented systems. In Zelkowitz, M., editor, *Advances in Computers*. Academic Press.

Briand, L., Wüst, J., Daly, J., et Porter, V. (2000). Exploring the relationships between design measures and software quality in object oriented systems. *Journal of Systems and Software*, 51:245–273.

Brocklehurst, S., Chan, P., Littlewood, B., et Snell, J. (1990). Recalibrating software reliability models. *IEEE Transactions on Software Engineering*, 4(16):458–469.

Calciu, M. et Benavent, C. (1992). L'analyse discriminante. Technical report, IAE des pays de l'Adour.

Calero, C., Sahraoui, H., Piattini, M., et Lounis, H. (2001). Estimating object-relational database understandability using structural metrics. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications*.

Carman, D., Dolinsky, A., Lyu, M., et Yu, J. (1995). Software reliability engineering study of a large-scale telecommunications system. In *Proceedings of 6th International Symposium on Software reliability Engineering*, pages 350–359, Toulouse, France.

Cherniavsky, J. et Smith, C. (1991). On weyukerś axioms for software complexity measures. *IEEE Transactions on Software Engineering*, 17(6):636–638.

Chidamber, S. et Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Transactions of Software Engineering*, 20(6):476–493.

Chooman, M. (1983). *Software engineering: Design, Reliability, and Management*. McGraw-Hill.

Chulani, S., Boehm, B., et Steece, B. (1999). Bayesian analysis of empirical software engineering cost models. *IEEE Transactions on Software Engineering*, 25(4):573–583.

Cohen, J. (1960). Coefficient of agreement for nominal scales. In *Proceedings of the Educational and Psychological Measurement*, pages 37–46.

Cohen, W. et Devanbu, P. (1997). A comparative study of inductive logic programming for software fault prediction. In *Proceedings of the Fourteenth International Conference on Machine Learning*.

Conte, S., Dunsmore, H., et Shen, V. (1986). *Software engineering Metrics and Models*. Benjamin-Cummings, Menlo Park,CA.

Cook, T. et Campbell, D. (1979). Quasi-experimentation-design and analysis issues for field settings. Technical report, Houghton Mifflin Company.

Cox, D. et Isham, V. (1980). *Point Processes*. Chapman-Hall,New York.

Curtis, B. (1980). Measurement and experimentation in software engineering. *IEEE software*, 68(9):1144–1157.

D´ Amore, F., Nguyen, V., Roos, T., et Widmayer, P. (1995). On optimal cuts of hyperrectangles. Technical report, Dipartimento di Informatica e Sistemistica, Universita di Roma, La Sapienza.

Daskalantonakis, M. (1992). A practical view of software measurement and implementation experiences within motorola. *IEEE Transaction in Software Engineeering*, 18(11):998–1010.

de Almeida, M., Lounis, H., et Melo, W. (1998). An investigation on the use of machine learned models for estimating correction costs. In *Proceedings of International Conference on Software Engineering*, pages 473–476.

de Almeida, M. et Matwin, S. (1999). Machine learning method for software quality model building. In *Proceedings of International Symposium on Methodologies for Intelligent System*.

DeMarco, T. (1982). *Controlling Software Projects Management*, chapter Measurement and Estimation. Yourdon Press.

Denton, J. (1999). Accurate software reliability estimation. Master's thesis, Colorado State University, Fort Collins, Colorado.

Dhama, H. (1994). Quantitative models of cohesion and coupling software. In *Proceedings of the annual Oregon Workshop on Software Metrics*, pages 111–121.

Dohi, T.and Nishio, Y. et Osaki, S. (1999). Optimal software release scheduling based on artificial neural networks. *Annals of Software Engineering*, 8(1):167–185.

Dolado, J. (2000). A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021.

Dolan, W., Cummings, P., et Le-Van, M. (1990). Algorithmic efficiency of simulated annealing for heat exchanger network design. In *Proceedings of the computers in Chemical Engineering conference*, pages 1039–1050.

Dromey, R. (1996). Cornering the chimera. *IEEE Software.*, 13(1):33–43.

Efron, B. (1979). Computers and the theory of statistics: thinking the unthinkable. *SIAM review*, 21:460–480.

Efron, B. et Tibshirani, R. (1993). *An Introduction to the Bootstrap*. London and NewYork, Chapman and Hall.

El Emam, K. (2000). A methodology for validating software product metrics. Technical report, Conseil national de recherches Canada Institut de Technologie de l'information.

El Emam, K., Benlarbi, S., Goel, N., et Rai, S. (2001a). Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software*, 55(3):301–320.

El Emam, K., Benlarbi, S., Goel, N., et Rai, S. (2001b). Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software*.

Elkan, C. (1997). Naive bayesian learning. Technical report, Department of Computer Science Harvard University.

Emerson, T. (1984). Path coverage, and the cohesion metric. In *Proceedings of the : IEEE Computer Software and Applications Conference*, pages 421–431.

Esteva, J. et Reynolds, R. (1991). Identifying reusable software components by induction. *International Journal of Software Engineering and Knowledge Engineering*, 1(3):271–292.

Evett, M., Khoshgoftar, T.and Chien, P., et Allen, E. (1998). Gp-based software quality prediction. In *proceedings of Third Annual Genetic Programming Conference*, pages 60–65.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. John Wiley and Sons.

Feathers, M. C. (1999). Stability through change. In *Proceedings of the Accomplishing Software Stability Workshop, OOPSLA 99 Denver, CO*.

Fenton, N. (1991). *Software Metrics : A Rigorous Approach*. Chapman and Hall.

Fenton, N. et Kitchenham, B. (1990). Validating software measures. *Journal of Software Testing Verification and Reliability*, 1(2):27–42.

Fenton, N., Krause, P., et Neil, M. (2002). Software measurement: Uncertainty and causal modelling. *IEEE Software*, 10(4):116–122.

Fenton, N. et Neil, M. (1999a). A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689.

Fenton, N. et Neil, M. (1999b). Software metrics and risk. In *Proceedings of the FESMA 99, 2nd European Software Measurement Conference*.

Fenton, N. et Neil, M. (2000a). The jury fallacy and the use of bayesian networks to present probabilistic legal arguments. *Mathematics Today (Bulletin of the IMA)*, 36(6):180–187.

Fenton, N. et Neil, M. (2000b). Software metrics: A roadmap. In Finkelstein, A., editor, *Proceedings of the The Future of Software Engineering, 22nd International Conference on Software Engineering*, pages 357–370. ACM Press.

Fenton, N. et Neil, M. (2001). Making decisions: Bayesian nets and MCDA. In *Proceedings of the Knowledge-Based Systems*, pages 307–325.

Fenton, N. et Ohlsson, N. (2000). Quantitative analysis of faults and failures in a complex sofware system. *IEEE Transactions on Software Engineering*, 26(8):797–814.

Fenton, N. et Pfleeger, S. (1997). *Software Metrics : A Rigorous and Practical Approach (2nd Edition)*. International Thomson Computer Press, Boston.

Ferland, J. et Costa, D. (2001). Heuridic search methods for combinatorial programming problems. Pub 1193, Department of Computer Sciences and operational researchs, Montreal University.

Freund, Y. et Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Gandibleux, X., Libert, G., Cartignies, E., et Millot, P. (1994). Smart: Étude de la faisabilité d'un solveur de problèmes de mobilisation de réserve tertiaire. *Revue des systemes de decision*, 1(3):45–67.

Ganesan, K., Khoshgoftaar, T., et Allen, E. (2000). Cased-based software quality prediction. *International Journal of Software Engineering and Knowledge Engineering*, 10(2):139–152.

Garey, R. (1979). *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman and Co.

Garvin, D. (1984). What does product quality really mean? *Sloan Management Review*, pages 25–45.

Genero, M., Piattini, M., et Calero, C. (2000). Early measures for uml class diagrams. *LÓbjet*, 6(4).

Glasberg, D., El Emam, K., Melo, W., Machado, J., et Madhavji, N. (2000). Evaluating thresholds for object-oriented design measures. Technical report, National Research Council Canada, Institute for Information Technology.

Glaube, R. (1963). Time-dependant statistics of the ising model. *Mathematics Physics Journal*.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. In *Computers and Operations Research*, pages 533–549.

Goel, A. et Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, 28:206–211.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Goldberg, D. (1991). Real-coded genetic algorithms, virtual alphabets and blocking. *Complex Systems*, 5:139–167.

Goodman, P. (1993). *Practical Implementation of Software Metrics*. McGraw Hill Company.

Gray, A. et MacDonell, S. (1997). A comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39:425–437.

Griech, B. et Pomerol, J. (1994). Design and implementation of a knowledge-based decision support system for estimating software development work-effort. *Joutnal of Systems Integration*, 4:171–184.

Grosser, D., Sahraoui, H., et Valtchev, P. (2003). An analogy-based approach for predicting design stability of java classes. In *Proceedings of IEEE metrics 2003*, pages 252–262.

Halstead, M. (1977). *Elements of Software Science*. Elsevier Science Inc.

Hanley, J.and Neil, M. (1982). The meaning and use of the area under a receiver operating characteristic curve. *Diagnostic Radiology*, 143(1):29–36.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial. In *Proceedings of the Congress on Numerical Methods in Combinatorial Optimization*.

Harrell, F., Lee, K., et Mark, D. (1996). Multivariate prognostic models: Issues in developing models,evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*, 15:361–387.

Helmer, H. (1966). *Social Technology*. Basic Books.

Henry, S. et Selig, C. (1990). Predicting source-code complexity at the design stage. *IEEE Software*, 7(2):36–44.

Hetzel, B. (1993). *Making Software Measurement Work - Building an Effective Measurement Program*. John Wiley & Sons, Inc.

Holland, J. (1975). *Adaptation in Natural Artificial Systems*. University of Michigan Press.

Hong, E. et Wu, C. (1997). Criticality models using sdl metrics set. In *Proceedings. of the 4th Asia-Pacific Software Engineering and International Computer Science Conference*, pages 23–30.

Humphrey, W. (1989). *Managing the software Process*. Addison-Wesley Reading, MA.

Iannino, A., Musa, J., Okumoto, K., et Littelwood, B. (1984). Criteria for software reliability model comparaisons. *IEEE Transactions on software Engineering*, 10:687–691.

ISO (2000). Iso/iec 14598 information technology - software product evaluation. Technical report, International Standards Organisation.

ISO (2003). Iso/iec 9126 software engineering - product quality. Technical report, International Standards Organisation.

Jacobs, R., Jordan, M., Nowlan, S., et Hinton, G. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Jeffery, D. et Low, G. (1990). Calibrating estimation tools for software development. *Philips Journal*, 5(1):215–221.

Jelinski, Z. et Moranda, P. (1972). *Software Reliability Research, Statistical Computer Performance Evaluation*. Academic Press.

Jones, C. (1978). Measuring programming quality and productivity. *IBM Systems Journal*, 17(1):39–63.

Jorgensen, M. (1995). Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering*, 21(8):674–681.

Kaaniche, K. et Kanoun, K. (1996). Reliability of a telecommunications system. In *Proceedings of the 7th Int. Symp. on Software Reliability Engineering*, pages 207–212, White Plains, NY, USA.

Kafura, D. et Canning, J. (1988). Using group and subsystem level analysis to validate software metrics on commercial software systems. Technical report, Polytechnic of Blacksburg, Virginia, USA.

Kai, M. et Boon, T. (1997). Model combination in the multiple-data-batches scenario. In *Proceedings of the European Conference on Machine Learning*, pages 250–265.

Kan, S. (1995). *Metrics and Models In Software Quality Engineering*. Addison Wesley.

Karunanithi, N., Whitely, D., et Malaiya, Y. (1992). Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering*, 18(7):563–574.

Kemerer, C. (1993). Reliability of function points measurement a field experiment. *Communications of the ACM*, 36(2):85–97.

Kenney, G. et Vouk, M. (1992). Measuring the field quality of wide-distribution commerical software. In *Proceedings of the 3th International Symposium on Software Reliability Engineering*, pages 351–357, research Triangle Park NC USA.

Khoshgoftaar, T., Allen, B., Kalaichelvan, K., et Goel, N. (1996a). Early quality prediction: a case study in telecommunications. *IEEE Software*, 13(1):65–71.

Khoshgoftaar, T., Allen, E., Bullard, L., Halstead, R., et Trio, G. (1996b). A tree based classification model for analysis of a military software system. In *Proceedings of the IEEE High-Assurance Systems Engineering Workshop*, pages 244–251.

Khoshgoftaar, T., Allen, E., Hudepohl, J., et Aud, S. (1997). Applications of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks*, 8(4):902–909.

Khoshgoftaar, T., Allen, E., Kalaichelvan, K., Goel, N., Hudepohl, J., et Mayrand, J. (1995a). Fault-prone program modules in a very large telecommunications system. In *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pages 24–33.

Khoshgoftaar, T., Allen, E., et Xu, Z. (2000). Predicting testability of program modules using a neural network. In *Proceedings of the IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pages 57–62.

Khoshgoftaar, T., Idri, A., Abran, A., et Szabo, R. (2002). Estimating software project effort by analogy based on linguistic values. In *Proceedings of the IEEE METRICS*, pages 21–31.

Khoshgoftaar, T., Lanning, D., et Munson, J. (1993). A comparative study of predictive models for program changes during system testing and maintenance. In *Proceedings of the Conference of Software Maintenance*, pages 72–79.

Khoshgoftaar, T. et Munson, J. (1990). The lines of code metric as a predictor of program faults: A critical analysis. In *Proceedings of Computer Software and Applications Conference*, pages 408–413.

Khoshgoftaar, T., Munson, J., Bhattacharya, B., et Richardson, G. (1992). Predictive modeling tech-

niques of software quality from software measures. *IEEE Transactions on Software Engineering*, 18(11):979–987.

Khoshgoftaar, T., Pandya, A., et Lanning, D. (1995b). Application of neural networks for predicting faults. *Annals of Software Engineering*, 13(1):141–154.

Khoshgoftaar, T. et Szabo, R. (1994a). Improving code churn prediction during the system test and maintenance phases. In *Proceedings of the International Conference of Software Maintenance*, pages 58–67.

Khoshgoftaar, T. et Szabo, R. (1994b). Improving code churn predictions during the system test and maintenance phases. In *Proceedings of the International Conference on Software Maintenance*, pages 58–67.

Khoshgoftaar, T., Szabo, R., et Voas, J. (1995c). Detecting program modules with low testability. In *Proceedings of the International Conference on Software Maintenance*, pages 242–250.

Kim, J. et Pearl, J. (1983). A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the IJCAI-83*, pages 190–193.

Kirkpatrick, S., Gellat, C., et Vecchi, M. (1983). Optimization by simulated annealing. In *Proceedings of the Science 220*, pages 671–680.

Kitchenham, B. et Pfleeger, S. (1996). Software quality: The elusive target. *IEEE Software*, 1:12–21.

Kitchenham, B., Pfleeger, S., et Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944.

Kolence, K. (1975). Software physics. *Datamation*, pages 48–51.

Lake, A. (1994). Use of factor analysis to develop oop software complexity metrics. In *Proceedings of the Annual Oregon Workshop on Software Metrics, Silver Falls,Oregon, USA*.

Lanubile, F. et Visaggio, G. (1997). Evaluating predictive quality models derived from software measures: lessons learned. *Journal of Systems and Software*, 38(1):225–234.

Levendel, Y. (1990). Reliability analysis of large software systems: defects data modelling. *IEEE Transactions on Software Engineering*, 16(2):141–152.

Lipow, M. (1982). Number of faults per line of code. *IEEE Trans. Software Eng.*, 8(4):437–439.

Littelwood, B. et Verrall, J. (1973). A bayesian reliability growth model for computer software. *Journal of the Royal Statistical Society*, 22:332–334.

Littlewood, B. (1991). Software reliability modelling: achievements and limitations. In *Proceedings of IEEE Computers Euro91,Bologna*, pages 13–16.

Liu, Y. et Khoshgoftaar, T. (2004). Reducing overfitting in genetic programming models for software quality classification. In *Proceedings of the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE 2004), Tampa, Florida, USA*, pages 25–26.

Louchet, J., Guyon, M., Lesot, M., et Boumaza, A. (2001). *Apprentissage et évolution*, chapter L'algorithme des mouches dynamiques Guider un robot par évolution artificielle en temps réel. Hermes.

Lounis, H., Sahraoui, H. A., et Melo, W. L. (1998). Vers un modèle de prédiction de la qualité du logiciel pour les systèmes à objets. *LŠObjet*, 4(4):407–429.

Lyu, M. et Nikora, A. (1992). Applying reliability models more effectively. *IEEE Software*, 7(9):43–52.

Mao, Y., Sahraoui, H., et Lounis, H. (1998). Reusability hypothesis verification using machine learning techniques:a case study. In *Proceedings of 13th IEEE International Conference on Automated Software Engineering*, pages 84–93.

Martin, R. (1997). Stability. *C++ Report*, 9(2).

McCabe, T. (1979). A complexity measure. *IEEE Transactions on Software Engineering*, 5:308–320.

McCall, J.A.and Richards, P. et Walters, G. (1977). Factors in software quality. Technical report, US Rome Air Development Center Reports.

Meir, R. (1994). Variance and the combination of estimators:the case of linear regression. TR 922, Department of Electrical Engineering, technion Haifa.

Meir, R., El-Yaniv, R., et Ben-David, S. (2000). Localized boosting. In *Proceedings of the 13th Annual Conference on Computational Learning Theory*, pages 190–199.

Merz, C. (1998). *Classification and Regression by Combining Models*. PhD thesis, university of California Irvine.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., et Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal. Chem. Phys.*, 27.

Moerland, P. et Mayoraz, E. (1999). ,DynaBoost: Combining boosted hypotheses in a dynamic way. Technical report, IDIAP, Switzerland.

Morris, K. (1989). Metrics for object-oriented software development environments. Technical report, Massachusetts Institute of Technology.

Mukhopadhyay, T., Vicinanza, S., et Prietula, M. (1992). Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS Quarterly*, 16(2):155–171.

Munson, J. et Khoshgoftaar, T. (1992). The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5):423–433.

Musa, J. et Okumoto, K. (1983). *Electronic Systems Effictivness and Life Cycle Costing*, chapter Software reliability models: Concepts, classification, comparaison, and practice. Springer-Verlag.

Nelson, E. (1966). *Management Handbook for the Estimation of Computer Programming Costs*. Systems Development Corp.

Ohlsson, N. et Alberg, H. (1996). Predicting error-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894.

Ottenstein, L. (1992). Using slice profiles and metrics during software maintenance. In *Proceedings of the 10th Annual Software Reliability Symposium*, pages 16–23.

Ottenstein, L. et Thuss, J. (1989). The relationship between slices and module cohesion. In *Proceedings of the 11th International Conference on Software Engineering.*, pages 198–204.

Ottenstein, L. et Thuss, J. (1991). Sliced based metrics for estimation cohesion. Technical report, Colorado State University.

Panthaki, M. J. et Sahu, R. (1999). Software robustness and stability mechanisms in the comet framework. In *Proceedings of the Accomplishing Software Stability Workshop, OOPSLA 99 Denver, CO.*

Parnas, D. (1994). Software aging. In *Proceedings of the 16th International Conference on Software Engineering.*

Pearl, J. (1982). Reverrend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the AAAI National Conference on AI*, pages 133–136.

Pearl, J. (2000). *Causality: Models, Reasoning and Inference*. Cambridge University Press.

Pekarek, B. (1999). Stable systems. In *Proceedings of the Accomplishing Software Stability Workshop, OOPSLA 99 Denver, CO*.

Perrone, M. et Cooper, L. (1993). *Artificial Neural Networks for Speech and Vision*, chapter When networks disagree: Ensemble Methods for hybrid neural networks. Chapman and Hall, London.

Perry, D.E Porter, A. et Votta, L. (2000). Empirical studies of software engineering: A roadmap. In Finkelstein, A., editor, *The Future of Software Engineering*. ACM Press.

Petty, M. et Mukherjee, A. (1998). Experimental comparison of d-rectangle intersection algorithms applied to hla data distribution. Technical report, University of Central Florida, Institute for Simulation and Training, Department of Computer Science, Orlando.

Pfleeger, S. (1989). *An Investigation of cost and productivity for Object-Oriented Development*. PhD thesis, George Mason University.

Pfleeger, S. (1998). *Software Engineering: Theory and Practice*. Prentice-Hall, Upper Saddle River, NJ.

Poels, G. et Dedene, G. (1997). Comments on property-based software engineering measurement: Refining the additivity properties. *IEEE Transactions on Software Engineering*, 23(3):190–195.

Porter, A. (1993a). Developing and analyzing classification rules for predicting faulty software components. In *Proceedings of the 5th International Conference of Software Engineering and Knowledge Engineering*, pages 453–461.

Porter, A. (1993b). Using measurement-driven modeling to provide empirical feedback to software developers. *Journal of Systems and Software*, 20:237–243.

Porter, A. et Selby, R. (1990a). Empirically guided software development using metric-based classification trees. *IEEE Software*, 4(4):46–54.

Porter, A. et Selby, R. (1990b). Evaluating techniques for generating metric-based classification trees. *Journal of Systems and Software*, 12:209–218.

Pressman, R. (1996). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education.

Putnam, L. (1978). A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions of Software Engineering*, 4:345–361.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ramoni, R. et Sebastiani, P. (1999). Robust bayesian classification. Technical report, Knowledge Media Institute, the Open University.

Ramsey, C. et Basili, V. (1989). An evaluation of expert systems for software engineering management. *IEEE Transactions of Software Engineering*, 15:747–759.

Rao, J. et Tibshirani, R. (1997). The out-of-bootstrop method for model averaging. Technical report, Department of statistics, university of Toronto.

Robins, A. (1995). Catastrophic forgetting. *Rehearsal and Pseudorehearsal ConnectionScience*, 7:123–146.

Rocacher, D. (1988). Metrics definitions for smalltalk. Technical report, Project ESPRIT 1257, MUSE WP9A.

Rousseeuw, P. (1984). Least median of squares regression. *Journal of American Statistical Association*, pages 871–880.

Sahraoui, H. et Azar, D. (1999). Quality estimation models optimisation using genetic algorithms: Case of maintainability. In *Proceedings of the European Software Measurement Conference , (FES-MA'99)*, pages 131–138.

Sahraoui, H., Boukadoum, M., Chawiche, H., Mai, G., et Serhani, M. (2002). A fuzzy logic framework to improve the performance and interpretation of rule-based quality prediction models for object-oriented software. In *Proceedings of the 26th Computer Software and Applications Conference (Computer Software and Applications Conference, Oxford)*.

Sahraoui, H. A., Serhani, M., et Boukadoum, M. (2001). Extending software quality predictive models domain knowledge. In *Proceedings of the 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*.

Schach, S. (1993). *Software Engineering, Second Edition.* IRWIN.

Schick, G. et Wolverton, R. (1978). Assessment of software reliability. In *Proceedings of the Operations Research, Physica-Verlag*, pages 395–422.

Schneidewind, N. (1994). Validating metrics for ensuring space shuttle flight software quality. *IEEE Computer*, pages 50–57.

Selby, R. et Porter, A. (1988). Learning from examples: generation and evaluation of decision trees for software resource analysis. *IEEE Transactions on Software Engineering*, 14(1):1743–1757.

Serrano, M., Calero, C., et Piattini, M. (2003). Experimental validation of multidimensional data models metrics. In *Proceedings of the 36th Hawaii International Conference on System Sciences*, pages 130–137.

Shao, J. (1996). Bootstrap model selection. *Journal of American Statistical Association*, pages 655–665.

Shen, V., Yu, T., Thebaut, S., et Paulsen, L. (1985). Identifying error-prone software – an empirical study. *IEEE Transactions on Software Engineering*, 11(4):317–323.

Sheppard, M. (1993). *Software Engineering Metrics - Volume I: Measures and Validations*. McGraw-Hill.

Shepperd, M. et Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12):736–743.

Shukla, K. (2000). Neuro-genetic prediction of software development effort. *Information and Software Technology*, 42(10):701–713.

Simmons, P. (1996). Quality outcomes: Determining business value. *IEEE Software*, 1:25–32.

Sollich, P. et Krogh, A. (1996). *Advances in neural information Processing Systems*, chapter Learning with ensembles: how overfitting can be usefu. MIT press.

Srinivasan, K. et Fisher, D. (1995). Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, 21(2):126–137.

Stone, M. (1977). Asymptotics for and against cross-validation. *Biometrika*, pages 29–35.

Sunita, D. (1999). *Bayesian Analysis ofSoftware Cost and Quality Modeles*. PhD thesis, University of Southern California.

Takahashi, R., Muraoka, Y., et Nakamura, Y. (1997). Building software quality classification trees: Approach, experimentation. In *Proceedings of the 9th International Symposium of Software Reliability Engineering, ISSRE'97*, pages 222–233.

Takang, A. et Grubb, P. (1996). *Software Maintenance*. Thomson Computer Press.

Talbi, E. (2001). Métaheuristiques pour l'optimisation combinatoire multi-objectif: Etat de l'art. TR 98-

757.33, Laboratoire d'Informatique Fondamentale de Lille, Université des Scieneces et Technologies de Lille.

Taylor, D. (1993). Software metrics for object technology. *Object Magazine*, pages 22–28.

Vicinanza, S., Mukhopadhyay, T., et Prietula, M. (1991). Software effort estimation: An exploratory study of expert per formance. *Information Systems Research*, 2:243–262.

Vicinanza, S., Prietulla, M., et Mukhopadhyay, T. (1990). Case-based reasoning in software effort estimation. In *Proceedings of the 11th Internatinal Conference on Information Systems*, pages 149–158.

Walston, C. et Felix, C. (1977). A method of programming measurement and estimation. *IBM Systems Journal*, 16(1):54–73.

Weiser, M. (1982). Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452.

Weiser, M. (1984). Program sclicing. *IEEE Transactions on Software Engineering*, 10(4):352–357.

Wolverton, R. (1974). The cost of developing large-scale software. *IEEE Transactions on computer*, 23(6):615–636.

Wright, A. (1991). Genetic algorithms for real parameter optimization. In *Proceedings of the Foundation Of Genetic Algorithms*.

Youden, W. J. (1961). How to evaluate accuracy. *Materials Research and Standards, ASTM*.

Zhang, D. et Tsai, J. (1986). Machine learning and software engineering. *Software Quality Journal*, pages 87–119.

Zuse, H. (1998). *A Framework of Software Measurement*. Walter de Gruyter.

Annexe A

Exemples de Modèles fixes de la qualité du logiciel

Dans cette annexe, nous donnons les structures de quelques modèles de la qualité qui ont connu une acceptation dans la communauté du génie logiciel et qui prétendent représenter les différentes vues de la qualité. Parmi ces modèles, nous citons le modèle de McCall (voir la figure A.1), le modèle de Boehm (voir la figure A.2) et le modèle standard de la qualité ISO9126 (voir la figure A.3).

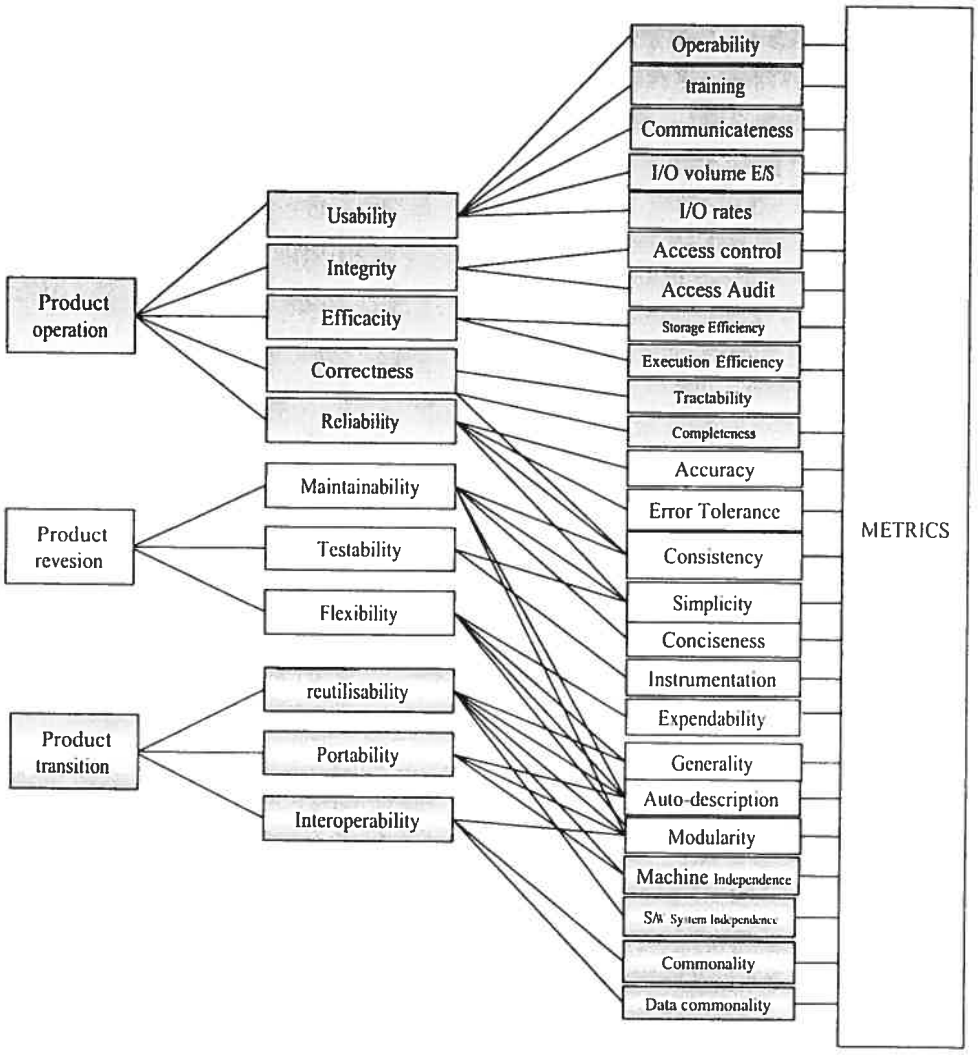


FIG. A.1 – *Modèle de McCall.*

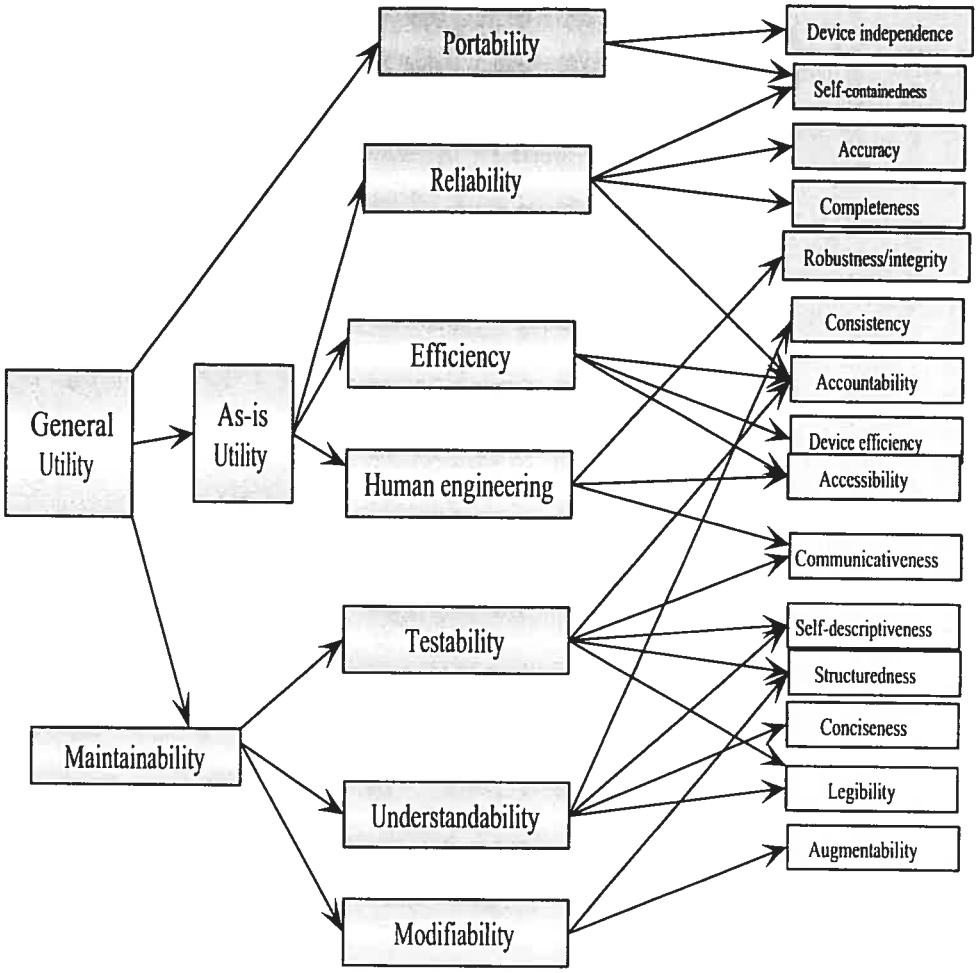


FIG. A.2 – *Modèle de Boehm.*

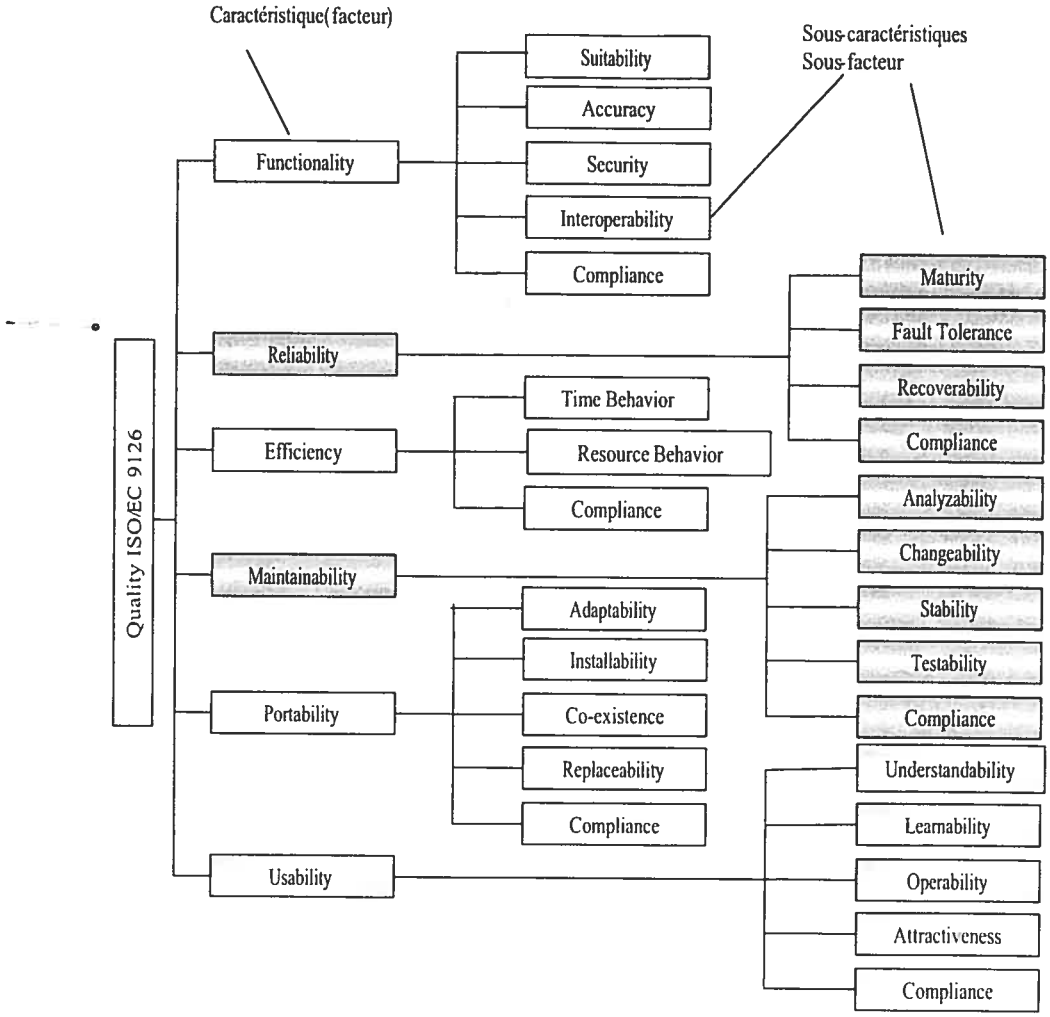


FIG. A.3 – *Modèle standard de la qualité ISO 9126.*

Annexe B

Techniques de construction de modèles de prédiction

Dans cet annexe, nous présentons les techniques utilisées dans la construction de modèles de qualité du logiciel. Ces techniques comprennent des méthodes statistiques basées généralement sur la régression et des méthodes d'intelligence artificielle basées sur l'apprentissage.

B.1 Techniques statistiques

B.1.1 Régression linéaire (RL)

Le modèle associé à cette technique est de forme :

$$\hat{f}(\mathbf{x}) = \hat{y} = \sum_{j=0}^{d} a_j x^{(j)} \tag{B.1}$$

où la variable de sortie $\hat{y}$ appelée aussi variable dépendante est continue, les variables d'entrée $x^{(j)}$ sont appelées variables explicatives, d est le nombre de variables explicatives et les $a_j, j = 1, \ldots, d$ sont les paramètres de la régression. La régression linéaire suppose que les variables d'entrée $x^{(j)}$ sont indépendantes les unes des autres et que la relation exprimée par le modèle est linéaire. Les procédures de détermination des paramètres de la régression consistent à minimiser l'erreur entre la réponse observée et la réponse prédite, qui est appelée « résidu ». Une façon de ce faire est de minimiser la somme des

carrées des résidus, exprimée par la fonction suivante :

$$\arg\min_{i} \sum_{i=1}^{n} r_i^2 \tag{B.2}$$

Le résidu ou l'écart résiduel est défini par :

$$r_i = y_i - \hat{y}_i, \textit{ avec } i = 1, \ldots, n,$$

avec n le nombre d'observations de construction (d'entraînement) du modèle, y_i la valeur de sortie réelle et $\hat{y}_i$ la valeur de sortie prédite de la $i^{ième}$ observation. Cette régression est appelée régression des moindres carrés (RMC). Elle est bien appropriée pour les situations suivantes:

- plusieurs degrés de liberté sont disponibles (c'est-à-dire, qu'il y a beaucoup plus d'observations que de paramètres à estimer) ;
- les données ont un comportement régulier (dans le sens statistique, par exemple, il n'y a aucune valeur extrême) ;
- un nombre restreint de variables d'entrées après transformations [1] ;
- pas de données manquantes.

Ces contraintes imposent des restrictions sévères sur l'utilisation de cette technique sur les données en génie logiciel qui respectent rarement toutes ces conditions [Gray et MacDonell, 1997; El Emam, 2000; Fenton et Neil, 1999a].

B.1.2 Régression robuste (RMMC)

La limitation de la régression linéaire des moindres carrés est due partiellement au fait que cette technique suppose que les données sont distribuées d'une manière statistiquement régulière. Les données en génie logiciel comportent souvent des points relativement éloignés par rapport à l'ensemble des données (valeurs extrêmes). Ce problème peut être contourné par l'application de la technique de la régression de médiane des moindres carrés (RMMC) [Gray et MacDonell, 1997] qui consiste à minimiser la médiane des résidus pour déterminer les paramètres de la régression (voir l'équation B.3). La RMMC revient à trouver (dans un plan) la pente de la droite de régression qui correspond à « la médiane des

1. Changement de variables, au besoin, afin de permettre une représentation interprétable de la variable de sortie.

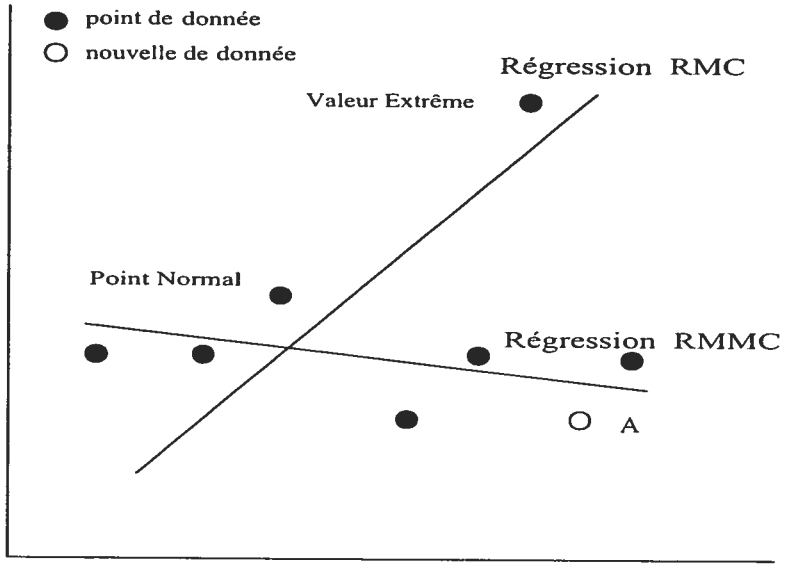


FIG. B.1 – *L'influence des points de données ayant des valeurs extrêmes sur la régression.*

pentes ». Elle utilise la même équation que la RMC sauf que les paramètres a_j, $j = 1, \ldots, d$ sont estimés en minimisant la médiane des carrés des écarts résiduels :

$$\arg\min_{i} \; mediane \; r_i^2 \tag{B.3}$$

La figure B.1 montre qu'une valeur extrême a généralement des effets indésirables sur la droite de régression utilisant RMC et elle est sans effet sur la droite de régression robuste (RMMC). Cette figure montre aussi que le point A ne peut pas être prédit par le modèle de régression utilisant RMC mais il l'est avec RMMC. Pour en connaître plus sur la régression de médiane de moindres carrés (voir [Rousseeuw, 1984]).

B.1.3 Régression logistique (RLog)

La régression linéaire a pour but de modéliser la relation entre une variable dépendante continue et un nombre de variables explicatives. Cependant, lorsque l'on veut prédire une variable dépendante binaire ou discrète, on a recours à une autre forme de régression connue sous le nom de régression logistique. C'est une technique employée pour la tâche de classification. Pour obtenir un modèle de classification, une méthode consiste à faire des transformations de la régression linéaire afin de convertir une variable continue y en une variable discrète. Pour ce faire, on choisit d'abord de transformer y

sous forme de probabilité (comprise entre 0 et 1). Dans le cas binaire, une codification très simple est effectuée de la façon suivante : si y est supérieure à un seuil alors on lui attribue le code 1, sinon on lui attribue le code 0. Après transformation la régression revient à prédire une probabilité d'appartenance à une classe ou à une catégorie. La régression linéaire est ainsi transformée en utilisant la fonction *exponentielle* et en relativisant y. Ces deux transformations sont montrées sur la figure B.2. On obtient enfin une équation d'une courbe sigmoïde qui représente la régression logistique de la forme :

$$\hat{y} = \frac{e^{(\Sigma_{j=0}^{d} a_j.x^{(j)})}}{1 + e^{(\Sigma_{j=0}^{d} a_j.x^{(j)})}}. \tag{B.4}$$

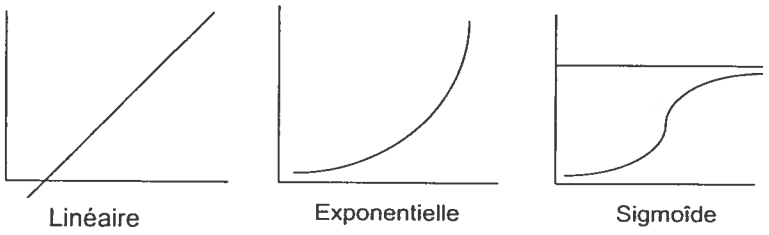


FIG. B.2 – *De la régression linéaire à la régression logistique.*

La figure B.2 représente le passage de la régression linéaire à la régression logistique dans le cas d'une seule variable d'entrée.

B.1.4 Analyse discriminante (ADis)

L'analyse discriminante est aussi utilisée pour produire un modèle de classification. Considérons la classification des composants logiciels en composants stables et instables à partir de deux variables d'entrée *LCOMB* et *NPPM*[2]. En analysant la figure B.3, on se rend compte que sur chacun des axes représentant les deux métriques, il y a une région importante d'incertitude dans laquelle, pour les mêmes valeurs d'une métrique, on trouve des composants appartenant aux deux classes. Le but de l'analyse discriminante est de trouver un nouvel axe sous forme d'une combinaison linéaire des variables d'entrée (métriques dans notre cas), qui permet de réduire cette zone d'incertitude et de séparer au mieux les deux classes. L'équation de cet axe est appelée fonction linéaire discriminante. Dans le cas de notre exemple,

2. Deux métriques OO, voir tableau 7.1.

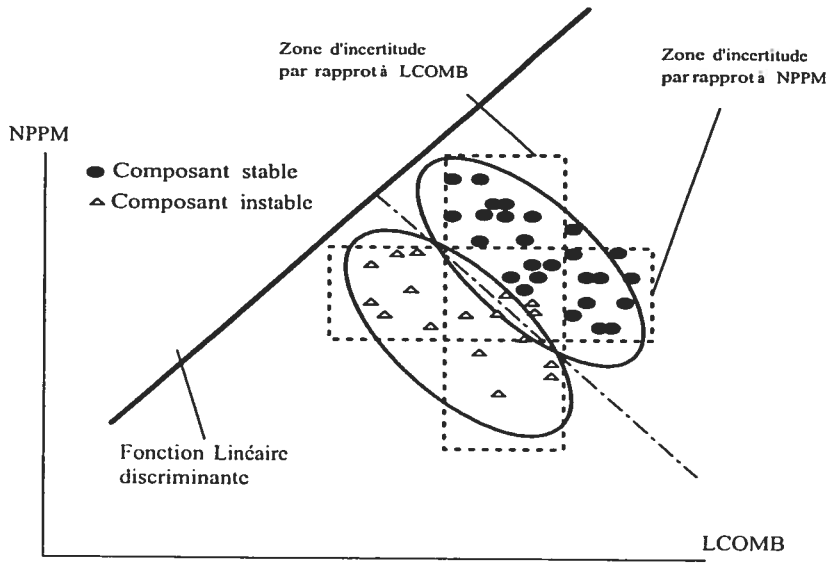


FIG. B.3 – *Principe de l'analyse discriminante.*

elle prend la forme de :

$$z = ax^{(1)} + bx^{(2)}, \tag{B.5}$$

avec $x^{(1)}$ et $x^{(2)}$ respectivement les variables correspondantes aux métriques *LCOMB* et *NPPM*. Les coefficients de cette fonction linéaire sont dérivés de telle manière que la variation de z soit la plus grande possible entre les deux classes et la plus petite possible à l'intérieur des classes. Plus de détails peuvent être trouvés dans [Calciu et Benavent, 1992]. Pour classifier les composants dans l'une des classes, on fixe un seuil z_s qui joue le rôle de frontière entre les classes. Normalement, ce seuil est la moyenne des valeurs de z si les nombres de composants par classe sont égaux, sinon on utilise la moyenne pondérée $z_s = \frac{n_2\bar{z}_1 + n_1\bar{z}_2}{n_1+n_2}$, où $\bar{z}_1$ et $\bar{z}_2$ sont les valeurs moyennes de z par classe et n_1 et n_2 sont les tailles de deux classes. Il est à noter que la moyenne de chaque classe est pondérée avec la taille de l'autre. Dans le cas de plusieurs classes, l'analyse discriminante permet de trouver *les meilleures fonctions linéaires discriminantes* qui séparent d'une manière maximale les différentes classes.

B.1.5 L'analyse en composantes principales (ACP)

L'analyse en composantes principales (ACP) est une technique permettant de trouver, à partir d'un ensemble initial de d attributs originels, un ensemble de d' nouveaux attributs qui prennent mieux en

compte la variance des données. Simplement, à partir des attributs originels $x^{(1)},\ldots,x^{(d)}$, l'ACP dérive de nouveaux attributs $X^{(1)},\ldots,X^{(d')}$. Un nouvel attribut $X^{(k)}$ est une transformation sous forme d'une combinaison linéaire des attributs d'origine, $X^{(k)} = a_{1k}x^{(1)} + \ldots + a_{dk}x^{(d)}$. Les coefficients a_{jk} sont déterminés de telle sorte que les $X^{(k)}$ soient :

- deux à deux non corrélés ;
- de variance maximale ;
- d'importance décroissante.

On dit que $X^{(1)}$ est la première composante principale, elle a la plus grande variance. Par la suite, la combinaison qui a la deuxième plus grande variance représente la deuxième composante principale $X^{(2)}$ et ainsi de suite. L'analyse en composantes principales projette alors les données d'un espace originel à d dimensions sur un sous-espace à d' dimensions (d' inférieur à d) défini par les nouveaux attributs $X^{(k)}$. On procède ainsi à une réduction de dimensionnalité en gardant généralement la plupart des informations intrinsèques des données originelles.

B.2 Techniques d'intelligence artificielle

B.2.1 Réseaux de neurones (RN)

Les réseaux de neurones sont des structures inspirées du fonctionnement des neurones du cerveau. Ils sont utilisés dans différents domaines pour modéliser les relations entre attributs. En général, un réseau de neurones comprend trois couches : une couche d'entrée qui contient les attributs d'entrées, une couche de sortie qui contient les attributs de sortie et une ou plusieurs couches « cachées » (voir la figure B.4).

Les réseaux de neurones les plus utilisés dans la littérature sont développés en choisissant d'abord une architecture appropriée de neurones. L'architecture décrit combien de couches de neurones sont utilisées, le nombre de neurones dans chaque couche et comment les neurones sont reliés entre eux. D'autres décisions sont également possibles, concernant la nature précise des neurones, tels que leur fonction de transfert et les paramètres pour l'algorithme d'entraînement. Généralement, les connexions entre couches de neurones sont pondérées.

Pour évaluer la réponse à une entrée, chaque neurone calcule la somme de ses entrées. Si cette

somme est supérieure à un seuil, le neurone s'active et délivre une sortie aux neurones de la couche suivante du réseau jusqu'à la sortie. Les fonctions d'activation peuvent être des fonctions à seuil, à valeurs discrètes, linéaires ou non linéaires. Le réseau est construit en ajustant les poids (qui fonctionnent comme les paramètres dans un modèle statistique, mais d'une façon beaucoup plus hiérarchique et souvent non linéaire) en utilisant le plus souvent l'algorithme connu sous le nom *rétro-propagation*. Ces ajustements sont calculés en réduisant l'erreur qui est alors la racine carrée de la moyenne d'erreurs (*RCME*) définie par :

$$RCME = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{s} (y - \hat{y})^2}{n \times s}},$$

où y est la valeur réelle, $\hat{y}$ est la valeur prédite, et n et s sont respectivement le nombre d'observations et le nombre de sorties.

L'exemple représenté sur la figure B.4 montre la structure d'un modèle de type réseau de neurones qui prédit l'effort de développement d'un système à partir d'un ensemble d'attributs (métriques) de spécification. Le réseau utilise l'algorithme de rétro-propagation afin de déterminer les poids qui permettent de réduire au minimum l'erreur prédictive.

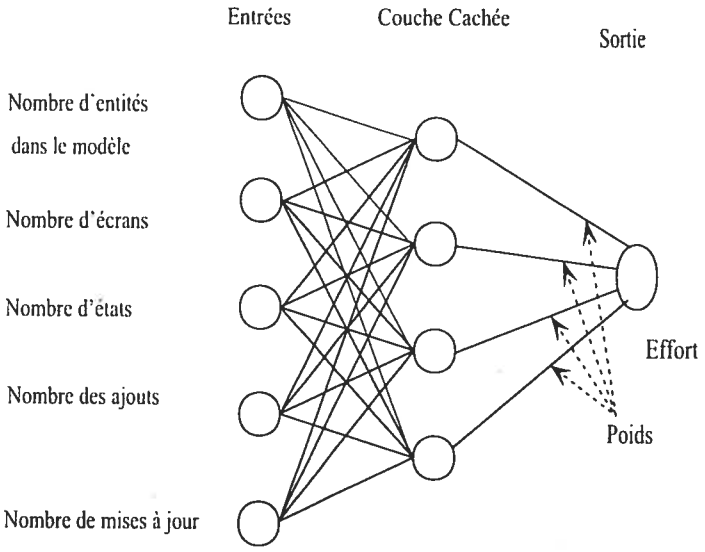


FIG. B.4 – *Une structure d'un réseau de neurones pour la prédiction de l'effort.*

Un inconvénient assez évident des réseaux de neurones est relié à leur nature « boîte noire », où les entrées et les sorties sont visibles, alors que le processus de passage des unes aux autres est caché.

Une voie intéressante pour éviter ce problème est l'utilisation des réseaux hybrides de neurones flous. Ces derniers héritent des avantages des réseaux de neurones, comme le comportement non linéaire et la bonne capacité de généralisation, et préservent le sens sémantique du modèle offert par la logique floue.

B.2.2 Arbres de régression et de décision (AD)

Bien que les arbres de régression et ceux de décision soient basés sur le même principe, ils ont deux buts différents. Les arbres de régression peuvent être utilisés quand la valeur de sortie à prédire est dans l'échelle intervalle[3], alors que les arbres de décision, également connus sous le nom d'arbres de classification, sont employés pour prédire la classe de sortie d'une observation : il s'agit alors de l'échelle nominale ou ordinale (catégorielle). Les deux algorithmes fonctionnent en apprenant les règles nécessaires pour classifier les cas pris dans un ensemble de données d'entraînement. Pour plus d'information sur les arbres de décision ainsi que sur leur utilisation, voir le chapitre 5 ou encore les travaux dans [Breiman et *al.*, 1993], [Selby et Porter, 1988] et [Porter et Selby, 1990a].

La figure B.5, montre un exemple d'arbre de régression binaire dont les noeuds sont des attributs d'entrée (comme le *Nombre d'écrans*) et les arcs sont des expressions booléennes (comme le *Nombre d'écrans* > *10*) et les feuilles sont les valeurs de sortie qui représentent différentes moyennes du temps de développement (comme la *Moyenne* de 112*h*).

3. Voir le livre de Zuse[1998] pour plus de détails sur les échelles de mesure

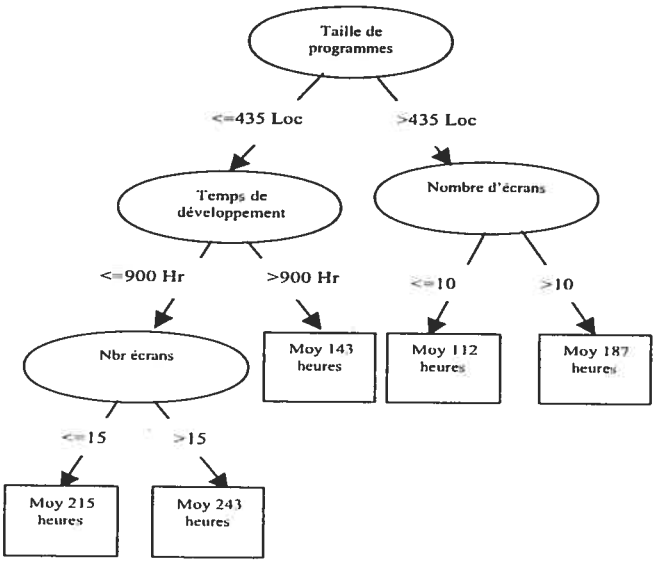


FIG. B.5 – *Arbre de régression pour la prédiction du temps de développement.*

Afin de créer l'arbre, l'algorithme d'apprentissage cherche quels attributs d'entrée peuvent être employés pour mieux classifier les données et construire itérativement l'arbre. Ces attributs formeront des noeuds et, si nécessaire, ces noeuds seront éclatés. Les algorithmes d'éclatement en deux sont les plus utilisés (des algorithmes pour subdiviser les intervalles dans le cas de notre exemple). Une expérience menée par Srinivasan et Fisher [1995] a permis de constater qu'en utilisant l'erreur résiduelle moyenne comme mesure de la capacité prédictive, la technique d'arbres de régression utilisée pour estimer l'effort est plus performante que COCOMO ou SLIM, mais moins performante que les réseaux de neurones et les points de fonction.

B.2.3 Raisonnement à base de cas (RBC)

Le raisonnement à base de cas est une technique basée sur la mémorisation d'observations. À chaque fois qu'on est confronté à une nouvelle observation, une recherche dans la base des cas (les observations enregistrées) est effectuée pour trouver les observations les plus proches de celle-ci. En employant les valeurs de sortie associées aux cas trouvés, on dérive une valeur de sortie pour le nouveau cas. Un système de raisonnement à base de cas a un pré-processeur pour préparer les données d'entrée, une fonction de similitude pour rechercher les cas similaires et une certaine fonction de combinaison de

cas pour dériver la valeur de sortie du nouveau cas. Si nécessaire, la base des cas est mise à jour par l'ajout du nouveau cas [Aha, 1991]. Les systèmes de raisonnement à base de cas sont conçus pour imiter le processus d'un expert prenant une décision basée sur ses expériences précédentes [Vicinanza et *al.*, 1991; Mukhopadhyay et *al.*, 1992]. La figure B.6 montre la structure générale d'un système RBC.

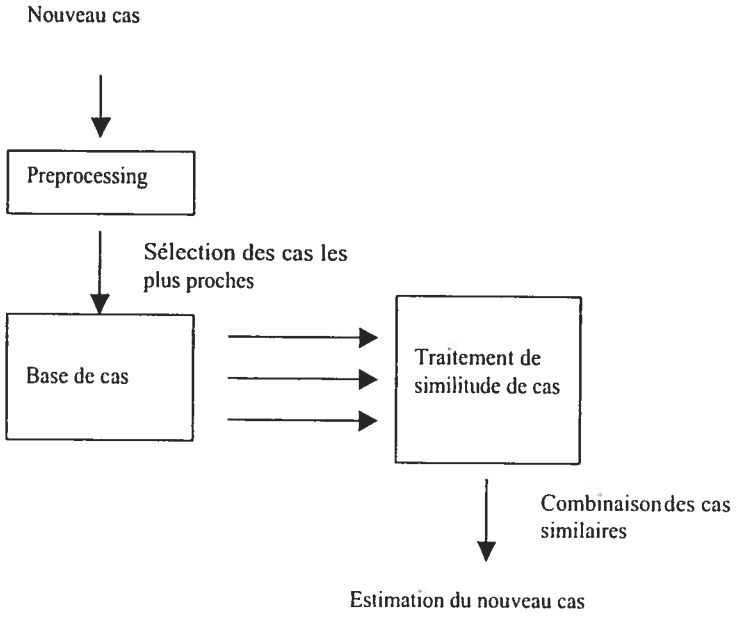


FIG. B.6 – *Le processus de classification du raisonnement à base de cas.*

Ce type de systèmes souffre de certains problèmes. Comme il est indiqué par Breiman et ses collègues [1993], ils sont intolérants au bruit. Les auteurs montrent également que la fonction de similitude utilisée exerce une forte influence sur la performance du raisonnement [Aha, 1991]. Ce problème fait de la création d'un système de raisonnement à base de cas une tâche non triviale.

Dans une expérience menée par Mukhopadhyay et ses collaborateurs [1992], la performance d'un système RBC est comparée, simultanément, à celle d'un expert humain et à celles des modèles standards de points de fonction (PF) et COCOMO. Le système RBC ESTOR, construit pour estimer l'effort, et l'expert humain ont été limités à utiliser les entrées standards des points de fonction et du modèle COCOMO. Les résultats ont montré une capacité prédictive du RBC meilleure que celles de PF et du modèle COCOMO mais légèrement inférieure à celle de l'expert humain. Ainsi, les auteurs ont conclu que la technique de raisonnement à base de cas mérite plus d'études en raison de ses résultats

encourageants.

B.2.4 Réseaux Bayésiens (RB)

Un réseau bayésien (RB) est un graphe orienté, toujours acyclique, constitué de noeuds et d'arcs. Les noeuds représentent des attributs généralement discrets (il y a quelques extensions aux réseaux bayésiens pour représenter les attributs continus mais avec des limitations). Les arcs sont orientés et représentent des liens de dépendance directe (de causalité) entres les noeuds, l'absence d'arc ne renseigne que sur l'inexistence d'une dépendance directe. Certains noeuds n'ont pas de noeuds parents (les attributs qu'ils représentent ne dépendent d'aucun autre attribut) : ils sont appelés *noeuds racines*. Le grand avantage des réseaux bayésiens est de permettre de modéliser des relations non déterministes. Ceci grâce au concept de **table de probabilités conditionnelles** (TPC) associée à chaque noeud. Pour comprendre le contenu de ces tables, nous considérons la modélisation du problème des retards à l'école[4]. Supposons que le retard d'un élève peut être causé par un mauvais état de santé ou par un retard du bus, alors que le retard d'un enseignant ne peut être causé que par un mauvais état de santé (voir figure B.7).

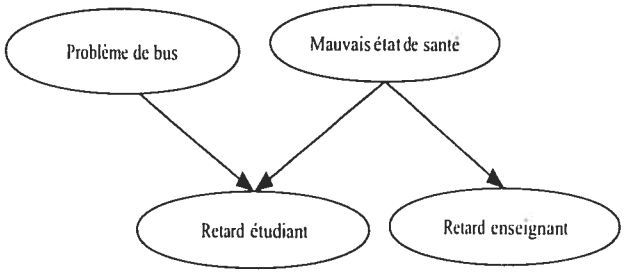


FIG. B.7 – *Un réseau bayésien modélisant le problème des retards à l'école.*

Quatre variables sont représentées dans ce réseau, qui prennent seulement l'une des deux valeurs *Vrai* et *Faux*. Le tableau B.1 montre ce que pourrait être la table de probabilités TPC associée au noeud *Retard enseignant*. Elle modélise la dépendance entre le retard de l'enseignant et son mauvais état de santé.

Cette table est en fait la distribution de la probabilité de l'attribut *Retard enseignant*, conditionnelle à

4. Faute d'exemples simples de RB en qualité du logiciel, nous choisissons d'illustrer par un exemple facile de la vie quotidienne.

	mauvais état de santé = Vrai	mauvais état de santé = Faux
Probabilité (Retard enseignant = Vrai)	0,6	0,1

TAB. B.1 – *Table de probabilités conditionnelles (TPC)*

l'attribut *Mauvais état de santé*. Elle donne seulement la probabilité de l'événement *Retard enseignant* = *Vrai*, car *Probabilité* (*Retard enseignant* = *Faux*) = 1 − *Probabilité* (*Retard enseignant* = *Vrai*). Les tables de probabilités associées respectivement aux noeuds *Mauvais état de santé* et *Retard du bus*, ont une nature particulière. Ces noeuds n'ont pas de noeuds parents dans ce modèle, ce sont des *noeuds racines*, et se voient donc assigner des probabilités pour leurs deux valeurs *Vrai* et *Faux*, en supposant par exemple que *Probabilité* (*Mauvais état de santé* = *Vrai*) = 0,4 et *Probabilité* (*Retard du bus* = *Vrai*) = 0,1.

Dans le cas de trois attributs $x^{(j)}$,$x^{(l)}$ et $x^{(m)}$, la table TPC associée à $x^{(j)}$ contient des probabilités conditionnelles de la forme $p(x^{(j)} = V_{jk}|x^{(l)} = V_{lp},x^{(m)} = V_{mq}) = p$. C'est-à-dire que l'attribut $x^{(j)}$ dépend conditionnellement des attributs $x^{(l)}$ et $x^{(m)}$ et que la probabilité avec laquelle l'attribut $x^{(j)}$ peut prendre la valeur V_{jk} est égale à p sachant que la valeur de l'attribut $x^{(l)}$ est V_{lp} et que la valeur de l'attribut $x^{(m)}$ est V_{mq}. Les tables TPC associées aux noeuds racines $x^{(r)}$ ont des probabilités marginales de la forme $p(x^{(r)}) = V_{rk}$.

La construction d'un RB de d attributs $x^{(1)},\ldots,x^{(d)}$, repose sur *la sémantique globale* du RB exprimée par l'équation B.6.

$$p(x^{(1)},\ldots,x^{(d)}) = \prod_{j=1}^{d} p(x^{(j)}|Parents(x^{(j)})). \tag{B.6}$$

À partir de cette même équation, on peut exprimer n'importe quelle préposition de la forme $p(x^{(i)} = u|x^{(l)} = v,\ldots,x^{(m)} = w)$ en fonction d'autres probabilités conditionnelles continues dans les TPC du RB. Des algorithmes sont proposés pour faire ce genre de calcul (voir dans [Pearl, 1982] et [Kim et Pearl, 1983]). C'est en se basant sur cette dernière propriété que se fait l'inférence dans un RB. En effet, la modification des probabilités, suite à l'introduction d'une évidence (valeurs certaines d'attributs), provoque la mise à jour de toutes les probabilités des autres attributs (par propagation et calcul). Concernant l'apprentissage d'un RB à partir des observations d'entraînement, il est réalisé par la mise à jour des

probabilités conditionnelles en utilisant des algorithmes comme celui de la méthode EM (*Expectation-Maximization*) [Pearl, 2000].

Un modèle de type réseau bayésien est une représentation explicite des dépendances causales entre des attributs de la qualité du logiciel (dans notre cas). Grâce à la sémantique globale du RB (voir l'équation B.6) et à la manipulation des probabilités conditionnelles [Fenton et Ohlsson, 2000], un RB permet de combiner des données empiriques et des jugement d'experts, de représenter explicitement des connaissances incertaines de phénomènes complexes de la qualité et d'introduire des évidences empiriques [Fenton et Ohlsson, 2000; Sunita, 1999]. Pour plus de détails sur les résultats de l'utilisation du RB dans la prédiction de la qualité, nous référons à un exemple [Fenton et Ohlsson, 2000] construit pour prédire la densité des défauts et le coût de développement des bibliothèques de classes à objets.

Un cas particulier des RB est celui des classificateurs bayésiens (CB). Ce sont des structures moins complexes qui utilisent les mêmes concepts qu'un RB comme les tables TPC et le processus d'inférence. Il sont, comme leur nom l'indique, utilisés pour les problèmes de classification. Une description plus détaillée des CB est fournie dans le chapitre 6.

B.2.5 Logique floue (LF)

La logique floue a été développée suite à des critiques envers la logique classique (tout ou rien). L'affirmation principale soutenue par cette technique est que les attributs des entités dans le monde réel ne prennent pas de valeurs précises. Par exemple, un projet peut n'être ni petit, ni moyen, ni grand. Il pourrait en fait, être entre deux, il est par exemple, beaucoup plus qu'un petit projet, mais moins qu'un projet moyen. Ceci peut être représenté comme par des degrés d'appartenance respectivement à la catégorie des moyens et à celle des petits projets.

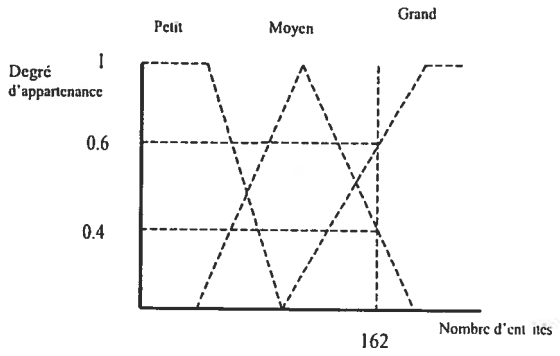


FIG. B.8 – *Fonctions d'appartenance à des ensembles flous.*

La figure B.8 montre un projet avec 162 entités qui appartient à la classe des projets moyens à un degré de 0,4 et à la classe des grands projets à un degré de 0,6.

Supposons que nous désirons prédire l'effort de développement d'un projet logiciel. Pour ce faire, plusieurs données doivent être estimées comme le nombre de fichiers, le nombre d'entités et d'autres attributs. Une telle évaluation peut être à l'origine d'une hésitation à s'engager avec des mesures précises. Dans une situation pareille, la logique floue fournit un moyen moins rigide d'engagement. Ainsi, un chef de projet peut dire qu'un projet aura un grand nombre d'entités, un nombre restreint de fichiers, et de la même manière sont évaluées d'autres variables. Ces données peuvent être représentées sous forme d'ensembles flous comme il est montré par la figure B.8. Une série de règles représentée sur la figure B.9 peut alors être employée pour dériver une certaine prédiction pour la sortie, qui est dans ce cas-ci l'« effort d'un projet ».

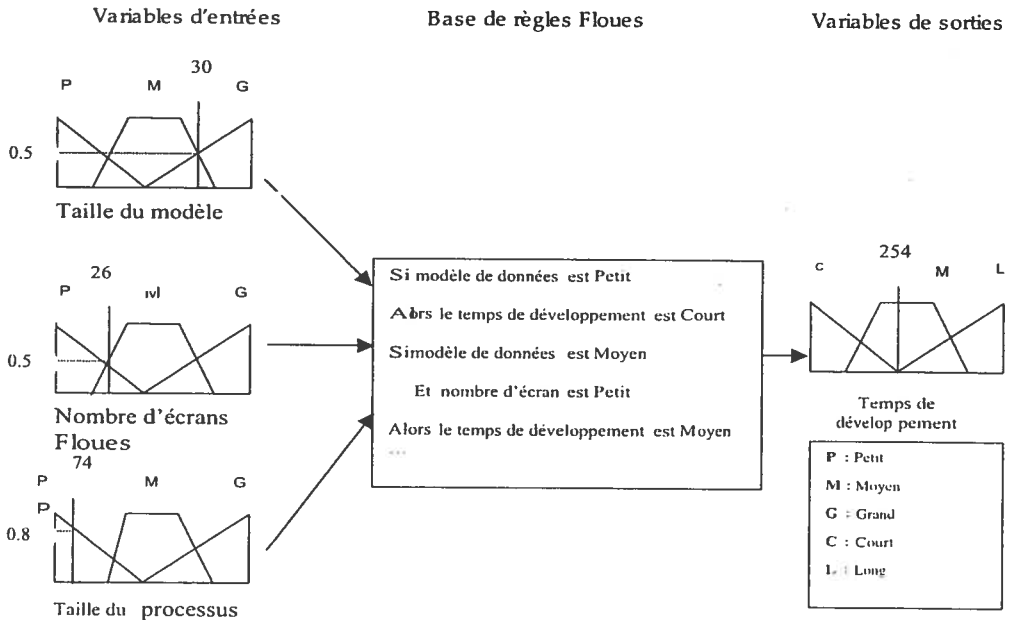


FIG. B.9 – *Système flou pour prédire la durée de développement d'un logiciel.*

Cette mesure de l'effort pourrait être « defuzzifiée »[5] en une valeur précise ou laissée comme une valeur linguistique (label ou étiquette) pour souligner qu'il s'agit seulement d'une simple estimation. Il est à noter que les systèmes flous les plus utilisés sont les classificateurs flous.

B.2.6 Systèmes basés sur les règles (SBR)

Les systèmes de règles floues sont une version élaborée des systèmes de règles précises. Un système basé sur les règles peut être simulé par un système flou. Pour cette raison, les systèmes précis sont considérés comme redondants [Ramsey et Basili, 1989; Griech et Pomerol, 1994]. Cependant, le plus grand avantage d'un système basé sur des règles précises est son adéquation avec le cas où beaucoup de variables d'entrée sont impliquées. En effet, un système basé sur les règles est organisé autour d'un ensemble de règles qui sont activées par des faits et qui peuvent activer d'autres faits, comme le montre dans la figure B.10.

5. Rendue précise après avoir été floue.

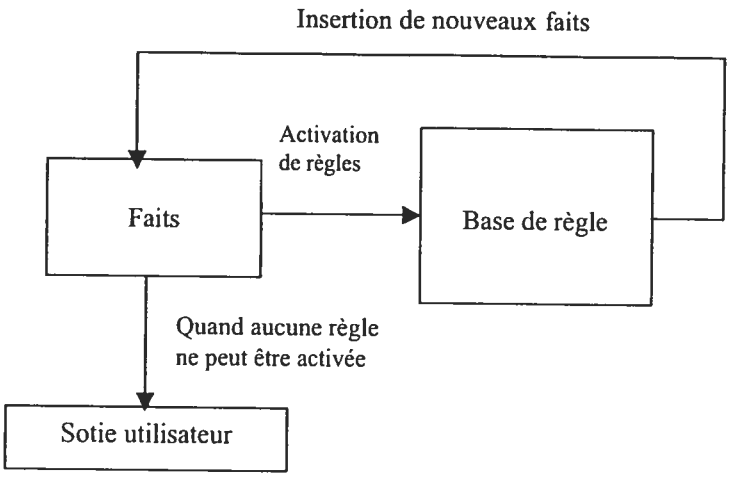


FIG. B.10 – *La structure générique des systèmes à base de règles.*

De cette façon, un enchaînement se produit entre les règles permettant leurs activations successives. Un exemple de système à base de règles peut être le suivant:

SI la *taille de module* > 40 *LOC* OU

SI la *taille de module* > 20 *LOC* ET le *temps de développement* > 2 *heures*

ALORS le module est à risque d'erreur élevé.

Même si les deux modules de tailles respectives 20 et 21 *LOC* nécessitent la même durée de développement, ils sont différenciés par la règle ci-dessus. Les deux modules sont très semblables, mais seulement le premier active la règle. Comparé à un système flou, le système à base de règles a l'inconvénient d'être rigide. En effet, toutes les conditions doivent être vraies ou fausses, sans degrés de vrai ou de faux permis.

Annexe C

Configurations des métaheuristiques

Dans cette annexe, nous présentons une synthèse des plus importantes exécutions effectuées dans le cadre de la recherche des bonnes valeurs des paramètres des différentes métaheuristiques. Les résultats de ces expériences sont organisés par métaheuristique et par paramètre. Pour chaque paramètre des bonnes valeurs sont sélectionnées pour le cas des arbres de décision et pour le cas des classificateurs bayésiens.

C.1 Algorithme génétique : Nombre des générations (T)

C.1.1 Cas des arbres de décision

JT Entraînement	T #Génért.	Temps Exécution	JT Entraînement	T #Génért.	Temps Exécution	JT Entraînement	T #Génért.	Temps Exécution	JT Entraînement	T #Génért.	Temps Exécution
56,50%	0	0	82,40%	54	3,30398333	82,49%	99	11,58255	82,88%	144	22,6314833
77,43%	10	0,19193333	82,40%	55	3,44085	82,49%	100	11,8294	82,88%	145	22,9013833
77,43%	11	0,22381667	82,40%	56	3,57871667	82,49%	101	12,06325	82,88%	146	23,1727833
77,91%	12	0,2602	82,40%	57	3,70205	82,49%	102	12,3051	82,88%	147	23,4515167
78,47%	13	0,29491667	82,40%	58	3,83775	82,49%	103	12,5204167	82,88%	148	23,7399333
78,47%	14	0,32713333	82,40%	59	3,97378333	82,49%	104	12,7457333	82,88%	149	24,01
78,47%	15	0,36368333	82,40%	60	4,1165	82,49%	105	12,9704	82,88%	150	24,3106
78,96%	16	0,39991667	82,40%	61	4,25986667	82,49%	106	13,1950667	82,88%	151	24,6102
78,96%	17	0,44113333	82,40%	62	4,41141667	82,60%	107	13,43425	82,88%	152	24,9183167
78,96%	18	0,48136667	82,40%	63	4,56915	82,60%	108	13,6829333	82,88%	153	25,2489667
78,96%	19	0,52526667	82,40%	64	4,72688333	82,60%	109	13,9248	82,88%	154	25,5520833
78,96%	20	0,56848333	82,40%	65	4,88161667	82,60%	110	14,16765	82,88%	155	25,8697
78,96%	21	0,61205	82,40%	66	5,03033333	82,60%	111	14,3802833	82,88%	156	26,1667833
78,96%	22	0,66045	82,40%	67	5,1999	82,60%	112	14,6128	82,88%	157	26,46935
78,96%	23	0,71103333	82,40%	68	5,36965	82,60%	113	14,8628333	82,88%	158	26,7752667
78,96%	24	0,7641	82,40%	69	5,53188333	82,60%	114	15,0948333	82,88%	159	27,0786833
79,06%	25	0,81851667	82,40%	70	5,70598333	82,60%	115	15,3258333	82,88%	160	27,3640667
79,06%	26	0,87843333	82,40%	71	5,87121667	82,60%	116	15,5369833	82,88%	161	27,65395
79,06%	27	0,9382	82,40%	72	6,04163333	82,60%	117	15,7558	82,88%	162	27,9520167
79,06%	28	1,00613333	82,40%	73	6,2234	82,60%	118	16,0028167	82,88%	163	28,2572667
79,06%	29	1,0754	82,40%	74	6,4025	82,60%	119	16,235	82,88%	164	28,5842167
79,06%	30	1,14566667	82,40%	75	6,58391667	82,60%	120	16,45965	82,88%	165	28,9273333
79,20%	31	1,21976667	82,40%	76	6,76601667	82,60%	121	16,6819833	82,88%	166	29,2601167
79,20%	32	1,29036667	82,40%	77	6,94311667	82,69%	122	16,9211667	82,88%	167	29,6009167
79,20%	33	1,36531667	82,40%	78	7,11436667	82,69%	123	17,16585	82,88%	168	29,914
79,20%	34	1,43841667	82,40%	79	7,28378333	82,69%	124	17,4090333	82,88%	169	30,2683167
79,20%	35	1,5142	82,40%	80	7,47171667	82,69%	125	17,6577333	82,88%	170	30,6137833
79,26%	36	1,59681667	82,40%	81	7,66	82,69%	126	17,9039333	82,88%	171	30,9931333
79,26%	37	1,6726	82,40%	82	7,84158333	82,69%	127	18,1441167	82,88%	172	31,3366333
79,65%	38	1,75038333	82,40%	83	8,03153333	82,69%	128	18,3642667	82,88%	173	31,67645
80,41%	39	1,8305	82,40%	84	8,2143	82,69%	129	18,5855833	82,90%	174	32,0283
80,41%	40	1,91845	82,40%	85	8,39473333	82,69%	130	18,8187667	82,90%	175	32,405
80,83%	41	2,00191667	82,40%	86	8,56781667	82,69%	131	19,0524333	82,90%	176	32,7955667
80,83%	42	2,08886667	82,49%	87	8,7636	82,69%	132	19,3143167	82,90%	177	33,2106667
80,93%	43	2,1795	82,49%	88	8,95238333	82,69%	133	19,5685167	82,90%	178	33,6382833
81,40%	44	2,25911667	82,49%	89	9,15885	82,69%	134	19,8254	82,90%	179	34,06525
81,60%	45	2,34525	82,49%	90	9,39518333	82,69%	135	20,08995	82,90%	180	34,4514667
81,60%	46	2,43203333	82,49%	91	9,62251667	82,69%	136	20,3498333	82,90%	181	34,8912667
81,60%	47	2,51883333	82,49%	92	9,84901667	82,79%	137	20,61605	82,90%	182	35,3808167
81,71%	48	2,61363333	82,49%	93	10,10205	82,79%	138	20,8912833	82,90%	183	35,84865
81,71%	49	2,71661667	82,49%	94	10,3587667	82,79%	139	21,1995667	82,90%	184	36,2654333
82,10%	50	2,82043333	82,49%	95	10,6042833	82,79%	140	21,49065	82,90%	185	36,6948833
82,40%	51	2,92993333	82,49%	96	10,8611667	82,88%	141	21,7684	82,90%	186	37,1634
82,40%	52	3,04926667	82,49%	97	11,1036833	82,88%	142	22,0498	82,90%	187	37,6264
82,40%	53	3,17161667	82,49%	98	11,3443667	82,88%	143	22,3414	82,90%	188	38,16585

FIG. C.1 – *Choix du nombre des générations (arbres de décision).*

C.1.2 Cas des classificateurs bayésiens

JT Entraînement	T #Génért.	Temps Exécution	JT Entraînement	T #Génért.	Temps Exécution	JT Entraînement	T #Génért.	Temps Exécution	JT Entraînement	T #Génért.	Temps Exécution
73,54%	10	0,70932417	74,37%	42	3,9266	75,81%	137	17,5232333	75,58%	105	12,2756833
73,54%	11	0,74152417	74,37%	43	4,05011667	74,51%	74	8,00926667	75,58%	106	12,4382667
73,54%	12	0,77727242	74,37%	44	4,1718	74,51%	75	8,1443	75,58%	107	12,5991667
73,54%	13	0,82777777	74,37%	45	4,2933	74,51%	76	8,27781667	75,58%	108	12,7634
73,54%	14	0,87454332	74,37%	46	4,41781667	74,57%	77	8,41468333	75,58%	109	12,9259833
73,83%	15	0,91024167	74,37%	47	4,5395	74,57%	78	8,54855	75,58%	110	13,0937167
73,83%	16	0,96544898	74,37%	48	4,66318333	74,68%	79	8,68441667	75,58%	111	13,26965
73,83%	17	1,00241667	74,37%	49	4,7867	74,80%	80	8,82095	75,58%	112	13,4223667
73,86%	18	1,12416667	74,37%	50	4,90955	74,80%	81	8,95715	75,58%	113	13,5971333
73,86%	19	1,17416667	74,37%	51	5,03205	74,80%	82	9,09218333	75,58%	114	13,74835
73,86%	20	1,26241667	74,37%	52	5,15456667	74,80%	83	9,22738333	75,64%	115	13,9094167
73,86%	21	1,35241667	74,37%	53	5,27691667	74,80%	84	9,36258333	75,64%	116	14,0556333
73,89%	22	1,49924167	74,37%	54	5,40043333	74,80%	85	9,50028333	75,64%	117	14,2041833
74,22%	23	1,61241667	74,37%	55	5,52211667	74,80%	86	9,63448333	75,64%	118	14,35575
74,22%	24	1,71241667	74,37%	56	5,64228333	74,80%	87	9,78036667	75,64%	119	14,5188167
74,22%	25	1,83241667	74,37%	57	5,76146667	74,80%	88	9,9164	75,64%	120	14,6727167
74,22%	26	1,95241667	74,37%	58	5,8813	74,99%	89	10,0522667	75,64%	121	14,8519667
74,22%	27	2,07241667	74,37%	59	6,00065	74,99%	90	10,18745	75,64%	122	15,0032
74,22%	28	2,19241667	74,37%	60	6,12115	74,99%	91	10,3223167	75,64%	123	15,1816167
74,22%	29	2,31241667	74,37%	61	6,242	74,99%	92	10,4605333	75,81%	124	15,3351833
74,22%	30	2,43241667	74,37%	62	6,36468333	74,99%	93	10,5979	75,81%	125	15,4905833
74,22%	31	2,5601	74,37%	63	6,48636667	75,33%	94	10,7349333	75,81%	126	15,6453
74,22%	32	2,68678333	74,37%	64	6,62038333	75,33%	95	10,8736333	75,81%	127	15,8200667
74,22%	33	2,8128	74,37%	65	6,7424	75,33%	96	11,012	75,81%	128	15,9908167
74,22%	34	2,94516667	74,37%	66	6,8771	75,33%	97	11,1463667	75,81%	129	16,1615667
74,22%	35	3,06918333	74,37%	67	7,0158	75,33%	98	11,2830667	75,81%	130	16,3289667
74,22%	36	3,19235	74,37%	68	7,15601667	75,58%	99	11,4172667	75,81%	131	16,50005
74,22%	37	3,31436667	74,37%	69	7,29855	75,58%	100	11,5546333	75,81%	132	16,67965
74,22%	38	3,43571667	74,37%	70	7,44043333	75,58%	101	11,69	75,81%	133	16,8367167
74,22%	39	3,5579	74,37%	71	7,5848	75,58%	102	11,8302	75,81%	134	17,0193167
74,22%	40	3,68123333	74,37%	72	7,74186667	75,58%	103	11,97275	75,81%	135	17,1792167
74,22%	41	3,80408333	74,48%	73	7,8754	75,58%	104	12,11695	75,81%	136	17,3638333

FIG. C.2 – *Choix du nombre de générations (classificateurs bayésiens).*

C.2 Recuit simulé

C.2.1 Longueur de la température (palier de la température *Nrs*)

C.2.1.1 Cas des arbres de décision

JT Entraînement	JT Test	Nrs	Temps Exécution
79,95%	83,45%	150	5
80,31%	81,14%	200	7
80,85%	81,88%	250	12
80,79%	81,88%	300	14
81,22%	81,88%	350	18
81,51%	78,55%	400	21
81,67%	79,16%	450	26
81,60%	82,29%	500	31
82,19%	83,64%	550	35
81,78%	80,25%	600	40
82,07%	84,80%	650	47
82,02%	83,03%	700	50
81,88%	79,09%	750	56
82,04%	83,31%	800	62
82,21%	81,27%	850	67
82,45%	81,34%	900	74
82,57%	80,73%	950	78
82,50%	79,57%	1000	88
82,65%	82,35%	1050	96

FIG. C.3 – *Choix de la longueur de la température de recuit(arbres de décision).*

C.2.1.2 Cas des classificateurs bayésiens

JT Entraînement	JT Test	Nrs	Temps Exécution
80,51%	82,00%	200	6
79,34%	76,55%	250	5
78,64%	76,44%	300	7
78,04%	74,12%	350	10
80,08%	81,60%	400	9
80,56%	76,65%	450	9
79,93%	82,20%	500	11
79,68%	77,66%	550	13
80,54%	83,32%	600	13
80,66%	80,79%	650	13
81,39%	78,97%	700	15
80,10%	80,59%	750	15
81,48%	82,51%	800	16
80,89%	78,88%	850	17
82,04%	81,40%	900	19
80,22%	82,91%	950	20
80,12%	82,61%	1000	21
80,57%	82,81%	1050	22
81,09%	80,79%	1100	23

FIG. C.4 – *Choix de la longueur de la température de recuit (classificateurs bayésiens).*

C.2.2 Stratégie d'acceptation de transition et Schéma de décroissance de la température

C.2.2.1 Cas des arbres de décision

STRATÉGIE GLAUBER

	JT Entraînement	JT Test	α	Temps Exécution
	72,72%	69,87%	0,9	6
	74,25%	74,55%	0,91	6
	73,51%	74,94%	0,915	6
	76,99%	77,72%	0,92	7
	78,20%	81,71%	0,925	9
	74,12%	74,09%	0,93	9
	76,48%	77,10%	0,935	10
	75,18%	78,14%	0,94	12
	76,14%	79,39%	0,945	11
	73,49%	76,52%	0,95	11
	74,55%	75,98%	0,955	12
	75,71%	76,89%	0,96	13
	76,33%	80,51%	0,965	13
	78,27%	79,92%	0,97	14
	77,44%	80,10%	0,975	14
	78,09%	78,72%	0,98	14
	78,82%	81,09%	0,985	16
	77,78%	78,93%	0,99	22
	77,98%	80,30%	0,995	25
Moyenne	76,11%	77,71%		

STRATÉGIE METROPOLIS

	JT Entraînement	JT Test	α	Temps Exécution
	71,07%	67,45%	0,9	6
	70,59%	73,69%	0,91	8
	72,26%	73,35%	0,915	7
	75,03%	74,33%	0,92	7
	75,02%	71,57%	0,925	9
	75,62%	78,56%	0,93	10
	74,77%	75,10%	0,935	10
	75,40%	76,27%	0,94	11
	75,59%	74,42%	0,945	11
	75,58%	75,18%	0,95	13
	74,04%	72,32%	0,955	12
	71,31%	71,22%	0,96	13
	73,38%	73,51%	0,965	14
	73,09%	71,62%	0,97	15
	76,25%	77,68%	0,975	14
	76,03%	77,80%	0,98	16
	74,62%	78,81%	0,985	16
	73,06%	73,77%	0,99	20
	73,18%	75,18%	0,995	24
Moyenne	73,99%	74,31%		

FIG. C.5 – *Stratégie d'acceptation de transition et Schéma de recuit (arbres de décision).*

C.2.2.2 Cas des classificateurs bayésiens

STRATÉGIE GLAUBER

	JT Entraînement	JT Test	α	Temps Exécution
	76,25%	78,88%	0,9	7
	76,58%	75,51%	0,91	7
	77,33%	80,22%	0,915	8
	77,48%	79,68%	0,92	8
	80,59%	92,50%	10	8
	73,32%	93,00%	10	9
	76,50%	76,35%	0,935	10
	77,71%	79,76%	0,94	10
	75,03%	75,80%	0,945	10
	77,24%	78,77%	0,95	10
	76,82%	80,80%	0,955	11
	76,96%	75,21%	0,96	11
	76,56%	79,60%	0,965	11
	76,22%	79,92%	0,97	12
	77,26%	78,35%	0,975	12
	77,83%	79,68%	0,98	13
	77,23%	78,33%	0,985	14
	77,86%	77,55%	0,99	16
	76,81%	79,89%	0,995	19
Moyenne	76,93%	79,99%		

STRATÉGIE METROPOLIS

	JT Entraînement	JT Test	α	Temps Exécution
	75,90%	79,05%	0,9	11
	76,21%	78,27%	0,91	10
	77,33%	77,00%	0,915	10
	75,24%	75,18%	0,92	11
	74,79%	75,22%	0,925	12
	76,62%	77,47%	0,93	12
	77,28%	78,38%	0,935	11
	75,55%	74,54%	0,94	13
	72,64%	74,09%	0,945	12
	77,33%	79,34%	0,95	14
	76,37%	77,80%	0,955	15
	73,69%	75,59%	0,96	16
	74,97%	80,34%	0,965	15
	73,98%	72,26%	0,97	12
	75,07%	76,97%	0,975	14
	77,00%	80,35%	0,98	14
	74,56%	78,22%	0,985	15
	74,80%	76,27%	0,99	17
	75,90%	76,97%	0,995	21
Moyenne	75,54%	77,02%		

FIG. C.6 – *Stratégie d'acceptation de transition et Schéma de recuit (classificateurs bayésiens).*

C.3 Recherche avec Tabous : Longueur ℓ de la liste taboue

C.3.1 Cas des arbres de décision

JT Entraînement	JT Test	Longueur Liste Taboue	Temps Exécution
81,10%	81,80%	3	10
80,94%	77,86%	4	11
80,87%	76,45%	5	10
81,63%	77,96%	6	12
81,75%	78,26%	7	12
81,13%	80,49%	8	13
81,25%	79,37%	9	12
81,38%	76,55%	10	14
80,95%	77,55%	11	11
81,14%	79,27%	12	13
81,17%	78,26%	13	14
81,31%	78,97%	14	15
81,18%	76,25%	15	14
81,49%	78,87%	16	15
81,26%	78,97%	17	15

FIG. C.7 – *Choix de la longueur de la liste taboue (arbres de décision).*

C.3.2 Cas des classificateurs bayésiens

JT Entraînement	JT Test	Longueur Liste Taboue	Temps Execution
79,92%	78,16%	3	8
78,98%	80,29%	4	8
80,31%	80,39%	5	8
80,19%	80,59%	6	7
80,27%	82,41%	7	8
80,04%	78,16%	8	8
80,46%	82,61%	9	8
79,10%	77,76%	10	9
80,04%	78,16%	11	9
80,32%	78,37%	12	10
80,04%	78,87%	13	10
80,04%	78,87%	14	11
79,40%	77,05%	15	12
80,70%	78,17%	16	13
80,16%	79,17%	17	13

FIG. C.8 – *Choix de la longueur de la liste taboue (classificateurs bayésiens).*