Fouille de motifs basée sur la programmation par contraintes - Appliquée à la validation de logiciels

Mehdi Maamar

UNIVERSITÉ D'ORAN 1 - Ahmed Ben Bella

T H È S E

pour obtenir le titre de

Doctorat

de l'Université d'Oran 1 - Ahmed Ben Bella
Mention : Informatique

Présentée et soutenue par

Mehdi Maamar

Fouille de motifs basée sur la programmation par contraintes

Appliquée à la validation de logiciels

Thèse dirigée par Yahia Lebbah et Nadjib Lazaar
Préparée à LITIO (Université d'Oran 1), LIRMM (Université de Montpellier/CNRS)

soutenue le 29 mai 2017

Jury :

Yahia Lebbah	Professeur, Université d'Oran 1, Algérie	Directeur
Karim Bouamrane	Professeur, Université d'Oran 1, Algérie	Président du jury
Moussa Benaissa	Professeur, Université d'Oran 1, Algérie	Examinateur
Hafid Haffaf	Professeur, Université d'Oran 1, Algérie	Examinateur
Sidi Mohamed Benslimane	Professeur, ESI Sidi Bel Abbes, Algérie	Examinateur
Arnaud Gotlieb	CR, Centre de recherche Simula, Norvège	Examinateur
Nadjib Lazaar	Maître de conférences, Université de Montpellier, France	Invité

Table des matières

Résumé

La découverte de motifs *fréquents fermés* est l'un des problèmes fondamentaux en fouille de données ensemblistes. Les approches récentes basées sur la programmation par contraintes (PPC) ont prouvé leur utilité et leur flexibilité. En effet, la programmation par contraintes est un cadre adéquat pour la fouille déclarative. Toutefois, l'utilisation des contraintes réifiées dans les approches actuelles, pose un sérieux problème de passage à l'échelle, face à des bases de tailles conséquentes. Nous proposons dans la première partie de cette thèse, une contrainte globale pour capturer la sémantique particulière de l'extraction de motifs fréquents fermés, sans faire appel aux contraintes réifiées ou aux variables supplémentaires. Nous présentons un algorithme de filtrage, qui maintient la consistance de domaine en un temps et espace polynomial.

Servant de cadre applicatif à la fouille de données, le test logiciel est connu pour être un processus long et coûteux. La principale préoccupation, appelée *localisation de fautes*, est d'identifier l'origine des défaillances. Dernièrement, plusieurs techniques ont été proposées, visant ainsi à développer des outils efficaces pour la localisation, en se basant sur un degré de suspicion, que chaque technique calcule pour chaque instruction d'un programme.

Nous proposons dans un second temps, une approche déclarative, basée sur la fouille de données ensemblistes pour assister la localisation de fautes. L'approche tire profit de l'extraction de données sous contraintes, offrant la possibilité et l'avantage de raisonner simultanément sur des ensembles d'instructions. Ainsi, l'approche permet de tenir compte des dépendances existantes, ce qui reflète la notion de *motif suspect*. Nous formalisons le problème de la localisation comme un problème d'extraction des k meilleurs motifs fermés, satisfaisant un ensemble de contraintes, modélisant les instructions les plus suspectes. Par la suite, nous exploitons la contrainte globale pour l'extraction de motifs au service de la localisation, ce qui permet une amélioration des performances et de l'évolutivité, pour extraire les top-k motifs suspects.

Mots clés : Extraction de motifs, Motifs fréquents fermés, Programmation par contraintes, Contraintes globales, Test logiciel, Localisation de fautes.

Abstract

Discovering the set of *closed frequent* itemsets is one of the fundamental problems in Data Mining. Recent Constraint Programming (CP) approaches for declarative itemset mining have proven their usefulness and flexibility. However, the first CP model that is based on a wide set of reified constraints is not able to cope with high dimensional datasets. The first contribution of this thesis is the proposition of a global constraint for mining closed frequent itemsets with a filtering algorithm that enforces domain consistency in polynomial time and space.

Moreover, itemset mining techniques have been used in a wide range of applications. Over the last years, software testing, and specially the *fault localization* task, becomes one of the challenging application domains for data mining. The fault localization task aims to locate automatically bugs in programs. The second contribution of this thesis is that we investigate, for the first time, how the fault localization problem can be reduced to a closed frequent itemset problem. We formalize the problem of fault localization as finding the k best itemsets satisfying a set of constraints modeling the most suspicious statements. We use a CP approach to model and to solve our itemset based fault localization problem. We propose a robust CP model based on our ClosedPattern global constraint.

Finally, we conduct several experiments to evaluate and to validate our contributions with the implementation of the ClosedPattern global constraint and the implementation of a fault localization tool, named F-CPminer.

Keywords : Itemset mining, Frequent Closed Itemset, Constraint Programming, Global constraints, Software testing, Fault localization.

Première partie .

Introduction

1 - Contexte :

L'extraction de connaissances à partir des données a pour objectif la découverte des informations pertinentes, destinées généralement à une interprétation par des experts. La question cruciale aujourd'hui n'est plus *comment acquérir des données*, mais plutôt *comment les analyser efficacement*, pour en extraire de la connaissance. Par ailleurs, la connaissance est connue pour prendre différentes formes, selon les domaines et les sources. Une des branches fondamentales la plus explorée est l'extraction de données orientée *motifs*. L'extraction de motifs, s'intéresse à la découverte de régularités dans un jeu de données, pouvant amener à un raisonnement sur le comportement et les tendances de ces données. Ces motifs sont appelés *motifs fréquents*. L'exemple classique de l'extraction de motifs fréquents est l'analyse des transactions commerciales d'un super marché, qui peuvent contenir des informations, sur les produits fréquemment achetés ensemble et par conséquent sur les profils des clients.

D'autre part, plusieurs travaux en extraction de motifs ont mis en avant l'intérêt d'utiliser les contraintes, pour indiquer le type de motifs à extraire, afin de mieux cibler le processus d'extraction, suivant les besoins et les centres d'intérêt de l'utilisateur. De tels besoins, se traduisent par des requêtes de plus en plus complexes, portant à la fois sur la forme des motifs et sur leur contenu. Les algorithmes classiques [Zaki et al., 1997, Pasquier et al., 1999a, Uno et al., 2003] proposés pour mener à bien cette tâche, se sont montrés très efficaces, mais néanmoins très rigides quant à la prise en compte de nouveaux besoins que les experts peuvent exprimer. En effet, une fois les algorithmes mis en place, l'émergence de nouveaux besoins et la combinaison de ces derniers, exigent de nouvelles mises en œuvre de ces algorithmes. Un tel constat, a incité la mise en place d'un cadre générique pour une fouille *déclarative*. La programmation par contraintes est un cadre générique adéquat, pour modéliser et résoudre les problèmes d'extraction de motifs, de manière déclarative. Effectivement, la PPC repose sur le mécanisme de filtrage et de propagation qui permettent d'interdire certaines valeurs ou des combinaisons de valeurs, qui ne répondent pas aux requêtes exprimées sur les motifs. Les contraintes globales forment une classe particulièrement efficace dans la PPC, permettant d'avoir un mécanisme de raisonnement sur toute la structure du problème. Dans cette perspective, plusieurs travaux ont récemment été entrepris, permettant d'exprimer et de combiner une large panoplie de contraintes qui portent sur les motifs [De Raedt et al., 2008, Khiari et al., 2010, Ugarte et al., 2015, Kemmar et al., 2015]. La clé de leur succès, est l'aspect déclaratif permettant à l'utilisateur d'ajouter/supprimer des contraintes, sans avoir besoin de développer une nouvelle approche. Néanmoins, la question de l'efficacité de ces approches en termes de performances reste discutable, étant donné que certaines de ces approches actuelles pour l'extraction de motifs [De Raedt et al., 2008, Guns et al., 2011] reposent sur l'utilisation d'un grand nombre de contraintes réifiées. Ce nombre équivaut au chargement en mémoire de l'équivalent de trois bases de transactions, ce qui en pratique handicape considérablement ce type d'approches, pour le traitement de bases contenant un grand nombre d'entrées.

Par ailleurs, le succès de la fouille de données d'une façon générale, a été mis en avant

par les nombreux champs d'application de cette activité. Un des domaines d'application prometteur de l'extraction de motifs, est le domaine de la vérification et la validation des logiciels, particulièrement, le *test logiciel*. Le test logiciel est connu pour être un processus long et coûteux. En effet, les trois activités du test logiciel, la *détection*, la *localisation* et la *correction*, consomment pas moins de 50% du budget total du développement et de maintenance d'un système. Parmi ces trois phases, l'aide à la localisation de fautes dans les programmes à partir de traces d'exécution, est une question cruciale lors de la mise au point et du test de logiciels. En effet, lorsque un programme contient des erreurs, analyser une trace d'exécution qui est souvent longue, se révèle d'un intérêt très limité pour le programmeur qui doit débugger son programme. La localisation des portions de code qui contiennent des erreurs, est donc souvent un processus difficile et coûteux, même pour des programmeurs expérimentés. Au cours de la dernière décennie, plusieurs techniques automatisées ont été proposées pour répondre à ce problème, en retournant des ensembles d'instructions de plus en plus précis, à examiner [Renieres and Reiss, 2003, Jones and Harrold, 2005, Eric Wong et al., 2010].

Parmi ces techniques, certaines ont commencé à mettre à profit la force de la fouille de données, au service de la localisation. Dans [Cellier et al., 2009], Cellier et al. proposent un processus de fouille de données, nommé DeLLIS, qui calcule des clusters d'instructions du programme et montre les dépendances entre ces instructions. Un calcul est ensuite mené, pour trouver toutes les différences entre les *traces d'exécution* et en même temps donner un ordre partiel de ces différences. Nessa et al. [Nessa et al., 2008] proposent de générer des sous-séquences d'instructions de longueur N, appelées N-grams, à partir des traces d'exécution. Ainsi, les traces d'exécution erronées sont examinées pour trouver les N-grams avec un taux d'occurrence supérieur à un certain seuil. Une analyse statistique est ensuite conduite pour déterminer la probabilité conditionnelle qu'une exécution échoue, sachant qu'un certain N-gram apparaît dans sa trace.

L'objectif majeur de toutes ces techniques est d'exploiter au mieux, les éléments d'un programme sous test et les dépendances qui peuvent exister entre ces éléments, pour accroître la qualité de localisation

2 - Objectifs et motivations :

Les travaux menés dans cette thèse sont motivés par un objectif double, qui met en relation à la fois l'extraction de motifs, le test logiciel et la programmation par contraintes :

i) **Améliorer le cadre de l'extraction déclarative des motifs :**

Bien que les approches basées sur la programmation par contraintes pour l'extraction de motifs se sont montrées attractives, particulièrement le cadre développé par l'équipe de *Luc De Raedt* en 2008 [De Raedt et al., 2008] et qui a aboutit suite à la thèse de Tias Guns [Guns, 2012] au projet CP4IM. Cette approche repose sur un nombre important de contraintes réifiées et des variables auxiliaires, pour représenter les transactions, ce qui ajoute une dimension au problème. Ceci remet clairement en question son passage à l'échelle, pour traiter des bases de

données de grandes tailles. Nous avons donc travaillé sur une contrainte globale, permettant de modéliser efficacement le problème de l'extraction de motifs. Ce travail a également été motivé par un récent travail [Kemmar et al., 2015] où les auteurs ont montré l'intérêt d'une contrainte globale, pour l'extraction de séquences.

ii) **Exploiter les régularités dans les exécutions d'instructions d'un programme pour la localisation de fautes :**

Les approches les plus connues [Jones and Harrold, 2005, Abreu et al., 2007, Yoo, 2012] dans la localisation de fautes, sont des approches statistiques qui se basent sur des fonctions de score, pour évaluer le caractère suspect de chaque instruction dans le programme. Ce type de fonction exploite les occurrences des instructions dans les traces négatives et positives. Toutefois, l'inconvénient de ces techniques est que les régularités entre les exécutions d'instructions sont ignorées, puisque chaque instruction est évaluée individuellement. De ce fait, nous avons travaillé sur une approche basée sur l'extraction déclarative de motifs, permettant de capturer et d'évaluer simultanément, un ensemble d'instructions, ce qui nous permet d'exploiter les dépendances qui peuvent exister entre les instructions exécutées ensemble. La seconde motivation de ce travail est que les approches basées sur la fouille de données pour la localisation [Cellier et al., 2009, Nessa et al., 2008] utilisent les traces d'exécution qui peuvent être longues et donc difficiles à exploiter. Nos travaux exploitent une abstraction des traces d'exécution, appelée *couverture de cas de test*, plus facile à comprendre.

3 - Contributions :

Les travaux menés durant cette thèse, ont abouti aux contributions suivantes et qui touchent à la fois le domaine de l'extraction de motifs et celui de la localisation de fautes :

3.1 - Une contrainte globale pour l'extraction de motifs fréquents fermés :

Dans la pratique, le modèle existant basé sur la programmation par contraintes, pour l'extraction de motifs fréquents fermés [De Raedt et al., 2008, Guns, 2012], offre une façon élégante pour exprimer une large collection de contraintes sur les motifs. Toutefois, l'encodage d'un tel modèle, qui repose sur l'utilisation des contraintes réifiées et des variables auxiliaires, pose une limitation quant au passage à l'échelle. Afin d'y remédier, nous proposons une contrainte globale nommée ClosedPattern pour résoudre efficacement le problème de l'extraction de motifs fréquents fermés, sans faire appel aux contraintes réifiées ou aux variables supplémentaires. En effet, les seules variables de décision sont celles représentant le motif recherché. Nous développons un algorithme de filtrage qui maintient la consistance de domaine en un temps et un espace polynomial.

L'aspect expérimental et pratique de la contrainte globale ClosedPattern est

évalué en utilisant (i) Les jeux de données transactionnelles du dépôt FIMI [1] dans le chapitre 4, (ii) Un cadre applicatif pour la localisation de fautes dans les programmes de la suite *Siemens* [2] dans le chapitre 6.

Publication : CP'16 [Lazaar et al., 2016]

3.2 - Modélisation du problème de localisation de fautes en extraction de motifs :

Nous formalisons le problème de localisation des fautes comme un problème d'extraction des k meilleurs motifs (top-k), satisfaisant un ensemble de contraintes, modélisant les instructions les plus suspectes d'un programme sous test. L'intérêt est de pouvoir raisonner sur un ensemble d'instructions qui forme un *motif suspect*. Ainsi, les relations entre les instructions exécutées dans les cas de test, peuvent être capturées, permettant ainsi, de fournir plus d'informations sur l'origine de la faute. L'aspect générique est particulièrement intéressant, car il nous permet de modifier le cadre selon les besoins, tel que raisonner sur une partie du programme, ou contraindre les top-k motifs à contenir (respectivement, ne pas contenir) une ou plusieurs instructions données. Le second avantage de disposer de motifs suspects est de capturer plusieurs instructions suspectes à la fois, ce qui peut être très intéressant pour la localisation de fautes multiples. Cette approche est mise en œuvre dans notre outil nommé F-CPminer.

Les expérimentations menées sur une série de programmes de la suite *Siemens*, montrent que cette approche peut offrir une localisation plus précise, comparée à des approches statistiques standard.

Publications : COSI'15 [Maamar et al., 2015], ASE Journal'16 [Maamar et al., 2016c], Workshop CP meets Verif'16 [Maamar et al., 2016a], JFPC'16 [Maamar et al., 2016b]

4 - Organisation du mémoire :

Cette thèse est organisée en deux parties. La **première partie** est consacrée à l'état de l'art des différents domaines sur lesquels portent nos travaux (Chapitre 1, 2 et 3). La **seconde partie** décrit nos contributions (Chapitre 4, 5 et 6).

Partie I

– *Le chapitre* 1 : Le premier chapitre présente les principales notions, liées à l'extraction de motifs en fouille de données. Il introduit successivement les différentes représentations possibles des bases de transactions et les représentations condensées de motifs, dont les motifs fermés qui nous intéressent de près. Nous présentons également dans ce chapitre, des motifs plus riches, appelés *ensemble de motifs*. Enfin, le chapitre dresse un petit panorama des différents algorithmes, proposés

1. http ://fimi.ua.ac.be/data/
2. http ://sir.unl.edu/portal/index.php

pour l'extraction de motifs, en expliquant particulièrement un des algorithmes le plus efficace nommé `LCM`.

– *Le chapitre 2* : Ce chapitre s'intéresse à la programmation par contraintes, en présentant les notions principales liées à un réseau de contraintes. Il illustre comment est résolu un réseau de contraintes, grâce aux mécanismes de filtrage et de propagation. Une partie de ce chapitre, est consacrée aux contraintes globales et aux contraintes réifiées. Les CSP dynamiques sont présentés brièvement. Enfin, pour conclure les deux premiers chapitres, nous présentons dans une section, une approche PPC pour l'extraction de motifs qui reposent sur les contraintes réifiées.

– *Le chapitre 3* : Dans ce chapitre, nous présentons le domaine du test logiciel. Ce chapitre introduit particulièrement la problématique de localisation de fautes et les éléments d'un programme sous test. Il présente la notion de code suspect et les mesures développées pour évaluer la suspicion. Enfin, il introduit quelques travaux proposés pour réaliser cette tâche de localisation et leurs avantages et inconvénients.

Partie II

– *Le chapitre 4* : Ce chapitre est dédié à notre première contribution, qui consiste en une contrainte globale nommée ClosedPattern pour l'extraction de motifs fréquents fermés. Nous y présentons un algorithme de filtrage qui maintient la consistance de domaine en un temps et espace polynomial. Les propriétés et l'apport pratique de cet algorithme sont évalués expérimentalement sur une série de bases transactionnelles.

– *Le chapitre 5* : Ce chapitre est réservé à notre seconde contribution. Dans lequel nous présentons une nouvelle approche nommée F-CPminer, basée sur l'extraction de motifs sous contraintes, pour la localisation de fautes. Cette approche formalise le problème de localisation des fautes, comme un problème d'extraction des k meilleurs motifs satisfaisant un ensemble de contraintes, modélisant les instructions les plus suspectes. Au niveau conceptuel, l'approche proposée fait appel à la contrainte globale ClosedPattern.

– *Le chapitre 6* : Ce chapitre, est dédié à une série d'expérimentations, sur l'approche F-CPminer. Les études portent sur l'approche F-CPminer en termes de performances et en termes de qualité de localisation. Les expérimentations menées sur une série de programmes, montrent que cette approche offre une localisation plus précise, comparée à certaines approches standards.

Pour finir, la conclusion fait le point sur les travaux menés au cours de cette thèse. Puis, les différentes perspectives envisagées sont présentées et discutées.

Deuxième partie .

État de l'art

1. Extraction de motifs ensemblistes

Ce chapitre introduit les principales notions liées à l'extraction de motifs. Nous introduisons la notion de motifs sous contraintes et la notion de base transactionnelle. Nous illustrons les différentes représentations d'une base de transactions. Nous détaillons les représentations condensées des motifs et leurs intérêts. Nous présentons les ensembles de motifs (patterns-set) qui forment une classe riche de motifs. Enfin, nous présentons brièvement quelques algorithmes d'extraction de motifs, et détaillons particulièrement l'algorithme `LCM`.

Sommaire

« *The goal is to turn data into information, and information into insight* » — Carly Fiorina

Dans l'extraction de données ensemblistes, un motif est un ensemble d'items, également appelé itemset. Ce terme provient du domaine d'analyse des paniers d'achat de supermarchés, où chaque transaction correspond à un client et chaque item dans une transaction à un produit acheté par le client. L'objectif est alors de trouver des ensembles d'articles intéressants dans ces transactions. Les modèles découverts peuvent être utilisés à des fins de marketing, pour mieux configurer les rayons des magasins, pour étudier le comportement des clients, etc.

1.1. Cadre formel et définitions

Nous commençons par présenter dans cette section, les notions d'*item*, de *motif*, de *base transactionnelle* et de *couverture* d'un motif.

Définition 1.1 (Items et Transactions)**.** *Soit $\mathcal{I} = \{i_1, ..., i_n\}$ un ensemble de littéraux (attributs) distincts appelés items. Une transaction t est un objet d'étude de données décrite par un ensemble d'items $t \subseteq \mathcal{I}$. Soit $\mathcal{T} = \{t_1, ..., t_m\}$ l'ensemble des identifiants des transactions.*

Définition 1.2 (Motif et Langage)**.** *Un motif ensembliste d'items X (itemset) est un sous-ensemble non vide de $\mathcal{I}$. Ces motifs sont regroupés dans un langage $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \setminus \emptyset$. Le langage $\mathcal{L}_\mathcal{I}$ représente l'espace de recherche de tous les motifs.*

Définition 1.3 (Base transactionnelle)**.** *Une base de données transactionnelle est l'ensemble $\mathcal{D} \subseteq \mathcal{I} \times \mathcal{T}$. La base $\mathcal{D}$ peut être représentée par une matrice, indiquant la présence ou non d'un item dans une transaction.*

La table 1.1 présente un jeu de données transactionnel $\mathcal{D}$ où chaque transaction t_i regroupe des articles décrits par des items dénotés A,...,E.

Transactions	*Items*				
t_1		B			E
t_2		B	C	D	
t_3	A				E
t_4	A	B	C	D	E
t_5		B	C	D	E
t_6		B	C	D	E
t_7	A	B	C	D	E

TABLE 1.1.: Base de données transactionnelle $\mathcal{D}$

Exemple 1.1. *Considérons la table 1.1, les ensembles $\{BE\}, \{A\}, \{ADE\}$ sont des exemples de motifs issus de cette base transactionnelle. Le langage $\mathcal{L}_\mathcal{I}$ de cette table est donné par le treillis de la figure 1.1.*

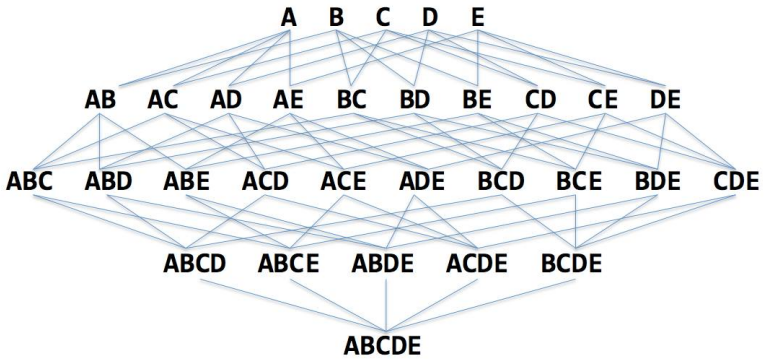

FIGURE 1.1.: Treillis de motifs avec $\mathcal{I} = \{A, B, C, D, E\}$

Définition 1.4 (Couverture)**.** *La couverture d'un motif X dans une base $\mathcal{D}$, notée $\mathcal{T}_{\mathcal{D}}(X)$, est l'ensemble de tous les identifiants de transactions dans lesquelles X apparaît :*

$$\mathcal{T}_{\mathcal{D}}(X) = \{t \in \mathcal{T} \mid X \subseteq t\}$$

Exemple 1.2. *Considérons le jeu transactionnel dans la table 1.1. Nous avons pour le motif $X = BE$, $\mathcal{T}_{\mathcal{D}}(X) = \{t_1, t_4, t_5, t_6, t_7\}$.*

La taille d'un motif X représente le nombre d'items que contient X : $taille(X) = |X|$.

Définition 1.5 (Fréquence d'un motif)**.** *La fréquence[1] d'un motif X dans une base transactionnelle $\mathcal{D}$ représente le cardinal de sa couverture $freq_{\mathcal{D}}(X) = |\mathcal{T}_{\mathcal{D}}(X)|$.*

Exemple 1.3. *Considérons la table 1.1, nous avons pour :*
$X = \{A\}$, $taille(X) = 1$ *et* $freq_{\mathcal{D}}(X) = 3$.
$X = \{BCD\}$, $taille(X) = 3$ *et* $freq_{\mathcal{D}}(X) = 5$.
$X = \{ABCDE\}$, $taille(X) = 5$ *et* $freq_{\mathcal{D}}(X) = 2$.

1.2. Contraintes sur les motifs et mesures d'intérêt

Dans l'extraction de motifs, l'espace de recherche défini par le langage $\mathcal{L}_{\mathcal{I}}$ contient $2^{|\mathcal{I}|} - 1$ motifs. Face à ce nombre exponentiel de candidats, l'utilisateur peut évaluer l'intérêt d'un motif selon certains critères. Pour se faire, l'utilisateur exprime des contraintes sur les motifs à extraire.

L'extraction de motifs sous contraintes consiste à extraire tous les motifs X de $\mathcal{L}_I$, à partir d'une base $\mathcal{D}$ satisfaisant une requête $q(x)$ (conjonction de contraintes) définissant une *théorie* [Mannila and Toivonen, 1997] : $\mathcal{T}h(\mathcal{L}_I, \mathcal{D}, q) = \{X \in \mathcal{L}_{\mathcal{I}} \mid q(X) \text{ est } vrai\}$. Il existe de nombreuses mesures, habituellement utilisées dans les contraintes, pour définir l'intérêt d'un motif [Geng and Hamilton, 2006]. Une des mesures les plus communes est la fréquence.

1. Le terme fréquence dans la littérature, peut être parfois remplacé par le terme support.

La contrainte de motifs fréquents est définie en sélectionnant les motifs dont la fréquence est supérieure à un seuil θ donné [2]. Un motif X est dit fréquent, si et seulement si $freq_{\mathcal{D}}(X) \geq \theta$.

Exemple 1.4. *Soit la table 1.1, considérons la requête suivante* $q(X) = freq_{\mathcal{D}}(X) \geq 5 \wedge taille(X) = 2$. *Les motifs en sortie sont :* BE, BC, BD *et* CD.

Il est à souligner que la valeur du seuil de fréquence minimale θ peut être indiquée de deux manières. La fréquence dite *absolue* où la valeur est indiquée avec une constante, quelque soit le nombre de transactions dans $\mathcal{D}$ (e.g., $\theta = v$ avec $v \in \mathbb{N}^+$) et la fréquence dite *relative* où la valeur est indiquée avec un pourcentage, par rapport à la taille de $\mathcal{D}$ (e.g., $\theta = 10\%$) [3].

Une autre contrainte très commune est la contrainte de *taille*, qui contraint le nombre d'items d'un motif X.

Anti-monotonie de la fréquence

Étant donné un motif X, la contrainte $C(X)$ est *anti-monotone* (respectivement, *monotone*) par rapport à la relation d'inclusion, si et seulement si, $\forall\ X, Y \subseteq \mathcal{I}, X \subseteq Y \implies (C(Y) \implies C(X))$ (respectivement, $(C(X) \implies C(Y))$). En d'autres termes, si une contrainte C est satisfaite par un motif X, C est satisfaite par tous les sous-ensembles (respectivement, sur-ensembles) de X. Par exemple, la contrainte de fréquence est une contrainte anti-monotone. La contrainte de taille est monotone.

Propriété 1.1. *Soit une base de transactions* $\mathcal{D}$ *avec l'ensemble d'items* $\mathcal{I}$. *Soient les deux motifs* $X, Y \subseteq \mathcal{I}$. *Ainsi, si* $X \subseteq Y$ *alors* $freq_{\mathcal{D}}(Y) \leq freq_{\mathcal{D}}(X)$.

En effet, si un motif est fréquent, alors tous ses sous-ensembles sont également fréquents. Par contraposée, si un motif est non-fréquent, alors tous ses sur-ensembles sont non-fréquents (voir la figure 1.2).

Exemple 1.5. *Soit la figure 1.2 qui illustre le treillis représentant le langage* $\mathcal{L}_I$ *avec les items* $\mathcal{I} = \{A, B, C, D, E\}$ *de la base transactionnelle 1.1. Soit le seuil* $\theta = 3$.

- BCE *est fréquent* $\implies B, C, E, BC, BE, CE$ *sont tous fréquents.*
- AD *est non-fréquent* $\implies ABD, ACD, ADE, ABCD, ABDE, ACDE, ABCDE$ *sont tous non-fréquents.*

La propriété d'anti-monotonie de la fréquence, est une propriété importante d'élagage de l'espace de recherche. En effet, un bon nombre d'algorithmes d'extraction de motifs fréquents, sont basés sur cette propriété (voir la section 1.6). Cette propriété est également capturée par une de nos règles de filtrage, de la contrainte globale ClosedPattern, présentée dans le chapitre 4.

2. La valeur du seuil est généralement fixée par l'utilisateur.
3. Dans la partie expérimentale du chapitre 4, la fréquence minimale est relative.

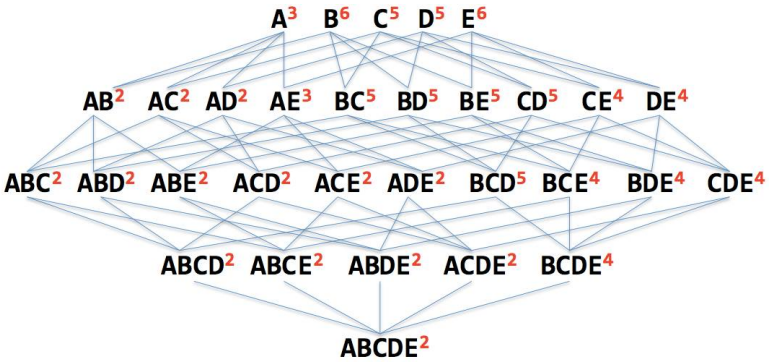

FIGURE 1.2.: Treillis de motifs de la base 1.1 avec fréquences

Mesures d'intérêt

D'autres mesures ont été proposées, pour enrichir l'information véhiculée par les motifs à extraire. Un bon nombre de ces mesures, sont liées à la fréquence. Nous donnons dans cette partie l'exemple deux autre mesures, à savoir, l'aire d'un motif et le taux d'émergence d'un motif.

Définition 1.6 (Aire d'un motif)**.** *L'aire d'un motif X dans une base $\mathcal{D}$, est définie par : $aire_{\mathcal{D}}(X) = freq_{\mathcal{D}}(X) \times taille(X)$.*

Ainsi, la contrainte d'aire minimale $aire_{\mathcal{D}}(X) \geq \alpha$ (avec $\alpha \in \mathbb{N}^+$), permet de rechercher des motifs à fois fréquents et grands, ces motifs occupent une partie importante de la base $\mathcal{D}$.

Dans certaines applications, il apparaît très approprié de relever les contrastes entre des sous-ensembles de transactions, tels que les cas de test positifs et négatifs pour la localisation de fautes (voir chapitre 5). Le taux d'émergence est une mesure de contraste très utilisée [Novak et al., 2009], elle permet d'extraire des motifs dont la fréquence varie fortement, d'un jeu de données à un autre.

Définition 1.7 (Taux d'émergence)**.** *Soit $\mathcal{D}$ une base transactionnelle partitionnée en deux sous-ensembles $\mathcal{D}_1$ et $\mathcal{D}_2$. Le taux d'émergence d'un motif X de $\mathcal{D}_2$ vers $\mathcal{D}_1$ est :*

$$Emergence_{(\mathcal{D}_1,\mathcal{D}_2)}(X) = \frac{|\mathcal{D}_2| \times freq_{\mathcal{D}_1}(X)}{|\mathcal{D}_1| \times freq_{\mathcal{D}_2}(X)} \tag{1.1}$$

Les motifs émergents [Dong and Li, 1999] sont ceux, dont le taux d'émergence est supérieur à un seuil donné. Les *Jumping Emerging Patterns* [Li et al., 2000] forment un cas particulier, de motifs émergents. Leur spécificité est d'avoir un taux d'émergence infini.

$\mathcal{V}_\mathcal{D}$

A	B	C	D	E
t_3	t_1	t_2	t_2	t_1
t_4	t_2	t_4	t_4	t_3
t_7	t_4	t_5	t_5	t_4
	t_5	t_6	t_6	t_5
	t_6	t_7	t_7	t_6
	t_7			t_7

$\mathcal{B}_\mathcal{D}$

Transaction	A	B	C	D	E
t_1	0	1	0	0	1
t_2	0	1	1	1	0
t_3	1	0	0	0	1
t_4	1	1	1	1	1
t_5	0	1	1	1	1
t_6	0	1	1	1	1
t_7	1	1	1	1	1

TABLE 1.2.: Représentation Verticale $\mathcal{V}_\mathcal{D}$ et Booléenne $\mathcal{B}_\mathcal{D}$ de la base $\mathcal{D}$

1.3. Représentations de la base transactionnelle

Les informations contenues dans une base de transactions, à savoir, la présence ou non d'un item donné, dans une transaction donnée, peuvent être représentées de différentes manières. Il existe plusieurs représentations de la base transactionnelle, chacune d'elles, offre des avantages lors de l'extraction de motifs. Les différents algorithmes d'extraction de motifs exploitent ces représentations, pour améliorer leurs performances.

Nous présentons brièvement dans ce qui suit, les représentations courantes, dans la littérature de la fouille de motifs ensembliste [Borgelt, 2012].

La représentation Horizontale $\mathcal{H}_\mathcal{D}$
C'est la représentation la plus répandue. La base est stockée sous forme d'une liste de transactions, où chaque transaction est une liste d'items qu'elle contient $\forall t \in \mathcal{T}, \mathcal{H}_\mathcal{D}(t) = \{i \in \mathcal{I} | i \in t\}$ (voir la table 1.1). Cette représentation est utilisée dans l'algorithme APRIORI [Agrawal and Srikant, 1994].

La représentation Verticale $\mathcal{V}_\mathcal{D}$
Dans cette représentation, les items sont d'abord stockés dans une liste. Ensuite, pour chaque item, les transactions le couvrant sont listées $\forall i \in \mathcal{I}, \mathcal{V}_\mathcal{D}(i) = \mathcal{T}_\mathcal{D}(i)$ (voir la table 1.2). Cette représentation est utilisée dans l'algorithme ECLAT [Zaki et al., 1997]. Cette représentation est également utilisée, dans une de nos contributions, au niveau de la contrainte globale CLOSEDPATTERN (chapitre 4).

La représentation Booléenne $\mathcal{B}_\mathcal{D}$
Les données transactionnelles sont représentées, sous la forme d'une matrice d'incidence booléenne de taille $n \times m$ avec $D_{ti} = 1$ si $(i \in t)$, 0 sinon. (voir la table 1.2). Cette représentation est utilisée dans le modèle d'extraction de motifs [De Raedt et al., 2008, Guns, 2012].

La représentation hybride $\mathcal{VH}_\mathcal{D}$
Cette représentation met à profit les informations, issues de la représentation verticale

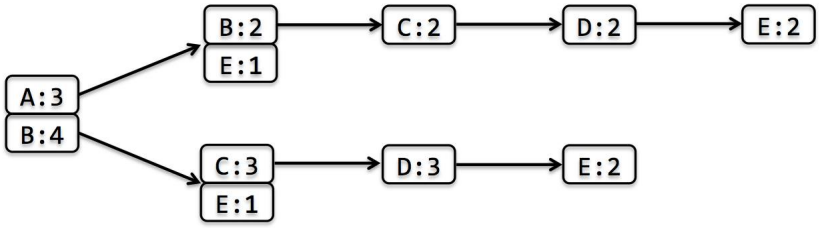

FIGURE 1.3.: Représentation arborescente $\mathcal{A}_\mathcal{D}$ de la base $\mathcal{D}$

et horizontale. Les schémas de représentation $\mathcal{V}_\mathcal{D}$ et $\mathcal{H}_\mathcal{D}$ sont maintenus en parallèle. Chaque transaction t dans $\mathcal{V}_\mathcal{D}$ dispose d'un pointeur sur son contenu $\{i \mid i \in t\}$ dans $\mathcal{H}_\mathcal{D}$. De cette manière, en parcourant la couverture $\mathcal{V}_\mathcal{D}(i)$ d'un item i donné, il est possible d'avoir le contenu d'une transaction le couvrant [4]. Cette représentation peut être très efficace, pour calculer l'intersection de transactions (calculer la couverture d'un motif). Cette représentation est utilisée dans l'algorithme LCM [Uno et al., 2004b].

La représentation arborescente $\mathcal{A}_\mathcal{D}$
Une autre représentation plus élaborée a été proposée, il s'agit de la représentation arborescente [4], appelée FP-tree, qui est essentiellement une représentation horizontale compressée. L'idée principale est de représenter un arbre préfixé, en fusionnant les transactions ayant les mêmes préfixes (voir la figure 1.3). Trier les items dans un ordre décroissant des fréquences, en commençant l'arbre par les items les plus fréquents, permet d'avoir un arbre plus compact. Cette représentation est utilisée dans l'algorithme FP-GROWTH [Han et al., 2000].

1.4. Représentations condensées de motifs

Une limite bien connue de l'extraction de motifs est que le nombre de motifs produits peut être très grand, dû d'une part à la taille de la base, et d'autre part à la redondance que peut contenir une collection de motifs. Bien que les motifs fréquents, représentent une partie fort intéressante de l'espace de recherche, leur nombre reste très grand et donc, d'un intérêt limité pour l'utilisateur. L'ensemble des motifs fréquents est considéré comme redondant, dans le sens où il est possible de les obtenir à partir d'autres ensembles de motifs moins larges [Pasquier et al., 1999b] appelés *représentations condensées*.

Les représentations condensées de motifs ont été introduites, pour augmenter la rapidité d'exécution des algorithmes d'extraction de motifs et réduire le nombre de motifs obtenus. L'idée centrale est que ces motifs soient suffisants, pour reproduire l'ensemble des motifs fréquents. Nous présentons deux représentations condensées de motifs, à savoir les *motifs fermés* [Pasquier et al., 1999b] et les *motifs maximaux* [Burdick et al., 2005]. Nous nous intéressons particulièrement aux motifs fermés, dans la suite du manuscrit.

4. Voir [Borgelt, 2012] pour plus de détails sur cette représentation.

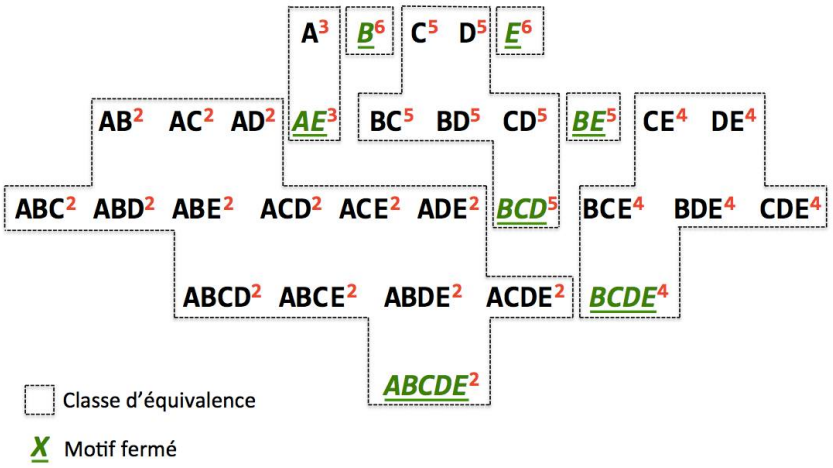

FIGURE 1.4.: Classes d'équivalences de la base 1.1 avec fréquences

1.4.1. Motifs fermés

Définition 1.8 (Motif fermé)**.** *Un motif $X \in \mathcal{L}_{\mathcal{I}}$ est fermé par rapport à la fréquence, si et seulement si, $\forall Y \supsetneq X, freq_{\mathcal{D}}(Y) < freq_{\mathcal{D}}(X)$. En d'autres termes, X est fermé, si tous ses sur-ensembles ont une fréquence inférieure à la sienne.*

Les motifs fermés structurent le treillis des motifs en classes d'équivalence. Tous les motifs d'une même classe d'équivalence, ont la même couverture. Pour chaque classe d'équivalence, seul le plus grand motif au sens de l'inclusion est retenu (c'est le motif fermé). Le treillis de la figure 1.4 illustre les classes d'équivalence de la base donnée dans l'exemple 1.1.

Exemple 1.6. *Soit l'exemple 1.1, avec la contrainte $freq_{\mathcal{D}}(X) \geq 5$, nous obtenons 9 motifs fréquents suivants :*
$B\langle 6\rangle$ [5], $C\langle 5\rangle, D\langle 5\rangle, E\langle 6\rangle, BC\langle 5\rangle, BD\langle 5\rangle, CD\langle 5\rangle, BE\langle 5\rangle, BCD\langle 5\rangle$.
Nous avons :
$C, D, BC, BD, CD \subset BCD$ *et tous ont la même fréquence (ils appartiennent à la même classe d'équivalence)* $\Rightarrow$ *seul BCD est retenu comme motif fermé.*
Les motifs $E, BE, BCDE$ ont des fréquences différentes $\Rightarrow$ *tous sont fermés.*

Nous présentons dans ce qui suit, la notion d'*extension propre*. Cette propriété est étroitement liée aux motifs fermés. Nous présentons également la notion de *clôture*, sur laquelle est basée l'algorithme LCM pour extraire les motifs fermés.

Définition 1.9 (Extension propre)**.** *Un motif non vide Y est une extension propre d'un autre motif X, si et seulement si, $\mathcal{T}_{\mathcal{D}}(X \cup Y) = \mathcal{T}_{\mathcal{D}}(X)$. En d'autres termes, $freq_{\mathcal{D}}(X) = freq_{\mathcal{D}}(X \cup Y)$.*

5. La valeur entre $\langle . \rangle$ indique la fréquence du motif

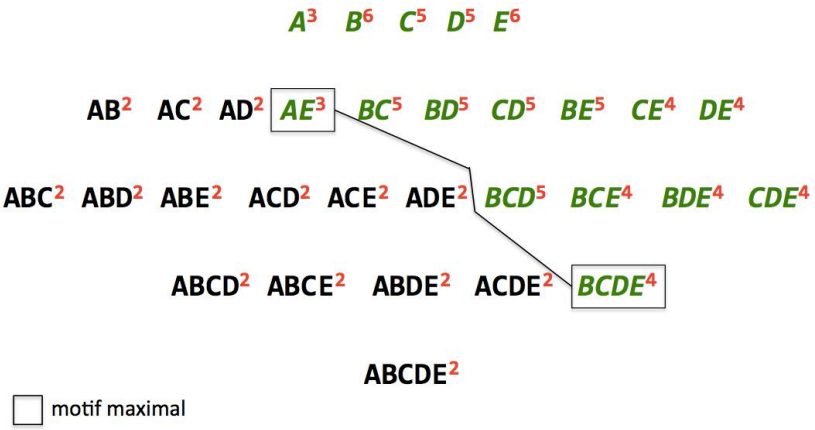

FIGURE 1.5.: Bordure positive formée par les motifs maximaux

Cette propriété est au cœur de l'extraction de motifs fermés [Wang et al., 2003], elle établit que, si un motif Y est une extension propre de X et que aucun sur-ensemble de Y n'est une extension propre de X, alors $X \cup Y$ forme un motif fermé. En d'autres termes, un motif fermé, est un motif qui n'a pas d'extension propre. Cette propriété est au cœur d'une de nos règles de filtrage, dans la contrainte globale CLOSEDPATTERN, proposée dans le chapitre 4.

Exemple 1.7. *Soit la table 1.1, le motif D est une extension propre du motif BC, puisque, $\mathcal{T}_\mathcal{D}(BC) = \mathcal{T}_\mathcal{D}(BC \cup D) = \{t_2, t_4, t_5, t_6, t_7\}$. Par conséquent, le motif BC ne peut pas être un motif fermé.*

Soit $\mathcal{S} \subseteq \mathcal{T}$ un sous-ensemble de transactions, $\mathcal{I}_\mathcal{D}(\mathcal{S})$ est l'ensemble des items communs à toutes les transactions de $\mathcal{S}$. Ainsi, $\mathcal{I}_\mathcal{D}(\mathcal{S}) = \bigcap_{t \in \mathcal{S}} t$.

Exemple 1.8. *Soit la table 1.1, nous avons $\mathcal{T}_\mathcal{D}(A) = \{t_3, t_4, t_7\}$, par conséquent, $\mathcal{I}_\mathcal{D}(\{t_3, t_4, t_7\}) = AE$.*

Définition 1.10 (Clôture)**.** *La clôture d'un motif X dans $\mathcal{D}$, notée $Clo(X)$, est l'ensemble des items communs dans la couverture de $\mathcal{T}_\mathcal{D}(X)$. Ainsi, $Clo(X) = \mathcal{I}_\mathcal{D}(\mathcal{T}_\mathcal{D}(X))$. Un motif X est fermé si et seulement si, $Clo(X) = X$.*

1.4.2. Motifs maximaux

Nous présentons brièvement dans cette partie, les motifs maximaux.

Définition 1.11 (Motif maximal)**.** *Un motif $X \in \mathcal{L}_\mathcal{I}$ est maximal par rapport à la fréquence, si et seulement si, $\forall Y \supsetneq X, freq_\mathcal{D}(Y) < \theta$ où, θ est le seuil de fréquence minimale. En d'autres termes, X est maximal si aucun de ses sur-ensembles n'est fréquent.*

Les motifs maximaux forment une *frontière*, dans le treillis de motifs. Cette frontière est appelée *frontière positive*, car au delà de cette frontière, aucun motif n'est fréquent. La figure 1.5 illustre les motifs maximaux pour l'exemple de la table 1.1 avec $\theta = 3$.

Exemple 1.9. *Prenons la table 1.1, soit $\theta = 3$. Le motif $BCD\langle 5\rangle$ est fréquent mais pas maximal car $BCD \subset BCDE$ et $BCDE$ est fréquent. Par ailleurs le motif $AE\langle 3\rangle$ est maximal car tous le motifs $(ABE, ACE, ADE, ABCE, ABDE, ACDE, ABCDE)$ incluant AE ne sont pas fréquents.*

Remarque : Les motifs maximaux, notés $\mathcal{M}$, sont moins nombreux que les motifs fermés, notés $\mathcal{C}$. Par définition, tout motif maximal est nécessairement fermé ($\mathcal{M} \subseteq \mathcal{C}$). Retrouver l'ensemble des motifs fermés à partir des motifs maximaux, consiste à calculer tous les maximaux pour chaque seuil θ possible, $\mathcal{C} = \bigcup_{i \in \theta} \mathcal{M}_i$

Motifs fréquents à partir des représentations condensées

Les représentations condensées sont considérées comme intéressantes, par le fait qu'elles permettent de régénérer l'ensemble des motifs fréquents. Néanmoins, ces représentations diffèrent dans la qualité de l'information, qu'elles permettent de reproduire. Il existe deux classes de représentations condensées, celles dites *exactes* et celles dites *approximatives*. Une représentation exacte permet de retrouver, l'ensemble des motifs fréquents, avec leur valeurs exactes de fréquence. Tandis qu'une représentation approximative permet de retrouver, uniquement les motifs fréquents. Les motifs fermés forment une représentation exacte, alors que les maximaux forment une représentation approximative.

a)- À partir des motifs maximaux

Régénérer les motifs fréquents à partir des motifs maximaux, consiste à calculer, tous les sous-ensembles possibles de motifs maximaux. En effet, ceci est une conséquence directe de la propriété 1.1 d'anti-monotonie de la contrainte de fréquence, à savoir, tous les sous-ensembles d'un motif fréquent, sont fréquents.

Exemple 1.10. *Soit la table 1.1 avec $\theta = 3$, les motifs maximaux sont AE et $BCDE$. L'ensemble de tous les motifs fréquents avec $\theta = 3$, sont tous des sous-ensembles de AE et $BCDE$ (voir la figure 1.5). Par ailleurs, l'utilisateur peut approximer la valeur de la fréquence d'un motif fréquent X par rapport à celle du motif maximal Y tel que $X \subset Y$ sachant que, $freq_{\mathcal{D}}(X) \geq freq_{\mathcal{D}}(Y)$.*

b)- À partir des motifs fermés

Il apparaît plus intéressant dans certains cas, de régénérer les motifs fréquents avec les valeurs exactes de leur fréquence. Pour se faire, considérons le motif fréquent X dont on veut calculer la fréquence, deux cas sont alors possibles :

- X est fermé. Dans ce cas, sa fréquence est connue.
- X n'est pas fermé. Dans ce cas, la fréquence de X est égale à la plus grande fréquence, parmi celles de ses sur-ensembles fermés.

$$freq_{\mathcal{D}}(X) = max\{freq_{\mathcal{D}}(Y) \mid Y \ est \ fermé \wedge X \subset Y\}.$$

Exemple 1.11. *Soit la table 1.1 avec $\theta = 3$. Les motifs fermés avec fréquences sont :*

$$B\langle 6\rangle, E\langle 6\rangle, AE\langle 3\rangle, BE\langle 5\rangle, BCD\langle 5\rangle, BCDE\langle 4\rangle, ABCDE\langle 2\rangle$$

Soit le motif fréquent BD, ce motif n'est pas fermé. Nous avons alors :

$$freq_{\mathcal{D}}(BD) = max\{freq_{\mathcal{D}}(BCD), freq_{\mathcal{D}}(BCDE), freq_{\mathcal{D}}(ABCDE)\} = 5.$$

1.5. Ensemble de motifs

En pratique, l'utilisateur est souvent intéressé par la découverte de motifs plus riches, satisfaisant des contraintes, portant sur un ensemble de motifs (patterns-set) et non plus un seul motif. Ces contraintes sont définies, comme étant des contraintes sur des ensembles de motifs [De Raedt and Zimmermann, 2007, Rojas et al., 2014] ou sur des motifs n-aires [Khiari et al., 2010]. Un type de ces motifs est le top-k motifs. L'extraction des top-k motifs (i.e. les k meilleurs motifs selon une fonction de score) est une voie prometteuse en fouille de données sous contraintes, permettant de produire des ensembles de motifs intéressants [Crémilleux and Soulet, 2008].

Définition 1.12 (top-k motifs)**.** *Soit une mesure d'intérêt m et k un entier. Le top-k motifs est l'ensemble des k meilleurs motifs par rapport à la mesure m.*

$$top\text{-}k = \{X \in \mathcal{L}_{\mathcal{I}} \mid \nexists Y_1, ...Y_k \in \mathcal{L}_{\mathcal{I}} : \forall 1 \leq j \leq k, m(Y_j) > m(X)\}$$

En d'autres termes, un motif X fait partie des top-k motifs, s'il n'existe pas plus de $k-1$ motifs, dont la valeur de la mesure m est meilleure que celle de X.

Exemple 1.12. *Soit la table 1.1, avec k =4 et m = fréquence. Les top-4 motifs fermés les plus fréquents (i.e., $m = freq$) sont : $B\langle 6\rangle, E\langle 6\rangle, BE\langle 5\rangle, BCD\langle 5\rangle$.*

L'extraction d'ensemble de motifs (patterns-set) est plus complexe que l'extraction d'un motif unique. En effet, cela nécessite de comparer les motifs entre eux. Plus concrètement, si $|\mathcal{I}| = n$, alors l'espace de recherche est en $O(2^n)$ dans le cas des motifs uniques, tandis que l'espace de recherche est en $O(2^{k \times n})$ lorsqu'il s'agit d'extraire k motifs [Khiari et al., 2010].

Dans le chapitre 5, nous proposons une approche qui porte sur l'extraction de top-k motifs.

1.6. Algorithmes pour l'extraction de motifs fréquents fermés

Plusieurs algorithmes ont été proposés, pour extraire les motifs fréquents fermés. Ces algorithmes exploitent diverses propriétés, telles que, l'anti-monotonie de la fréquence et l'opérateur de clôture, pour extraire efficacement les motifs en question. Une méthode naïve pour extraire les motifs fréquents fermés, est d'extraire l'ensemble des motifs, puis

sélectionner les fréquents, pour ensuite les grouper dans les classes d'équivalence, et enfin sélectionner les fermés. Cette méthode devient rapidement impraticable tant l'espace de recherche est exponentiel en taille $O(2^n)$ où n est le nombre d'items. Nous présentons brièvement dans ce qui suit, quelques algorithmes les plus connus. Particulièrement, nous nous intéressons à l'algorithme `LCM`, qui est le plus efficace et dont nous détaillons le fonctionnement.

L'algorithme Close [Pasquier et al., 1999b] est le premier algorithme, pour l'extraction de motifs fréquents fermés. Cet algorithme est principalement basé sur une recherche naïve. En effet, après avoir trouvé les motifs fréquents de taille k, Close compare la fréquence de chaque motif, avec celle de ces sous-ensembles (ceux dont la taille est inférieure à k). Si les fréquences de deux motifs correspondent, alors le sous-ensemble correspondant est retiré. Dans une seconde étape, Close calcule l'intersection des transactions où les motifs gardés dans le niveau k apparaissent, pour déterminer si ces motifs sont des motifs fermés.

L'algorithme Charm [Zaki and Hsiao, 2002] est un algorithme de recherche en profondeur d'abord, basé sur la représentation verticale de la base de transactions (voir la figure 1.2). Cet algorithme est plus efficace que Close, il utilise une recherche hybride qui lui permet d'éviter beaucoup de niveaux. En effet, Charm explore simultanément l'espace de recherche des motifs et celui des transactions dans un nouvel espace de recherche appelé *IT-tree*, où chaque nœud est une paire (motif-couverture : X-$\mathcal{T}_{\mathcal{D}}(X)$) pour identifier plus rapidement, les motifs fermés.

Les inconvénients de la plupart des algorithmes d'extraction de motifs fermés, sont (i) Ces algorithmes sont obligés de passer par beaucoup de motifs non-fermés lors de la recherche, pour énumérer les fermés. (ii) Ils nécessitent également beaucoup de mémoire, pour stocker les motifs fermés déjà trouvés lors de la recherche. (iii) Ils nécessitent plusieurs parcours de la base de transactions lors de la recherche. Tous ces inconvénients ont été solutionnés par l'algorithme `LCM`.

LCM (Linear time Closed itemset Miner)
L'algorithme spécialisé le plus populaire et le plus performant pour l'extraction des motifs fréquents fermés est l'algorithme `LCM` [Uno et al., 2003, Uno et al., 2004a, Uno et al., 2004b, Uno et al., 2005]. `LCM` utilise la représentation hybride de la base transactionnelle (voir section 1.3). `LCM` est principalement basé sur une technique appelée *prefix preserving closure extension* (ppc extension) permettant d'étendre directement un motif fermé trouvé, au prochain motif fermé [6]. Grâce à cette stratégie de branchement, `LCM` développe un arbre de recherche en profondeur d'abord dont la taille est égale au nombre de motifs fermés. Ce qui implique une énumération des motifs fermés, sans aucune duplication. `LCM` n'a également pas besoin de stocker, les précédents motifs fermés trouvés. Contrairement, aux autres algorithmes [Pasquier et al., 1999a, Pei et al., 2000, Zaki and Hsiao, 2002] l'espace mémoire utilisé par `LCM`, ne dépend pas des motifs fréquents. Il est à noter que l'algorithme `LCM` nécessite

6. Dans [Jr., 1998] les auteurs définissent une technique nommée *tail-extension* qui permet d'avoir un motif fréquent à partir d'un précédent motif fréquent trouvé.

Algorithme 1 : LCM

```
1 Données 𝒟, θ
2 Enum-Closed-Patterns(∅) ;
3 Procédure Enum-Closed-Patterns(X : motif fréquent fermé) ;
4 Pour (i = core(X) + 1 a |ℐ|) ∧ (i ∉ X) faire
5 |   Y = Clo(X ∪ {i}) ;
6 |   si X(i − 1) = Y(i − 1) alors
7 |   |   Enum-Closed-Patterns(Y) ;
8 End
```

que les items soit ordonnés.

Définition 1.13 (ppc extension)**.** *[Uno et al., 2004a] Soit un motif fermé X et $X(i) = X \cap \{1...i\}$ l'ensemble des items appartenant à X et qui ne sont pas supérieurs à i. $core(X)$ [7] est l'index minimal de X tel que $\mathcal{T}_{\mathcal{D}}(X(i)) = \mathcal{T}_{\mathcal{D}}(X)$. Un motif Y est ppc extension de X si :*

- *$i > core(X) \wedge i \notin X$.*
- *$Y = Clo(X \cup \{i\})$, Y résulte de l'opérateur de clôture sur X augmenté de l'item i.*
- *$X(i-1) = Y(i-1)$, ce qui garantit que le (i-1)-prefix de X est conservé.*

Tout motif fermé $Y \neq \emptyset$ est une *ppc extension* d'au moins un autre motif fermé X. Un tel motif fermé X est unique pour Y. Ceci garantit la complétude de l'énumération faite par LCM et permet d'énumérer les motifs fermés, sans duplication. L'algorithme 1 décrit la procédure de LCM en se basant sur la notion de *ppc extension*.

Exemple 1.13. *Soit $\mathcal{D}$ la base transactionnelle donnée dans la table 1.1 avec un support minimal $\theta = 2$. La figure 1.6 illustre l'arbre de recherche déployé par LCM pour générer l'ensemble des motifs fermés fréquents. Pour chaque nœud (motif fermé) il est nécessaire d'indiquer l'index minimal $core(X)$ ainsi que le résultat de l'opération de clôture $Clo(X \cup \{i\})$.*

Cet exemple montre clairement que LCM possède une stratégie très efficace. La taille de son arbre de recherche est égale au nombre de motifs fermés à extraire.

Complexité

Grâce à la technique *ppc extension*, la complexité en temps de LCM est bornée par une fonction linéaire sur le nombre de motifs fermés. L'algorithme LCM calcule au niveau de chaque nœud $Clo(X \cup \{i\})$. Ce calcul est fait n fois dans le pire des cas sur chaque nœud ($i = core(X) + 1$ à $|\mathcal{I}|$). La complexité en temps est donc en $O(n \times m \times \mathcal{C})$ avec $n = |\mathcal{I}|$, $m = |\mathcal{T}|$ et $\mathcal{C}$ le nombre de motifs fréquents fermés.

7. $core(\emptyset) = 0$

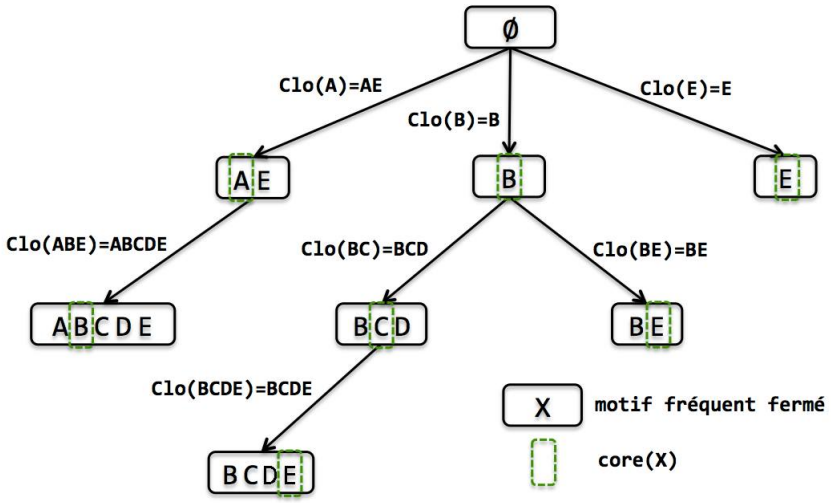

FIGURE 1.6.: Arbre de recherche de LCM

LCM reste l'algorithme le plus efficace pour l'énumération de motifs fermés. Basé sur la technique *ppc extension* qui combine le principe de tail-extension [Jr., 1998] et l'opérateur de clôture [Pasquier et al., 1999b], ce qui permet une énumération directe des motifs fermés, sans duplications et sans avoir besoin de stocker les motifs fermés déjà trouvés. Néanmoins, tous ces algorithmes sont des approches dédiées aux motifs fréquents fermés, il est donc très difficile de paramétrer ou prendre en compte de nouvelles propriétés pour enrichir les motifs produits.

1.7. Conclusion

Dans ce chapitre, nous avons présenté, un état de l'art de l'extraction de motifs. Nous nous sommes, particulièrement intéressés aux (i) représentations condensées, plus précisément, la représentation des motifs fermés, qui joue un rôle central dans nos contributions. Nous avons vu que ces motifs sont fort intéressants, car ils ne perdent aucune information et permettent de reconstruire l'ensemble des motifs fréquents avec leurs fréquences. (ii) L'algorithme d'extraction de motifs fermés LCM : Cet algorithme nous intéresse de part son efficacité et sa stratégie de recherche des motifs fermés.

Au regard des nombreuses mesures d'intérêt qui peuvent être exprimées sur les motifs (telles que, avoir des motifs fermés avec un taux d'émergence donné), ce genre de requête visant à extraire des motifs, de plus en plus riches, n'est pas géré par les algorithmes spécialisés, présentés précédemment. Un nouveau cadre a été entrepris dernièrement avec les premiers travaux connexes entre l'extraction de données et la programmation par contraintes. L'objectif est d'offrir à l'utilisateur, des méthodes déclaratives, génériques et plus expressives, pour modéliser les problèmes qu'il souhaite résoudre. L'utilisateur n'a alors plus à se soucier de l'étape de résolution, celle-ci étant prise en charge par la PPC.

2. Programmation par contraintes (PPC)

Ce chapitre regroupe les principales notions de la PPC qui seront utilisées tout au long de ce manuscrit. Nous présentons le formalisme pour définir un réseau de contraintes dans la secion 2.1 et comment s'opère le filtrage et la résolution dans les sections 2.2 et 2.3. Nous présentons ensuite les contraintes réifiées dans la section 2.4. Nous présentons également la notion de contrainte globale et de CSP dynamique. Enfin, nous présentons pour conclure les deux premiers chapitres, une approche PPC pour l'extraction de motifs basée sur les contraintes réifiées.

Sommaire

« *Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming : the user states the problem, the computer solves it.* » — E. Freuder

2.1. Cadre général

La programmation par contraintes est un paradigme de programmation *déclarative* puissant, destiné à résoudre des problèmes combinatoires [Rossi et al., 2006]. Ce paradigme permet d'exprimer le problème sous forme d'un ensemble de contraintes. En effet, l'utilisateur exprime ses contraintes en fonction des solutions qu'il désire obtenir, sur un ensemble de variables dites *variables de décision*.

La programmation par contraintes regroupe plusieurs champs, tels que l'intelligence artificielle, l'optimisation combinatoire et la logique. Un fait intéressant dans la PPC, est qu'elle sépare la partie modélisation de la résolution. En effet, l'utilisateur pose l'ensemble des contraintes de son problème (modélisation), mais ne se soucie pas de la stratégie de recherche des solutions (résolution). Le problème qui consiste à trouver une solution à un réseau de contraintes est appelé problème de satisfaction de contraintes (CSP) [Montanari, 1974].

Définition 2.1 (CSP). *[Rossi et al., 2006] Un CSP ou réseau de contraintes est défini formellement par un triplet ($\mathcal{X}$,$\mathcal{D}$, $\mathcal{C}$) tel que :*

- *$\mathcal{X}$ est un ensemble fini de variables de décision $\mathcal{X} = \langle x_1, x_2, ..., x_n \rangle$.*
- *$\mathcal{D} = \langle D_1, D_2, ..., D_n \rangle$ est l'ensemble des domaines des variables, où chaque D_i est l'ensemble fini des valeurs possibles de la variable x_i.*
- *$\mathcal{C} = \langle C_1, C_2, ..., C_m \rangle$ est l'ensemble des contraintes, où chaque contrainte est une relation exprimée sur un sous-ensemble de variables $C_i(x_{i_1}, ..., x_{i_k})$. La portée d'une contrainte C_i est le nombre de variables sur lesquelles porte C_i notée $var(C_i)$. D'un point de vue sémantique, une contrainte $C_i(x_{i_1}, ..., x_{i_k})$ est l'ensemble des tuples $\{\langle v_{i_1}, ..., v_{i_k} \rangle \mid \langle v_{i_1}, ..., v_{i_k} \rangle \in D_{i_1} \times ... \times D_{i_k}$ et $C_i(v_{i_1}, ..., v_{i_k})$ est vraie $\}$.*

Exemple 2.1. *Soit le CSP suivant avec :*
- $\mathcal{X} = \{x_1, x_2, x_3\}$
- $\mathcal{D} = \{D_1, D_2, D_3\}$, avec $D_1 = \{1, 2\}$, $D_2 = \{0, 1, 2, 3\}$, $D_3 = \{2, 3\}$
- $\mathcal{C} = \{C_1(x_1, x_2), C_2(x_1, x_2, x_3), C_3(x_1)\}$, avec :

$$C_1(x_1, x_2) : x_1 > x_2$$
$$C_2(x_1, x_2, x_3) : x_1 + x_2 = x_3$$
$$C_3(x_1) : x_1 \neq 0$$

- Avec le tuple $\langle 2, 0, 2 \rangle$ la contrainte C_2 est vraie (satisfaite).

Définition 2.2 (Instanciation). *[Rossi et al., 2006] Une instanciation I sur les variables $\mathcal{Y} = \{x_{i_1}, ..., x_{i_k}\}$, est une affectation des valeurs $v_{i_1}, ..., v_{i_k}$ aux variables $x_{i_1}, ..., x_{i_k}$, ordonnées par l'ordre lexicographique, et notée $I[x_{i_1}, ..., x_{i_k}] = \langle v_{i_1}, ..., v_{i_k} \rangle$, telle que chaque $v_{i_j} \in D(x_{i_j})$. Si une instanciation porte sur toutes les variables ($\mathcal{Y} = \mathcal{X}$), alors on parle d'instanciation complète, autrement on parle d'instanciation partielle.*

On peut noter une instanciation complète $I = \langle v_1, ..., v_n \rangle$ en déclarant uniquement les valeurs, dont l'ordre suit l'ordre lexicographique entre toutes les variables du CSP. Sinon, la notation $I[x_{i_1}, ..., x_{i_k}] = \langle v_{i_1}, ..., v_{i_k} \rangle$ est de rigueur. Soit I une instanciation partielle quelconque. La notation $I[x_{i_1}, ..., x_{i_k}]$ représente aussi la projection de I sur les variables $x_{i_1}, ..., x_{i_k}$, à savoir $\langle v_{i_1}, ..., v_{i_k} \rangle$.

Étant donné une instanciation partielle I, on peut écrire $I \in C_i$, pour exprimer que I satisfait C_i. Si l'instanciation partielle I ne porte pas sur certaines variables de C_i, l'appartenance $I \in C_i$ est assurée, si on peut compléter les autres variables non instanciées, pour former une solution de C_i. En contrepartie, si la complétion n'est pas possible, on dit que I viole C_i.

Exemple 2.2. *Soit le CSP précédent vu dans l'exemple 2.1, soit :*
- $I_1 = \langle 2, 3, 3 \rangle$ est une instanciation complète sur $\mathcal{X}$.
- $I_2[x_2, x_3] = I_1[x_2, x_3] = \langle 3, 3 \rangle$ est l'instanciation complète I_1 restreinte aux variables $\{x_2, x_3\}$.

Définition 2.3 (Solution d'un CSP)**.** *[Rossi et al., 2006] Une solution A d'un CSP $(\mathcal{X},\mathcal{D},\mathcal{C})$ est le tuple $A = \langle a_1, a_2, ..., a_n \rangle$, où $a_i \in D(x_i)$ est une instanciation de la variable x_i, telle que toutes les contraintes du CSP sont satisfaites : $\forall\ C_i \in \mathcal{C},\ A[var(C_i)] \in C_i$.*

Exemple 2.3. *Le CSP vu dans l'exemple 2.1 admet deux solutions qui satisfont toutes les contraintes C_1, C_2 et C_3 :*

- $A_1 = \langle 2, 0, 2 \rangle$
- $A_2 = \langle 2, 1, 3 \rangle$

2.2. Consistance et filtrage

Une des notions incontournables dans la PPC est le mécanisme de *filtrage*. Étant donné un état d'un CSP, le filtrage cherche des valeurs ou des combinaisons de valeurs que l'on peut interdire, sachant que l'on a la garantie que ces valeurs ne nous amènent pas vers une solution. Ces valeurs peuvent alors être filtrées des domaines. Dans cette partie, nous expliquons le processus de consistance et de filtrage.

Définition 2.4 (Consistance locale)**.** *[Rossi et al., 2006] Une instanciation I sur $\mathcal{Y}$ viole une contrainte C_i, si et seulement si, $var(C_i) \subseteq \mathcal{Y}$ et $I[var(C_i)] \notin C_i$. Une instanciation I est dite localement consistante, si elle ne viole aucune des contraintes qui impliquent les variables de $\mathcal{Y}$. Toutes les valeurs v_i de I sont alors dites localement consistantes.*

Exemple 2.4. *Soit le CSP précédent vu dans l'exemple 2.1, où :*

- L'instanciation $I_1[\mathcal{Y}] = \langle 1, 0 \rangle$ avec $\mathcal{Y} = \{x_1, x_2\}$ est localement consistante puisque, $var(C_1) \subseteq \mathcal{Y}$, $var(C_3) \subseteq \mathcal{Y}$ et C_1, C_3 sont satisfaites.

- L'instanciation $I_2[\mathcal{Y}] = \langle 2, 3 \rangle$ avec $\mathcal{Y} = \{x_1, x_2\}$ est localement inconsistante puisque, $var(C_1) \subseteq \mathcal{Y}$ et C_1 n'est pas satisfaite.

Remarque : Nous pouvons alors dire qu'une solution est une instanciation complète I sur $\mathcal{X}$ localement consistante.

Définition 2.5 (Instanciation partielle globalement consistante)**.** *Une instanciation partielle I sur $\mathcal{Y}$ est globalement consistante, si elle peut être étendue à une solution. En d'autres termes, il existe une solution A telle que $I[\mathcal{Y}] = A[\mathcal{Y}]$.*

Exemple 2.5. *Soit le CSP précédent vu dans l'exemple 2.1, soient les exemples :*

- L'instanciation $I_1[\mathcal{Y}] = \langle 1, 0\rangle$ avec $\mathcal{Y} = \{x_1, x_2\}$ est localement consistante et globalement inconsistante puisque, I_1 ne peut pas être étendue à une solution.

- L'instanciation $I_2[\mathcal{Y}] = \langle 2, 0\rangle$ avec $\mathcal{Y} = \{x_1, x_2\}$ est globalement consistante puisque, I_2 peut être étendue à la solution $I[x_1, x_2, x_3] = \langle 2, 0, 2\rangle$.

Définition 2.6 (Réseau globalement consistant)**.** *Un réseau $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ est globalement consistant, si et seulement si, pour toute instanciation partielle I sur $\mathcal{Y}$, telle que $\mathcal{Y} \subseteq \mathcal{X}$, I est globalement consistante.*

Le *filtrage* consiste à éliminer les valeurs du domaine d'une variable qui n'appartiendront à aucune solution du CSP (i.e., valeurs localement inconsistantes). Le filtrage le plus communément utilisé pour détecter les valeurs inconsistantes est le filtrage par *Consistance de Domaine* (Dans la littérature l'appellation *arc consistance* est aussi répandue).

Définition 2.7 (Consistance de domaine DC)**.** *[Rossi et al., 2006] Une contrainte C_i vérifie la consistante de domaine sur les variables $var(C_i)$, si et seulement si, pour toute variable $x_i \in var(C_i)$ et pour toute valeur $v_i \in D(x_i)$ il existe une instanciation $I[x_i] = v_i$ et I est localement consistante $I[var(C_i)] \in C_i$.*

Exemple 2.6. *Soit un CSP avec les variables $\mathcal{X} = \{x_1, x_2, x_3\}$ telles que :*

- $D(x_1) = \{0, 4, 7\}, D(x_2) = \{2, 3, 6\}, D(x_3) = \{4, 5\}$
- $C(x_1, x_2, x_3) : x_1 - x_2 > x_3$

En appliquant le filtrage par consistance de domaine sur la contrainte C, le domaine de chaque variable devient :

$$D(x_1) = \{7\},\ D(x_2) = \{2\},\ D(x_3) = \{4\}$$

Toutes les valeurs ne pouvant pas former une instanciation localement consistante, tenant compte de la contrainte $x_1 - x_2 > x_3$ sont filtrées.

Remarque : On dit que la valeur 7 de $D(x_1)$ possède un ***support*** dans $D(x_2)$ et $D(x_3)$. En d'autres termes, il existe des valeurs dans $D(x_2)$ et $D(x_3)$ combinées à $x_1 = 7$ et qui satisfont les contraintes qui impliquent ces variables.

Lorsque l'ensemble des contraintes $\mathcal{C}$ d'un réseau $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ vérifient la consistance de domaine, on dit que le réseau est *domaine-consistant* (ou *arc-consistant*), mais cela ne

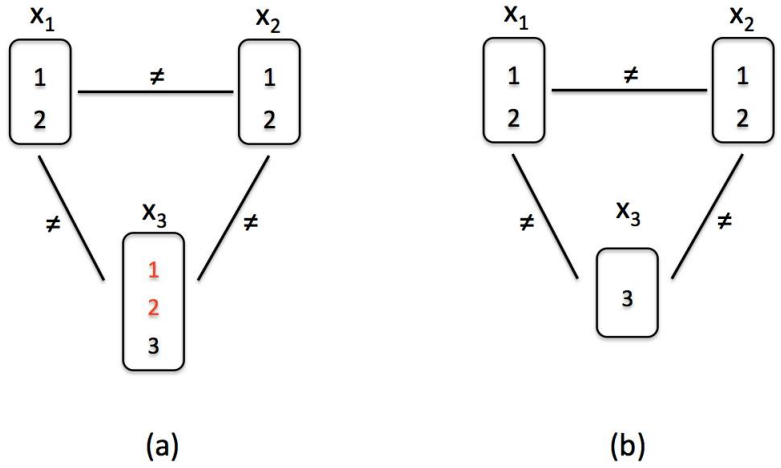

FIGURE 2.1.: Réseau domaine-consistant (a) et Réseau globalement consistant (b)

garantit pas que le réseau soit globalement consistant, puisque la consistance est une notion purement locale.

Définition 2.8 (Réseau domaine-consistant)**.** *Un réseau $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ est domaine-consistant, si et seulement si, pour toute contrainte $C_i \in \mathcal{C}$ et pour toute variable $x_i \in var(C_i)$, $\forall v_i \in D(x_i)$, il existe une instanciation I telle que $I[x_i] = v_i$ et $I[var(C_i)] \in C_i$.*

Exemple 2.7. *Soit le CSP suivant avec :*
- $\mathcal{X} = \{x_1, x_2, x_3\}$
- $D(x_1) = D(x_2) = \{1, 2\}$, $D(x_3) = \{1, 2, 3\}$
- $C_1(x_1, x_2) : x_1 \neq x_2$, $C_2(x_1, x_3) : x_1 \neq x_3$, $C_2(x_2, x_3) : x_2 \neq x_3$

En appliquant le filtrage par consistance de domaine sur chaque contrainte C_i, aucune valeur ne peut être filtrée, car il y a toujours un support pour une valeur donnée. Le réseau est domaine-consistant, mais il existe des valeurs qui n'appartiennent à aucune solution (figure 2.1-(a)).

Le CSP possède deux solutions :

$$-A_1 = \langle 1, 2, 3 \rangle$$
$$-A_2 = \langle 2, 1, 3 \rangle$$

Les valeurs 1 et 2 de $D(x_3)$ sont globalement inconsistantes (elle ne font partie d'aucune solution). Cet exemple montre que l'on peut tout à fait garantir la consistance de domaine sur le réseau, alors que, le réseau n'est pas globalement consistant.

En effet, lorsque un réseau $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ est globalement consistant, ceci implique nécessairement que le réseau est domaine-consistant, car pour toute instanciation partielle I, celle-ci est globalement consistante, donc localement consistante (voir l'exemple

de la figure 2.1-(b)). Par ailleurs, lorsque le réseau est domaine-consistant, il n'y pas de garantie qu'il soit globalement consistant, car pour toute instanciation partielle I, celle-ci est localement consistante, mais pas forcément globalement consistante (voir l'exemple 2.5).

2.2.1. Filtrage par consistance de domaine

La propriété de domaine-consistance permet d'avoir des algorithmes de filtrage assurant un bon compromis entre le temps gagné (en éliminant certaines valeurs) et celui nécessaire pour réaliser ce filtrage. On dit qu'un algorithme de filtrage associé à une contrainte C_i établit la consistance de domaine, s'il élimine toutes les valeurs des variables impliquées dans C_i et qui violent C_i. Si la consistance de domaine est établie sur toutes les contraintes du CSP, on dit, que le CSP est domaine-consistant. Plusieurs algorithmes de filtrage, ont été proposés pour rendre un CSP domaine-consistant. L'algorithme le plus connu est AC-3 [Waltz, 1972, Mackworth, 1977] qui a été proposé dans un premier temps pour les cas des contraintes binaires. La complexité de AC-3 dans un réseau de contraintes est en $O(er^3d^{r+1})$ en temps et $O(er)$ en espace, où $e = |\mathcal{C}|$, d la cardinalité du plus grand domaine et r la plus grande arité parmi les contraintes $\mathcal{C}$. L'algorithme AC-3 repose sur une fonction Revise qui supprime du domaine d'une variable x_i les valeurs qui n'ont pas de support dans une des contraintes contenant x_i.

Par la suite, plusieurs extensions de AC-3 ont été proposées pour améliorer la complexité, en insistant particulièrement sur les informations que peut stocker la fonction Revise pour améliorer le filtrage :

- AC4 [Mohr and Henderson, 1986] : Contrairement à AC-3, AC-4 stocke un certain nombre d'informations concernant les supports des valeurs dans une pré-étape. Le but est de réduire les appels de la fonction Revise sur les mêmes contraintes. AC-4 maintient l'AC en une complexité temporelle optimale $O(erd^r)$ et $O(ed^2)$ en espace.
- AC6 [Bessiere and Cordier, 1993] : AC-6 est un compromis entre AC-3 et AC-4. En effet, l'idée est de conserver la complexité optimale dans le pire des cas de AC-4 et d'arrêter la recherche de support pour une valeur le plutôt possible (dés que le premier support est trouvé, comme le fait AC-3). AC-6 maintient également une structure de données plus légère que AC-4. AC-6 a une complexité temporelle en $O(erd^r)$ et $O(ed)$ en espace.
- AC-2001 [Bessiere and Régin, 2001] : Améliore la complexité en moyenne, en améliorant la façon de stocker les supports. AC-2001 utilise des pointeurs sur les support au lieu des listes. AC-2001 maintient l'AC en une complexité temporelle optimale $O(er^2d^r)$ et $O(ed^2)$ en espace.

D'autres consistances moins fortes que la consistance de domaine peuvent être appliquées [Debruyne and Bessiere, 2001], dans le cas où le maintien de la consistance de domaine se révèle coûteux sur de larges domaines. Une des consistances moins fortes est la *consistance de bornes* [Lhomme, 1993] adoptée dans les CSP avec des domaines de

valeurs continues.

2.3. Résolution d'un CSP

La résolution d'un CSP réside dans l'établissement d'une consistance globale. Un algorithme de résolution permet d'atteindre les instanciations globalement consistantes du CSP. Nous commençons par introduire la notion de propagation dans la section suivante.

2.3.1. La propagation

A l'issue d'un filtrage opéré sur le domaine d'une variable x_i impliquée dans une contrainte C_i, il peut s'avérer intéressant d'examiner si ce changement de domaine affecte d'autres variables. Pour cela, on *réveille* toutes les contraintes C_k impliquant x_i. En effet, il est possible qu'une valeur $v_i \in D(x_j)$ telle que la variable $x_j \in var(C_k)$, étant au départ localement consistante, devienne inconsistante, suite au filtrage d'une valeur dans $D(x_i)$. En d'autres termes, la valeur filtrée de $D(x_i)$ représentait un support pour une ou plusieurs valeurs de x_j. Ce mécanisme est appelé *propagation*. Il permet de réduire successivement les domaines, grâce aux appels de filtrage sur plusieurs variables. Ceci augmente considérablement la puissance du filtrage, particulièrement si une variable est impliquée dans beaucoup de contraintes (la propagation touchera de plus en plus de variables).

Une des méthodes de résolution de base d'un CSP, est la méthode par Retour-arrière (Backtrack) [Golomb and Baumert, 1965a].

2.3.2. Algorithme par retour-arrière

Le principe consiste à générer des instanciations partielles consistantes, puis étendre ces instanciations par une nouvelle variable. De cette manière, les variables sont instanciées une à une, jusqu'à ce que toutes les contraintes soient satisfaites (i.e., génération d'une solution), ou qu'une contrainte soit violée (i.e., instanciation non consistante) menant vers un retour arrière. L'algorithme remet en cause la dernière valeur affectée, il revient sur cette variable et essaye une autre valeur de son domaine. Dans le cas où toutes les valeurs d'une variable sont testées, sans succès, l'algorithme revient sur la variable précédente. A chaque retour arrière sur une variable donnée, toutes ces valeurs sont restaurées. La complexité de cette méthode est en $O(ed^n)$, avec $e = |\mathcal{C}|$, $n = |\mathcal{X}|$ et d la cardinalité du plus grand domaine. Sur des problèmes de grandes tailles, cette méthode devient impraticable.

Des améliorations de l'algorithme par *retour-arrière* ont été proposées pour rendre le retour arrière *intelligent*. On distingue deux catégories :

- Algorithme rétrospectif (look-back) : L'idée est de tirer profit des informations contenues dans un échec, pour une part, sélectionner un meilleur point de retour et

ne pas retomber sur les mêmes échecs (ex. Backjumping [Dechter, 1990]).

- Algorithme prospectif (look-ahead) : L'idée est d'anticiper l'effet d'une instanciation, en enlevant les branches qui conduisent manifestement vers une non solution. La démarche consiste à instancier une variable x_i et tenter de supprimer les valeurs des variables suivantes $x_{i+1}...x_n$ incompatibles avec x_i. Il existe deux méthodes, (i) Vérification en avant (Forward Checking FC) [Golomb and Baumert, 1965b] : Dans cette technique dés qu'une variable x_i est instanciée à v_i, on filtre toutes les valeurs incompatibles avec $x_i = v_i$. (ii) Maintient de l'arc consistance (MAC) [Bessiere and Régin, 1996] : à la différence de FC, cette méthode est basée sur la *propagation*. A chaque fois qu'une valeur v_i est éliminée d'une variable x_i, ce filtrage est propagé à toutes les autres contraintes qui impliquent x_i.

Heuristiques de choix de variables/valeurs :

A chaque étape de l'algorithme par retour-arrière, l'algorithme doit choisir d'une part une nouvelle variable qui n'a pas encore été instanciée, et d'autre part la valeur à tester sur cette variable. Ces deux choix s'avèrent avoir une incidence non négligeable sur le reste de la procédure de recherche de solutions. Tenant compte de cela, des heuristiques de choix de *variables/valeurs* ont été proposées, dans l'intention d'améliorer l'efficacité des algorithmes de résolution.

- Heuristique de choix de variables : Deux heuristiques peuvent être envisagées, *statiques* ou *dynamiques*. Dans l'heuristique statique, on choisit un ordre sur les variables (par exemple, la variable la plus contrainte). Cet ordre est conservé tout le long de la recherche. Dans l'heuristique dynamique, l'ordre choisi peut changer d'une branche à une autre (par exemple, la variable ayant le plus petit domaine). Une des heuristiques les plus efficace est celle minimisant le rapport : (taille de domaine $D(x_i)$) / (nombre de contraintes impliquant x_i).
- Heuristique de choix de valeurs : Les choix les plus communs pour les valeurs sont, la plus petite (respectivement, grande) valeur du domaine de la variable. En pratique, trouver la meilleure valeur peut s'avérer coûteux.

2.4. Les contraintes réifiées

La programmation par contraintes offre un type de contraintes très expressif. On peut dénoter par exemple, les contraintes prédéfinies (i.e., les contraintes arithmétiques, les contraintes d'inégalité), les contraintes données en extension (liste des combinaisons de valeurs autorisées/interdites), les combinaisons logiques de contraintes.

Les contraintes réifiées [Apt, 2003] sont un type particulier de contraintes. Une contrainte réifiée associe une variable booléenne b à une contrainte C, de manière à capturer si la contrainte est satisfaite ($b = 1$), ou non satisfaite ($b = 0$). La variable b ne doit pas faire partie des variables sur lesquelles porte C.

$$b \Longleftrightarrow C$$

Ce genre de contraintes est utile pour exprimer des expressions logiques, ou exprimer qu'un certain nombre de contraintes doivent être satisfaites (min, max, exactly). En termes d'expressivité, les contraintes réifiées peuvent former une contrainte sur des contraintes, notamment, pour exprimer une disjonction de contraintes. Par exemple, la condition qu'au moins une contrainte parmi C_1 et C_2 doit être satisfaite, peut être exprimée en associant les contraintes avec les variables b_1 et b_2 et en ajoutant la contrainte $b_1 + b_2 \geq 1$.

$$b_1 \iff C_1$$

$$b_2 \iff C_2$$

$$b_1 + b_2 \geq 1$$

On retrouve les contraintes réifiées dans le modèle d'extraction de motifs présenté dans la section 2.7.

La propagation des contraintes réifiées s'opère de la manière suivante :

- Si la contrainte C est satisfaite alors on propage $b = 1$.
- Si la contrainte C est insatisfiable alors on propage $b = 0$.
- Si la variable $b = 1$ alors la contrainte C doit être satisfaite.
- Si la variable $b = 0$ alors la négation de la contrainte C doit être satisfaite.

Le dernier point est particulièrement coûteux, car capturer la sémantique de la négation d'une contrainte n'est pas triviale, mais aussi nous n'avons pas l'assurance d'avoir le propagateur de $\neg C$, ce qui réduit considérablement la force de la propagation. Une technique pour avoir la négation est de faire appel à la *négation par l'échec* [Clark, 1977].

2.5. Contraintes globales

Une contrainte globale est une contrainte qui permet de capturer une relation entre un nombre quelconque et non-borné de variables de décision. L'intérêt est d'avoir un mécanisme de raisonnement sur toute la structure d'un sous-problème, auquel est associé un algorithme de filtrage dédié. En effet, le mécanisme de filtrage présenté précédemment, possède un raisonnement sur chaque contrainte indépendamment des autres, ce qui représente une faiblesse. En d'autres termes, un CSP domaine-consistant ne garantit pas l'existence de solutions (voir l'exemple 2.8). Par conséquent, un raisonnement global peut améliorer le filtrage opéré sur les domaines.

Par ailleurs, le filtrage qu'opère une contrainte globale ne peut pas être efficacement propagé par les algorithmes génériques d'arc-consistance tels que AC-3 [Mackworth, 1977], AC-2001 [Bessiere and Régin, 2001], sauf le cas où la décomposition de la contrainte globale est Berge-Acyclique [Beeri et al., 1983], on dit

alors que la contrainte globale est AC-décomposable, ce qui veut dire que l'on peut assurer le même niveau de consistance sur cette décomposition. Par ailleurs, dans un travail récent [Cohen and Jeavons, 2016] les auteurs ont montré que lorsqu'une décomposition AC-décomposable existe, celle ci est forcement Berge acyclique. Pour les contraintes globales, des algorithmes de filtrage (propagateurs) sont construits à partir de la sémantique du problème abordé, dans le but d'établir un filtrage efficace en une complexité polynomiale en temps sur la taille du problème.

Afin d'illustrer l'intérêt que peut apporter une contrainte globale, en termes de filtrage, comparé à de simples contraintes exprimant le même problème, nous présentons une des contraintes globales[1] les plus connues : ALL-DIFFERENT [Régin, 1994]. La contrainte ALL-DIFFERENT$(x_1, x_2, ..., x_n)$ exprime le fait que toutes les valeurs affectées aux variables $x_1, x_2, ..., x_n$ doivent être distinctes deux à deux.

Exemple 2.8. *Soit le CSP suivant :*
- $\mathcal{X} = \{x_1, x_2, x_3\}$
- $\mathcal{D} = \{D_1, D_2, D_3\}$, *avec* $D_1 = D_2 = D_3 = \{1, 2\}$
- $\mathcal{C} = \{C_1(x_1, x_2), C_2(x_1, x_3), C_3(x_2, x_3)\}$, *avec :*

$$C_1(x_1, x_2) : x_1 \neq x_2$$
$$C_2(x_1, x_3) : x_1 \neq x_3$$
$$C_3(x_2, x_3) : x_2 \neq x_3$$

Le CSP ci-dessus est équivalent à la contrainte globale ALL-DIFFERENT(x_1, x_2, x_3). Ce CSP est domaine-consistant, car aucune valeur ne peut être filtrée. En d'autres termes, dans chaque contrainte $C(x_i, x_j)$, chaque valeur v_i d'une variable x_i possède un support dans le domaine $D(x_j)$ et vice versa. Par ailleurs, une analyse simple de ce CSP montre qu'il n'existe pas de solution. Ceci illustre clairement la faiblesse d'une telle représentation (voir la figure 2.2-a) avec des contraintes binaires.

La contrainte ALL-DIFFERENT(x_1, x_2, x_3) associe un algorithme de filtrage dédié, basé sur le couplage maximum (variable - valeur). La contrainte de différence est représentée par un graphe bi-parti (voir la figure 2.2-b). Le propagateur peut alors détecter que l'on ne peut obtenir un couplage maximum et donc que le CSP ne contienne pas de solution.

Plusieurs contraintes globales ont été proposées, comme la contrainte REGULAR [Pesant, 2004], la contrainte ELEMENT [Hentenryck and Carillon, 1988]. Il existe un catalogue où les contraintes globales sont regroupées [Beldiceanu et al., 2007]. Dans le chapitre 4, nous proposons une contrainte globale CLOSEDPATTERN [Lazaar et al., 2016] pour l'extraction de motifs *fréquents fermés*.

2.6. CSP dynamiques

Un CSP dynamique, noté DCSP [Verfaillie and Jussien, 2005] est une séquence $\langle P_1, P_2, ..., P_n \rangle$ de CSP, telle que chaque CSP P_i est le résultat des modifications apportées au CSP précèdent P_{i-1}. Ces modifications (ajout/suppression) peuvent s'opérer

1. Un catalogue des contraintes globales est donné sur http://web.emn.fr/x-info/sdemasse/gccat/sec5.html

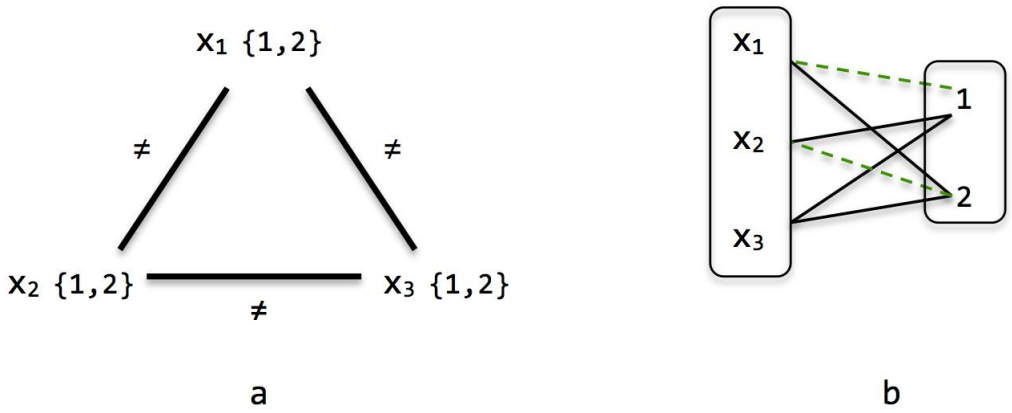

FIGURE 2.2.: Représentation de la contrainte de différence

sur les variables, les domaines ou les contraintes [2]. L'intérêt de base d'un tel CSP est de modéliser un problème dont les solutions n'ont pas toutes la même structure, par exemple, elles n'impliquent pas le même nombre de variables et de contraintes. Un second intérêt est de faire de l'optimisation, en faisant en sorte que le solveur produise des solutions optimales.

A chaque nouvelle solution trouvée, le CSP courant évolue au niveau du triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Dans le cas de l'ajout de contraintes, les solutions suivantes, doivent satisfaire à la fois l'ensemble courant des contraintes $\mathcal{C}$ ainsi que les contraintes ajoutées au nouveau CSP. Résoudre un tel CSP dynamique, consiste à résoudre un seul CSP avec des contraintes supplémentaires postées pendant la recherche.

Contrairement à un CSP classique, l'algorithme de résolution d'un DCSP avec ajout de contraintes, effectue une opération en plus après avoir retourné une solution. Cette opération, consiste en l'ajout de contraintes sur les variables $\mathcal{X}$, notées $\phi(\mathcal{X})$ dans l'ensemble de contraintes de départ $\mathcal{C}$ ($\mathcal{C}.add(\phi(\mathcal{X}))$).

Nous présentons dans la section suivante, une approche PPC pour l'extraction de motifs (voir chapitre 1). Cette approche associe à chaque item/transaction une variable. Des contraintes sont ensuite exprimées sur ces variables, pour modéliser des propriétés intéressantes sur la nature des motifs à extraire. L'intérêt d'une telle approche est l'aspect déclaratif.

2.7. Extraction de motifs par la programmation par contraintes

En tenant compte du fait que les algorithmes classiques présentés dans le chapitre précèdent [Pasquier et al., 1999b, Zaki and Hsiao, 2002, Uno et al., 2004a] souffrent d'un manque de généricité, plusieurs travaux récents ont été menés pour

2. Dans nos travaux (chapitre 5) les modifications concernent uniquement l'ajout de contraintes.

modéliser le problème de l'extraction de motifs dans un cadre de programmation par contraintes [De Raedt et al., 2008, Khiari et al., 2010, Ugarte et al., 2015]. Un des modèles les plus élaborés permettant d'exprimer un bon nombre de contraintes est le modèle présenté par *De Raedt et al* [De Raedt et al., 2008, Guns, 2012]. Ce modèle est basé sur les contraintes réifiées.

Le modèle réifié

Ce modèle exploite la représentation booléenne de la base de transactions (voir la figure 1.2), il est basé sur deux ensembles de variables booléennes.

- Variables :

- Les variables de décision d'items $X = \{X_1, ..., X_n\}$, où, $(X_i = 1)$ si et seulement si, l'item i est présent dans le motif recherché X.
- Les variables auxiliaires de transactions $T = \{T_1, ..., T_m\}$, où, $(T_t = 1)$ si et seulement si, le motif recherché $X \subseteq t$.

- Contraintes :

La relation entre un motif recherché X et la base transactionnelle $\mathcal{D}$ est établie à travers la contrainte de *couverture* ci-dessous :

$$\forall t \in \mathcal{T} : (T_t = 1) \longleftrightarrow \sum_{i \in \mathcal{I}} X_i(1 - \mathcal{D}_{ti}) = 0 \tag{2.1}$$

Cette contrainte établit le fait que pour chaque transaction t, la variable $T_t = 1$ si le motif X est un sous-ensemble de t. Au moyen de la contrainte de couverture, une variable de transaction sera égale à 1 si la transaction couvre le motif. Par conséquent, la fréquence de ce motif peut être obtenue en comptant le nombre de transactions $T_t = 1$. La contrainte de fréquence est établie par la contrainte ci-dessous :

$$\forall i \in \mathcal{I} : (X_i = 1) \longrightarrow \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geqslant \theta \tag{2.2}$$

La contrainte de fermeture est exprimée par la contrainte ci-dessous. Elle établit le fait que l'item i d'un motif X doit apparaître dans toutes les transactions T_t qui couvrent X.

$$\forall i \in \mathcal{I} : (X_i = 1) \longleftrightarrow \sum_{t \in \mathcal{T}} T_t(1 - \mathcal{D}_{ti}) = 0 \tag{2.3}$$

Ce modèle offre une façon élégante d'exprimer un large éventail de contraintes que l'on retrouve dans l'extraction de motifs [Guns et al., 2011], telles que la contrainte de maximalité et la contrainte de taille minimale. Cependant, ce modèle souffre d'un inconvénient majeur qui constitue une réelle limite quant au passage à l'échelle. En

effet, ce modèle nécessite à la fois :

(i) Des variables booléennes auxiliaires représentants les transactions, ce qui ajoute une dimension combinatoire au problème lors de la résolution.
(ii) Un large déploiement de contraintes réifiées pour exprimer les différentes contraintes sur les motifs.

Il est aussi important de noter que, le filtrage utilisé pour propager les contraintes de ce modèle est un filtrage classique (voir la partie 2.4 sur la propagation des contraintes réifiées) qui est totalement indépendant de la nature du problème.

Nombre de contraintes réifiées et de variables
Comme mentionné précédemment, afin d'extraire les motifs fréquents fermés, le modèle réifié exprime 3 contraintes : la couverture, la fréquence minimale, la clôture.

- Pour exprimer la couverture, le modèle utilise la contrainte 2.1 qui est d'arité $n+1$ (n items et 1 variable booléenne T_t). Cette contrainte et déployée m fois (le nombre de transactions).

- Pour exprimer la fréquence minimale, le modèle utilise la contrainte 2.2 qui est d'arité $m+1$ (m transactions et 1 variable booléenne I_i). Cette contrainte et déployée n fois (le nombre d'items).

- Pour exprimer la clôture, le modèle utilise la contrainte 2.3 qui est d'arité $m+1$ (m transactions et 1 variable booléenne I_i). Cette contrainte et déployée n fois (le nombre d'items).

Nous avons au total un nombre de contraintes réifiées égal à : (m d'arité $(n+1)$) + (n d'arité $(m+1)$) + (n d'arité $(m+1)$).

Exemple 2.9. *Soit une base de transactions contenant* 1 000 *items et* 100 000 *transactions (base T10I4D100K*[3]*). Le modèle réifiée nécessitera alors :* 102 000 *contraintes réifiées qui portent sur un total de* 101 000 *variables.*

Ce dernier point nous a particulièrement motivé pour le travail que nous introduisons dans le chapitre 4, où, nous présentons une contrainte globale pour encoder efficacement le problème de l'extraction des motifs fréquents fermés.

2.8. Conclusion

Dans ce chapitre, nous avons présenté les notions principales de la PPC reprises dans nos contributions. Nous avons présenté la notion de CSP et montré comment s'opère la résolution d'un CSP notamment grâce au filtrage basé sur la consistance. Nous nous sommes particulièrement intéressés aux points suivants :

– Les contraintes réifiées, utilisées pour modéliser les problèmes d'extraction de motifs.
– Les contraintes globales en PPC, sur lesquelles repose une de nos contributions présentées dans le chapitre 4, où nous proposons une contrainte globale pour l'extraction de motifs fréquents fermés.

3. http ://fimi.ua.ac.be/data/

– Les CSP dynamiques, qui permettront l'extraction des motifs suspects, dans notre contribution liée à la localisation de fautes (chapitre 5).

Le chapitre 3 suivant présente une des phases du test logiciel, à savoir, la localisation de fautes dans les programmes, qui représente le champ de recherche où s'inscrit le cadre applicatif de nos travaux sur la fouille de motifs basée sur la programmation par contraintes.

3. Test de logiciels : localisation de fautes

Ce chapitre introduit la problématique de la localisation de fautes dans les programmes, une des activités les plus importantes dans le test logiciel. La section 3.1 présente le processus de test de logiciels. La section 3.2 introduit le cadre général de la localisation et les éléments liés à cette activité. La section 3.3 présente la notion de code suspect, qui joue un rôle central dans la localisation et qui est au cœur de notre contribution au chapitre 5. La section 3.4 propose un bref panorama des techniques courantes dans la localisation de fautes. La section 3.5 présente une métrique permettant d'évaluer, la qualité des méthodes proposées.

Sommaire

« *Software and cathedrals are much the same : first we build them, then we pray.* » — Anonymous

3.1. Processus de test d'un logiciel

Au cours du processus de développement de logiciels, ces derniers sont sujets aux erreurs ou bugs, d'autant plus que leurs sources sont multiples, de part (i) la phase de développement dans laquelle se trouve le produit logiciel et (ii) le nombre de développeurs impliqués dans cette phase du développement. Ce risque est tant répandue et préjudiciable, qu'il impacte fortement la fiabilité et la qualité des systèmes. D'autre part, un effort conséquent est mené à la fois pour augmenter la qualité des logiciels produits et réduire les conséquences néfastes, que peuvent avoir ce genre de défaillances.

Dans ce contexte, un large éventail d'approches et de méthodes sont utilisées, pour assurer un certain niveau de fiabilité et de qualité d'un logiciel. Les méthodes basées sur la preuve essayent de certifier qu'un code est dépourvu d'erreurs, une version formalisée de l'exactitude du code par rapport à la spécification mathématique du système est alors établit. Dans l'analyse statique, on procède par l'examen du programme sans l'exécuter, contrairement à l'analyse dynamique qui surveille l'exécution des programmes. Les méthodes formelles s'appuient sur une logique mathématique et un haut niveau d'abstraction, telles que, l'analyse statique par interprétation abstraite [Cousot and Cousot, 1977], qui permet d'obtenir une approximation du comportement du système, où encore, la vérification de modèles (model checking) [Queille and Sifakis, 1982], qui est un ensemble de techniques de vérification automatique de propriétés temporelles sur des systèmes. Ces approches sont très rigoureuses mais elles sont aussi coûteuses et difficiles à mettre en œuvre dans la pratique.

Le *test logiciel* est de loin la méthode la plus utilisée, pour valider les programmes informatiques. Celle ci repose sur l'exécution du logiciel en observant son comportement. Le test logiciel offre moins de garanties sur la qualité d'un logiciel, car contrairement aux méthodes formelles, avec le test il est possible de prouver la présence d'erreurs dans un système, mais jamais leurs absences (Edsger Dijkstra, 1972). En d'autres termes, le test permet d'augmenter la confiance portée au logiciel sous test. En pratique, le test logiciel est relativement coûteux. Généralement cette activité consomme pas moins de 50% du budget total de développement et de maintenance d'un système. Ce coût croît considérablement, lorsqu'il s'agit de logiciels destinés à des systèmes critiques.

Plus concrètement, le *test logiciel* est le processus qui permet d'exécuter un programme ou un logiciel avec des données d'entrée particulières, dans l'intention de vérifier si ce dernier répond correctement à ses exigences et spécifications. Pour chaque exécution, un oracle, humain ou automatisé, permet de déterminer si le comportent est conforme à sa spécification. Typiquement, réaliser du test sur un logiciel, s'effectue en trois activités majeures, à savoir : (i) La *détection* (ii) La *localisation* et enfin (iii) La *correction*. Nous décrivons brièvement dans ce qui suit chacune de ces 3 étapes :

- *Détection* : Cette première étape vise à exécuter le logiciel sous diverses conditions,

dans le but de révéler s'il comporte des erreurs en observant des défaillances. On dit généralement que l'on détecte des défaillances.
- *Localisation* : Cette seconde étape vise à identifier la région ou partie du système potentiellement responsable de l'incapacité du logiciel, à répondre correctement aux spécifications, pour lesquelles il est conçu.
- *Correction* : Cette dernière étape consiste à corriger les erreurs trouvées dans le logiciel et ainsi, valider le comportement de ce dernier.

Dans les travaux menés dans cette thèse, nous nous intéressons à la seconde activité de ce processus : *La localisation.*

3.2. La localisation de fautes dans les programmes

Une des phases coûteuses dans la mise au point d'un programme, est la localisation des fautes [Vessey, 1985]. Pour mener à bien cette phase, les développeurs doivent identifier les instructions impliquées dans la défaillance du programme en question. En pratique, cette tâche commence lorsque le développeur exécute son programme et se rend compte que le résultat n'est pas conforme, il en résulte une longue phase de recherche de l'origine de cet écart.

Améliorer la localisation permet d'apporter un gain non négligeable dans le processus du test, car cette phase avec laquelle sont repérées les erreurs, puis corrigées, est au cœur du processus de test. Nous donnons dans ce qui suit, le cadre général et les définitions, que l'on retrouve dans la localisation de fautes.

3.2.1. Cadre général et objectifs

En test logiciel, une *défaillance* est une déviation entre le résultat attendu et le résultat actuel obtenu par un système. Une *erreur* est une partie du système qui est susceptible de conduire à une défaillance. Une *faute* est la cause supposée ou adjugée d'une erreur [Laprie et al., 1992] (voir la figure 3.1). Il est à noter que, les sources d'une faute sont diverses, dans [Avizienis et al., 2004] les auteurs proposent une taxonomie des fautes, selon leurs natures et leurs causes.

La relation entre une faute, une erreur et une défaillance est résumée dans la figure 3.1 [Avizienis et al., 2004]. Lorsqu'une faute est active (le programme ou le système est amené à passer par cette faute) elle produit une erreur. Cette erreur se propage dans une ou plusieurs parties du système, conduisant à un certain degré de défaillance dans son comportement. Il est à noter également, qu'une défaillance peut aussi, être à l'origine d'une faute. Par exemple, si la défaillance est un résultat erroné d'une méthode et que cette méthode est appelée ailleurs dans le programme, l'appel de cette méthode constitue alors une faute pour la suite de l'exécution.

L'objectif de la localisation de fautes, est d'identifier la cause principale des symptômes observés durant l'exécution des cas de test. En d'autres termes, le besoin est de retourner au programmeur/testeur un sous ensemble d'instructions (ou partie du système) pouvant potentiellement, expliquer l'origine de la défaillance observée dans le résultat

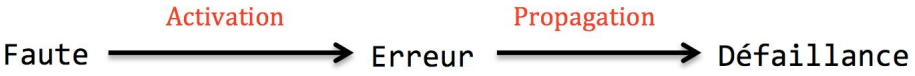

FIGURE 3.1.: Chaîne causale de menaces à la fiabilité d'un logiciel

Programme : Compteur de caractères	Cas de test tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8
`function count (char *s) {`								
`int let, dig, other, i = 0;`								
`char c;`								
e_1 : `while (c = s[i++]) {`	1	1	1	1	1	1	1	1
e_2 : `if('A'<=c && 'Z'>=c)`	1	1	1	1	1	1	0	1
e_3 : `let += 2; //- faute -`	1	1	1	1	1	1	0	0
e_4 : `else if ( 'a'<=c && 'z'>=c )`	1	1	1	1	1	0	0	1
e_5 : `let += 1;`	1	1	0	0	1	0	0	0
e_6 : `else if ( '0'<=c && '9'>=c )`	1	1	1	1	0	0	0	1
e_7 : `dig += 1;`	0	1	0	1	0	0	0	0
e_8 : `else if (isprint (c))`	1	0	1	0	0	0	0	1
e_9 : `other += 1;`	1	0	1	0	0	0	0	1
e_{10} : `printf("%d %d %d\n", let, dig, other);}`	1	1	1	1	1	1	1	1
Positif/Négatif	N	N	N	N	N	N	P	P

FIGURE 3.2.: Exemple d'un programme avec la matrice de couverture associée

du programme sous test. Par ailleurs, examiner ce sous-ensemble d'instructions peut se révéler très coûteux et fastidieux. Par conséquent, plus ce sous ensemble est de taille réduite, plus on considérera que la méthode utilisée pour localiser ce sous-ensemble est précise. En effet, cela se traduit par le fait que le programmeur/testeur aura peu d'instructions à examiner avant de pointer sur la faute.

Soit un programme P avec faute ayant n lignes $L = \{e_1, e_2, ..., e_n\}$. La figure 3.2 montre un exemple d'un programme simple nommé "Compteur de caractères", où nous avons $L = \{e_1, e_2, ..., e_{10}\}$. Ce programme contient une faute à la ligne 3, l'instruction correcte devrait en effet être "`let += 1;`", ce qui cause une erreur lorsque le programme "Compteur de caractères" exécute cette instruction.

3.2.2. Cas de test et couvertures

Afin d'examiner le comportement d'un système ou programme sous diverses conditions, ce dernier est exécuté avec des données d'entrée particulières. Une donnée de test est appelée cas de test.

Définition 3.1 (Cas de test)**.** *Un cas de test tc_i est un tuple $\langle D_i, O_i \rangle$, où D_i est l'ensemble des paramètres d'entrée pour déterminer, si un programme P se comporte comme prévu ou non, O_i est la sortie attendue du programme.*

Considérons le cas de test $tc_i = \langle D_i, O_i \rangle$. Soit A_i la sortie courante retournée par un programme P suite à l'exécution de l'entrée D_i. Si $A_i = O_i$, alors le cas de test tc_i est considéré comme correct (i.e. positif), erroné (i.e. négatif) dans le cas contraire.

Définition 3.2 (Suite de test). *Une suite de test $T = \{tc_1, tc_2, ..., tc_m\}$ est l'ensemble des m cas de test, destinés à tester si le programme P respecte l'ensemble des exigences spécifiées.*

Lorsque un programme P est exécuté avec un cas de test tc_i, il passe par un certain nombre d'instructions. Cet ensemble d'instructions forme ce qu'on appelle la *couverture*.

Définition 3.3 (Couverture de cas de test). *Étant donné un cas de test tc_i et un programme P, l'ensemble des instructions de P exécutées (au moins une fois) avec tc_i désigne la couverture du cas de test $I_i = (I_{i,1}, ..., I_{i,n})$, où $I_{i,j} = 1$ si l'instruction à la ligne j est exécutée, 0 sinon. La couverture d'un cas de test indique, quelles sont les parties actives du programme durant une exécution spécifique*

Par exemple, le cas de test tc_4 dans la figure 3.2 couvre les instructions $(e_1, e_2, e_3, e_4, e_6, e_7, e_{10})$. Le vecteur de couverture correspondant est $I_4 = (1, 1, 1, 1, 0, 1, 1, 0, 0, 1)$.

Lors de la génération d'une suite de test T pour tester un programme P, cette suite de test doit répondre à des critères particuliers. Un des critères est celui de la couverture, qui assure que T doit garantir l'exécution de toutes les instructions de P.

3.2.3. Hypothèse de base sur la faute

Il est important de souligner que la majeure partie des approches de localisation, telles que [Jones and Harrold, 2005, Abreu et al., 2007, Nessa et al., 2008], sont basées sur deux hypothèses de travail. La première hypothèse est habituellement appelée l'*hypothèse du développeur compétent* [DeMillo et al., 1978]. Cette hypothèse suppose que, même si plusieurs fautes *simples* sont introduites, le programme résultant, pourrait certainement répondre à la quasi-totalité de ses spécifications. En d'autres termes, le programme contient des fautes, mais il va plus ou moins, résoudre le problème pour lequel il a été conçu.

La deuxième hypothèse sur laquelle les approches de localisation sont basées, est l'*hypothèse de la faute simple*, c'est-à-dire, qu'il y a seulement une seule instruction fautive dans le programme. Cette exigence peut paraître restrictive, mais il a été montré que les fautes dites *complexes* sont le résultat d'une combinaison de fautes simples (i.e,. l'effet de couplage [Jones et al., 2002]).

3.2.4. Localisation des fautes multiples

Les cas de localisation étudiés par la plupart des approches, portent sur des programmes contenant une seule instruction fautive. Par ailleurs, dans la pratique un programme sous test, peut contenir plusieurs instructions fautives. Ce genre de programmes soulève un challenge important pour les méthodes classiques de localisation.

L'idée principale des approches conçues pour traiter des programmes à fautes multiples [Podgurski et al., 2003, Liu and Han, 2006, Jones et al., 2007, Eric Wong et al., 2010], est de construire des clusters de cas de test négatifs, appelés *fault-focusing*, tels que, les cas de test négatifs d'un même cluster, sont ceux ayant

exécutées la même faute. Il y a donc autant de clusters de cas de test, que de fautes dans le programme. Par ailleurs, pour construire de manière précise de tels clusters, ces méthodes dépendent fortement de l'information qui consiste à connaître le nombre de fautes dans le programme, pour savoir si un cas de test négatif est passé par une faute donnée. Il est clair que dans la pratique, nous ne pouvons disposer de ce type de informations sur le programme à corriger. L'idée de ces méthodes, est alors de regrouper les cas de test négatifs dans les mêmes clusters, en étudiant les similarités des traces d'exécution de cas négatifs. Ensuite, l'idée est de combiner chaque groupe de cas de test négatives avec de cas de test positifs pour trouver la faute du cluster correspondant. Ce processus est appliqué sur chaque cluster.

Une autre idée consiste à localiser une faute à la fois (*one-bug-at-a-time* strategy) [Wong and Debroy, 2009]. Après avoir localiser et corriger une faute, le programme est re-exécuté avec les mêmes cas de test. Si une nouvelle exécution échoue, alors une seconde localisation est menée, et ainsi de suite, jusqu'à ce qu'aucune défaillance n'est observée [1].

3.3. Instructions suspectes et mesures de suspicion

La notion de suspicion est étroitement liée à la nature des cas de test ayant exécuté le programme. En effet, lorsqu'une déviation dans le résultat survient, l'objectif est de repérer le point de déviation. Une déviation est le résultat d'une exécution négative. Il est alors intéressant d'explorer les informations d'une telle exécution. C'est pourquoi, une partie du programme sous test est considérée comme suspecte, si cette partie du code est souvent exécutée par une exécution négative. Cela se traduit par le fait que cette partie du programme peut expliquer la raison de la défaillance. Étant donné, un ensemble de cas de test positifs et négatifs, une instruction dans un programme est dite *suspecte*, si cette instruction est principalement exécutée par les cas de test négatifs et occasionnellement par les cas de test positifs.

La notion de suspicion donnée ici, peut être différente de la notion de suspicion donnée dans [Agrawal et al., 1995], où une instruction est considérée suspecte, si elle est exécutée uniquement par les cas de test négatifs. Cette seconde définition peut se révéler parfois restrictive, d'autant plus qu'il a été montré que dans la pratique, une faute peut être exécutée par des cas de test positifs.

Ainsi, avec une telle notion, les entités d'un programme peuvent être catégorisées selon leur suspicion et ordonnées en fonction du degré de suspicion. Une des classes de méthodes exploitant cette notion de suspicion pour classer les instructions d'un programme, est la classe des mesures de suspicion.

Mesures de suspicion :

Les techniques basées sur les mesures de suspicion permettent d'assigner un score mesurant le degré de suspicion de chaque instruction dans un programme, elles rapportent

1. Par définition du test logiciel, nous ne pouvons pas affirmer que le programme est alors dépourvu de fautes.

ensuite, un classement des instructions dans un ordre décroissant de la valeur de suspicion. La plupart de ces mesures sont définies manuellement et de manière analytique, sur la base de plusieurs hypothèses ; sur les programmes, les cas de test et les fautes introduites. L'hypothèse de base est que le programme produit une défaillance lorsqu'il passe par l'instruction fautive. Ceci dit, les mesures partagent la même intuition : les instructions les plus suspectes sont celles exécutées le plus souvent par les cas de test négatifs et le moins souvent par les cas de test positifs. Ce qui traduit une corrélation entre les cas de test négatifs et l'instruction fautive. La suspicion d'une instruction augmente proportionnellement avec son exécution par les cas de test négatifs, et inversement.

Nous donnons dans ce qui suit, les mesures les plus connues dans l'état de l'art. Chaque mesure dispose d'une fonction lui permettant de donner un sens à la suspicion d'une instruction, en capturant la corrélation avec les cas de test négatifs exécutés. La figure 3.3 illustre également le comportement de chaque mesure étant donné un nombre d'exécution de cas de test positifs et négatifs.

Tarantula [Jones and Harrold, 2005] : C'est la mesure de suspicion la plus connue dans l'état de l'art. La particularité de Tarantula est le fait que cette mesure ne permet pas beaucoup de tolérance à une instruction fautive d'être exécutée par les cas de test positifs (Figure 3.3) [2]. Cette particularité peut se montrer efficace lors de la localisation [Jones, 2008]. Le degré de suspicion affecté par Tarantula pour une instruction e_i donnée est calculé avec la mesure 3.1 :

$$Tarantula(e_i) = \frac{\frac{|negatif(e_i)|}{|negatif(T)|}}{\frac{|positif(e_i)|}{|positif(T)|} + \frac{|negatif(e_i)|}{|negatif(T)|}} \tag{3.1}$$

Avec $positif(T)$ (respectivement, $negatif(T)$) est l'ensemble de tous les cas de test positifs (respectivement. négatifs). $positif(e_i)$ (resp. $negatif(e_i)$) est l'ensemble des cas de test positifs (resp. négatifs) couvrant e_i.

Ochiai [Abreu et al., 2007] : Cette mesure est connue dans le domaine de la biologie et a été utilisée dans [Abreu et al., 2007], dans le but d'améliorer la précision de la localisation. La spécificité de cette mesure est sa tolérance moyenne à une instruction fautive, d'être couverte par les cas de test positifs (Figure 3.3). La fonction de la mesure Ochiai pour une instruction e_i est la suivante :

$$Ochiai(e_i) = \frac{|negatif(e_i)|}{\sqrt{\left(|negatif(e_i)| + |positif(e_i)|\right) \times |negatif(T)|}} \tag{3.2}$$

Jaccard [Abreu et al., 2007] : Cette mesure est moins tolérante à la présence d'une instruction dans les cas de test positifs que la mesure Ochiai. Cette mesure à

2. Dans la figure 3.3, la couleur rouge (respectivement, bleu) indique un très haut (respectivement, bas) degré de suspicion.

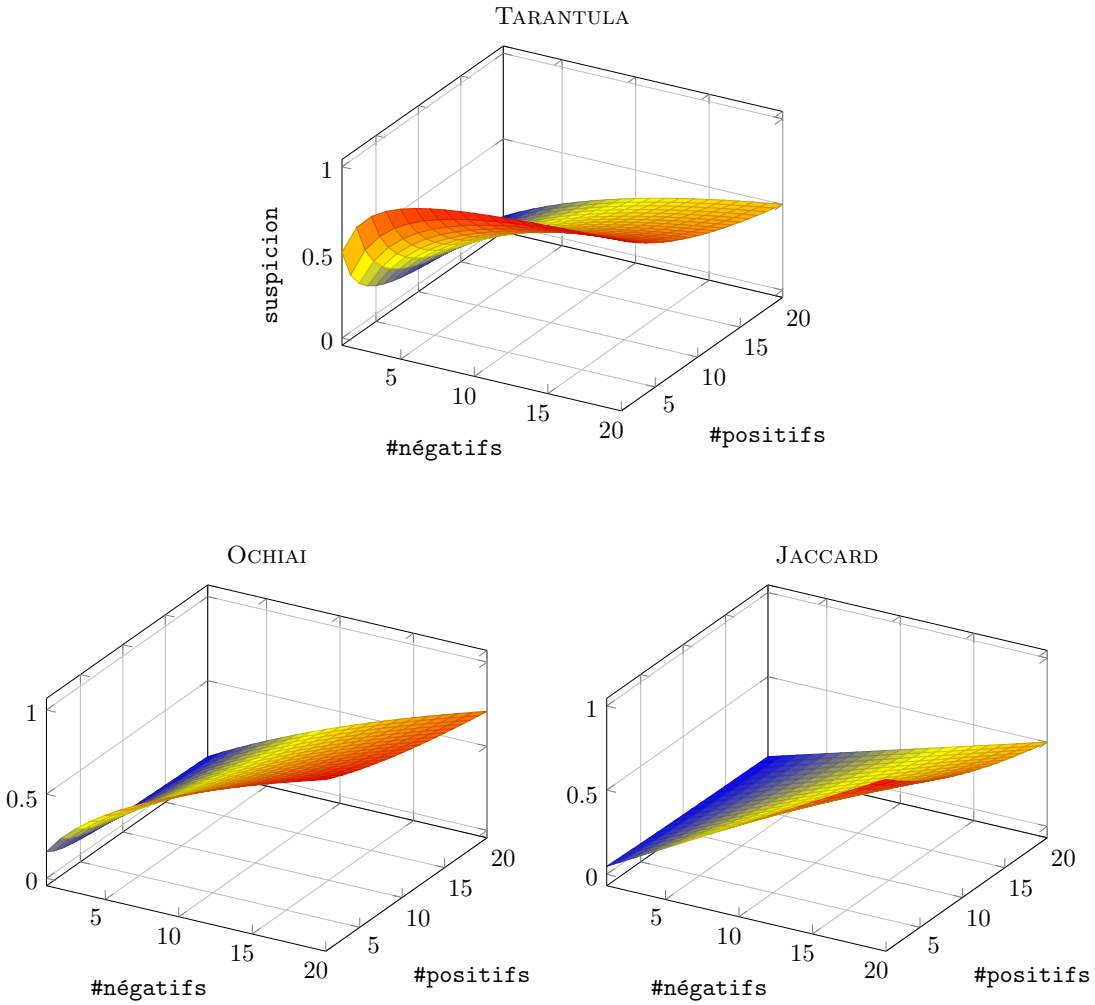

Figure 3.3.: Tendances des mesures de suspicion

été présentée également dans [Abreu et al., 2007] avec l'hypothèse qu'une instruction exécutée par un cas de test positif a peu de chance d'être suspecte (Figure 3.3). La fonction de la mesure Jaccard pour une instruction e_i est la suivante :

$$Jaccard(e_i) = \frac{|negatif(e_i)|}{|positif(e_i)| + |negatif(T)|} \tag{3.3}$$

L'avantage majeur de ce genre de mesures est que l'évaluation des instructions d'un programme se fait de manière rapide. En effet, le coût des algorithmes de ce genre de mesures est linéaire suivant la taille du programme (nombre d'instructions) et linéaire suivant la taille de la suite de test (nombre de cas de test). Pour chaque instruction, une mesure doit calculer le nombre de cas de test (positifs et négatifs) l'ayant exécutée.

Cette requête prend pour chaque instruction m tests, où m est la taille de la suite de test T. Une fois les couvertures positives et négatives calculées, le degré de suspicion est calculé en un temps constant C. Par conséquent, le calcul de la valeur de la mesure pour une instruction donnée est $(m + C)$. Pour évaluer l'ensemble des instructions n du programme P, le temps total est $n(m + C)$. Ainsi, la complexité en temps est de $O(mn)$ [Jones, 2008].

Exemple 3.1. *Les courbes de la figure 3.3, illustrent le résultat de chaque mesure sur un nombre donné de cas de test (Pour cet exemple nous avons choisi une suite de test avec 20 cas de test positifs et 20 cas de test négatifs). L'axe des x, représente le nombre de cas de test négatifs ayant exécuté une instruction e donné. L'axe des y, représente les cas positifs. Enfin, l'axe z, représente le degré de suspicion.*

Nous constatons que chaque mesure a sa façon de raisonner, étant donné un nombre de cas de test négatifs et positifs, pour donner un sens à la notion de suspicion d'une instruction e.

En effet, lorsqu'une instruction est considérée comme très suspecte par une mesure donnée, elle ne l'est pas forcement par une autre mesure. Prenons par exemple la courbe de Tarantula*, une instruction exécutée par* 10 *cas de test négatifs et entre* 0 *et* 5 *fois par les cas de test positifs, est considérée comme très suspecte (plus qu'une instruction exécutée* 20 *fois par les cas de test négatifs et* 10 *fois par les cas de test positifs), ce qui traduit le manque de tolérance à une instruction d'être exécutée par les cas de test positifs dans le raisonnement de la fonction 3.1 de* Tarantula*. Cette observation n'est pas vraie dans les deux autres courbes de* Ochiai *et* Jaccard*.*

Dans la courbe de Ochiai *l'importance est donnée aux instructions exécutées par* 20 *cas de test négatifs, quelques soient leurs exécutions dans les cas de test positifs, il en résulte un classement différent par rapport aux autres mesures.*

Exemple 3.2. *Les constatations observées dans l'exemple 3.1 sont confirmées sur cet exemple, où à l'aide du programme de la figure 3.2. Nous donnons dans le tableau 3.1 les valeurs des degrés de suspicion affectés par les différentes mesures, sur chaque instruction du programme. Nous avons donc repris les mesures* Tarantula, Ochiai *et* Jaccard*. Nous donnons également dans le tableau 3.1 le classement retourné par chaque mesure (colonne Rang) basé sur la valeur de suspicion (colonne susp).*

Remarque : Ces mesures ont reçu une attention considérable, mais ont été également critiquées pour leur manque de précision [Parnin and Orso, 2011]. Une des causes de ce manque de précision est l'existence de cas de test positifs par *coincidence*, qui forment des cas de test positifs malgré leur passage par une instruction fautive - *Coincidentally Correct Test cases (CCTs)* - [Richardson and Thompson, 1993] ; ce qui a tendance à affaiblir la corrélation entre la faute et les cas de test négatifs. En plus de cette analyse, nous dressons les observations suivantes sur l'exemple 3.1 :

- Le classement final retourné par les mesures sont différents (Tarantula $\neq$ Jaccard = Ochiai). En effet, une instruction peut être considérée comme suspecte par une mesure donnée et l'être moins dans une autre mesure. Par exemple, l'instruction e_7 considérée comme très suspecte par Tarantula ($rang = 1$) et beaucoup moins suspecte dans les

	Cas de Test								Mesures					
	négatifs						positifs		Tarantula		Jaccard		Ochiai	
Inst	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8	*susp*	*Rang*	*susp*	*Rang*	*susp*	*Rang*
e_1	1	1	1	1	1	1	1	1	0.5	8	0.75	4	0.86	4
e_2	1	1	1	1	1	1	0	1	0.66	4	0.85	2	0.92	2
e_3	1	1	1	1	1	1	0	0	1	1	1	1	1	1
e_4	1	1	1	1	1	0	0	1	0.62	5	0.71	5	0.83	5
e_5	1	1	0	0	1	0	0	0	1	1	0.5	7	0.70	7
e_6	1	1	1	1	0	0	0	1	0.57	6	0.57	6	0.73	6
e_7	0	1	0	1	0	0	0	0	1	1	0.33	8	0.57	8
e_8	1	0	1	0	0	0	0	1	0.4	10	0.28	10	0.47	10
e_9	1	0	1	0	0	0	0	1	0.4	10	0.28	10	0.47	10
e_{10}	1	1	1	1	1	1	1	1	0.5	8	0.75	4	0.86	4

Table 3.1.: Degrés de suspicion affectés par différentes mesures sur le même programme

deux autres mesures ($rang = 8$). A cela vient s'ajouter le fait que, le degré de suspicion ne concorde pas tout le temps avec les fréquences de l'instruction en question. Par exemple, l'instruction e_4 est considérée plus suspecte (rang = 5) que l'instruction e_1 (rang = 8) par Tarantula, bien que e_1 soit plus fréquente dans les cas de test négatifs que e_4.

- Un deuxième constat important, est que dans ce genre de mesures, les instructions sont évaluées de manière individuelle et indépendamment les unes des autres. La conséquence est que, les dépendances qui peuvent exister entre l'exécution des instructions sont totalement ignorées. Une des questions que l'on peut alors se poser, est comment capturer ces dépendances pouvant fortement aider dans la localisation.

3.4. Techniques existantes de localisation de fautes

Dans cette section, nous présentons en plus des mesures de suspicion vues précédemment, quelques travaux connus de l'état de l'art en localisation de fautes utilisant différents champs et domaines de recherche pour assister cette activité.

Techniques utilisant les exécutions positives/négatives : Comme introduit précédemment, les mesures de suspicion sont parmi les techniques de localisation de fautes les plus populaires, qui font appel à l'information liée aux exécutions positives/négatives, en termes de nombre de cas de test couvrant les instructions. Une autre technique proche de ce genre de mesures, est celle proposée par Cleve et Zeller [Cleve and Zeller, 2005], basée sur les états du programme. La technique compare les états des cas de test positifs et négatifs. Dans un travail précédent [Zeller, 2002], Zeller montre, que localiser une faute juste en considérant l'espace de recherche (variables, valeurs) n'est pas suffisant en général. En effet, il est possible de retourner des instructions comme des fautes, en comparant les différents états dans les cas de test positifs/négatifs, mais une faute dans un programme peut produire une différence sur tous les états suivants dans le programme. Dans [Cleve and Zeller, 2005] une recherche durant l'exécution est menée pour localiser la première transition qui conduit à un échec.

Techniques utilisant l'analyse des dépendances : Des techniques ont été proposées afin de prendre en compte les chaînes de cause à effet avec une analyse des dépendances, en utilisant par exemple, le découpage de programmes [Agrawal et al., 1993]. L'inconvénient ici, est que la faute peut être localisée dans une partie assez large (découpage statique) et/ou peut être coûteuse en temps/espace (découpage dynamique). Dans la même perspective, Renieris et Reiss [Renieres and Reiss, 2003] utilisent la notion de *plus proche voisin*, où ils confrontent une trace d'exécution négative avec la trace positive la plus proche. Ici, la distance entre deux traces est exprimée avec la différence entre l'ensemble des instructions exécutées. Dans le cas où aucune trace positive proche ne peut être obtenue, la technique construit le graphe des dépendances du programme et vérifie les nœuds adjacents d'une trace négative, un par un, avec le but de trouver l'endroit de la faute.

Techniques utilisant la fouille de données : Denmat et al [Denmat et al., 2005] donne une analyse de l'approche Tarantula en re-interprétant la mesure de suspicion, comme un moyen pour effectuer de l'extraction de données. En fouille de données, des règles d'association sont définies entre les données. La mesure de suspicion est vue comme une des règles d'association. Pour les besoins de la localisation, la mesure peut être utilisée pour caractériser les règles qui associent les instructions couvertes avec un cas de test négatif. Dans [Cellier et al., 2009], Cellier et al. proposent un processus de fouille de données DeLLIS, qui calcule des clusters d'éléments du programme et montre les dépendances entre ces éléments. Cette approche calcule toutes les différences entre les traces d'exécution et en même temps donne un ordre partiel de ces différences. Dans [Cellier et al., 2008], Cellier et al. proposent une approche qui combine les règles d'association et l'analyse formelle de concepts (FCA) pour assister la localisation de fautes. Ils essayent d'identifier les règles entre l'exécution d'une instruction et le cas de test négatif correspondant, et ensuite mesurer la fréquence de chaque règle. Nessa et al. [Nessa et al., 2008] proposent eux, de générer des sous-séquences d'instructions de longueur N, appelées N-grams, à partir des traces. Les traces d'exécution erronées sont alors examinées pour trouver les N-grams avec un taux d'occurrence supérieur à un certain seuil. Une analyse statistique est ensuite conduite, pour déterminer la probabilité conditionnelle qu'une exécution échoue, sachant qu'un certain N-gram apparaît dans sa trace.

3.5. Mesure d'efficacité des méthodes de localisation

L'objectif de l'ensemble des méthodes présentées précédemment est de fournir un classement des instructions, de la plus suspecte de contenir une faute, à la moins suspecte. Ce classement se fait en fonction du degré de suspicion de chaque instruction. En effet, chaque instruction possède un rang qui définit sa profondeur dans la liste retournée par une méthode donnée. L'ensemble des instructions avec les plus grandes valeurs de suspicion, est l'ensemble des instructions à considérer/vérifier en premier lieu par le programmeur pour trouver la faute. Si, après l'examen des instructions en haut de classement, la faute n'est toujours pas trouvée, la recherche se poursuit dans le reste de la

liste triée dans l'ordre décroissant, en termes de degré de suspicion.

L'efficacité et la précision d'une méthode de localisation de fautes est mesurée à l'aide du classement qu'elle retourne, en particulier, le rang qu'elle affecte à l'instruction fautive recherchée. Plus cette instruction sera en haut (respectivement, bas) du classement (i.e, elle sera examinée très rapidement (respectivement, très tardivement)), plus la méthode sera efficace (respectivement, inefficace).

Dans la communauté de test, il existe une métrique très connue, à savoir l'EXAM score [Wong and Debroy, 2009, Eric Wong et al., 2010]. Cette métrique mesure l'efficacité d'une méthode de localisation donnée.

Définition 3.4 (EXAM score)**.** *L'*EXAM *score (équation 3.4) donne le pourcentage des instructions qu'un développeur doit examiner, jusqu'à atteindre celle contenant la faute.*

$$\text{EXAM } score = \left(\frac{rang\ de\ la\ faute}{nombre\ total\ d'instructions}\right) \times 100 \tag{3.4}$$

Il est alors logique que la meilleure méthode est celle qui a le pourcentage d'EXAM score le plus petit. Par conséquent, les méthodes peuvent être comparées en utilisant l'EXAM score.

Lors de l'évaluation des instructions d'un programme P donné, par une méthode $\mathcal{M}$, plusieurs instructions peuvent être considérées comme équivalentes par $\mathcal{M}$, c'est-à-dire, que le même degré de suspicion leur ai affecté (voir l'exemple 3.1). Lorsque ce cas se produit, deux variantes de l'EXAM score sont définies pour mesurer l'efficacité, à savoir : l'EXAM score *pessimiste* noté P-EXAM et *optimiste* noté O-EXAM.

Définition 3.5 (O-EXAM)**.** *Cette variante est utilisée lorsque, l'instruction fautive est examinée en premier, parmi le groupe d'instructions équivalentes, dont elle fait partie. Dans ce cas, l'efficacité de la méthode est dite sur-estimée.*

Définition 3.6 (P-EXAM)**.** *A l'inverse du* O-EXAM*, ici, l'instruction fautive est examinée en dernier, parmi le groupe d'instructions équivalentes, dont elle fait partie. Dans ce cas, l'efficacité de la méthode est sous-estimée.*

Il existe également une troisième métrique, le Δ-EXAM = O-EXAM - P-EXAM, représentant la marge de l'EXAM score . En d'autres termes, Δ-EXAM représente la distance entre le score optimiste et pessimiste. Cette dernière reflète également la précision d'une méthode donnée. En effet, plus le Δ-EXAM est petit, plus la méthode est précise en retournant un classement distinct des instructions et non des ensembles larges d'instructions équivalentes.

Pour résumer, une méthode $\mathcal{M}_1$ est plus efficace qu'une méthode $\mathcal{M}_2$, si l'EXAM score de $\mathcal{M}_1$ est plus petit que l'EXAM score de $\mathcal{M}_2$. $\mathcal{M}_1$ nécessitera l'examen de moins d'instructions que $\mathcal{M}_2$ pour localiser la faute.

3.6. Conclusion

Dans ce chapitre, nous avons d'une part, présenté l'ensemble des outils et des techniques nécessaires, pour la compréhension de l'objectif de la localisation de fautes.

D'autres part, nous avons également situé, la phase de localisation dans le processus de test logiciel, dans laquelle nous apportons une contribution. Nous nous sommes surtout intéressés, aux mesures de suspicion, avec lesquelles, nous nous comparons dans le chapitre 6. Nous avons présenté, quelques approches de localisation, notamment celles basées sur la fouille de données, car c'est dans ce champ de recherche, que s'inscrivent nos contributions.

Troisième partie .

Contributions

4. Une contrainte globale pour l'Extraction de Motifs Fréquents Fermés

Dans ce chapitre, nous présentons une contrainte globale nommée ClosedPattern qui capture le problème de l'Extraction de Motifs Fréquents Fermés (EMFF). La contrainte ClosedPattern ne fait appel ni aux contraintes réifiées, ni à des variables auxiliaires pour encoder et capturer efficacement la sémantique particulière du problème de EMFF. Nous présentons un algorithme de filtrage pour la contrainte ClosedPattern qui assure la consistance de domaine DC à chaque nœud en temps et espace polynomiale.

Sommaire

4.1. Contexte et motivations

Les récentes approches PPC proposées pour modéliser et résoudre le problème de l'extraction de motifs fréquents fermés (EMFF) de manière déclarative [De Raedt et al., 2008, Guns et al., 2011], ont reçu dernièrement une attention considérable, et ont prouvé leur efficacité et leur flexibilité quant à l'ajout/-suppression des contraintes définies par l'utilisateur. L'inconvénient majeur de ces approches, est que la modélisation fait appel à un grand nombre de contraintes réifiées et de variables auxiliaires. Charger toutes ces contraintes en mémoire est équivalent à charger trois bases de transactions, ce qui rend ce type d'approches particulièrement sensible et questionnable pour le passage à l'échelle et le traitement des bases de transactions de taille considérable (voir l'exemple 2.9 donné dans la section 2.7). Les approches classiques [Pei et al., 2000, Zaki and Hsiao, 2002, Uno et al., 2003] offrent une bonne efficacité pour le traitement des grandes bases, mais restent néanmoins très rigides pour la prise en considération de nouveaux besoins, qui se traduisent par la nécessité d'une modification assez conséquente des algorithmes dédiés. Cette dernière remarque est particulièrement vraie pour le modèle riche de localisation de fautes que nous proposons dans nos travaux [Maamar et al., 2016c] (voir chapitre 5).

L'objectif de la contrainte globale ClosedPattern est de se placer à la croisée de ces deux constatations, à savoir, (i) La rigidité des approches classiques pour la prise en compte de nouvelles contraintes et (ii) La lourdeur du modèle réifié. L'idée est d'avoir un modèle où l'espace de recherche est défini uniquement sur des variables décisionnelles qui sont les variables représentant le motif recherché, avec un algorithme de filtrage efficace assurant la consistance de domaine. Notre travail a également été motivé par l'approche Prefix-Projection qui a été présentée dans [Kemmar et al., 2015], où les auteurs ont proposé une contrainte globale pour l'extraction de séquences et qui a montré son apport pratique, comparé au modèle réifié existant pour la fouille de séquences.

Une autre source de motivation, est naît du besoin d'avoir un modèle performant pour la phase de localisation de fautes, basée sur l'extraction de motifs ensemblistes traitée dans le chapitre 5. Un tel modèle nous permettra en effet, d'avoir à notre disposition un cadre riche et générique, permettant de raisonner sur les régularités des exécutions et d'attaquer la combinatoire des combinaisons d'instructions, qui est de nature complexe.

4.2. Contrainte globale ClosedPattern

Cette section présente la contrainte globale ClosedPattern, en décrivant formellement la définition de la contrainte, l'encodage et les propriétés utiles autour de la contrainte. Nous présentons également, les règles de filtrage proposées et l'algorithme qui assure les règles associées à la contrainte ClosedPattern.

TABLE 4.1.: Base de transactions $\mathcal{D}$: représentation horizontale (a) représentation binaire (b)

Trans.	Items							
t_1		B	C			G	H	
t_2	A			D				
t_3	A		C	D			H	
t_4	A				E	F		
t_5		B			E	F	G	

(a)

Trans	A	B	C	D	E	F	G	H
t_1	0	1	1	0	0	0	1	1
t_2	1	0	0	1	0	0	0	0
t_3	1	0	1	1	0	0	0	1
t_4	1	0	0	0	1	1	0	0
t_5	0	1	0	0	1	1	1	0

(b)

4.2.1. Encodage et notations

Étant donné n items, soit P le motif fréquent fermé recherché, encodé à l'aide des variables booléennes $P_1, \ldots, P_n$ représentant les items du motif. On dénote par σ une instanciation partielle obtenue à partir des variables $P_1, \ldots, P_n$ et dont les domaines sont des singletons. Les items de P peuvent être alors en trois sous-ensembles :

- Items présents :
  $\sigma^+ = \{j \in 1..n \mid P_j = 1\}$. σ^+ représente l'ensemble des items instanciés à 1.
- Items absents :
  $\sigma^- = \{j \in 1..n \mid P_j = 0\}$. σ^- représente l'ensemble des items instanciés à 0.
- Items libres :
  $\sigma^* = \{1, .., n\} \setminus \{\sigma^+ \cup \sigma^-\}$. σ^* représente l'ensemble des items non instanciés. Par conséquent, si $\sigma^* = \emptyset$ alors σ est une instanciation complète.

4.2.2. Définition

La contrainte globale CLOSEDPATTERN assure à la fois, la propriété de fréquence minimale et la propriété de fermeture.

Définition 4.1 (CLOSEDPATTERN)**.** *Soit $P_1, \ldots, P_n$ les variables booléennes représentant les items. Soit $\mathcal{D}$ la base de transactions et θ le seuil de fréquence minimale. Étant donné une instanciation complète σ sur $P_1, \ldots, P_n$,* $\text{CLOSEDPATTERN}_{\mathcal{D},\theta}(\sigma)$ *est satisfaite, si et seulement si, $freq_{\mathcal{D}}(\sigma^+) \geq \theta$ et σ^+ forme un motif fermé.*

Exemple 4.1. *Soit la base de transactions de la table 4.1 avec $\theta = 2$. Soit $P = \langle P_1, ..., P_8 \rangle$ avec $dom(P_i) = \{0, 1\}$ pour $i \in \{1..8\}$. Considérons le motif AD encodé par $P = \langle 10010000 \rangle$, où $\sigma^+ = \{A, D\}$ et $\sigma^- = \{B, C, E, F, G, H\}$.* $\text{CLOSEDPATTERN}_{\mathcal{D},2}(P)$ *est satisfaite, car $freq(\{A, D\}) \geq 2$ et $\{A, D\}$ forme un motif fermé.*

Soit σ une instanciation partielle des variables de P et i un item libre. Nous adoptons la représentation verticale de la base de transactions (voir section 1.3), dénotée par $\mathcal{V}_{\mathcal{D}}$ où pour chaque item, la liste des transactions le contenant sont stockées : $\forall i \in \mathcal{I}, \mathcal{V}_{\mathcal{D}}(i) = \mathcal{T}_{\mathcal{D}}(i)$. Par ailleurs, on dénote par $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i)$ la projection de la couverture courante de l'instanciation σ^+ sur la couverture de l'item i :

$$\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i) = \mathcal{T}_{\mathcal{D}}(\sigma^+ \cup \{i\}) = \mathcal{T}_{\mathcal{D}}(\sigma^+) \cap \mathcal{T}_{\mathcal{D}}(\{i\}).$$

Définition 4.2 (Instanciation extensible)**.** *Soit la contrainte* ClosedPattern$_{\mathcal{D},\theta}$ *sur* $P_1, .., P_n$*, une instanciation partielle est dite extensible, si et seulement si, elle peut être étendue à une instanciation complète des variables* $P_1, .., P_n$ *qui satisfait* ClosedPattern$_{\mathcal{D},\theta}$*.*

Nous caractérisons dans ce qui suit, le cas où une instanciation partielle, peut être extensible par rapport à la contrainte ClosedPattern.

Proposition 4.1. *Soit* σ *une instanciation partielle des variables* $P_1, \ldots, P_n$*.* σ *est une instanciation extensible, si et seulement si,* $freq_{\mathcal{D}}(\sigma^+) \geq \theta$ *et* $\nexists j \in \sigma^-$ *tel que* $\{j\}$ *est une extension propre de* σ*.*

Preuve. D'après la propriété d'anti-monotonie de la fréquence (cf. Propriété 1.1), si l'instanciation partielle σ n'est pas fréquente (i.e., $freq_{\mathcal{D}}(\sigma^+) < \theta$), elle ne peut donc sous aucune circonstance, être étendue à un motif *fréquent* fermé.
Soit maintenant une instanciation partielle fréquente σ (i.e., $freq_{\mathcal{D}}(\sigma^+) \geq \theta$), prenons $j \in \sigma^-$ tel que $\{j\}$ est une extension propre de σ. Il en résulte que $\mathcal{T}_{\mathcal{D}}(\sigma^+) = \mathcal{T}_{\mathcal{D}}(\sigma^+ \cup \{j\}) = \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)$. Par conséquent, $Clos(\sigma^+) = Clos(\sigma^+ \cup \{j\})$. Sachant que σ^+ sans j (j étant dans σ^-) ne peut être étendu à un motif fermé. S'il n'existe pas d'item $j \in \sigma^-$ tel que $\{j\}$ est une *extension propre* de σ, alors l'instanciation courante σ peut être certainement prolongée à un motif fermé en adoptant une extension propre pour former un motif fermé. □

4.3. Règles de filtrage

Nous donnons dans cette section les règles de filtrage de la contrainte ClosedPattern, en caractérisant quand est ce qu'une valeur est inconsistante.

Proposition 4.2 (ClosedPattern : *Règles de filtrage*)**.** *Soit* σ *une instanciation partielle consistante des variables* $P_1, \ldots, P_n$*, et* P_j *(*$j \in \sigma^*$*) une variable libre. Les trois cas suivants caractérisent l'inconsistance des valeurs* 0 *et* 1 *de la variable* P_j *:*

- $0 \notin dom(P_j)$ *ssi :* $\{j\}$ *est une extension propre de* σ*.* *(règle 1)*
- $1 \notin dom(P_j)$ *ssi :*
  - $|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)| < \theta \quad \vee$ *(règle 2)*
  - $\exists k \in \sigma^-, \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(k)$*.* *(règle 3)*

règle 1 : Cette règle prend son origine dans la notion de *merging item* proposée dans [Wang et al., 2003], cela se traduit par la question suivante : existe-il un item j qui

est nécessairement, une extension de l'instanciation partielle courante σ, sans changer sa couverture ? (i.e., cet item j est présent dans toutes les transactions couvrant le σ actuel).

règle 2 : Cette règle est une règle de base, dérivée de la propriété d'anti-monotonie de la fréquence (Propriété 1.1). Elle se traduit par la question suivante : Peut-on potentiellement construire un motif *fréquent*, à partir de l'instanciation partielle courante σ ?

règle 3 : Cette règle est originale, elle tire son raisonnement des items absent du motif courant, la question que l'on se pose est : existe-il des item(s) dont l'absence est nécessairement impliquée par l'absence d'autres items. La couverture de ce type d'items est un sous-ensemble de la couverture de l'item absent.

Preuve. Soit σ une instanciation partielle extensible et P_j une variable libre.

$0 \notin dom(P_j)$: ($\Rightarrow$) Soit 0 une valeur inconsistante. Dans ce cas, P_j peut prendre seulement la valeur 1. Ce qui implique que $Clos(\sigma^+) = Clos(\sigma^+ \cup \{j\})$. Donc, $\mathcal{T}_{\mathcal{D}}(\sigma^+) = \mathcal{T}_{\mathcal{D}}(\sigma^+ \cup \{j\})$. Par définition 1.9, $\{j\}$ est une extension propre de σ.
($\Leftarrow$) Soit $\{j\}$ une extension propre de σ, ce qui signifie que $Clos(\sigma^+) = Clos(\sigma^+ \cup \{j\})$ (définition. 1.9). La valeur 0 est inconsistante sachant que j ne peut pas être dans σ^- (Proposition.4.1).

$1 \notin dom(P_j)$: ($\Rightarrow$) Soit 1 une valeur inconsistante. Ceci peut être le cas si la fréquence du motif courant σ^+ devient inférieure au seuil θ lorsqu'on y ajoute l'item j (i.e., $|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)| < \theta$). Ou, $\sigma^+ \cup \{j\}$ ne peut pas être étendu à un motif fermé : Ceci est le cas lorsqu'il existe un item $k \in \sigma^-$ tel que, à chaque fois que l'item j appartient à une transaction dans la base transactionnelle, k y appartient aussi ($\mathcal{V}_{\mathcal{D}}^{\sigma^+}(j) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(k)$). Par conséquent, l'absence de k (i.e., $k \in \sigma^-$) implique nécessairement l'absence de j aussi (j ne peut être dans un motif fermé sans k). Cela se traduit par : $(P_k = 0 \Rightarrow P_j = 0)$.
($\Leftarrow$) Ceci est une conséquence directe de la Proposition 4.1. □

Exemple 4.2. *Poursuivant l'exemple 4.1 avec $\theta = 2$, considérons une instanciation partielle σ telle que la variable P_1 est mise à 0 (item A). Ceci donne, $\sigma^- = \{A\}$ et $\sigma^+ = \emptyset$. La valeur 1 de $dom(P_4)$ (item D) est inconsistante, car l'absence de A implique celle de D dans la base $\mathcal{D}$ (i.e., $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(D) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(A)$). Soit maintenant $P_1 = 0, P_4 = 0$, Ceci donne, $\sigma^- = \{A, D\}$ et $\sigma^+ = \emptyset$. Si la variable P_3 est mise à 1 (item C), la valeur 1 de P_i, avec $i = \{2, 5, 6, 7\}$ (items B, E, F, G) est inconsistante car $|\mathcal{V}_{\mathcal{D}}^{\sigma^+}(P_i)| < 2$. La valeur 0 de P_8 (item H) est aussi inconsistante, car $\{H\}$ est une extension propre de σ.*

4.4. Algorithme de filtrage

Dans cette section, nous présentons un algorithme de filtrage, nommé FilterClosedPattern (Algorithm 2) pour maintenir la consistance de domaine sur la

contrainte ClosedPattern, à chaque nœud de l'arbre de recherche des motifs fréquents fermés.

L'algorithme Filter-ClosedPattern (Algorithme 2) maintient de manière incrémentale, les structures de données $\sigma =< \sigma^+, \sigma^-, \sigma^* >$ et la couverture correspondante à cette instanciation σ à savoir $\mathcal{T}_{\mathcal{D}}(\sigma^+)$. Plus précisément, Filter-ClosedPattern maintient une pile où chaque σ est stocké avec sa couverture $\mathcal{T}_{\mathcal{D}}(\sigma^+)$, en cas de retour arrière (backtrack), on dépile la paire $\langle \sigma, \mathcal{T}_{\mathcal{D}}(\sigma^+) \rangle$ correspondante. A l'aide de ces deux structures de données, nous pouvons à tout moment constater, si un item donné est présent ou pas dans la base verticale $\mathcal{V}_{\mathcal{D}}$.

L'algorithme Filter-ClosedPattern prend en paramètres d'entrée, la base verticale $\mathcal{V}_{\mathcal{D}}$, le seuil de support minimal θ, l'instanciation courante σ où $\sigma^* \neq \emptyset$ et les variables de P. En sortie, l'algorithme Filter-ClosedPattern réduit les domaines des variables P_i et par conséquent, il augmente la taille de l'ensemble σ^+ et/ou σ^- et réduit celle de σ^*.

L'algorithme commence par tester, si l'instanciation partielle courante est *extensible* ou pas (Proposition 4.1). Ceci est réalisé en testant :

1. Si la taille de la couverture courante est inférieure au support minimal (ligne 4).
2. Si aucun item d'une variable instanciée à zero n'est une extension propre de σ^+ (ligne 5), c'est-à-dire que toutes les transactions couvrant l'instanciation courante, contiennent cet item.

Les lignes (6-14) sont une application directe des *règle 1* et *règle 2* de la proposition 4.2. Pour chaque variable non instanciée, nous testons :

1. Si la valeur 0 est consistante : cet item n'est pas une extension propre de σ (lignes 7-10).
2. Si la valeur 1 est consistante : la taille de la nouvelle couverture du motif courant augmenté de cet item, reste fréquente (i.e., $\geq$ à θ) (lignes 11-14).

Enfin, les lignes (15-20) implémentent la *règle 3* de la proposition 4.2. L'algorithme filtre la valeur 1 de chaque variable d'item libre $i \in \sigma^*$ tel que, sa couverture est incluse dans la couverture d'un item absent $j \in \sigma^-$ ($\mathcal{V}_{\mathcal{D}}^{\sigma^+}(i) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(j)$).

4.5. Analyse de la complexité

Nous présentons dans cette section, une étude de la complexité en temps de calcul et en espace de l'algorithme de filtrage (Algo.2), associé à la contrainte ClosedPattern.

Théorème 4.1. *Étant donnée une base de transactions $\mathcal{D}$ avec n items et m transactions, et un support minimal θ. L'algorithme* Filter-ClosedPattern *maintient la consistance de domaine sur la contrainte* ClosedPattern*, ou prouve son inconsistance dans un temps $O(n^2 \times m)$ avec une complexité en espace en $O(n \times m)$.*

Preuve.

▶ **Consistance de domaine DC :**

L'algorithme Filter-ClosedPattern implémente précisément la Proposition 4.1 et

Algorithme 2 : FILTER-CLOSEDPATTERN($\mathcal{V}_\mathcal{D}, \theta, \sigma, P$)

1 **Données** : $\mathcal{V}_\mathcal{D}$: base verticale ; θ : support minimal
2 **Données internes** : $P = \{P_1 \dots P_n\}$: variables booléennes d'items ; σ : instanciation courante.
3 **Début**
4 **si** ($|\mathcal{T}_\mathcal{D}(\sigma^+)| < \theta$) **alors retourner** $false$;
5 **si** $\exists\ i \in \sigma^- : |\mathcal{V}_\mathcal{D}^{\sigma^+}(i)| = |\mathcal{T}_\mathcal{D}(\sigma^+)|$ **alors retourner** $false$;
6 **Pour chaque** $i \in \sigma^*$ **faire**
7 **si** ($|\mathcal{V}_\mathcal{D}^{\sigma^+}(i)| = |\mathcal{T}_\mathcal{D}(\sigma^+)|$) **alors**
8 $dom(P_i) \leftarrow dom(P_i) - \{0\}$;
9 $\sigma^+ \leftarrow \sigma^+ \cup \{i\}$;
10 $\sigma^* \leftarrow \sigma^* \setminus \{i\}$;
11 **else if** ($|\mathcal{V}_\mathcal{D}^{\sigma^+}(i)| < \theta$) **then**
12 $dom(P_i) \leftarrow dom(P_i) - \{1\}$;
13 $\sigma^- \leftarrow \sigma^- \cup \{i\}$;
14 $\sigma^* \leftarrow \sigma^* \setminus \{i\}$;
15 **Pour chaque** $i \in \sigma^-$ **faire**
16 **Pour chaque** $j \in \sigma^*$ **faire**
17 **si** ($\mathcal{V}_\mathcal{D}^{\sigma^+}(j) \subseteq \mathcal{V}_\mathcal{D}^{\sigma^+}(i)$) **alors**
18 $dom(P_j) \leftarrow dom(P_j) - \{1\}$;
19 $\sigma^- \leftarrow \sigma^- \cup \{j\}$;
20 $\sigma^* \leftarrow \sigma^* \setminus \{j\}$;
21 **retourner** $true$;

les trois règles de filtrage données dans la Proposition 4.2. Ceci nous amène à dire que l'algorithme FILTER-CLOSEDPATTERN assure la consistance de domaine.

▶ **Temps :**
Soit $n = |\mathcal{I}|$ et $m = |\mathcal{T}|$. Tout d'abord, nous avons besoin de calculer $\mathcal{T}_\mathcal{D}(\sigma^+)$, ce qui requiert dans le pire des cas $O(n \times m)$. Ce calcul est fait une seule fois durant toute la résolution. La couverture $\mathcal{V}_\mathcal{D}^{\sigma^+}(i)$ peut être calculée en faisant l'intersection de $\mathcal{T}_\mathcal{D}(\sigma^+)$ (déjà calculé) et $\mathcal{T}_\mathcal{D}(\{i\})$ (donné par la base verticale) dans le pire des cas en $O(m)$. Tester les *règles 1* et *2* sur l'ensemble des variables libres se fait en $O(n \times m)$ (lignes 6-14). Toutefois, le test de la *règle 3* est cubique (lignes 15-20), plus précisément en $O(n \times (n \times m))$, où le test d'inclusion de la couverture $\mathcal{V}_\mathcal{D}^{\sigma^+}(i)$ peut être fait en $O(m)$. Au final, dans le pire des cas la complexité est en $O(n \times (n \times m))$.

▶ **Espace :**
La complexité en espace de FILTER-CLOSEDPATTERN, réside dans le stockage des

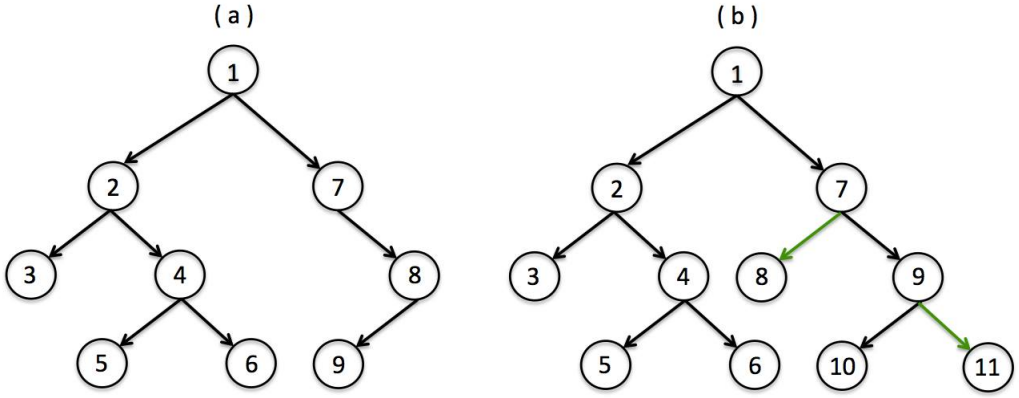

FIGURE 4.1.: (a) Arbre binaire simple (b) Arbre binaire entier

structures de données $\mathcal{V}_\mathcal{D}$, σ et la couverture $\mathcal{T}$. La base verticale $\mathcal{V}_\mathcal{D}$ nécessite au plus $(n \times m)$ en espace mémoire. Dans le pire des cas, nous avons à stocker n items dans σ et m transactions dans $\mathcal{T}$. Ce qui donne dans le pire des cas, une complexité en $O(n \times m + n + m) = O(n \times m)$.

Corollaire. Durant le processus de résolution en profondeur d'abord, la complexité totale en espace est en $O(n \times (m + n))$ car : (1) la profondeur de l'arbre dans le pire des cas est n ; (2) σ et $\mathcal{T}$ nécessitent alors $O(n \times (m + n))$; (3) la base verticale est la même durant tout le processus de résolution $O(n \times m)$; (4) Nous avons donc au final : $O(n \times (m + n)) + O(n \times m) = O(n \times (m + n))$.

Proposition 4.3 (Résolution sans retour arrière)**.** *L'extraction de tous les motifs fréquents fermés, noté $\mathcal{C}$ avec* ClosedPattern*, est sans retour arrière (on retourne en arrière uniquement lorsqu'on atteint une feuille, l'arbre ne contient aucun nœud échec). La complexité totale de* Filter-ClosedPattern *est alors en $O(\mathcal{C} \times n^2 \times m)$ pour propager la contrainte* ClosedPattern*.*

Preuve. Filter-ClosedPattern assure la consistance de domaine DC à chaque nœud de l'arbre de recherche. Par conséquent, on a la garantie de produire les motifs fréquents fermés, sans retour arrière. L'arbre de recherche est en conséquence un arbre *binaire entier*, où chaque nœud est soit une feuille (une solution), ou bien possède exactement deux nœuds fils (voir la figure 4.1). Le nombre total de nœuds est donc égal à $O(2 \times \mathcal{C})$. Sachant que la DC est assurée en $O(n^2 \times m)$, l'extraction de l'ensemble total des motifs fréquents fermés est donc en $O(\mathcal{C} \times n^2 \times m)$.

4.6. Illustration

Dans cette section, nous illustrons la propagation de notre contrainte ClosedPattern, ainsi qu'une comparaison avec le modèle de contraintes réifiées, détaillé dans la

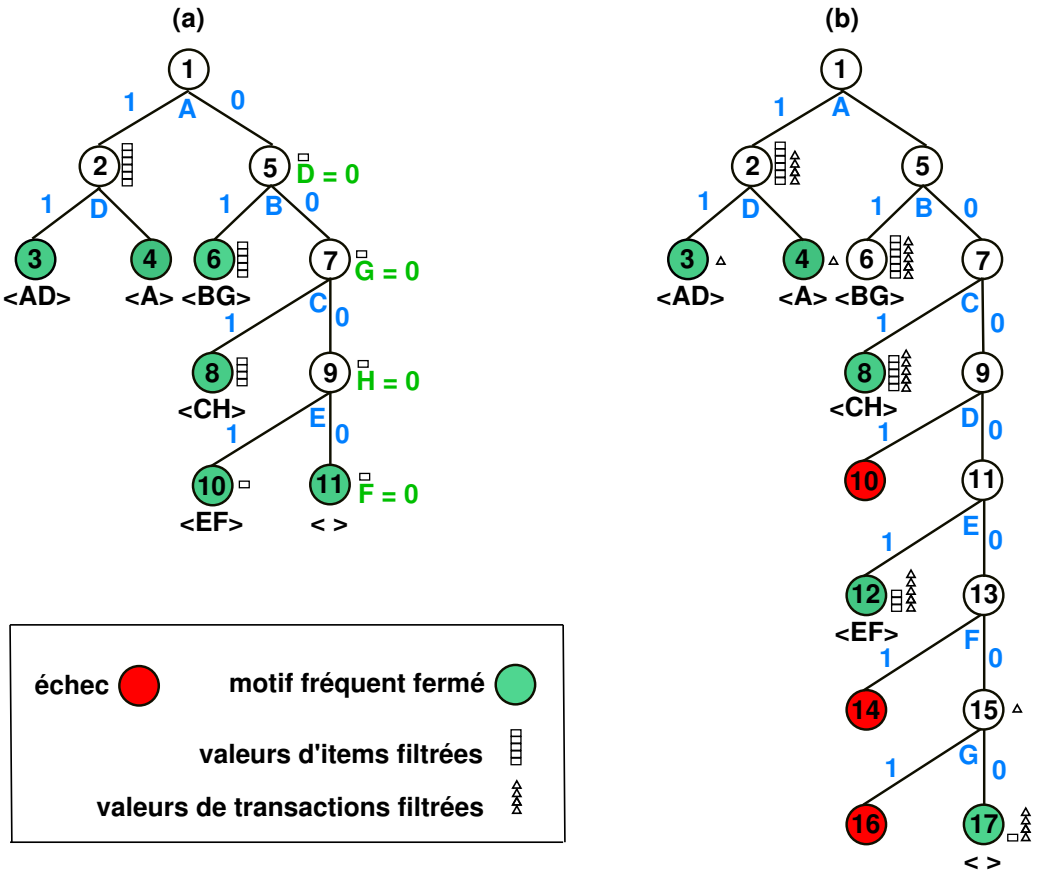

FIGURE 4.2.: (a) Contrainte CLOSEDPATTERN (b) Modèle de Contraintes Réifiées (RCM)

section 2.7 et dénoté par RCM (Reified Constraint Model). Pour cela, nous reprenons l'exemple de la base de transactions de la table 4.1 avec 5 transactions $\{t_1, .., t_5\}$ et 8 items $\{A, .., H\}$.

La figure 4.2 montre l'arbre de recherche exploré par la contrainte CLOSEDPATTERN (partie(a)) et l'arbre de recherche exploré avec le modèle réifié (partie(b)) pour extraire les motifs fréquents fermés avec une fréquence minimale $\theta = 2$. Dans cet exemple, les deux approches utilisent les mêmes heuristiques de branchement, à savoir `Lex` sur les variables (ordre lexicographique) et `Max_val` pour les valeurs (valeur maximale).

Il est important de noter, que l'espace de recherche exploré par CLOSEDPATTERN est défini uniquement sur les variables de décision représentant les items, tandis que le modèle réifié ajoute une autre dimension avec des variables auxiliaires représentant les transactions.

Au niveau du nœud racine (nœud 1), aucun filtrage n'est opéré dés lors que tous les items sont fréquents et qu'il n'existe aucun item qui est une extension propre du motif vide (en d'autres termes, il n'y a aucun item présent dans toutes les transactions, faisant

ainsi que cet item sera nécessairement dans les motifs fermés trouvés, voir la table 4.1).

Par la suite, ClosedPattern et RCM se comportent de la même manière sur la branche $A = 1$. Avec $A = 1$ les items B, C, E, F, G, H deviennent non fréquents puisque seuls les transaction t_2, t_3, t_4 sont à considérer (couverture de A), la valeur 1 est donc filtrée des domaines de ces variables en appelant la *règle 2*. Avec le modèle RCM sur le nœud 2, le filtrage de ces cinq variables de décision (variables d'items) provoque un filtrage sur quatre autres variables auxiliaires (transactions). Sur la branche $A = 1$, deux solutions (motifs fréquents fermés) sont trouvées : $\langle AD \rangle$ et $\langle A \rangle$.

En branchant sur $A = 0$ (nœud 5), la valeur 1 de la variable D est filtrée avec la *règle 3*, car nous avons $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(D) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(A)$ avec $\sigma^+ = \langle\rangle$ et par conséquent $D \Rightarrow A$ et donc le branchement $A = 0$ réduit D à 0. Nous pouvons ainsi dire qu'on maintient bien la DC sur le nœud 5 avec ClosedPattern, ce qui n'est pas le cas du modèle RCM. La même observation est faite sur les nœuds suivants :

- nœud 7 : nous avons le branchement $B = 0$ et $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(G) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(B)$, par conséquent $G \Rightarrow B$.
- nœud 9 : nous avons le branchement $C = 0$ et $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(H) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(C)$, par conséquent $H \Rightarrow C$.
- nœud 11 : nous avons le branchement $E = 0$ et $\mathcal{V}_{\mathcal{D}}^{\sigma^+}(F) \subseteq \mathcal{V}_{\mathcal{D}}^{\sigma^+}(E)$, par conséquent $F \Rightarrow E$.

Prenons le nœud 6, les branchements sur $A = 0$ et $B = 1$ rendent les items C, D, E, F, H non fréquents (*règle 2*). Par ailleurs, la *règle 1* peut être appliquée puisque l'item G est une extension propre du motif $\langle B \rangle$ (i.e., nous ne pouvons en aucun cas avoir un motif fréquent fermé comprenant l'item B sans l'item G). La valeur 0 est alors filtrée du domaine de G, ce qui permet d'atteindre la solution $\langle BG \rangle$. La même observation est faite sur le nœud 8 où H est une extension propre de $\langle C \rangle$.

En conclusion, Filter-ClosedPattern assure la consistance de domaine à chaque nœud et par conséquent énumère toutes les solutions sans retour arrière (pas de nœud d'échec). Cette même analyse ne peut pas être faite sur le modèle `RCM`, car le filtrage qu'effectue la *règle 3* n'est jamais déclenché, ce qui implique des échecs sur ces valeurs inconsistantes.

4.7. Etude expérimentale de ClosedPattern

Dans cette section, nous présentons l'étude expérimentale menée sur plusieurs bases de transactions, pour comparer et évaluer les apports pratiques de notre contrainte globale, par rapport aux approches de l'état de l'art. Nous commençons par décrire la base de benchmarking. Puis, nous décrivons le protocole expérimental, ensuite nous discutons les résultats obtenus en termes de statistiques PPC et de temps de calcul.

TABLE 4.2.: Descriptif des bases de transactions retenues pour nos expérimentations

Base	$\|\mathcal{T}\|$	$\|\mathcal{I}\|$	$\|\widehat{\mathcal{T}}\|$	ρ	Type de données	Taille
Chess	3 196	75	37	49%	game steps	239 700
Splice1	3 190	287	60	21%	genetic sequences	915 530
Mushroom	8 124	119	23	19%	species of mushrooms	966 756
Connect	67 557	129	43	33%	game steps	8 714 853
BMS-Web-View1	59 602	497	2.5	0.5%	web click stream	29 622 194
T10I4D100K	100 000	1 000	10	1%	synthetic dataset	100 000 000
T40I10D100K	100 000	1 000	40	4%	synthetic dataset	100 000 000
Pumsb	49 046	7 117	74	1%	census data	349 060 382
Retail	88 162	16 470	10	0.06%	retail market basket data	1 452 028 140

4.7.1. Jeux de données de FIMI

Nous avons sélectionné plusieurs bases réelles et synthétiques [Zaki and Hsiao, 2002, Grahne and Zhu, 2005] de grande taille du dépôt FIMI [1]. Ces bases possèdent différentes caractéristiques selon les domaines d'application. La table 4.2 résume pour chaque base, le nombre de transactions $|\mathcal{T}|$, le nombre d'items $|\mathcal{I}|$, la taille moyenne des transactions $|\widehat{\mathcal{T}}|$, la densité de la base ρ qui est donnée par $|\widehat{\mathcal{T}}|/|\mathcal{I}|$ et la taille de la base (i.e., $|\mathcal{T}| \times |\mathcal{I}|$). Le choix des bases est motivé par la variété de leur nombre de transactions, nombre d'items et la densité. Certaines bases comme Chess et Connect sont très denses (respectivement, 49% et 33%), d'autres bases comme Retail et BMS-Web-View1 sont très creuses (respectivement, 0.06% et 0.5%). La différence entre ces deux types de bases est que les bases denses ont tendance à produire un nombre important de motifs fermés, comparées aux bases dites creuses. A noter que la taille des bases de notre benchmark varie de l'ordre de $\approx 10^5$ à 10^9.

4.7.2. Protocole expérimental

La mise en œuvre de notre approche a été réalisée en C++, sous GECODE [2], un solveur ouvert, libre et portable, pour le développement des programmes à contraintes. Toutes nos structures de données ont été implémentées, à l'aide des conteneurs de la bibliothèque standard de C++ STL. Les expérimentations ont été menées sous LINUX avec un processeur Intel Xeon E3-1245 V2 @ 3.40 Ghz avec une mémoire vive de 32 Gb et un timeout de 3600 secondes. Pour chaque base de transactions, nous avons fait varier le seuil de fréquence θ jusqu'à ce qu'il soit impossible pour les méthodes d'extraire l'ensemble des motifs avec le temps/mémoire alloué. Nous comparons notre approche CLOSEDPATTERN avec :

i) `CP4IM`, une référence dans l'état de l'art des approches basées sur la programmation par contraintes, qui utilise le modèle réifié `RCM`.

ii) `LCM`, une référence dans l'état de l'art des algorithmes classiques spécialisées.

1. http://fimi.ua.ac.be/data/
2. www.gecode.org

Pour ClosedPattern et CP4IM, nous avons utilisé les mêmes stratégies de branchement, à savoir Lex sur les variables et Max_val pour les valeurs. Nous avons expérimenté avec les distributions disponibles de LCM-v5.3,[3] et CP4IM,[4]. A noter que, CP4IM est également mis en œuvre sous Gecode.

4.7.3. Résultats et Discussions

a)- Approches PPC : ClosedPattern vs CP4IM

Nous commençons par comparer dans un premier temps les deux approches PPC, ClosedPattern et CP4IM, en termes de nombre de propagations, de nombre de nœuds explorés, de la mémoire consommée, de la profondeur maximale de l'arbre de recherche et du nombre de nœuds où la valeur de branchement n'est pas consistante (nœud échec). Le tableau 4.3 résume pour chaque base et pour chaque seuil de fréquence donnée, les performances des deux approches.

Une première remarque est que ClosedPattern peut traiter toutes les bases proposées, ce qui n'est pas tout le temps le cas de CP4IM où dans certains cas, le modèle réifié est sujet à un *out of memory*, dû au nombre très important de contraintes réifiées généré par le modèle, c'est le cas des bases : Retail et BMS-Web-View1. A titre d'exemple prenons la base Retail, le modèle CP de CP4IM contient $|\mathcal{T}| + 2 \times |\mathcal{I}| = 121\ 102$ contraintes réifiées portant sur $\mathcal{T}| + |\mathcal{I}| = 104\ 596$ variables que le modèle doit charger en mémoire.

Nombre de propagations : Pour ClosedPattern le nombre de propagations est de manière générale très réduit par rapport à CP4IM (jusqu'à un facteur de 30 537 dans le cas de la base T40I10D100K avec le seuil 10%), ceci n'est pas étonnant et s'explique par le nombre de contraintes dans le store de contraintes que doit propager le modèle réifié. Prenons par exemple la base Connect, avec un seuil $\theta = 80\%$, pour ClosedPattern il y a 49 985 propagations, là ou CP4IM fait 9 453 279 propagations.

Nombre de nœuds : En termes de nombre de nœuds, l'observation principale est que ClosedPattern, en assurant la consistance de domaine, produit un nombre de nœuds toujours plus réduit que CP4IM (jusqu'à un facteur $\simeq 2$, voir la base *mushroom* avec $\theta = 0.5\%$). Nous avons montré, que ce nombre peut être calculé de manière exacte (voir la proposition 4.3 et la section 4.6). Ce nombre est égal à $(\mathcal{C} \times 2 - 1)$, avec $\mathcal{C} =$ le nombre de solutions (motifs fréquents fermés). Ce résultat est confirmé par les résultats expérimentaux (voir le tableau 4.3). Sur la base *connect* et sur 3 instances de la base *splice-1*, ClosedPattern et CP4IM produisent le même nombre de nœuds. En effet, sur ces derniers cas CP4IM est également *sans retour arrière* (voir la dernière colonne de

3. http://research.nii.ac.jp/~uno/codes.htm
4. https://dtai.cs.kuleuven.be/CP4IM/

TABLE 4.3.: Statistiques des approches CLOSEDPATTERN et CP4IM

$\mathcal{D}$	θ	#Motifs	#Propagations		#Noeuds		Mémoire		Profondeur max		# Échecs	
	(%)		CLOPAT	CP4IM	CLOPAT	CP4IM	CLOPAT	CP4IM	CLOPAT	CP4IM	CLOPAT	CP4IM
mushroom	30	428	**1 316**	430 016	**855**	1 039	**26 200**	88 079	**21**	27	0	93
	20	1 198	**3 805**	1 030 445	**2 395**	3 071	**45 016**	90 831	**28**	42	0	339
	10	4 898	**15 820**	2 771 719	**9 795**	13 281	**64 192**	93 839	**31**	55	0	1 744
	5	12 855	**41 353**	5 574 143	**25 709**	36 495	**89 816**	97 552	**33**	72	0	5 394
	1	51 672	**154 886**	13 813 312	**103 345**	168 999	133 688	**102 608**	**34**	95	0	32 828
	0.5	76 199	**221 938**	18 018 929	**152 397**	259 427	138 808	**104 017**	**34**	103	0	53 516
	0.1	164 118	**455 888**	31 222 435	**328 235**	529 289	162 600	**106 641**	**34**	116	0	100 528
	0.05	203 882	**555 768**	36 520 438	**407 765**	622 145	166 696	**107 025**	**34**	118	0	107 191
chess	60	98 393	**314 425**	2 661 395	**196 785**	196 787	46 096	**15 623**	**28**	34	0	1
	50	369 451	**1 196 509**	9 201 740	**738 901**	738 907	60 456	**16 520**	**31**	37	0	3
	40	1 366 834	**4 487 065**	30 541 475	**2 733 667**	2 733 735	77 560	**18 568**	**31**	40	0	34
	30	5 316 468	OOM	**104 618 207**	OOM	**10 635 019**	OOM	**19 976**	OOM	**50**	OOM	1 042
connect	90	3 487	**10 993**	1 865 236	**6 973**	**6 973**	**31 632**	561 872	**12**	21	0	0
	80	15 108	**49 985**	9 453 279	**30 215**	**30 215**	**31 728**	585 040	**14**	28	0	0
	70	35 876	**119 651**	24 968 701	**71 751**	**71751**	**32 752**	594 640	**14**	31	0	0
	60	68 350	**227 027**	51 648 114	**136 699**	**136 699**	**41 208**	661 583	**15**	36	0	0
	50	130 102	**435 695**	98 600 221	**260 203**	**260 203**	**47 424**	665 871	**16**	38	0	0
	40	239 373	OOM	**177 644 355**	OOM	**478 745**	OOM	**675 088**	OOM	**41**	OOM	0
pumsb	95	111	**372**	176 422	**221**	235	**13 576**	33 597 055	**12**	13	0	7
	90	1 467	**4 735**	1 253 450	**2 933**	3 071	**30 680**	33 603 712	**17**	20	0	69
	85	8 514	**28 860**	5 658 514	**17 027**	17 739	**35 160**	33 605 567	**19**	24	0	356
	80	33 296	**121 506**	18 810 299	**66 591**	69 201	**39 616**	33 614 336	**19**	25	0	1305
	75	101 048	**377 292**	49 238 437	**202 095**	209 527	**44 048**	33 615 487	**21**	27	0	3716
retail	10	10	**28**	OOM	**19**	OOM	**8 760**	OOM	**5**	OOM	0	OOM
	5	17	**49**	OOM	**33**	OOM	**11 096**	OOM	**6**	OOM	0	OOM
	1	160	**527**	OOM	**319**	OOM	**106 696**	OOM	**70**	OOM	0	OOM
	0.5	581	**1 943**	OOM	**1 161**	OOM	**819 136**	OOM	**221**	OOM	0	OOM
	0.1	7 696	**25 786**	OOM	**15 391**	OOM	**73 332 792**	OOM	**2 140**	OOM	0	OOM
	0.05	19 699	**65 778**	OOM	**39 397**	OOM	**218 161 256**	OOM	**3 991**	OOM	0	OOM
T10I*	1	386	**1 289**	TO	**771**	TO	**2 346 408**	TO	**375**	TO	0	TO
	0.5	1 074	**3 311**	TO	**2 147**	TO	**5 215 176**	TO	**569**	TO	0	TO
	0.1	26 807	**71 976**	TO	**53 615**	TO	**9 571 712**	TO	**796**	TO	0	TO
	0.05	46 994	**131 335**	TO	**93 987**	TO	**10 311 392**	TO	**832**	TO	0	TO
	0.01	283 398	**907 389**	TO	**566 795**	TO	**10 910 080**	TO	**846**	TO	0	TO
T40I*	10	83	**276**	8 428 377	165	165	**144 472**	8 446 800	**82**	**82**	0	0
	1	65 237	**176 995**	TO	**130 473**	TO	**9 077 648**	TO	**755**	TO	0	TO
	0.5	1 275 940	**2 886 947**	TO	**2 551 879**	TO	**10 344 160**	TO	**838**	TO	0	TO
BMS-W-V1	0.16	32	**103**	OOM	**63**	OOM	**28 728**	OOM	**30**	OOM	0	OOM
	0.08	9 392	**29 782**	OOM	**18 783**	OOM	**2 160 600**	OOM	**352**	OOM	0	OOM
	0.06	64 763	**206 577**	OOM	**129 525**	OOM	**2 300 872**	OOM	**368**	OOM	0	OOM
	0.04	155 652	**510 007**	OOM	**311 303**	OOM	**2 400 184**	OOM	**384**	OOM	0	OOM
	0.02	422 693	**1 369 603**	OOM	**845 385**	OOM	**2 759 032**	OOM	**423**	OOM	0	OOM
	0.01	1 240 701	**4 084 099**	OOM	**2 481 401**	OOM	**3 223 416**	OOM	**450**	OOM	0	OOM
splice-1	20	244	**812**	913 779	**487**	**487**	827 352	**88 461**	**222**	**222**	0	0
	10	1 606	**5 376**	1 810 466	**3 211**	**3 211**	931 096	**89 677**	**240**	**240**	0	0
	5	31 441	**105 913**	35 816 836	**62 881**	**62 881**	931 096	**89 677**	**240**	**240**	0	0
	1	60 778 76	**20 332 362**	2 280 218 821	**12 155 751**	12 165 843	1 017 568	**93 070**	**240**	**240**	0	5 046

(OOM : Out Of Memory ; TO : TimeOut)

la table 4.3).

Mémoire : Concernant la consommation mémoire, CLOSEDPATTERN consomme de manière générale, moins de mémoire que CP4IM, sauf pour la base *splice-1* et 4 instances de la base *mushroom*, (voir la section 4.5 pour la complexité en espace de CLOSEDPATTERN).

Profondeur de l'arbre : La colonne de la profondeur maximale indique clairement que CLOSEDPATTERN développe un arbre de recherche dont la profondeur des branches est plus réduite, comparée à l'arbre produit par CP4IM, ceci est un résultat attendu car

toutes les valeurs inconsistantes sont filtrées par ClosedPattern (moins de valeurs de branchement à tester).

Nœuds échec : La dernière colonne du tableau 4.3 confirme la proposition 4.3 : ClosedPattern assure la consistance de domaine et donc il ne peut y avoir aucun échec dans l'arbre de recherche (sans retour arrière). Tandis que `CP4IM` produit un nombre important de nœuds échec sur certaines instances, jusqu'à 107 191 nœuds pour la base *mushroom* avec $\theta = 0.05\%$.

b) Comparaison des Temps de calcul : ClosedPattern vs `CP4IM` vs `LCM`

Nous comparons dans cette seconde partie, les trois approches ClosedPattern, `CP4IM` et `LCM` en termes de temps CPU. La figure 4.3 montre les courbes d'évolution du temps de calcul, en fonction du seuil de fréquence θ pour chaque base.

ClosedPattern **vs `CP4IM` :**
En plus des bases de transactions que `CP4IM` ne peut pas traiter pour raison de *out of memory* (respectivement, *time out*), à savoir les bases Retail et BMS-Web-View1 (respectivement, les bases T10I4D100K, T40I10D100K), ClosedPattern domine `CP4IM` sur les bases *mushroom* (jusqu'à un facteur de 3 avec $\theta = 0.05\%$), *Pumsb* (jusqu'à un facteur $\simeq 3000$ avec $\theta = 95\%$) et *splice-1* (jusqu'à un facteur de 62 avec $\theta = 20\%$).

Sur les deux bases *Chess* et *Connect*, `CP4IM` domine ClosedPattern, l'explication vient du fait que ces deux bases sont de nature *dense* et produisent un nombre important de motifs fréquents fermés (par exemple, 5 316 468 de motifs pour la base *Chess* avec $\theta = 30\%$), ce qui réduit la force du filtrage de ClosedPattern (peu de valeurs inconsistantes à filtrer). Nous avons par conséquent un algorithme de filtrage cubique sur chaque nœud, mais qui ne filtre pas un nombre important de valeurs inconsistantes.

ClosedPattern **vs `LCM` :**
Au niveau du temps CPU, `LCM` reste l'approche la plus performante, toutefois ClosedPattern peut traiter toutes les bases traitées par `LCM`. Par ailleurs, sur certaines bases telles que Retail, BMS-Web-View1 et Splice-1 l'écart reste raisonnable sur certaines valeurs de seuil θ. Prenons par exemple la base Retail avec $\theta \leq 0.5$, l'extraction de l'ensemble des motifs fréquents fermés est réalisée en moins de $1s$ avec les deux approches.

c) - Extraction de k-motifs :

Une voie prometteuse pour la découverte de motifs intéressants ; est de poser des contraintes sur un ensemble de k motifs (k-patterns sets) [Khiari et al., 2010, Guns et al., 2013]. Dans ce contexte, l'intérêt d'un motif est évalué par rapport à un ensemble de motifs. Nous proposons de modéliser et résoudre une instance particulière, où l'objectif est d'extraire les k motifs fermés $\{P^1, ..., P^k\}$ tels que :

(i) $\forall i \in [1, k]$: ClosedPattern (P^i). Ce qui implique que les k motifs recherchés doivent être fermés.

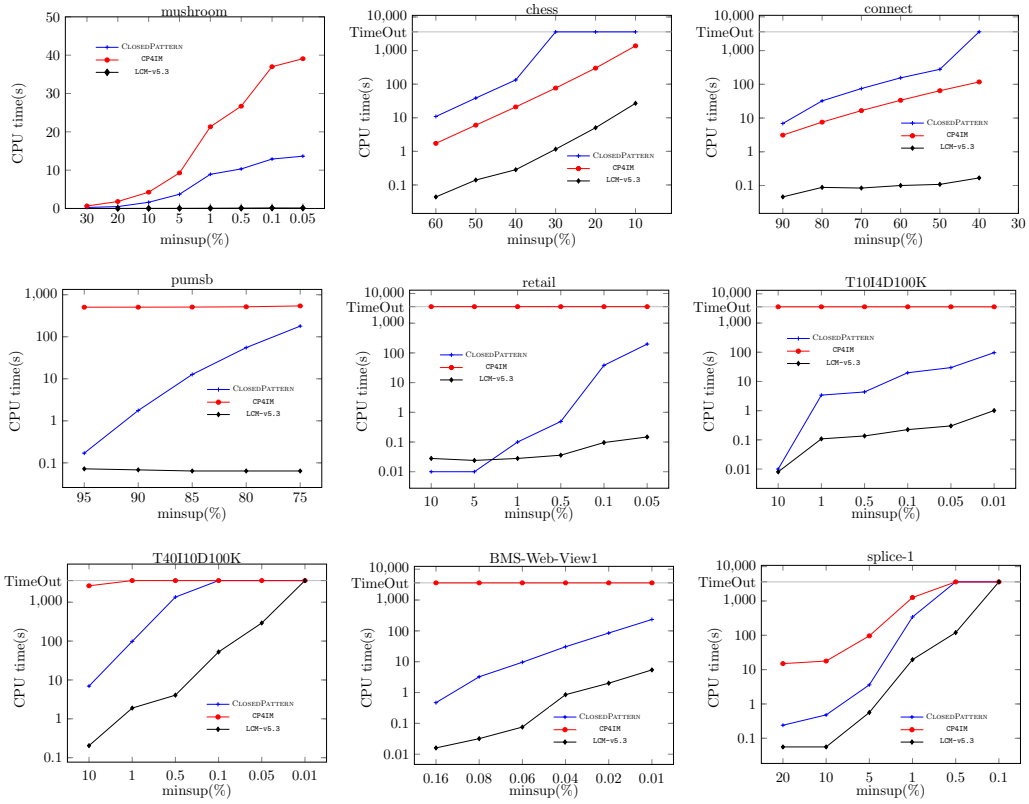

FIGURE 4.3.: Comparaison des temps CPU pour l'extraction des motifs fréquents fermés

(ii) $\forall i, j \in [1, k] : P^i \cap P^j = \emptyset$. Les k motifs doivent être totalement distincts.

(iii) $\forall i \in [1, k] : lb < |P^i| < ub$. La taille de chaque motif est contrainte par une borne min (lb) et une borne max (ub).

La figure 4.4 montre une comparaison entre la contrainte globale ClosedPattern, le modèle réifié `CP4IM` et l'algorithme `LCM`. Nous avons sélectionné deux bases de transactions où le nombre de motifs fermés est raisonnable, *chess* avec $\theta = 80\%$ (5084 motifs fermés) et *connect* avec $\theta = 90\%$ (3487 motifs fermés), et nous avons fait varier la valeur de k avec un timeout de 3600s. Après quelques tests préliminaires, les bornes lb et ub sur la taille des motifs, ont été fixées respectivement à 2 et 10.

Sur la base *chess*, l'approche ClosedPattern est robuste et se comporte bien : elle est linéaire sur k et ne dépasse jamais 6 minutes même avec $k = 12$ ($323.53s$). `CP4IM` Suit une échelle exponentielle et va au-delà du timeout avec seulement $k = 8$ ($7222.41s$). La même observation est faite sur la base *connect*, mais d'une manière plus prononcée sur l'échelle exponentielle suivi par `CP4IM`. Avec $k = 4$, `CP4IM` va au-delà du timeout avec

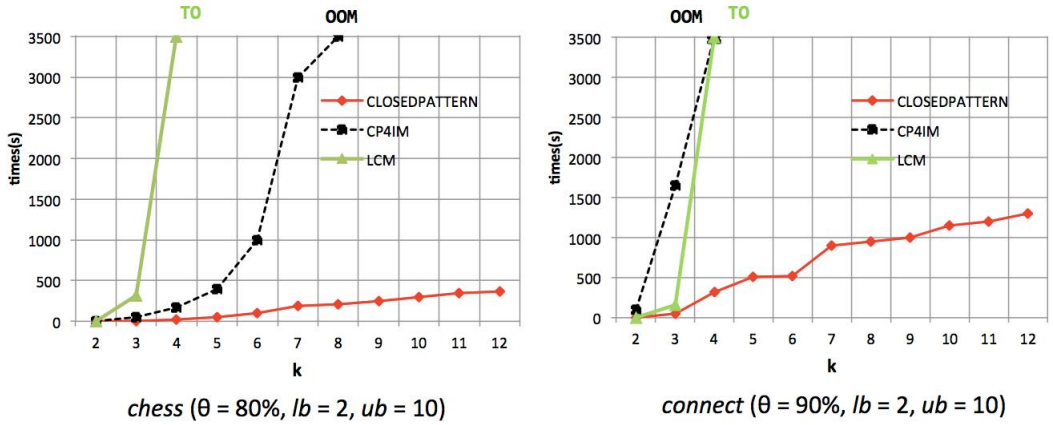

FIGURE 4.4.: Extraction de k motifs avec CLOSEDPATTERN, `RCM`, `LCM`

5428.05s tandis que CLOSEDPATTERN confirme son comportement linéaire en variant k de 2 à 12.

Pour de tels problèmes, une ligne de base peut être l'utilisation des algorithmes spécialisés avec un post-traitement. On peut imaginer (i) L'utilisation de `LCM` pour extraire le nombre total de motifs fermés et (ii) Une recherche générer-et-tester en essayant de trouver des motifs distincts, pour une taille donnée. Une telle approche peut être très coûteuse. Ici, le post-traitement générera toutes les k combinaisons possibles de motifs fermés. Par exemple, nous rappelons que pour *chess* avec $\theta = 80\%$, nous avons 5084 motifs fermés. Avec $k = 12$ et en utilisant CLOSEDPATTERN, nous avons besoin de moins de 6 min, la où avec l'utilisation d'un algorithme spécialisé, nous devons faire face à un nombre massif de combinaisons. Ainsi, cette dernière expérience confirme que si `LCM` est plus rapide sur les requêtes de base (par exemple, demander les motifs fréquents fermées avec une taille donnée), il ne peut pas faire face à des requêtes complexes. Ceci nécessite de réfléchir et de proposer un nouvel algorithme ad-hoc, tandis que la PPC permet à un utilisateur novice, d'exprimer sa requête sous forme de contraintes.

4.8. Conclusion

Dans ce chapitre, nous avons proposé une contrainte globale CLOSEDPATTERN pour l'extraction de motifs fréquents fermés. CLOSEDPATTERN capture la sémantique particulière du problème. Afin de propager efficacement la contrainte, nous avons défini trois règles de filtrage qui assurent la consistance de domaine. Nous avons ensuite, conçu un algorithme de filtrage, qui établit la Consistance de Domaine avec une complexité cubique en temps et quadratique en espace. Nous avons proposé, une implémentation sous GECODE reposant sur la représentation verticale des bases de transactions. Nous avons montré, que CLOSEDPATTERN offre un apport pratique sur les bases de grande taille, ce qui est un enjeu majeur pour la communauté de fouille de données.

Finalement, Pour montrer l'applicabilité et la flexibilité de ClosedPattern, par rapport aux méthodes spécialisées, nous avons effectué une expérimentation sur une instance de k pattern-set où ClosedPattern est intégré avec un ensemble de contraintes. Ce type d'expérimentation est particulièrement intéressant, sachant que dans la partie localisation de fautes avec l'extraction de motifs, l'objectif est d'extraire les top-k motifs fermés suspects.

5. F-CPminer : Localisation de fautes par extraction de motifs sous contraintes

Dans ce chapitre, nous présentons une approche nommée F-CPminer basée sur l'extraction de motifs, pour la localisation de fautes dans les programmes. Nous formalisons cette tâche comme étant un problème d'extraction des k meilleurs motifs, satisfaisant un ensemble de contraintes, modélisant les instructions les plus suspectes. Nous faisons appel à la Programmation Par Contraintes pour modéliser et résoudre le problème de localisation. L'approche procède sur deux étapes : i) 1ère localisation : extraction des top-k suites d'instructions les plus suspectes ; ii) Raffinement de la 1ère localisation : par un post-traitement des top-k motifs suspects. Pour l'extraction des k motifs suspects, nous proposons deux modèles, l'un basé sur le modèle réifié, le second sur la contrainte globale ClosedPattern.

Sommaire

5.1. Contexte et Motivations

Au cours de la dernière décennie, plusieurs techniques automatisées ont été proposées, pour répondre au problème de localisation de fautes et assurer un certain niveau de qualité du logiciel. Les méthodes formelles basées sur la preuve, tentent de démontrer formellement, l'exactitude du code par rapport à une spécification mathématique. Par ailleurs, plusieurs de ces techniques automatisées issues du test logiciel, comparent deux types de traces d'exécution, les exécutions dites correctes (*positives*) et les exécutions dites erronées (*négatives*) [Renieres and Reiss, 2003, Cleve and Zeller, 2005]. Un des inconvénients majeurs, est que les traces à analyser, peuvent être de grande taille et de ce fait, d'un intérêt limité.

Une autre catégorie, qui regroupe un bon nombre de méthodes du test logiciel, se base sur une fonction de score, pour évaluer le caractère suspect de chaque instruction dans un programme, en exploitant les occurrences des instructions dans les traces *négatives* et/ou *positives* [Jones and Harrold, 2005, Abreu et al., 2007, Yoo, 2012]. L'intuition sous-jacente, est que les instructions ayant les scores les plus élevés, sont les plus suspectes pour contenir des fautes, ce qui permet d'ordonner les instructions d'un programme de la plus suspecte à la plus innocente. Toutefois, l'inconvénient de ces techniques, réside dans le fait qu'ils ne font aucune différence entre deux traces d'exécution négatives (respectivement, positives) et évaluent chaque instruction de manière individuelle. Par conséquent, les régularités entre les exécutions des instructions, sous les cas de test sont totalement ignorées.

En outre, ces dernières années, le problème de localisation de fautes a été traité avec des techniques de fouille de données [Nessa et al., 2008, Cellier et al., 2008]. Dans ce travail, nous formalisons le problème de localisation de fautes comme un problème d'extraction des k meilleurs motifs, satisfaisant un ensemble de contraintes, modélisant les instructions les plus suspectes. En effet, l'extraction des top-k motifs est une voie prometteuse en fouille de données sous contraintes, permettant de produire des ensembles de motifs intéressants. L'approche que nous proposons bénéficie des travaux récents sur la fertilisation croisée entre la fouille de motifs et la programmation par contraintes [Khiari et al., 2010, Guns et al., 2011, Lazaar et al., 2016]. L'intérêt est de disposer d'heuristiques puissantes pour (i) attaquer la combinatoire due à l'exécution des instructions, et ainsi prendre en compte les dépendances existantes entre les instructions et le cas de test (ii) avoir un modèle générique pouvant aisément prendre en considération les propriétés d'un programme sous test, dans le but d'offrir une localisation de plus en plus précise. L'approche que nous proposons est réalisée en deux étapes :

i) Première localisation : Extraction des top-k suites d'instructions les plus suspectes, selon une *relation de dominance* et utilisant des contraintes postées dynamiquement (DCSP, voir la section 2.6).

ii) Raffinement de la première localisation : Classement des instructions issues des top-k motifs, pour localiser la faute avec un algorithme *ad-hoc*.

Programme : Compteur de caractères	Cas de test							
	tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7	tc_8
`function count (char *s) {`								
`int let, dig, other, i = 0;`								
`char c;`								
e_1 : `while (c = s[i++]) {`	1	1	1	1	1	1	1	1
e_2 : `if('A'<=c && 'Z'>=c)`	1	1	1	1	1	1	0	1
e_3 : `let += 2; //- faute -`	1	1	1	1	1	1	0	0
e_4 : `else if ( 'a'<=c && 'z'>=c )`	1	1	1	1	1	0	0	1
e_5 : `let += 1;`	1	1	0	0	1	0	0	0
e_6 : `else if ( '0'<=c && '9'>=c )`	1	1	1	1	0	0	0	1
e_7 : `dig += 1;`	0	1	0	1	0	0	0	0
e_8 : `else if (isprint (c))`	1	0	1	0	0	0	0	1
e_9 : `other += 1;`	1	0	1	0	0	0	0	1
e_{10} : `printf("%d %d %d\n", let, dig, other);}`	1	1	1	1	1	1	1	1
Positif/Négatif	N	N	N	N	N	N	P	P

FIGURE 5.1.: Exemple d'un programme avec la matrice de couverture associée

5.2. Localisation de fautes par extraction de motifs

Dans cette section, nous présentons notre modélisation du problème de localisation de fautes, comme une tâche d'extraction des k meilleurs motifs *fréquents fermés*. L'extraction est réalisée sur deux bases transactionnelles, une base dite positive et une base dite négative, formées à partir des ensembles de cas test positifs et négatifs. Nous adoptons la couverture des cas de test d'un programme collectée durant la phase de test (définition 3.3) pour construire ces bases.

Dans notre modélisation, nous montrons qu'il est possible de faire appel, soit au modèle réifié, présenté dans la section 2.7 et que nous adaptons à notre besoin de localisation, soit à la contrainte globale CLOSEDPATTERN présentée dans le chapitre 4. L'idée est de comparer par la suite les deux modèles. Nous commençons par donner notre formalisation du problème, nous présentons ensuite la notion de top-k motifs *suspects*, puis nous détaillons l'algorithme de recherche des top-k, basé sur une relation de dominance qui reflète la notion de *suspicion* d'un motif. Enfin, nous décrivons comment exploiter ces top-k motifs pour retourner à la fin, un classement précis des instructions, pour localiser la faute.

Dans l'approche F-CPMINER que nous proposons, nous nous basons sur les deux hypothèses de faute *simple* et *multiple* (voir le chapitre 3). En effet, dans le chapitre 6, nos études expérimentales portent dans un premier temps sur des programmes à faute unique, dans un second temps nous présentons une étude sur un programme avec plusieurs fautes.

Étant donné un programme sous test P. Soit $L = \{e_1, ..., e_n\}$ l'ensemble des n instructions qui compose P et $T = \{t_1, ..., t_m\}$ l'ensemble des m cas de test. Ainsi, la base transactionnelle $\mathcal{T}$ est définie comme suit :

i) Chaque instruction de L, correspond à un item de $\mathcal{I}$.

ii) La couverture de chaque cas de test t_i forme une transaction dans la base transactionnelle $\mathcal{T}$.

Par ailleurs, pour découvrir les contrastes entre les sous-ensembles de cas de test positifs et négatifs, la base $\mathcal{T}$ est partitionnée en deux sous-ensembles disjoints $\mathcal{T}^+$ et $\mathcal{T}^-$, tels que, $\mathcal{T}^+$ représente l'ensemble des couvertures des cas de test positifs et $\mathcal{T}^-$ représente l'ensemble des couvertures des cas de test négatifs.

Soit d la matrice booléenne représentant la base transactionnelle $\mathcal{T}$. Nous avons alors, $\forall t \in \mathcal{T}, \forall i \in \mathcal{I}$, $(d_{t,i} = 1)$, si et seulement si, l'instruction i est exécutée (au moins une fois) sous le cas de test t.

Exemple 5.1. *La figure 5.1 montre la matrice booléenne de transactions, associée au programme "compteur de caractères". Soit la couverture (définition 3.3) du cas de test* tc_5 *est* $\langle 1,1,1,1,1,0,0,0,0,1\rangle$. *Comme* tc_5 *est un cas de test qui est négatif, alors sa couverture est ajoutée à* $\mathcal{T}^-$.

5.2.1. Variables du modèle de localisation

Soit P un motif suspect recherché (i.e., suite d'instructions). Nous introduisons l'ensemble des variables suivantes dans notre modèle :

i) n variables booléennes $\{P_1, ..., P_n\}$ pour représenter le motif P, où, $P_i = 1$ si l'instruction i fait partie des instructions du motif P.

ii) m variables booléennes $\{T_1, ..., T_m\}$ pour représenter les transactions de $\mathcal{T}$, où, $T_t = 1$ si toutes les instructions du motif P, sont exécutées sous le cas de test t. Comme pour la base $\mathcal{T}$, l'ensemble des variables $\{T_1, ..., T_m\}$ est partitionné en deux sous-ensembles disjoints : les variables $\{T_t\} \in \mathcal{T}^+$ et les variables $\{T_t\} \in \mathcal{T}^-$ telles que, $(\mathcal{T}^+ \cup \mathcal{T}^- = \mathcal{T})$ et $(\mathcal{T}^+ \cap \mathcal{T}^- = \emptyset)$.

iii) Ainsi, nous définissons deux variables entières $freq_{\mathcal{T}^+}$ et $freq_{\mathcal{T}^-}$ pour mesurer la fréquence de P (son exécution) dans la base $\mathcal{T}^+$ et dans la base $\mathcal{T}^-$.

Exemple 5.2. *Étant donné le programme de l'exemple 5.1. Soit le motif* $P = \langle e_2, e_6, e_8\rangle$ *qui correspond à* $\langle 0,1,0,0,0,1,0,1,0,0\rangle$ *avec l'encodage booléen. Nous avons alors :* $freq_{\mathcal{T}^+} = 1$ *et* $freq_{\mathcal{T}^-} = 2$, *car la suite des instructions de P sont exécutées ensemble par un cas de test positifs (*tc_8*) et par deux cas de test négatifs (*tc_1, tc_3*).*

Dans la partie suivante, nous présentons les contraintes posées dans le modèle de localisation, aussi bien pour la version utilisant le modèle réifié (premier modèle), que celle faisant appel à la contrainte globale ClosedPattern (deuxième modèle).

5.2.2. Modèle réifié pour la localisation

Dans le premier modèle de localisation, inspiré du modèle réifié, les contraintes réifiées posées, correspondent aux contraintes de couverture et aux contraintes de fréquence. Ces deux contraintes, sont posées sur $\mathcal{T}^+$ et $\mathcal{T}^-$ car nous devons connaître les couvertures et les fréquences d'un motif P dans les deux bases.

Les contraintes de couverture sont données dans l'équation 5.1 pour $\mathcal{T}^+$ et l'équation 5.2 pour $\mathcal{T}^-$:

$$\forall t \in \mathcal{T}^+ : (T_t = 1) \leftrightarrow \sum_{i \in \mathcal{I}} P_i \times (1 - d_{t,i}) = 0 \tag{5.1}$$

$$\forall t \in \mathcal{T}^- : (T_t = 1) \leftrightarrow \sum_{i \in \mathcal{I}} P_i \times (1 - d_{t,i}) = 0 \tag{5.2}$$

Ainsi, la variable T_t d'un cas de test t, est égale à 1, si et seulement si, le motif P est exécuté sous ce cas de test.

Les contraintes pour mesurer les fréquences sont alors données dans l'équation 5.3 pour $\mathcal{T}^+$ et dans l'équation 5.4 pour $\mathcal{T}^-$:

$$freq_{\mathcal{T}^+} = \sum_{t \in \mathcal{T}^+} T_t \tag{5.3}$$

$$freq_{\mathcal{T}^-} = \sum_{t \in \mathcal{T}^-} T_t \tag{5.4}$$

Ainsi, nous avons pour chaque motif P, le nombre de ses exécutions dans $\mathcal{T}^+$ et $\mathcal{T}^-$.

Afin de réduire la redondance dans les suites d'instructions extraites, nous imposons la contrainte de fermeture sur le motif P. Dans notre cas, cette contrainte est imposée sur $\mathcal{T}^+ \cup \mathcal{T}^-$, ce qui garantit que P n'a pas de sur-ensemble avec les mêmes fréquences dans les deux bases ($freq_{\mathcal{T}^+}$ et $freq_{\mathcal{T}^-}$).
Les contraintes de fermeture sont posées sur $\mathcal{T}$ car nous recherchons des motifs fermés sur toute la base. La contrainte est encodée avec l'équation. 5.5 :

$$\forall i \in \mathcal{I} : (P_i = 1) \leftrightarrow \sum_{t \in \mathcal{T}^+ \cup \ \mathcal{T}^-} T_t \times (1 - d_{t,i}) = 0 \tag{5.5}$$

Dans la propriété 5.1, nous allons expliquer pourquoi nous avons fait ce choix de modélisation pour la contrainte de fermeture.

5.2.3. Localisation basée sur la contrainte ClosedPattern

Dans le second modèle, nous proposons de faire appel à la contrainte globale ClosedPattern [1] pour extraire les motifs fermés sur la base $\mathcal{T}$ et qui remplace l'ensemble des contraintes de fermeture 5.5 du premier modèle :

$$\text{ClosedPattern}_{\mathcal{T},\theta}(P) \tag{5.6}$$

La contrainte globale ClosedPattern prend comme paramètre la base entière $\mathcal{T}$.

Ainsi, nous posons également les contraintes de couverture 5.1, 5.2 et les contraintes de fréquence 5.3, 5.4, afin de savoir si l'occurrence du motif fermé P est dans $\mathcal{T}^+$ ou dans $\mathcal{T}^-$ et par conséquent, comptabiliser correctement les valeurs des variables $freq_{\mathcal{T}^+}$ et $freq_{\mathcal{T}^-}$.

Il est important de noter que dans ce second modèle, le filtrage pour extraire un motif suspect P, est opéré avec l'algorithme de filtrage 2 de la contrainte globale ClosedPattern.

Intérêt de la contrainte de fermeture sur $\mathcal{T}$:

Dans la suite de ce chapitre, nous allons exploiter un ensemble de motifs suspects $\langle P^1, ..., P^k \rangle$, nous avons donc besoin de comparer plusieurs motifs entre eux. Pour cela, nous allons surcharger la notation de fréquence, avec $freq_{\mathcal{T}^+}(P^i)$ et $freq_{\mathcal{T}^-}(P^i)$ pour indiquer la valeur de la fréquence d'un motif P^i dans $\mathcal{T}^+$ et dans $\mathcal{T}^-$.

Exemple 5.3. *Étant donné le programme de l'exemple 5.1. Soit le motif* $P^1 = \langle e_2, e_6, e_8 \rangle$. *Nous avons* $freq_{\mathcal{T}^+}(P^1) = 1$ *et* $freq_{\mathcal{T}^-}(P^1) = 2$. *Par ailleurs, il existe d'autres motifs* P^j *tel que* $P^j \subset P^1$ *et* $(freq_{\mathcal{T}^+}(P^1) = freq_{\mathcal{T}^+}(P^j))$ *et* $(freq_{\mathcal{T}^-}(P^1) = freq_{\mathcal{T}^-}(P^j))$.
Soient les motifs suivants $P^2 = \langle e_8 \rangle, P^3 = \langle e_2, e_8 \rangle, P^4 = \langle e_6, e_8 \rangle$, *nous avons :*

- $P^2, P^3, P^4 \subset P^1$
- $freq_{\mathcal{T}^+}(P^1) = freq_{\mathcal{T}^+}(P^2) = freq_{\mathcal{T}^+}(P^3) = freq_{\mathcal{T}^+}(P^4) = 1$
- $freq_{\mathcal{T}^-}(P^1) = freq_{\mathcal{T}^-}(P^2) = freq_{\mathcal{T}^-}(P^3) = freq_{\mathcal{T}^-}(P^4) = 2$

La contrainte de fermeture, permet donc d'éviter les motifs P^2, P^3, P^4, *en gardant uniquement le motif* P^1, *qui est le plus grand motif au sens de l'inclusion et qui représente un niveau de suspicion formé par la fréquence positive et la fréquence négative.*

Il est important de noter que la contrainte de fermeture telle que nous l'avons exprimée sur $\mathcal{T}$, n'impose pas qu'un motif P^i soit fermé dans $\mathcal{T}^+$ et/ou qu'il soit fermé dans $\mathcal{T}^-$, mais qu'il soit fermé sur $\mathcal{T}^+ \cup \mathcal{T}^-$.

Propriété 5.1. *(motifs fermés dans une partition) Soient* $\mathcal{S}^+$ *l'ensemble des motifs fermés de la base* $\mathcal{T}^+$, $\mathcal{S}^-$ *l'ensemble des motifs fermés de la base* $\mathcal{T}^-$ *et* $\mathcal{S}$ *l'ensemble des motifs fermés calculés sur* $\mathcal{T}$. *Nous avons alors* $\mathcal{S}^+ \cup \mathcal{S}^- \subseteq \mathcal{S}$.

1. Dans le chapitre 4, l'encodage et la définition de la contrainte ClosedPattern sont détaillées

classe	*trans*	*items*
$\mathcal{T}^+$	t_1	a
	t_2	$a\ b$
$\mathcal{T}^-$	t_3	$b\ c$

TABLE 5.1.: Exemple de partition $\mathcal{T}^+$ et $\mathcal{T}^-$

Preuve. Pour monter que $\mathcal{S}^+ \cup \mathcal{S}^- \subseteq \mathcal{S}$, nous devons monter que :
(i) $\forall\ P \in \{\mathcal{S}^+ \cup \mathcal{S}^-\} \Rightarrow S_i \in \mathcal{S}$
(ii) $\exists\ P \in \mathcal{S} : P \notin \{\mathcal{S}^+ \cup \mathcal{S}^-\}$

(i) $\Rightarrow$ Soit un motif fermé $P \in \mathcal{S}^+$ (le cas $P \in \mathcal{S}^-$ est symétrique), avec $freq_{\mathcal{T}^+}(P)$ pour la fréquence de P dans la base $\mathcal{T}^+$. Nous avons :

- $\forall\ P' \supset P : freq_{\mathcal{T}^+}(P') < freq_{\mathcal{T}^+}(P)$ (voir la définition 1.8 d'un motif fermé).
- $\forall\ t \in \mathcal{T} \setminus \mathcal{T}^+, \forall\ P' \supset P$, nous avons deux cas :
  1. $(P \subseteq P') \subseteq t : freq_{(\mathcal{T}^+ \cup\ t)}(P') < freq_{(\mathcal{T}^+ \cup\ t)}(P)$
  2. $(P \subseteq P') \nsubseteq t : (freq_{(\mathcal{T}^+ \cup\ t)}(P') = freq_{\mathcal{T}^+}(P')) \wedge (freq_{(\mathcal{T}^+ \cup\ t)}(P) = freq_{\mathcal{T}^+}(P))$
- Nous avons alors : $\forall\ P' \supset P : freq_{\mathcal{T}}(P') < freq_{\mathcal{T}}(P)$ donc $P \in \mathcal{S}$.

(ii) $\Rightarrow$ Cependant, il existe des cas où la base $\mathcal{T}$ produit plus de motifs fermés que les bases $\mathcal{T}^+$ et $\mathcal{T}^-$ séparément, donc $\mathcal{S}^+ \cup \mathcal{S}^- \subset \mathcal{S}$.

Exemple 5.4. *Soit l'exemple de la table 5.1.* $\mathcal{S}^+ = \{\langle a\rangle, \langle ab\rangle\}$ *est l'ensemble des motifs fermés de* $\mathcal{T}^+$ *et* $\mathcal{S}^- = \{\langle bc\rangle\}$ *est l'ensemble des motifs fermés de* $\mathcal{T}^-$. *Considérons* $\mathcal{T} = \mathcal{T}^+ \cup \mathcal{T}^-$, *l'ensemble de motifs fermés sur* $\mathcal{T}$ *est* $\mathcal{S} = \{\langle a\rangle, \langle ab\rangle, \langle b\rangle, \langle bc\rangle\}$. *Nous avons alors* $\mathcal{S}^+ \cup \mathcal{S}^- \subset \mathcal{S}$ *car le motif* $\langle b\rangle$ *n'est pas fermé sur* $\mathcal{T}^+$ *ou sur* $\mathcal{T}^-$. □

Il est donc plus intéressant pour nous, de calculer l'ensemble $\mathcal{S}$, pour ne pas rater des motifs fermés, qui représentent un niveau de suspicion donné. Ce qui explique notre choix au niveau conceptuel, de poser la contrainte de fermeture sur toute la base $\mathcal{T}$.

5.3. Les top-k motifs suspects

L'intuition qui est derrière les méthodes de localisation de fautes basées sur les cas de test, illustre le fait que les instructions qui apparaissent le plus souvent dans les traces d'exécution négatives, sont considérées comme étant les plus suspectes, tant dis que, les instructions qui apparaissent seulement dans les traces d'exécutions positives, sont considérées comme étant les plus innocentes [Renieres and Reiss, 2003, Jones and Harrold, 2005, Nessa et al., 2008, Eric Wong et al., 2010].

Suivant le même raisonnement, notre approche a pour objectif d'extraire non pas les instructions suspectes une à une, mais les motifs les plus suspects. Un motif étant un ensemble d'instructions ayant été exécutées ensembles. Pour se faire, nous définissons une relation de dominance notée $\succ_{\mathcal{S}}$ entre les motifs, en fonction de leur niveau de suspicion.

Définition 5.1 (Relation de dominance $\succ_{\mathcal{S}}$)**.** *Étant donnée une bipartition de $\mathcal{T}$ en deux sous-ensembles disjoints $\mathcal{T}^+$ et $\mathcal{T}^-$, un motif S_i domine un autre motif S_j (noté $S_i \succ_{\mathcal{S}} S_j$), si et seulement si :*

$$S_i \succ_{\mathcal{S}} S_j \Leftrightarrow [freq_{\mathcal{T}^-}(S_i) > freq_{\mathcal{T}^-}(S_j)] \vee [(freq_{\mathcal{T}^-}(S_i) = freq_{\mathcal{T}^-}(S_j)) \wedge (freq_{\mathcal{T}^+}(S_i) < freq_{\mathcal{T}^+}(S_j))] \quad (5.7)$$

La relation de dominance $\succ_{\mathcal{S}}$ exprime le fait que S_i est considéré comme plus suspect que S_j, noté $S_i \succ_{\mathcal{S}} S_j$, si S_i est plus fréquent que S_j dans $\mathcal{T}^-$. Si les deux motifs couvrent exactement le même ensemble de transactions dans $\mathcal{T}^-$, alors le motif le moins fréquent dans $\mathcal{T}^+$ est préféré. Cette relation permet d'établir un ordre strict partiel sur les motifs en termes de suspicion. En effet, la relation $\succ_{\mathcal{S}}$ est :

- *irréflexive* : $\forall\ S_i : \neg(S_i \succ_{\mathcal{S}} S_i)$. On ne peut pas exprimer qu'un motif soit plus suspect que lui même.
- *transitive* : $\forall\ S_i, S_j, S_k : (S_i \succ_{\mathcal{S}} S_j) \wedge (S_j \succ_{\mathcal{S}} S_k) \Rightarrow (S_i \succ_{\mathcal{S}} S_k)$.
- *asymétrique* : $\forall\ S_i, S_j : (S_i \succ_{\mathcal{S}} S_j) \Rightarrow \neg(S_j \succ_{\mathcal{S}} S_i)$, si S_i est plus suspect que S_j alors S_j ne peut pas dominer S_i en termes de suspicion.

Exemple 5.5. *Étant donné le programme de l'exemple 5.1. Soient les deux motifs (suite d'instructions) $S_1 = \langle e_2, e_6, e_8\rangle$ et $S_2 = \langle e_3, e_4, e_{10}\rangle$. Nous avons $S_2 \succ_{\mathcal{S}} S_1$ car S_2 est exécuté plus de fois dans $\mathcal{T}^-$ ($freq_{\mathcal{T}^-}(S_2) = 5$) que S_1 ($freq_{\mathcal{T}^-}(S_1) = 2$).*

Notons que deux motifs S_i et S_j peuvent avoir le même degré de suspicion. C'est-à-dire qu'ils sont exécutés le même nombre de fois par les cas de test positifs et par les cas de test négatifs.

Définition 5.2 (motifs équivalents $=_{\mathcal{S}}$)**.** *Étant donné une bipartition de $\mathcal{T}$ en deux sous-ensembles disjoints $\mathcal{T}^+$ et $\mathcal{T}^-$, deux motifs S_i et S_j sont équivalents en termes de suspicion (noté $S_i =_{\mathcal{S}} S_j$) si et seulement si :*

$$S_i =_{\mathcal{S}} S_j \Leftrightarrow [freq_{\mathcal{T}^-}(S_i) = freq_{\mathcal{T}^-}(S_j)] \wedge [freq_{\mathcal{T}^+}(S_i) = freq_{\mathcal{T}^+}(S_j)] \quad (5.8)$$

Exemple 5.6. *Étant donné le programme de l'exemple 5.1. Soient les deux motifs (suite d'instructions) $S_1 = \langle e_1, e_4, e_8\rangle$ et $S_2 = \langle e_6, e_8, e_{10}\rangle$. Nous avons $S_1 =_{\mathcal{S}} S_2$ car S_1 et S_2 sont exécutés le même nombre de fois dans $\mathcal{T}^+$ et $\mathcal{T}^-$ ($freq_{\mathcal{T}^+}(S_1) = freq_{\mathcal{T}^+}(S_2) = 2$) et ($freq_{\mathcal{T}^-}(S_1) = freq_{\mathcal{T}^-}(S_2) = 1$).*

En mettant à profit la relation de dominance $\succ_{\mathcal{S}}$, notre objectif est de produire l'ensemble des top-k motifs les plus suspects en fonction de la relation $\succ_{\mathcal{S}}$.

Définition 5.3 (top-k motif suspect)**.** *Un motif P fait partie des top-k motifs les plus suspects (par rapport à $\succ_{\mathcal{S}}$) si et seulement si $\nexists S_1, \ldots S_k, \in \mathcal{L}_{\mathcal{I}}, \forall 1 \leq j \leq k,\ S_j \succ_{\mathcal{S}} P$.*

Ainsi, un motif P est un top-k motif suspect, s'il n'existe pas plus de $(k-1)$ motifs qui sont plus suspects que P. Un ensemble de top-k motifs suspects est défini comme suit :

$$\{P \in \mathcal{L}_{\mathcal{I}} \mid Suspect(P) \wedge \nexists S_1, \ldots S_k, \in \mathcal{L}_{\mathcal{I}}, \forall 1 \leq j \leq k,\ S_j \succ_{\mathcal{S}} P\}$$

Le réseau de contraintes *Suspect* permet de spécifier que le motif P doit satisfaire des propriétés de base. Dans notre cas, nous imposons que le motif recherché doit satisfaire les propriétés suivantes :

$$Suspect(P) \equiv fermeture(P) \wedge freq_{\mathcal{T}^-}(P) \geq 1 \wedge taille(P) \geq 1,$$

La contrainte $fermeture(P)$ indique que le motif P doit être fermé.

Ce qui signifie qu'un motif suspect P doit :

(i) être fermé sur la base $\mathcal{T} = \mathcal{T}^+ \cup \mathcal{T}^-$, ceci permet d'avoir le motif le plus grand au sens de l'inclusion pour chaque degré de suspicion ;

(ii) le motif doit apparaître au moins une fois dans les cas de test négatifs (évitant ainsi de générer les motifs n'ayant jamais été exécutés par les cas de test négatifs) ;

(iii) le motif P doit inclure au moins une instruction.

5.4. 1ère Localisation : extraction des top-k motifs les plus suspects

Dans cette section, nous présentons comment les top-k motifs suspects peuvent être extraits, en faisant appel aux contraintes postées dynamiquement durant la recherche [Rojas et al., 2014]. L'idée principale est d'exploiter la relation de dominance (noté $\succ_{\mathcal{S}}$), entre les ensembles des instructions, pour produire à chaque recherche, des motifs de plus en plus suspects, grâce aux contraintes postées dynamiquement durant le processus d'extraction. Nous avons de ce fait, un CSP dynamique (voir la section 2.6) où chaque nouvelle contrainte dynamique, va imposer que le prochain motif qui est recherché, doit dominer au moins le moins suspect (au sens de la relation $\succ_{\mathcal{S}}$) des k motifs déjà trouvés. Ce processus s'arrête, lorsque aucune meilleure solution, ne peut être encore obtenue.

L'algorithme 3 extrait les top-k motifs (i.e., les plus suspects), par rapport à la relation $\succ_{\mathcal{S}}$. Il prend comme entrées, l'ensemble des couvertures des cas de tests positifs $\mathcal{T}^+$ et négatifs $\mathcal{T}^-$, un entier positif k, et retourne comme sortie, les top-k motifs les plus suspects. Cet ensemble de k motifs forme une première localisation, puisque l'un ou plusieurs des motifs extraits sont susceptibles de contenir la faute recherchée. L'algorithme commence avec un réseau de contraintes $Suspect(P)$ (ligne 3). Nous rappelons que, le réseau $Suspect(P)$ correspond aux contraintes réifiées 5.1, 5.2, 5.3, 5.4 et 5.5, si on adopte le premier modèle, ou à la contrainte ClosedPattern 5.6 et aux contraintes 5.1, 5.2, 5.3, 5.4, si on adopte le second modèle.

D'abord, nous recherchons les k premiers motifs suspects, qui sont des solutions du système de contraintes courant, en utilisant la fonction $SolveNext(\mathcal{C})$ (lignes 4-7). La fonction $SolveNext(\mathcal{C})$ demande au solveur CP de retourner une solution à $\mathcal{C}$, qui est différente des solutions déjà retournées auparavant. Initialement, le premier appel de $SolveNext(\mathcal{C})$ retourne un motif S_1, le second appel retourne un motif $S_2 \neq S_1$, et ainsi de suite. Durant la recherche, une liste $\mathcal{S}$ des top-k motifs suspects est maintenue. Une fois que les k motifs sont trouvés, ils sont triés dans un ordre décroissant selon $\succ_{\mathcal{S}}$ et $=_{\mathcal{S}}$ (ligne 8). C'est-à-dire, du plus suspect au moins suspect.

Algorithme 3 : Extraction des top-k motifs les plus suspects $\mathcal{S} = \langle S_1, ..., S_k \rangle$

```
1  Données : T+, T−, k ;
2  Sorties : S : top-k motifs les plus suspects ;
3  C ← Suspect(P) ;   S ← ∅ ;   i ← 1 ;   θ = 1 ;
4  répéter
5  |  P ← SolveNext(C)
6  |  si P ≠ ∅ alors S.add(P) ;   i ← i + 1;
7  jusqu'à (i > k) ou (P = ∅) ;
8  Trier S selon l'ordre décroissant avec ≻S et =S ;
9  tant que P ≠ ∅ faire
10 |  C.add(P ≻S Sk) ;
11 |  P ← SolveNext(C) ;
12 |  si P ≠ ∅ alors
13 |  |  S.remove(Sk) ;
14 |  |  Insérer P dans S selon l'ordre décroissant avec ≻S et =S ;
15 retourner S ;
```

Par la suite, à chaque fois qu'un nouveau motif est trouvé, nous supprimons de $\mathcal{S}$ le motif le moins suspect selon $\succ_{\mathcal{S}}$ (ligne 13), nous ajoutons le nouveau motif dans $\mathcal{S}$ dans l'ordre décroissant selon $\succ_{\mathcal{S}}$ et $=_{\mathcal{S}}$ (ligne 14) et postons dynamiquement une nouvelle contrainte $(P \succ_{\mathcal{S}} S_k)$ à la ligne 10, qui garantit que le nouveau motif recherché P doit être meilleur (plus suspect), selon la relation $\succ_{\mathcal{S}}$, que le moins préféré des motifs S_k dans la liste $\mathcal{S}$ courante des top-k. Par conséquent, les solutions suivantes vérifient à la fois le système de contraintes courant et les nouvelles contraintes dynamiques. Ce processus est répété jusqu'à ce que, aucun motif ne peut être généré.

5.5. Raffinement : post-traitement des top-k motifs suspects

Notre algorithme d'extraction des top-k (algorithme 3), retourne une liste ordonnée des k meilleurs motifs (suspects) $\mathcal{S} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$. Cette liste forme une première localisation, où chaque motif $\mathcal{S}_i$ représente un sous-ensemble d'instructions, pouvant expliquer et localiser la faute. Cependant, en fonction du nombre d'instructions, du programme traité et de leurs exécutions, les motifs peuvent contenir un nombre important d'instructions, ce qui rend leur analyse peu pratique.

Nous proposons une seconde étape, visant à affiner la première localisation, en dissociant les instructions composant les top-k motifs en fonction de la nature et le contenu des motifs. En effet, d'un motif à un autre, des instructions apparaissent/disparaissent en fonction de leurs fréquences dans les bases positives et négatives. Nous proposons dans cette section, l'algorithme 4 qui prend en entrée les top-k motifs les plus suspects

Algorithme 4 : Raffinement des top-k motifs

1 **Données** top-k $\mathcal{S} = \langle S_1, \ldots, S_k \rangle$, fréquences de chaque S_i : $(freq_{\mathcal{T}^+}, freq_{\mathcal{T}^-})$
2 **Sorties** une liste ordonnée des instructions $Loc = \langle \Omega_1, \Omega_2, \Omega_3 \rangle$
3 $\mathcal{SM} \leftarrow Fusionner(\mathcal{S})$;
4 $\Omega_1 \leftarrow \langle\rangle$; $\Omega_3 \leftarrow \langle\rangle$; $Loc \leftarrow \langle\rangle$;
5 $\Omega_2 \leftarrow \mathcal{SM}_1$; $l \leftarrow |\mathcal{SM}|$;
6 **pour chaque** $i \in 2..l$ **faire**
7 $\quad \Delta_D \leftarrow \Omega_2 \setminus \mathcal{SM}_i$;
8 $\quad$ **si** $\Delta_D \neq \emptyset$ **alors**
9 $\quad\quad \Omega_1.add(\Delta_D)$;
10 $\quad\quad \Omega_2.removeAll(\Delta_D)$;
11 $\quad \Delta_A \leftarrow \mathcal{SM}_i \setminus \mathcal{SM}_1$;
12 $\quad \omega \leftarrow \emptyset$;
13 $\quad$ **pour chaque** $b \in \Delta_A$ **faire**
14 $\quad\quad$ **si** $(\forall \omega' \in \Omega_3, \forall \omega" \in \Omega_1 : b \not\in \omega' \wedge b \notin \omega")$ **alors** $\omega \leftarrow \omega \cup \{b\}$
15 $\quad$ **si** $\omega \neq \emptyset$ **alors** $\Omega_3.add(\omega)$
16 $Loc.add(\Omega_1)$; $Loc.add(\Omega_2)$; $Loc.add(\Omega_3)$;
17 **retourner** Loc ;

et retourne une liste Loc, contenant un classement précis des instructions susceptibles de localiser la faute (ligne 2). La liste Loc inclut trois listes ordonnées notées Ω_1, Ω_2 et Ω_3. Les éléments de Ω_1 sont classés en premier, suivis des éléments de Ω_2 et enfin des éléments de Ω_3 qui contient les instructions les moins suspectes.

L'algorithme 4 commence par fusionner les motifs équivalents de $\mathcal{S}$ (i.e., de même degré de suspicion), donc ayant les mêmes fréquences dans les cas de test positifs et les cas de test négatifs.

Définition 5.4 (Fusion des motifs équivalents)**.** *Soit $\mathcal{S} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_k \rangle$ la liste des top-k motifs les plus suspects. La liste $\mathcal{S}$ peut contenir des motifs équivalents (e.g., $S_i =_{\mathcal{S}} S_j$), dans ce cas, S_i et S_j sont fusionnés pour former un seul motif $\mathcal{SM}_i = S_i \cup S_j$. Ainsi, la liste $\mathcal{SM} = \{\mathcal{SM}_1, ..., \mathcal{SM}_l\}$ avec $l \leq k$ est obtenue à partir de $\mathcal{S}$ en fusionnant tous les motifs équivalents.*

La fusion des motifs équivalents de $\mathcal{S}$ est réalisée par la fonction $Fusionner(\mathcal{S})$ (ligne 3) qui renvoie la nouvelle liste $\mathcal{SM}$. Un motif obtenu à partir de la fusion de plusieurs motifs, est un motif contenant toutes les instructions des motifs originaux. Notons que la liste retournée $\mathcal{SM}$ peut être égale à la liste initiale des top-k $\mathcal{S}$, s'il n'existe aucune paire de motifs équivalents. Les listes Ω_1 et Ω_3 sont initialisées à vide (ligne 4), alors que la liste Ω_2 est initialisée avec les instructions qui apparaissent dans $\mathcal{SM}_1$ (le plus suspect des motifs) (ligne 5).

A partir de cet ensemble d'instructions suspectes Ω_2, l'algorithme 4 essaye de différencier entre les instructions, en tirant profit des trois observations suivantes 5.1, 5.2 et 5.3.

Observation 5.1. *Étant donné les top-k motifs $\mathcal{S}$, $\mathcal{SM}_1$ est une sur-approximation de la localisation de la faute :* $\forall S_i \in \mathcal{S} : (freq_{\mathcal{T}^-}(S_1) = freq_{\mathcal{T}^-}(S_i) \wedge (freq_{\mathcal{T}^+}(S_1) = freq_{\mathcal{T}^+}(S_i)) \Rightarrow S_i \subseteq \mathcal{SM}_1$

Comme $\mathcal{SM}_1$ correspond au motif le plus suspect i.e., il contient les instructions qui sont les plus fréquentes dans la base négative et peu fréquentes dans la base positive, il est très susceptible de contenir l'instruction fautive. Toutefois, il contient aussi d'autres instructions, qui ont un certain degré de suspicion. C'est pourquoi, dans l'algorithme 4, la liste Ω_2 est initialisée à $\mathcal{SM}_1$ à la ligne 5. L'idée est de dissocier les instructions $e \in \mathcal{SM}_1$ en raisonnant sur les motifs suivants.

Observation 5.2. *Soit l'ensemble de motifs $\mathcal{SM}$ (résultant des top-k motifs $\mathcal{S}$) et une sur-approximation de la localisation de la faute $\mathcal{SM}_1$ (Ω_2), certaines instructions de $\mathcal{SM}_1$ disparaissent dans les motifs suivants $\mathcal{SM}_i \in \mathcal{SM}$ ($i = 2..|\mathcal{SM}|$) :* $(\mathcal{SM}_1 \setminus \mathcal{SM}_i) \neq \emptyset$.

Dans l'algorithme 4, les instructions qui sont dans $\mathcal{SM}_1$ (Ω_2) et disparaissent dans un certain motif $\mathcal{SM}_i$ sont notées Δ_D (ligne 7). Selon les valeurs de fréquences, nous avons deux cas :

1. $\mathcal{SM}_i$ a la même fréquence que $\mathcal{SM}_1$ dans la base négative, mais le motif $\mathcal{SM}_i$ est plus fréquent dans la base positive que le motif $\mathcal{SM}_1$. Ainsi, les instructions de Δ_D sont moins fréquentes dans la base positive comparée à celles de $(\Omega_2 \setminus \Delta_D)$. Ainsi, les instructions de Δ_D sont alors plus suspectes que les instructions restantes dans $(\Omega_2 \setminus \Delta_D)$ et doivent par définition, être classées en premier (supprimées de Ω_2 (ligne 10) et ajoutées à Ω_1 (ligne 9)).
2. $\mathcal{SM}_i$ est moins fréquent dans la base négative que $\mathcal{SM}_1$. Dans ce deuxième cas, les instructions de Δ_D doivent aussi être classées en premier et ajoutées à Ω_1.

Par conséquent, Ω_1 contient les instructions les plus suspectes provenant de l'état initial de Ω_2. Les instructions restantes dans Ω_2 qui apparaissent dans tous les motifs $\mathcal{SM}_i$ sont classées après celles de Ω_1.

Observation 5.3. *Étant donné l'ensemble de motifs $\mathcal{SM}$ et la sur-approximation de la localisation de la faute $\mathcal{SM}_1$, certaines instructions apparaissent dans les motifs suivants* $\mathcal{SM}_i \in \mathcal{SM} : (\mathcal{SM}_i \setminus \mathcal{SM}_1) \neq \emptyset$.

Selon la propriété 5.3, certaines instructions qui ne sont pas dans $\mathcal{SM}_1$ (Ω_2) et qui apparaissent dans les autres motifs $\mathcal{SM}_i$ (ligne 11) sont notées Δ_A. La liste Δ_A est ajoutée à Ω_3 (ligne 15) et classée après Ω_2 comme étant les instructions les moins suspectes du programme (ligne 16).

En résumé, la première liste Ω_1 classe les instructions de $\mathcal{SM}_1$ selon leur ordre de disparition dans $\mathcal{SM}_2..\mathcal{SM}_l$ (ligne 6). La liste Ω_2 contient les instructions restantes de $\mathcal{SM}_1$

TABLE 5.2.: 1ère localisation - $\mathcal{S}$: top-k motifs suspects retournés par l'algorithme 3.

$\mathcal{S}$: top-k motifs suspects	$freq_{\mathcal{T}^+}(S_i)$	$freq_{\mathcal{T}^-}(S_i)$	rang
$S_1 : \{e_1, e_2, e_3, e_{10}\}$	0	6	1
$S_2 : \{e_1, e_2, e_{10}\}$	1	6	2
$S_3 : \{e_1, e_{10}\}$	2	6	3
$S_4 : \{e_1, e_2, e_3, e_4, e_{10}\}$	0	5	4
$S_5 : \{e_1, e_2, e_4, e_{10}\}$	1	5	5
$S_6 : \{e_1, e_2, e_3, e_4, e_6, e_{10}\}$	0	4	6
$S_7 : \{e_1, e_2, e_4, e_6, e_{10}\}$	1	4	7
$S_8 : \{e_1, e_2, e_3, e_4, e_5, e_{10}\}$	0	3	8
$S_9 : \{e_1, e_2, e_3, e_4, e_6, e_7, e_{10}\}$	0	2	9
$S_{10} : \{e_1, e_2, e_3, e_4, e_6, e_8, e_9, e_{10}\}$	0	2	9

qui apparaissent dans tous les motifs $\mathcal{SM}_i (i = 1..k)$. Finalement, la liste Ω_3 contient les instructions qui n'appartiennent pas à $\mathcal{SM}_1$ et qui apparaissent graduellement dans $\mathcal{SM}_2..\mathcal{SM}_l$.

5.6. Illustration à l'aide d'un exemple

Dans cette section, nous donnons un exemple pour illustrer le résultat de notre méthode, à travers un programme simple nommé *Compteur de caractères*, introduit dans [Gonzalez-Sanchez et al., 2011] et donné dans la figure 5.1. Le programme contient dix instructions exécutées sur huit cas de test, notés de tc_1 à tc_8 avec les cas de test négatifs (tc_1 à tc_6) et positifs (tc_7 et tc_8). Dans cet exemple, la faute est introduite à la ligne 3 où l'instruction correct `"let += 1"` est remplacée par `"let += 2"`. La figure 5.1 rapporte la couverture de chaque instruction, avec la valeur 1 si l'instruction est exécutée au moins une fois par le cas de test et la valeur 0, si elle n'est pas exécutée. D'après notre modélisation, la couverture d'un cas de test négatif (respectivement, positif) forme une transaction dans la base négative (respectivement, positive), nous avons alors, $\mathcal{T}^- = \{tc_1, tc_2, tc_3, tc_4, tc_5, tc_6\}$ et $\mathcal{T}^+ = \{tc_7, tc_8\}$.

Notre modèle de contraintes présenté dans la section 5.2, a pour objectif l'extraction des k motifs les plus suspects. Nous rappelons que le sens donné à la notion de suspicion, est lié à la fréquence d'un motif donné dans la base négative et/ou positive. Dans cet exemple, nous avons choisi la valeur de k de manière intuitive, en fixant k au nombre d'instructions du programme (i.e., $k = 10$). Justement, une étude expérimentale est menée et présentée dans la section 6.4 du chapitre 6, pour étudier l'impact de k sur la localisation. La table 5.2 donne le classement des top-k motifs, calculés par la première étape (algorithme 3) de l'approche avec leurs fréquences respectives. La table 5.4 donne le résultat obtenu par l'algorithme 4 prenant en entrée la table 5.2 et qui est la seconde étape de notre approche.

Table 5.3.: 2ème étape - $\mathcal{SM}$: top-k motifs suspects après fusion (algorithme 4)

$\mathcal{SM}$: top-k motifs suspects	$freq_{\mathcal{T}^+}(\mathcal{SM}_i)$	$freq_{\mathcal{T}^-}(\mathcal{SM}_i)$	rang
$\mathcal{SM}_1 : \{e_1, e_2, e_3, e_{10}\}$	0	6	1
$\mathcal{SM}_2 : \{e_1, e_2, e_{10}\}$	1	6	2
$\mathcal{SM}_3 : \{e_1, e_{10}\}$	2	6	3
$\mathcal{SM}_4 : \{e_1, e_2, e_3, e_4, e_{10}\}$	0	5	4
$\mathcal{SM}_5 : \{e_1, e_2, e_4, e_{10}\}$	1	5	5
$\mathcal{SM}_6 : \{e_1, e_2, e_3, e_4, e_6, e_{10}\}$	0	4	6
$\mathcal{SM}_7 : \{e_1, e_2, e_4, e_6, e_{10}\}$	1	4	7
$\mathcal{SM}_8 : \{e_1, e_2, e_3, e_4, e_5, e_{10}\}$	0	3	8
$\mathcal{SM}_9 : \{e_1, e_2, e_3, e_4, e_6, e_7, e_8, e_9, e_{10}\}$	0	2	9

Table 5.4.: Raffinement des résultats avec la 2ème étape (algorithme 4)

Instructions	Rang	Liste
e_3	1	Ω_1
e_2	2	Ω_1
$\{e_1, e_{10}\}$	4	Ω_2
e_4	5	Ω_3
e_6	6	Ω_3
e_5	7	Ω_3
e_7	10	Ω_3
$\{e_8, e_9\}$	10	Ω_3

Table 5.5.: Classement des instructions retourné par Tarantula et le taux d'émergence.

Instructions	Tarantula	Taux d'émergence	Rang
e_3	1	∞	1
e_5	1	∞	1
e_7	1	∞	1
e_2	0.66	2	4
e_4	0.62	1.67	5
e_6	0.57	1.33	6
e_1	0.5	1	8
e_{10}	0.5	1	8
e_8	0.4	0.67	10
e_9	0.4	0.67	10

a)- Tarantula et Taux d'émergence :
La formule 3.1 de TARANTULA qui calcule la suspicion d'une instruction e dans un programme, est très similaire à la formule de taux d'émergence (donnée dans la définition 1.7), qui mesure l'émergence d'un motif donné d'une base vers une autre (dans notre cas, des cas de test négatifs vers les cas de test positifs). En effet, les deux formules évaluent le ratio de la fréquence négative, comparée avec la fréquence positive, à la seule différence où TARANTULA évite de diviser par une fréquence nulle. Par conséquent, Les valeurs de TARANTULA ont le même accroissement que celle du taux d'émergence, et donc, les deux mesures donnent le même classement des instructions, comme illustré dans la table 5.5.

b)- F-CPMiner et Tarantula :
Le résultat de la première localisation de notre approche réalisée par l'algorithme 3, est une liste des top-k motifs les plus suspects. La table 5.2 donne les top-k motifs pour l'exemple 5.1. Le motif S_1 est le plus suspect car il apparaît le plus dans la base négative ($freq_{\mathcal{T}^-}(S_1) = 6$), contrairement aux motifs en bas de liste qui apparaissent peu dans la base négative ($freq_{\mathcal{T}^-}(S_{10}) = 2$). L'instruction fautive e_3 est capturée par le motif S_1 qui contient également 3 autres instructions. La seconde étape a pour objectif d'affiner les motifs de la liste $\mathcal{S}$ (particulièrement le motif S_1), pour avoir la faute, sans devoir examiner les autres instructions.

Dans l'algorithme 4, à la ligne 3, la première étape avant le classement des instructions, consiste à fusionner les motifs de même degré de suspicion. Dans la table 5.2, nous avons $S_9 =_{\mathcal{S}} S_{10}$, ces deux motifs S_9, S_{10} sont fusionnés dans un seul motif $\mathcal{SM}_9$ qui va contenir l'union des instructions de S_9, S_{10} (voir la table 5.3). Par la suite, l'algorithme 4 compare les motifs suspects, pour en extraire un classement des instructions en termes de suspicion. Nous comparons les résultats de notre méthode donnés dans la table 5.4, avec ceux donnés par TARANTULA dans la table 5.5 (2^{eme} colonne) et le taux d'émergence (3^{eme} colonne). Nous dressons les observations suivantes :

- Dans nos résultats, la faute qui est introduite dans e_3 est classée en premier. Elle est l'instruction la plus suspecte. Selon l'algorithme 4, cette instruction est dans le motif le plus suspect $\mathcal{SM}_1$ ayant les fréquences ($freq_{\mathcal{T}^+}(\mathcal{SM}_1) = 0, freq_{\mathcal{T}^-}(\mathcal{SM}_1) = 6$). Nous notons que e_3 disparaît dans le prochain motif $\mathcal{SM}_2$, ayant les fréquences ($freq_{\mathcal{T}^+}(\mathcal{SM}_2) = 1, freq_{\mathcal{T}^-}(\mathcal{SM}_2) = 6$). Ici, e_3 sera ajoutée à Δ_D (ligne 7) et supprimée de Ω_2. Cependant, TARANTULA considère les trois instructions e_3, e_5, e_7 comme les instructions les plus suspectes, avec le même rang, bien que ces instructions ont des fréquences différentes dans la base des cas de test négatifs. En effet, la mesure de TARANTULA est incapable de différencier ces instructions entre elles.

- En utilisant la méthode TARANTULA, les instructions $\{e_5, e_7\}$ sont classées avant $\{e_2, e_4, e_6\}$, bien que les instructions $\{e_2, e_4, e_6\}$ soient plus suspectes, car elles sont plus présentes dans les cas de test négatifs que $\{e_5, e_7\}$. Ce classement

est dû à la faible précision donnée par la formule 3.1 de Tarantula. Cette faiblesse est compensée par les instructions qui *apparaissent* dans l'algorithme 4. Revenons à notre exemple, les instructions e_4 et e_6 ne sont pas dans le motif $\mathcal{SM}_1$ et apparaissent dans les motifs suivants ($\mathcal{SM}_4$ et $\mathcal{SM}_5$ respectivement). Par conséquent, ces instructions sont ajoutées à Δ_A (ligne 11) et donc ajoutées à Ω_3.

- Concernant les instructions $\{e_8, e_9\}$ et $\{e_1, e_{10}\}$, Tarantula révèle que ces instructions ont la même suspicion, sans pour autant les distinguer. Notre approche donne le même résultat, car l'algorithme 4 relève ces instructions en même temps (même groupe). Dans notre approche, l'instruction e_7 a aussi la même suspicion que $\{e_8, e_9\}$ car ces instructions apparaissent dans différents motifs, mais ces motifs avaient le même rang, donc ces instructions vont se retrouver dans un seul motif après l'étape de fusion.

Cette comparaison entre l'efficacité de Tarantula et notre modèle de fouille sous contraintes, révèle la précision que notre approche peut apporter, en exploitant au mieux les relations entre les instructions du programme.

5.7. Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche basée sur l'extraction de motifs ensemblistes et la programmation par contraintes pour traiter le problème de localisation de fautes. Notre approche procède en deux étapes. En premier, nous avons formellement défini le problème de localisation de fautes, comme une problématique de fouille en faisant appel à la PPC, pour modéliser et résoudre les contraintes, tenant compte de la nature bien particulière des motifs recherchés. La résolution du modèle à contraintes, nous permet d'obtenir les top-k suites des instructions les plus suspectes. La seconde étape *ad-hoc* a pour objectif, de classer de façon plus précise l'ensemble des instructions des top-k, en tirant profit de deux observations majeures :

i) Les fautes introduites dans un programme, peuvent être vues comme un motif (ensemble d'instructions) qui est plus fréquent dans les exécutions négatives que celles positives.

ii) La différence entre un motif pertinent et un autre qui l'est moins, est un sous ensemble d'instructions, qui apparaissent/disparaissent d'un motif à un autre. Cette différence, nous aide à mieux repérer la position de la faute dans le programme.

Nous avons montré, comment ces deux propriétés peuvent être exploitées dans un algorithme de classement *ad-hoc*, pour produire une localisation précise.

Dans le chapitre 6 suivant, nous présentons une série d'études expérimentales sur l'approche F-CPminer, afin de montrer l'apport pratique que peut amener cette approche, comparée à d'autres méthodes.

6. F-CPMINER : Études expérimentales

DANS ce chapitre, nous présentons une série d'études expérimentales, visant à évaluer l'approche F-CPMINER sur des programmes de la suite *Siemens*, connue en test logiciel. Les expérimentations que nous menons portent sur (i) L'évaluation de l'approche F-CPMINER en termes de performances, en adoptant le modèle réifié et en adoptant la contrainte globale CLOSEDPATTERN, lors de l'extraction de k motifs suspects. (ii) La Comparaison de F-CPMINER avec les mesures de suspicion de l'état de l'art, en termes de qualité de localisation. Nous présentons également à la fin de ce chapitre, une première expérimentation de F-CPMINER sur un programme avec plusieurs fautes.

Sommaire

Ce chapitre reporte les différentes études expérimentales que nous avons réalisées. Nous commençons par rappeler dans la partie 6.1, l'essentiel de notre approche nommée F-CPMINER. Pour les expérimentations, dans un premier temps, nous présentons la base de programmes, que nous avons utilisés durant nos expérimentations dans la partie 6.2. En second lieu, nous détaillons le protocole expérimental, pour l'évaluation de notre approche dans la partie 6.3. Ensuite, nous étudions et discutons l'influence du paramètre k, pour l'extraction des top-k motifs suspects, sur l'efficacité de l'approche F-CPMINER dans la partie 6.4. Nous comparons dans un premier temps, notre approche en termes de performances dans la partie 6.5, lorsque celle ci fait appel au modèle réifié (nommée F-CPMINER$^+$) avec la version exploitant la contrainte globale (nommée F-CPMINER*). Par la suite, nous comparons nos résultats avec ceux obtenus avec TARANTULA [Jones and Harrold, 2005], OCHIAI et JACCARD [Abreu et al., 2007] dans la partie 6.6. Nous évaluons par la suite, l'impact de la variation du nombre de cas de test, sur la performance des différentes approches dans la partie 6.7. Enfin, nous expérimentons et analysons le comportement de notre approche, comparée aux mesures de suspicion, sur un programme à fautes multiples dans la partie 6.8.

6.1. L'outil F-CPMINER

Dans le chapitre 5, nous avons présenté l'approche F-CPMINER [Maamar et al., 2015, Maamar et al., 2016c, Maamar et al., 2016a], une technique visant à assister le programmeur pendant l'activité de localisation de fautes, lors du test logiciel. L'approche proposée, tire profit de la fouille de données sous contraintes, offrant la possibilité et l'avantage de raisonner simultanément sur un ensemble d'instructions (motif). Ainsi, nous pouvons traiter les régularités qui peuvent exister, entre les exécutions des instructions d'un programme donné, reflétant le concept de *motif suspect*.

Nous rappelons que, l'approche F-CPMINER procède sur deux étapes, à savoir :

1. Extraction des top-k motifs suspects, en fonction d'une relation de dominance 5.1, permettant d'ordonner l'ensemble des motifs, issus des bases de couvertures de cas de test. L'extraction de motifs avec l'algorithme 3, peut se faire en utilisant le modèle réifié proposé dans la sous-section 5.2.2, ou en faisant appel à la contrainte globale CLOSEDPATTERN qui a été mise au service de la localisation (voir la sous-section 5.2.3). Cette première étape représente une première localisation des fautes.

2. Raffinement de la première localisation, grâce à un mécanisme de raisonnement sur les top-k motifs suspects, illustré dans l'algorithme 4, permettant de raffiner le résultat de la première localisation et de produire un classement plus précis, des instructions issues de ces top-k motifs.

Dans la suite de ce chapitre, nous présentons une série d'expérimentations réalisées sur les programmes de la suite *Siemens*. Ces expérimentations montrent l'efficacité en termes de performances de F-CPMINER, lorsque celui-ci fait appel à la contrainte globale CLOSEDPATTERN, par rapport au modèle réifié. Ces expérimentations montrent également

TABLE 6.1.: La base de programmes Siemens (111 programmes)

Programme	Description	Versions erronées	LOC	LEC	Cas de tests
Replace	Pattern replacement	29	514	245	5542
PrintTokens2	Lexical analyzer	9	358	200	4056
PrintTokens	Lexical analyzer	4	348	195	4071
Schedule	Priority scheduler	5	294	152	2650
Schedule2	Priority scheduler	8	265	128	2680
TotInfo	Information measure	19	272	123	1052
Tcas	Altitude separation	37	135	65	1578

LOC : lignes de code dans la version correcte – LEC : lignes de code exécutables

que F-CPMiner, permet d'obtenir une meilleure qualité de localisation, comparé aux mesures de suspicion, vues dans le chapitre 3.

6.2. Base de programmes

Nous utilisons dans nos expérimentations, la suite de programmes de *Siemens* [Hutchins et al., 1994], qui est la base la plus utilisée, pour la comparaison des différentes techniques de localisation de fautes. Cette base est composée de sept programmes écrits en C, où chaque programme se décline en plusieurs versions ayant différentes fautes. La base est illustrée dans la table 6.1. Notons que des suites de cas de test sont disponibles, pour tester chaque catégorie de programmes [1].

Nous rappelons que la suite de programmes *Siemens*, a été assemblée aussi bien pour les études de localisation de fautes, que pour la détection des fautes avec les critères basés sur le graphe de flot de contrôle et le graphe de flot de données. Durant nos expérimentations, nous avons exclu 21 versions, qui sont hors de portée de la tâche de localisation de fautes traitée par notre approche (e.g., erreurs de segmentation). En effet, lorsqu'une erreur de segmentation se produit, nous ne sommes pas en mesure de générer correctement la couverture des instructions (la couverture s'arrête à l'instruction ayant causée l'erreur de segmentation). Ainsi, ces versions défectueuses posent de nombreux problèmes, qui pourraient perturber une discussion objective, des résultats expérimentaux. En somme, nous avons 111 programmes avec différentes fautes, détaillés dans la table 6.1.

6.3. Protocole expérimentale et implémentation

Tout d'abord, nous avons besoin de savoir, quelle instruction est couverte lors d'une exécution donnée. Pour cela, nous faisons appel à l'outil Gcov [2]. Cet outil est

1. Une description complète de *Siemens suite* est donnée dans [Hutchins et al., 1994]
2. https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

appelé lors de la compilation d'un programme, pour repérer quelles instructions sont actuellement exécutées, par un cas de test donné. GCOV peut aussi être utilisé, pour connaître le nombre des exécutions d'une instruction donnée. Par ailleurs, dans notre approche, nous avons seulement besoin de la matrice de couverture, dont la génération nécessite l'exécution du programme P sur chaque cas de test, puis le résultat retourné est comparé avec le résultat attendu (Oracle). Si les deux résultats sont égaux, nous ajoutons la couverture du cas de test à la base transactionnelle positive, autrement, la couverture est ajoutée à la base négative.

Notre mise en œuvre informatique est nommée F-CPMINER, cet outil est écrit en C++. L'implémentation a été réalisée en utilisant GECODE [3], un solveur ouvert, libre et portable, pour le développement des programmes à contraintes. Notre implémentation inclut :

(i) L'algorithme 3 pour l'extraction des top-k motifs suspects (section. 5.2). Cet algorithme implémente les contraintes 5.1, 5.2, 5.3, 5.4 et 5.5, lorsque celui-ci utilise le modèle réifié, conçu dans la partie 5.2.2. Dans la seconde version, l'algorithme 3 inclut la contrainte CLOSEDPATTERN illustrée dans le chapitre 4 et les contraintes 5.1, 5.2, 5.3, 5.4. Dans la sous-section 5.2.1, nous avons donné les variables du modèle de localisation, à savoir, l'ensemble $\{P_1, ..., P_n\}$ de variables booléennes pour représenter les n instructions d'un motif suspect P et l'ensemble $\{T_1, ..., T_m\}$ pour représenter les m cas de test pour tester un programme donné.

(ii) L'algorithme 4 qui traite les top-k motifs et retourne une localisation précise des fautes, avec un classement des instructions, selon leurs degrés de suspicion.

Nos expérimentations ont été réalisées sur un processeur Intel Xeon CPU E3-1245 V2 @ 3.40 GHz et 32 Gb de RAM.

Afin d'avoir une comparaison précise entre F-CPMINER et les autres mesures (notées MEASURES), nous avons implémenté ces mesures et avons évalué la suspicion des instructions de la même manière, que celle présentée dans la section 3.3. D'autre part, nous avons adopté une métrique très connue dans la localisation de fautes, à savoir l'EXAM score [Wong and Debroy, 2009] présentée dans la section 3.5, qui mesure l'efficacité d'une approche de localisation donnée.

F-CPMINER et MEASURES peuvent retourner un ensemble d'instructions équivalentes en termes de suspicion (i.e., avec le même degré de suspicion). Dans ce cas, l'efficacité dépend de l'instruction qui est examinée en premier. Pour cela, nous considérons les deux EXAM score, à savoir l'EXAM score *optimiste* (noté O-EXAM) et *pessimiste* (noté P-EXAM) (définitions 3.5 et 3.6). Nous reprenons également la troisième métrique, i.e., Δ-EXAM.

3. www.gecode.org

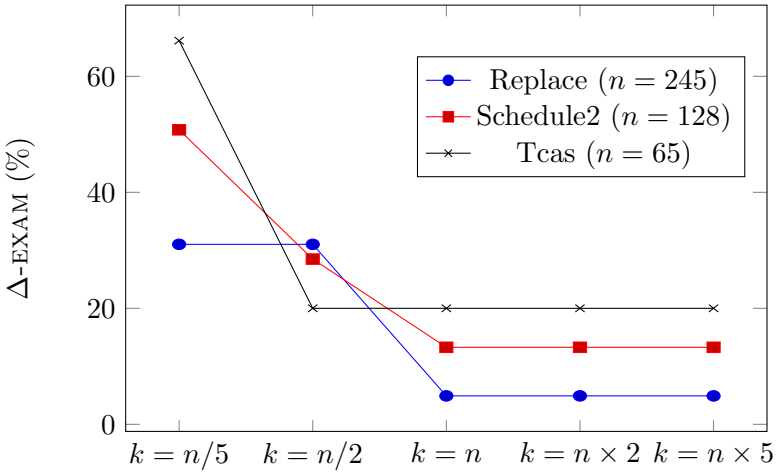

FIGURE 6.1.: Comparaison de différentes valeurs de k.

6.4. Étude 1 : Influence de k sur la localisation

Notre première expérimentation a pour but de déterminer la meilleure valeur pour k, permettant d'atteindre le Δ-EXAM optimal. En effet, une petite valeur pour le Δ-EXAM, signifie que la méthode donne un ensemble réduit d'instructions avec une suspicion équivalente, ce qui traduit une localisation précise. Nous rappelons que l'algorithme 3 prend en entrée le paramètre k pour l'extraction des top-k motifs suspects. Dans cette expérimentation, nous sélectionnons trois programmes de Replace, Schedule2 et Tcas qui ont différentes tailles (resp. 245, 128, 65 instructions exécutables). Nous avons fait varier la valeur de k à partir de (n / 5) à (5 $\times n$), où n représente la taille du programme en termes d'instructions exécutables.

La figure 6.1 montre l'impact de k sur la précision du Δ-EXAM retourné. La première observation que nous pouvons notée, traduit le fait que, en augmentant la valeur de k, nous réduisons la distance entre O-EXAM et P-EXAM. En effet, plus nous avons de motifs dans la liste des top-k, mieux nous pouvons tirer d'observations sur les formes de ces motifs, et donc en tirer des ensembles d'instructions de plus en plus précis. La seconde observation est qu'à partir d'une certaine valeur de k supérieure ou égale à la taille du programme (i.e., $k = n$), le Δ-EXAM devient stable. Dans le reste de cette section expérimentale et d'après nos tests, k est fixé à la taille du programme. Cette expérimentation est valable pour les deux versions F-CPMINER* et F-CPMINER$^+$, car dans les deux cas, l'algorithme 3 prend en entrée le paramètre k.

6.5. Étude 2 : F-CPMINER* versus F-CPMINER$^+$

Dans cette sous section, nous présentons dans le table 6.2 une analyse des statistiques des performances des deux versions, à savoir, F-CPMINER* comparé à la version F-CPMINER$^+$. Pour chaque classe de programme, nous reportons le temps CPU moyen et

TABLE 6.2.: Performances d'extraction des top-k (F-CPMINER* vs F-CPMINER$^+$).

		Temps CPU (s)			#Propagations		#Noeuds	
Prog	k	F-CPMINER*	F-CPMINER$^+$	Speed up	F-CPMINER*	F-CPMINER$^+$	F-CPMINER*	F-CPMINER$^+$
(1)	245	**20.03**± 20.46	147.69 ± 86.02	**7.37**	**8116650**	216625216	**37530**	2450119
(2)	200	**4.62** ± 0.47	146.25 ± 65.89	**31.65**	**2590350**	180761014	**5468**	3588108
(3)	195	**4.62** ± 0.39	61.49 ± 34.35	**13.28**	**1914429**	70013851	**3331**	1235562
(4)	152	**0.85** ± 0.26	27.41 ± 15.28	**32.24**	**489246**	15821395	**2350**	210884
(5)	128	**1.46** ± 1.78	12.66 ± 5.37	**8.67**	**807141**	7602101	**4870**	113438
(6)	123	**0.20** ± 0.07	2.53 ± 1.01	**12.65**	**229505**	1698834	**1023**	10445
(7)	65	**0.04** ± 0.01	0.16 ± 0.02	**4**	**99241**	245331	**60**	133

(1) Replace, (2) Print tokens2, (3) Print Tokens, (4) Schedule, (5) Schedule2, (6) Tot info, (7) Tcas

l'écart type pour les deux versions. Nous reportons également, des statistiques de PPC, telles que, le nombre moyen de propagations et de nœuds explorés, pour extraire les top-k motifs suspects.

Il est important de rappeler que les approches telle que TARANTULA, évaluent le degré de suspicion de chaque instructions en utilisant la formule 3.1, en ne prenant en compte aucune dépendance entre les instructions. Ainsi, l'explosion combinatoire due aux combinaisons possibles d'instructions, n'est pas traitée par les mesures (MEASURES). En effet, nous avons montré dans la section 3.3, que la complexité de l'algorithme de ces mesures est linéaire sur le nombre de cas de test. Par conséquent, le temps CPU obtenu par une telle approche est négligeable (en millisecondes).

Nous rappelons que l'approche F-CPMINER* (respectivement, F-CPMINER$^+$) fait appel à la contrainte globale CLOSEDPATTERN (respectivement, modèle réifié) pour l'extraction des motifs suspects. Il est important de souligner que la combinatoire des instructions/cas de test, pour extraire les motifs les plus suspects est assez importante, au vu du nombre considérable de motifs candidats (i.e. $|\mathcal{L}_\mathcal{I}| = 2^\mathcal{I}$, où $\mathcal{I}$ est l'ensemble des instructions du programme). Ceci explique en majeure partie les temps CPU retournés par l'approche proposée.

La première observation est que le modèle intégrant la contrainte globale, offre une nette amélioration en termes de temps CPU (voir la colonne Speed up). Ce facteur varie en fonction de la taille du programme traité et atteint une valeur de 32 (respectivement, 31) pour le programme Schedule (respectivement, PrintTokens2). Ce qui fait que nous avons une extraction des top-k motifs suspects nettement plus rapide avec CLOSEDPATTERN.

Le temps requis par la version F-CPMINER$^+$ pour calculer les top-k motifs suspects, est un réel inconvénient en localisation de fautes. Par exemple, pour le programme PrintTokens2, F-CPMINER$^+$ nécessite en moyenne 146 secondes (contre 4 secondes pour F-CPMINER*), ce qui en pratique représente un temps d'attente assez important pour le programmeur qui doit debugger son code. En effet, ceci est dû essentiellement, au modèle réifié sur lequel repose ce modèle d'extraction de motifs. Par conséquent, le filtrage de ce modèle pour propager les contraintes est un filtrage classique (voir la section 2.4).

TABLE 6.3.: Temps CPU de F-CPMINER sur la *suite Siemens* (en secondes).

Programme	k	top-k extraction (algo.3)	raffinement (algo.4)
Replace	245	**20.03** ± 20.46	**0.051** ± 0.015
PrintTokens2	200	**4.62** ± 0.47	**0.045** ± 0.013
PrintTokens	195	**4.63** ± 0.39	**0.030** ± 0.003
Schedule	152	**0.85** ± 0.26	**0.012** ± 0.004
Schedule2	128	**1.46** ± 12.66	**0.016** ± 0.004
TotInfo	123	**0.20** ± 0.07	**0.014** ± 0.006
Tcas	65	**0.04** ± 0.01	**0.001** ± 0.000

Concernant le nombre de propagations/nœuds, l'approche intégrant la contrainte globale est nettement plus efficace, avec un nombre de propagations plus réduit, dû au nombre de contraintes réifiées du premier modèle, outre les contraintes supplémentaires ajoutées dynamiquement durant la recherche. En termes de nœuds, comme nous l'avons montré dans le chapitre 4, la contrainte CLOSEDPATTERN assure la consistance de domaine avec un propagateur dédié (Algorithme 2), en filtrant les motifs non-fréquents et non-fermés, ce qui explique clairement la différence obtenue entre les deux modèles.

Remarque : Pour la suite de ce chapitre, F-CPMINER désigne notre approche de localisation avec la contrainte globale CLOSEDPATTERN et que nous utilisons pour le reste des expérimentations.

Temps CPU de la première localisation et raffinement :

La table 6.3 reporte pour chaque classe de programmes, le temps CPU moyen et l'écart type pour les deux étapes de F-CPMINER (i.e., l'extraction des top-k motifs correspondant à l'algorithme 3 et l'étape de raffinement correspondant à l'algorithme 4).

La première observation que l'on peut faire est que l'extraction des top-k motifs est l'étape la plus coûteuse dans F-CPMINER. Ceci étant en partie expliqué par le nombre très important de motifs candidats à explorer, pour extraire les top-k (i.e., $|\mathcal{L}_{\mathcal{I}}|$). En effet, l'espace de recherche est défini par le nombre d'instructions et de cas de test.

Toutefois, dans nos expérimentations, les temps CPU ne dépassent pas 20 secondes dans le pire des cas (voir le programme Replace). La seconde observation est que les temps CPU consommés par l'étape de raffinement pour la localisation de l'instruction fautive sont négligeables (de l'ordre de millisecondes dans tous les cas).

La figure 6.2 montre la variation du temps CPU pour l'extraction des top-k sur les 29 versions différentes (de v1 à v29) du programme Replace (le plus coûteux). Pour cette classe, l'écart type du temps CPU est assez large (i.e., 20.46 s) par rapport à un temps CPU moyen, égal à 20.03s. L'écart type n'est pas négligeable, dû au fait que toutes les versions d'une classe de programme ont les mêmes cas de test, mais elles ont des bases $\mathcal{T}^+$ et $\mathcal{T}^-$ différentes, tant au niveau de la taille de ces bases (nombre de transactions), qu'au niveau de leurs contenus (densités). Cette différence est due principalement, au

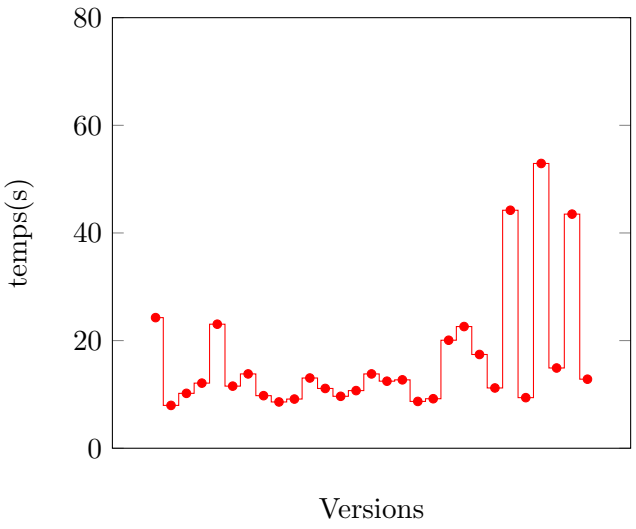

FIGURE 6.2.: Variation du temps CPU pour l'extraction des top-k (Replace)

type de faute introduite dans le programme. En effet, chaque paire de bases ($\mathcal{T}^+$,$\mathcal{T}^-$) produit un arbre de recherche différent pour l'extraction des k motifs, par conséquent, un temps de recherche différent.

6.6. Étude 3 : F-CPMINER versus MEASURES

La table 6.4 donne une comparaison entre F-CPMINER et les différentes mesures, vues dans le chapitre 3, notées MEASURES. La comparaison est basée sur l'EXAM score, pour les 111 programmes de la suite *Siemens*. Pour chaque classe de programmes (e.g., Tcas-* inclut 37 versions), nous reportons le nombre moyen de cas de test positifs $|\mathcal{T}^+|$, le nombre moyen de cas de test négatifs $|\mathcal{T}^-|$, la valeur de k utilisée dans F-CPMINER pour extraire les top-k motifs suspects (correspondant à la taille du programme), les valeurs moyennes de P-EXAM, O-EXAM et Δ-EXAM $\pm$ l'écart-type de chaque classe de programme.

La première observation est que F-CPMINER est meilleur dans 6 classes de programmes sur 7, selon les valeurs de P-EXAM et O-EXAM reportées :

– P-EXAM : Par exemple, si nous prenons la classe PrintTokens2-* avec ses 9 versions erronées (voir table 6.1), avec F-CPMINER la faute est localisée après avoir examiné seulement 2.55% du code avec un écart type = 3.11%. Cependant, TARANTULA, OCHIAI et JACCARD ont besoin d'examiner respectivement 16.52%, 10.83% et 16.07% avec un écart type dans l'intervalle [12% - 17%]. Cette tendance est observée dans 5 autres classes de programmes (Replace, PrintTokens, Schedule2, TotInfo et Tcas). Cependant, dans la classe de programme Schedule, F-CPMINER

TABLE 6.4.: F-CPMINER versus MEASURES (Comparaison des EXAM score).

				$\widehat{\text{P-EXAM}}(\%)$			
Programme	$\widehat{\lvert\mathcal{T}^+\rvert}$	$\widehat{\lvert\mathcal{T}^-\rvert}$	k	F-CPMINER	TARANTULA	OCHIAI	JACCARD
Replace	5450	92	245	**6.18** ± 8.56	11.09 ± 13.99	8.07 ± 10.46	10.81 ± 13.85
PrintTokens2	3827	299	200	**2.55** ± 3.11	16.52 ± 17.94	10.83 ± 12.74	16.07 ± 17.57
PrintTokens	4016	55	195	**3.97** ± 2.38	13.59 ± 11.21	6.66 ± 6.68	11.28 ± 10.01
Schedule	2506	144	152	21.45 ± 31.48	7.71 ± 3.65	**7.19** ± 4.06	7.21 ± 3.45
Schedule2	2646	34	128	**47.04** ± 23.03	61.70 ± 28.64	55.29 ± 27.57	61.61 ± 28.62
Totinfo	1015	37	123	**11.81** ± 7.78	23.62 ± 16.30	18.05 ± 12.09	21.52 ± 14.55
Tcas	1542	36	65	**39.28** ± 28.32	43.94 ± 31.92	42.11 ± 29.97	43.86 ± 31.96
				$\widehat{\text{O-EXAM}}(\%)$			
Programme	$\widehat{\lvert\mathcal{T}^+\rvert}$	$\widehat{\lvert\mathcal{T}^-\rvert}$	k	F-CPMINER	TARANTULA	OCHIAI	JACCARD
Replace	5450	92	245	**4.45** ± 7.29	9.35 ± 12.17	6.34 ± 8.47	9.07 ± 12.04
PrintTokens2	3827	299	200	**1.61** ± 1.82	15.58 ± 17.46	9.88 ± 12.11	15.13 ± 17.07
PrintTokens	4016	55	195	**2.05** ± 2.66	11.66 ± 11.39	4.74 ± 7.32	9.35 ± 10.48
Schedule	2506	144	152	19 ± 30.27	5.74 ± 3.59	**5.22** ± 3.63	5.25 ± 3.39
Schedule2	2646	34	128	**36.24** ± 20.71	50.90 ± 25.96	44.49 ± 25.42	50.80 ± 25.95
Totinfo	1015	37	123	**6.07** ± 6.86	17.88 ± 14.62	12.32 ± 10.74	15.78 ± 12.88
Tcas	1542	36	65	**15.26** ± 11.34	19.91 ± 14.75	18.09 ± 12.81	19.83 ± 14.81
				$\widehat{\Delta\text{-EXAM}}(\%)$			
Programme	$\widehat{\lvert\mathcal{T}^+\rvert}$	$\widehat{\lvert\mathcal{T}^-\rvert}$	k	F-CPMINER	TARANTULA	OCHIAI	JACCARD
Replace	5450	92	245	1.73 ± 3.15	1.73 ± 3.15	1.73 ± 3.15	1.73 ± 3.15
PrintTokens2	3827	299	200	0.94 ± 1.48	0.94 ± 1.48	0.94 ± 1.48	0.94 ± 1.48
PrintTokens	4016	55	195	1.92 ± 1.71	1.92 ± 1.71	1.92 ± 1.71	1.92 ± 1.71
Schedule	2506	144	152	2.45 ± 1.23	1.96 ± 0.45	1.96 ± 0.45	1.96 ± 0.45
Schedule2	2646	34	128	10.80 ± 5.51	10.80 ± 5.51	10.80 ± 5.51	10.80 ± 5.51
Totinfo	1015	37	123	5.73 ± 5.14	5.73 ± 5.14	5.73 ± 5.14	5.73 ± 5.14
Tcas	1542	36	65	24.02 ± 17.51	24.02 ± 17.51	24.02 ± 17.51	24.02 ± 17.51

est la moins bonne approche, comparée aux autres mesures. La mesure OCHIAI est la plus efficace avec un EXAM score = 7.19%.

– O-EXAM : Toujours avec la classe PrintTokens2-*, F-CPMINER localise la faute en moyenne après avoir examiné seulement 1.61% avec un écart type = 1.82%. Cependant, TARANTULA, OCHIAI et JACCARD ont besoin d'examiner respectivement 15.58%, 9.88% et 15.13% du code, avec un écart type dans l'intervalle [12% - 17%]. Cette tendance est observée dans 5 autres classes de programmes (Replace, PrintTokens, Schedule2, TotInfo et Tcas). Comme pour le P-EXAM, dans la classe de programme Schedule, F-CPMINER est la moins bonne approche, comparée aux autres mesures. La mesure OCHIAI est la plus efficace avec un EXAM score = 5.22%.

Notre seconde observation dans la table 6.4, concerne les valeurs de Δ-EXAM qui sont identiques, pour toutes les approches (sauf le cas Schedule ; 2.45% pour F-CPMINER contre 1.96% pour MEASURES). Ce qui se traduit par le fait que toutes ces approches retournent les mêmes ensembles d'instructions équivalentes (de même degré de suspicion). Toutefois, nous rappelons que le Δ-EXAM dans F-CPMINER est étroitement lié aux valeurs de k (voir figure 6.1) et qu'en utilisant k égale à la taille du programme nous arrivons à obtenir un Δ-EXAM optimal.

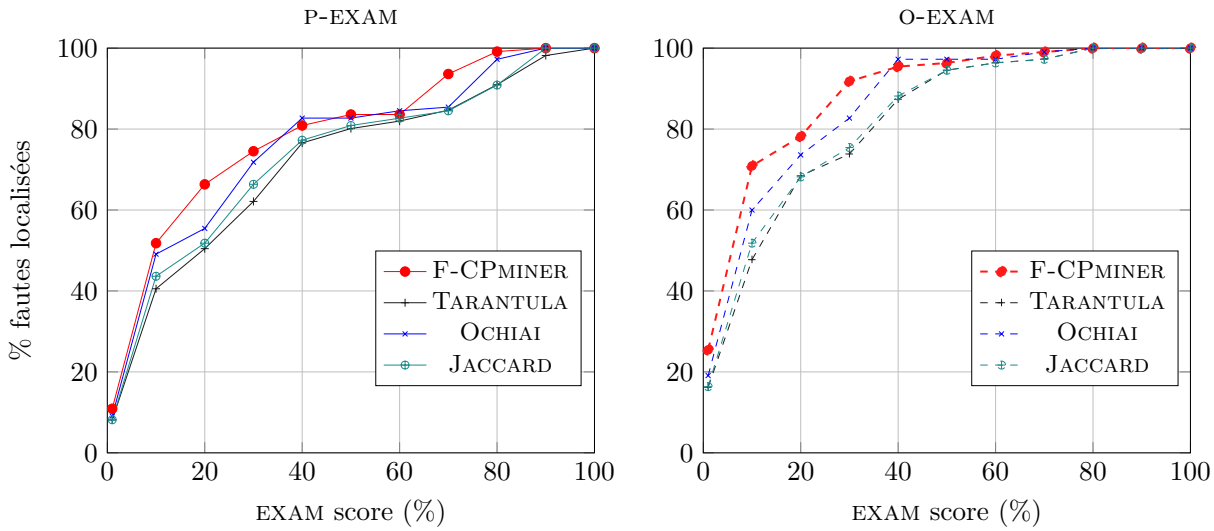

FIGURE 6.3.: TARANTULA et F-CPMINER : Comparaison d'efficacité

Dans le but de compléter les résultats de la table 6.4, la figure 6.3 montre une comparaison de l'efficacité basée sur le P-EXAM et O-EXAM entre F-CPMINER et les différentes mesures (TARANTULA, OCHIAI, JACCARD). L'axe des x donne les valeurs de l'EXAM score (avec des intervalles de 10%) et l'axe des y donne le pourcentage cumulatif des fautes localisées sur les 111 programmes.

Commençons par le cas pessimiste (i.e, P-EXAM) ; avec une valeur de P-EXAM = 30%, F-CPMINER localise plus de fautes que les MEASURES (un écart jusqu'à 11% de fautes localisées en plus par rapport à OCHIAI avec 20% du P-EXAM). Dans l'intervalle de 40% à 60%, F-CPMINER et OCHIAI agissent de la même manière en localisant plus de 80% des fautes. De 60% à 80% , nous observons une différence qui approche les 10% de fautes localisées en faveur de F-CPMINER, comparé à OCHIAI (93% pour F-CPMINER au lieu de 85% des fautes localisées pour OCHIAI). Toutefois, les 20% des fautes restantes sont rapidement localisées par F-CPMINER. Concernant TARANTULA et JACCARD, ces deux mesures sont tout le temps dominées par F-CPMINER. Pour le cas optimiste (i.e, O-EXAM) illustré par les courbes pointillées, F-CPMINER agit assez rapidement dés le début en localisant plus de fautes que les autres mesures.

Table 6.5.: exam score avec la 1ère localisation de F-CPminer

Programme	$\|P\|$	$\widehat{\|S_i\|}$	$\widehat{\text{p-exam}}$ (%)	$\widehat{\text{o-exam}}$ (%)	$\widehat{\Delta\text{-exam}}$ (%)
Replace	245	89	36.63 ± 12.67	0.40 ± 0	36.23 ± 12.67
PrintTokens2	200	110	55.16 ± 2.75	0.5 ± 0	54.66 ± 2.75
PrintTokens	192	87	44.87 ± 2.93	0.51 ± 0	44.36 ± 2.93
Schedule	152	103	68.15 ± 12.59	0.65 ± 0	67.50 ± 12.59
Schedule2	128	91	71.48 ± 6.69	0.78 ± 0	70.70 ± 6.69
TotInfo	128	54	44.50 ± 19.23	0.81 ± 0	43.73 ± 19.28
Tcas	65	49	76.34 ± 4.97	1.53 ± 0	74.81 ± 4.97

Nombre total d'instructions examinées : Nous donnons dans cette partie, un autre résultat intéressant, qui est le nombre total d'instructions qui doivent être examinées par les différentes méthodes, sur l'ensemble des 111 programmes. Au total, F-CPminer doit examiner 2364 instructions contre 3482 pour Tarantula, 2868 pour Ochiai et 3381 pour Jaccard dans le cas du p-exam, sur un total de 16211 instructions. Dans le cas du o-exam, F-CPminer doit examiner 1373 instructions, contre 2491 pour Tarantula, 1877 pour Ochiai et 2390 pour Jaccard. Cette dernière comparaison montre clairement qu'avec F-CPminer, nous devons examiner moins d'instructions pour trouver toutes les fautes sur les 111 versions, comparé aux mesures statistiques classiques.

D'une manière générale, nous pouvons conclure à partir des résultats présentés, que F-CPminer est très concurrentiel, par rapport aux autres mesures en termes d'efficacité (i.e., la métrique exam score), que ce soit en pessimiste, ou en en optimiste. En effet, F-CPminer est capable de localiser la plupart des fautes plus rapidement que les Measures.

Efficacité de l'algorithme de raffinement : Enfin, nous concluons cette partie avec les observations suivantes sur les 111 programmes testés :

- Dans 110 programmes, l'instruction fautive est dans le premier motif (i.e., le plus suspect $\mathcal{SM}_1$).
- Dans 96 programmes, la faute est une instruction qui disparaît de $\mathcal{SM}_1$, donc qui est dans Ω_1.
- Dans 14 programmes, la faute est localisée dans Ω_2.
- Dans seulement un programme, la faute est localisée dans Ω_3.

Ces observations montrent clairement, l'efficacité de la stratégie de classement, adoptée dans l'algorithme 4.

Discussion : Afin de voir clairement les limites et les avantages de chaque étape de notre approche (1ère localisation et raffinement), nous proposons dans la table 6.5 les

EXAM score retournés par la première étape. En d'autres mots, nous utilisons uniquement les top-k motifs pour trouver la faute sans aucun post-traitement. Pour se faire, nous parcourons simplement les motifs, instruction par instruction, jusqu'à trouver la faute. Comme nous avons constaté précédemment, que sur quasiment tous les programmes (à savoir 110), la faute est dans le premier motif, nous aurons à parcourir uniquement ce motif.

Dans la table 6.5, nous donnons pour chaque programme, la taille moyenne [4] du motif $\widehat{|S_i|}$ dans lequel se trouve la faute, les valeurs moyennes de P-EXAM, O-EXAM et Δ-EXAM $\pm$ l'écart-type de chaque classe de programme. La première observation est que le motif S_i où apparaît la faute pour la première fois, est de taille assez grande ($\geq 36\%$ de la taille du programme P). Par conséquent, la valeur du P-EXAM est naturellement élevée (entre $36.63\% - 76.34\%$), puisqu'il faut parcourir l'ensemble des instructions du motif S_i pour trouver la faute. Le O-EXAM ne dépasse jamais 1.53%, car dans ce cas nous considérons le motif S_i comme un seul ensemble d'instructions avec le même degré de suspicion, ce qui dans le cas optimiste revient à considérer la faute en premier, donc examiner une seule instruction. L'écart type dans le cas de O-EXAM est égale à 0, car dans toutes les versions, nous examinons une seule instructions. Le Δ-EXAM reflète l'écart entre le fait de parcourir tout le motif S_i et d'examiner une seule instruction.

Les résultats de cette expérimentation, montre nettement l'utilité de la seconde étape, dont le rôle est de dissocier au mieux les instructions d'un motif, en raisonnant sur sa forme et celle des motifs suivants. Par ailleurs, dans la section 6.5 nous avons montré que cette seconde étape de raffinement, n'est pas coûteuse en termes de temps CPU, comparée à la première localisation.

6.7. Étude 4 : Impact des cas de test sur la localisation

Dans cette section, nous étudions l'impact des cas de test sur la qualité de localisation. Nous avons comparé dans cette expérimentation F-CPMINER et TARANTULA (nous avons choisi TARANTULA, car c'est la mesure la plus connue de l'état de l'art). Pour se faire, nous avons changé le nombre de cas de test donné en entrée, en faisant varier la taille des bases $\mathcal{T}^+$ et $\mathcal{T}^-$. Pour chaque programme, nous réduisons la taille de ses bases de 100% à 10% en retirant à chaque fois de manière aléatoire 10% des cas de test en reportant l'EXAM score (O-EXAM et P-EXAM) pour F-CPMINER et TARANTULA. Dans cette section, nous avons sélectionné quatre programmes (PrintTokens2-v3, Tcas-v28, TotInfo-v18 et Replace-v22) que nous considérons comme représentatifs au vue de leur nature et la taille de leurs bases (nombre de cas de test). La figure 6.4 donne les résultats sur les programmes sélectionnés.

PrintToken2-v3. Pour cette instance de programme, lorsque nous considérons la totalité des bases (100% des cas de test), F-CPMINER et TARANTULA obtiennent approximativement le même résultat en termes de EXAM score. Une fois que nous commençons à réduire le nombre de cas de test, nous observons que F-CPMINER garde approximative-

4. En termes du nombre d'instructions que le motif contient.

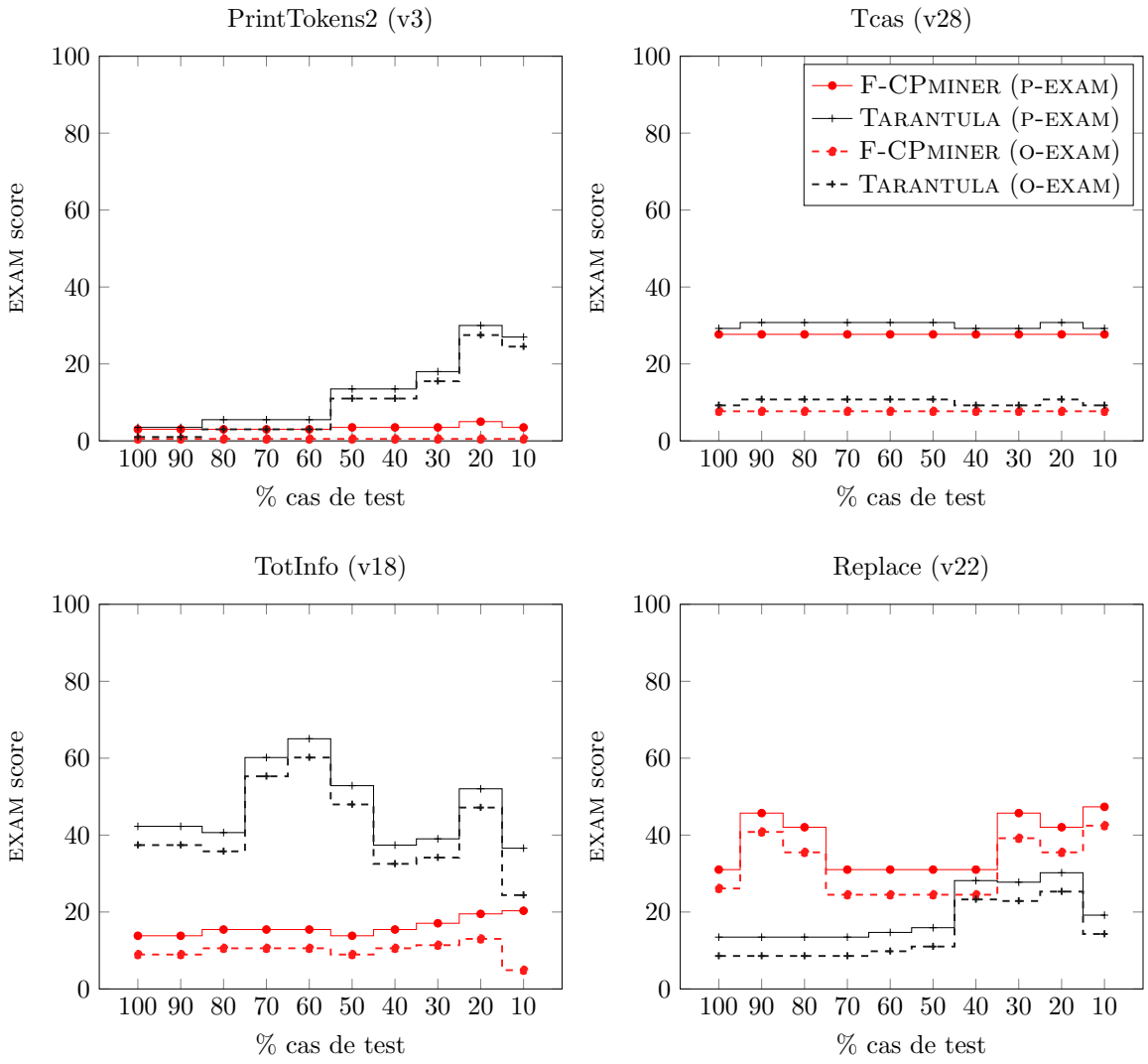

FIGURE 6.4.: Impact de $|\mathcal{T}^+|$ et $|\mathcal{T}^-|$ sur l'EXAM score de F-CPMINER et TARANTULA.

ment la même précision jusqu'à atteindre 10% de cas de test. D'autre part, TARANTULA montre un comportement chaotique où la précision diminue (i.e., EXAM score augmente) de manière significative après une réduction de 60% des cas de test, atteignant un score de 30%. Ce genre d'observations sont faites sur 6 versions parmi les 111 de la *suite Siemens*. Une seule de ces versions appartient aussi à la classe PrintToken2.

Tcas-v28. Pour cette instance du programme, F-CPMINER et TARANTULA obtiennent approximativement le même résultat avec un léger gain pour notre approche. Nous pouvons observer que la réduction des bases n'a pas un impact significatif sur la précision

des deux outils. Ce résultat est observé sur 32 programmes de la *suite Siemens*, dont 14 appartiennent à la classe Tcas.

TotInfo-v18. Ici, F-CPMINER montre un comportement stable et plus ou moins la même précision durant la réduction des bases de cas de test, alors que TARANTULA est très impacté par cette réduction (i.e., EXAM score change de manière significative). Ce même comportement est observé sur 22 programmes de la *suite Siemens*, avec 8 de la classe TotInfo.

Replace-v22. Pour cette instance de programme, nous pouvons voir que les deux approches sont plus ou moins affectées par la réduction du nombre de cas de test, TARANTULA est plus stable jusqu'à une réduction de 50% des cas de test, alors que F-CPMINER est stable sur l'intervalle [75% - 35%] des cas de test. Pour ce cas, les mêmes comportements (i.e., les deux approches sont affectées par la réduction) sont observés sur 34 autres programmes de la *suite Siemens*, dont 7 appartiennent à la classe Replace.

Ces observations sont particulièrement intéressantes et soulignent le fait que notre approche, peut être moins sensible, au nombre de cas de test considérés. Cela est particulièrement vrai, sachant qu'il y a 60 programmes sur 111 où F-CPMINER se comporte de manière stable (voir les 3 premières courbes de la fig 6.4). Dans 62 programmes TARANTULA a un comportement plus ou moins chaotique et sur seulement 17 programmes, TARANTULA est plus stable que F-CPMINER. Ceci est dû à la qualité de nos top-k motifs extraits et l'étape de post-traitement qui permet une analyse des régularités entre les motifs extraits. Ce qui n'est pas le cas pour TARANTULA, où l'ajout/suppression de cas de test peut amener à des résultats moins précis.

6.8. Étude 5 : Localisation de fautes multiples

Dans cette dernière partie, nous menons une étude sur la localisation de fautes multiples où nous adoptons la stratégie *one-bug-at-a-time* présentée dans le chapitre 3. L'idée est de localiser une faute à la fois, en régénérant entre chaque localisation, les nouvelles bases $\mathcal{T}^+$ et $\mathcal{T}^-$ pour extraire de nouveaux top-k motifs suspects permettant de localiser le reste des fautes efficacement. L'idée de cette première expérimentation est aussi d'observer le comportement de l'approche F-CPMINER sur une telle situation.

Nous avons construit une version du programme Schedule 2, où nous avons combiné 3 fautes provenant d'autres versions de la classe de programme Schedule 2 (v1,v5,v7). Le tableau 6.6 donne les fautes mises dans ce programme, à savoir, l'instruction 34, 45 et 116. Nous donnons la taille des bases $\mathcal{T}^+$ et $\mathcal{T}^-$ de ce programme. Le but est de suivre l'évolution de ces bases après chaque localisation.

Pour chaque itération (localisation), nous donnons pour chaque approche, la faute localisée, l'EXAM score nécessaire pour trouver une première faute (colonne P-EXAM-O-EXAM). Dans le reste du tableau, nous donnons pour F-CPMINER, les fréquences du motif dans lequel, chaque instruction fautive apparaît pour la première fois, noté S_A, ainsi que les fréquences du motif dans lequel, chaque instruction fautive disparaît, noté S_D. Pour les autres mesures, nous donnons le nombre d'exécution de chaque instruction

TABLE 6.6.: Localisation de fautes multiples - Schedule 2

	Fautes = (34,45,116) - $\lvert\mathcal{T}^+\rvert = 2556, \lvert\mathcal{T}^-\rvert = 124$									
Itération	**Outil : Faute**	O-EXAM - P-EXAM	**Motif**	**n°34**		**n°45**		**n°116**	
				$freq^+$	$freq^-$	$freq^+$	$freq^-$	$freq^+$	$freq^-$
1	F-CPminer : 45	**14.72 - 27.13**	S_A	2098	124	2098	124	2098	124
			S_D	2518	124	2518	124	2556	124
	Tarantula : 45	58.91 - 68.99	/	2518	124	2517	124	2525	124
	Jaccard : 45	58.13 - 68.21							
	Ochiai : 45	48.83 - 58.91							
2	F-CPminer : 34	**14.72 - 27.13**	S_A	2157	63	/	/	2157	63
			S_D	2579	63	/	/	2617	63
	Tarantula : 34	65.11 - 66.66	/	2579	63	/	/	2586	63
	Jaccard : 34	65.11 - 66.66							
	Ochiai : 34	59.68 - 61.24							
3	F-CPminer : 116	**51.93 - 65.89**	S_A	/	/	/	/	1632	31
			S_D	/	/	/	/	2649	31
	Tarantula : 116	75.96 - 89.92	/	/	/	/	/	2618	31
	Jaccard : 116	75.96 - 89.92							
	Ochiai : 116	66.66 - 80.62							
$\widehat{\textbf{Exam}}$	F-CPminer	**27.12 - 40.05**							
	Tarantula	66.66 - 75.19							
	Jaccard	66.40 - 74.93							
	Ochiai	58.39 - 66.92							

fautive, par les cas de test négatifs/positifs.

a)- Séquence de localisation : La première observation est que toutes les méthodes F-CPminer et Measures, localisent les fautes suivant la même séquence, à savoir, les instructions 45, 34 et ensuite 116. L'explication est que, même si ces méthodes ont différentes façons d'interpréter la suspicion d'une instructions (voir la figure 3.3), elles suivent plus ou moins la même intuition générale, qui consiste à dire que la suspicion est proportionnelle à l'exécution des cas de test négatifs.

b)- Exam score : La seconde observation concerne les exam score retournés. Sur chacune des trois itérations, F-CPminer localise plus rapidement la faute, que les approches Measures. En effet, pour la première itération, F-CPminer examine 14.72% avec o-exam (resp, 27.13% avec p-exam) du code pour trouver la faute 45. Là où, Tarantula et Jaccard nécessitent l'examen de plus de 58% du code avec o-exam (resp, plus de 68% avec p-exam). Ochiai est plus efficace que Tarantula et Jaccard, en examinant 48% du code avec o-exam (resp, 58% avec p-exam). Cette même tendance est observée sur les deux autres itérations. En moyenne, F-CPminer examine plus de deux fois moins de code, pour trouver toutes les fautes comparativement à Measures. En effet, F-CPminer examine 27.12% avec o-exam (resp, 40.05% avec

P-EXAM) du code. OCHIAI qui est la meilleure mesure examine en moyenne 58.39% avec O-EXAM (resp, 66.92% avec P-EXAM) du code.

c)- Évolution de $\mathcal{T}^+$ et $\mathcal{T}^-$: La dernière observation concerne l'évolution des bases $\mathcal{T}^+$ et $\mathcal{T}^-$ après chaque localisation. Dans la première itération, le motif S_A dans lequel apparaissent les instructions fautives pour la première fois, est le même (celui avec $freq_{\mathcal{T}^+}(S_A) = 2098$ et $freq_{\mathcal{T}^-}(S_A) = 124$). En effet, c'est le premier motif S_1 de la liste des top-k. Par ailleurs, l'instruction 116, ne disparaît dans le même motif S_D que les instructions 34 et 45. Ceci s'explique par le fait que, l'instruction 116 ne possède pas les mêmes fréquences que les instructions 34 et 45. Par conséquent, l'instruction 116 peut persister dans la liste top-k dans des motifs moins suspects, alors que, les instructions 34 et 45 disparaissent plus tôt, du fait de leur fréquences positives moins élevées, comparativement à celle de l'instruction 116. Ici, nous avons les instructions 34 et 45 dans le même motif fermé. En effet, nous avons constaté que l'instruction 34 est une extension propre de l'instruction 45 (voir la définition 1.9 de l'extension propre).

Dans l'itération 2, après avoir corrigé la faute 45, nous constatons que les motifs dans lesquels apparaissent les instructions fautives 34 et 116, deviennent moins suspects, comparés à la première itération, ce qui semble logique, puisque le programme contient une faute en moins. L'instruction 34 disparaît dans un motif S_D, qui est plus haut dans la liste top-k, que le motif où disparaît l'instruction 116. Ceci est dû encore une fois, à la fréquence positive de l'instruction 116, qui est plus élevée que celle de l'instruction 34. C'est donc l'instruction 34 qui est localisée puis corrigée en premier. Pour finir, nous avons la faute restante à l'instruction 116, dont le motif S_A devient moins suspect en corrigeant la faute 34.

En somme, l'étude que nous avons menée, porte seulement sur un programme, mais nous laisse penser que F-CPMINER, ouvre une voie intéressante pour la localisation de fautes multiples. D'autant plus, qu'une telle étude, nous a permis de mieux comprendre le comportement de notre outil F-CPMINER en suivant l'évolution de l'effet, que peut avoir la présence d'une ou plusieurs fautes, sur la répartition des cas de test. Aussi, nous avons observé qu'en corrigeant les fautes, les motifs composant la liste top-k deviennent moins suspects.

6.9. Conclusion

Nous avons présenté dans ce chapitre, une série d'études expérimentales pour évaluer notre approche F-CPMINER, pour la localisation de fautes. Nous avons montré qu'à l'aide de la contrainte globale CLOSEDPATTERN, assurant un bon niveau de filtrage des motifs non fréquents ou non fermés, la localisation de fautes est plus efficace en termes de résolution.

Par la suite, nous avons comparé expérimentalement notre approche implémentée dans l'outil F-CPMINER avec les approches standards TARANTULA, OCHIAI et JACCARD sur la base de programmes *Siemens*. Les résultats obtenus ont montré, que notre approche

permet dans un bon nombre de cas, de proposer une localisation plus précise. Nous avons également montré, que sur un programme à fautes multiples, F-CPminer permet de localiser plus rapidement l'ensemble des fautes. L'objectif de cette dernière étude étant de suivre le comportement de notre approche.

Quatrième partie .

Conclusions et perspectives

7. Conclusions et perspectives

7.1. Conclusions

L'ensemble des travaux réalisés tout le long de cette thèse, ont ciblé deux objectifs majeurs :

1)- Proposer un modèle de fouille déclarative efficace, pour le problème de l'extraction de motifs fréquents fermés. La motivation était double : (i) Le modèle existant basé sur la programmation par contraintes [De Raedt et al., 2008, Guns, 2012], possède un inconvénient majeur lié au nombre de contraintes de réification et de variables impliquées dans l'encodage du modèle. (ii) L'extraction de motifs fréquents fermés étant au centre de l'approche applicative F-CPMiner que nous proposons, pour la localisation de fautes, où l'objectif est d'extraite les top-k motifs suspects.

2)- Proposer un nouveau cadre pour la localisation de fautes, en faisant le parallèle entre le problème de localisation et la fouille de motifs ensemblistes. Une approche qui tient compte des dépendances entre les instructions d'un programme sous test. Ce cadre étant établi à l'aide d'une modélisation PPC, il a aussi pour objectif une évolutivité, en prenant en considération les différentes propriétés, que peut soulever un programme sous test.

Contrainte globale pour l'extraction de motifs fréquents fermés

Nous avons proposé dans le chapitre 4, une contrainte globale, pour l'extraction de motifs fréquents fermés. La contrainte ClosedPattern capture la sémantique particulière de ce problème, à savoir la fréquence minimale et la fermeture des motifs. Pour propager efficacement cette contrainte, nous avons dans un premier temps, proposé trois règles de filtrage pour assurer la consistance de domaine lors du processus de recherche des solutions. Par la suite, nous avons défini un algorithme de filtrage, qui maintient la consistance de domaine, avec une complexité cubique en temps et quadratique en espace. Nous avons montré, que la contrainte ClosedPattern offre un apport pratique sur les bases de grande taille, ce qui est un enjeu majeur pour la communauté de fouille de données, orientée motifs. Enfin, cette contrainte globale a été utilisée dans l'approche de localisation de fautes, que nous avons proposée dans le chapitre 5. Grâce au bon niveau de filtrage des motifs non-fréquents ou non-fermés qu'assure la contrainte, l'extraction des motifs dits *suspects* est plus efficace en termes de temps de réponse.

Localisation de fautes par l'extraction de motifs sous contraintes

Nous avons proposé dans le chapitre 5, une nouvelle approche, basée sur l'extraction de motifs ensemblistes et la programmation par contraintes, pour traiter le problème de localisation de fautes. En premier, nous avons formellement défini le problème de localisation de fautes, comme une problématique de fouille en faisant appel à la PPC, pour modéliser et résoudre les contraintes, tenant compte de la nature bien particulière des motifs recherchés. La résolution du modèle à contraintes, nous permet d'obtenir une première localisation avec les top-k suites d'instructions suspectes. La seconde étape de l'approche, a pour objectif de raffiner le résultat, en classant de façon plus fine, l'ensemble des instructions des top-k ; en tirant profit de deux observations majeures : i) Les fautes introduites dans un programme peuvent être vues comme un motif (ensemble d'instructions), qui est plus fréquent dans les exécutions non-réussies que dans celles réussies ; ii) La différence entre un motif pertinent et un autre qui l'est moins, est un sous-ensemble d'instructions qui apparaissent/disparaissent d'un motif à un autre ; cette différence nous aide à mieux repérer la position de la faute dans le programme. Nous avons montré, comment ces deux propriétés peuvent être exploitées dans un algorithme de classement ad-hoc, pour augmenter la précision de la localisation. Finalement, nous avons évalué expérimentalement notre approche dans le chapitre 6, implémentée dans l'outil F-CPMiner. Les résultats obtenus ont montré, que notre approche permet de proposer une localisation précise.

7.2. Perspectives

Les perspectives de recherche s'inscrivent dans la continuité des travaux menés, que ce soit pour l'extraction de motifs ou la localisation de fautes.

1- Fouille déclarative de motifs :

Améliorer les performances de la contrainte globale :

– La complexité de l'algorithme de filtrage de la contrainte globale ClosedPattern est cubique $O(n^2 \times m)$ en temps, où n est le nombre d'items et m est le nombre de transactions. Cette complexité est due à la troisième règle, qui calcule l'intersection entre deux couvertures. Il serait intéressant d'essayer de réduire cette complexité, en exploitant des structures de données paresseuses/intelligentes, ou exploiter une autre représentation de la base de transactions, telle que la représentation hybride ou la représentation arborescente [Borgelt, 2012], pour réduire la complexité de ce calcul.

Étendre la contrainte globale à d'autres types de motifs :

– La contrainte globale ClosedPattern a été mise en œuvre pour l'extraction de motifs fréquents fermés. Nous souhaitons étendre son raisonnement à d'autres types

de motifs, tels que les motifs maximaux. Il serait intéressant de mettre en œuvre d'autres règles de filtrage pour cibler les motifs maximaux. En effet, nous savons que les motifs maximaux forment une classe particulière (sous-ensemble) des motifs fermés. Cela permet d'avoir une idée sur les propriétés à exploiter en termes de filtrage, pour extraire ces motifs. Au niveau conceptuel, l'idée est de produire une contrainte globale générique paramétrable, permettant l'extraction de motifs fréquents/fermés/maximaux.

– Dans un travail récent [Kemmar et al., 2015], les auteurs ont proposé une nouvelle contrainte globale nommée PrefixProjection, pour l'extraction de séquences fréquentes. Il serait fort intéressant, de voir si la notion de fermeture, traitée dans la contrainte globale ClosedPattern, permet de produire des règles de filtrage efficaces pour l'extraction de séquences fermées.

2- Localisation de fautes par la fouille déclarative :

– Dans nos travaux nous avons tiré profit des motifs fermés, pour construire les top-k motifs suspects. Par ailleurs, d'autres classes de motifs tels que, les motifs maximaux et les motifs libres, peuvent offrir d'autres informations intéressantes pour la localisation. L'idée est de penser un nouvel algorithme ad-hoc, pour raisonner sur ces motifs qui sont de formes/contenus différents, par rapport aux motifs fermés.

– La localisation de fautes par le biais de motifs, ouvre une voie intéressante quant à la localisation des fautes multiples. En effet, la présence de plusieurs instructions fautives peuvent être capturées par un seul motif qui regroupe plusieurs instructions. Évitant ainsi la stratégie *one-bug-at-time* (adopté dans notre première étude à la section 6.8) et qui nécessite une série de re-exécutions du programme.

– Une piste intéressante pour améliorer le cadre de localisation proposé, est d'explorer d'autres observations sur les programmes avec fautes, pour enrichir le modèle avec plus de contraintes. Ces contraintes peuvent porter sur uniquement, une partie des instructions. Une autre solution envisageable, serait de calculer les sky-patterns *suspicieux* [Soulet et al., 2011], au lieu des top-k motifs suspects, en faisant appel notamment à la dominance au sens de Pareto, pour le calcul des motifs dominants et exploiter des mesures/critères, qui favorisent une meilleure qualité de la localisation.

– Une suite qui nous semble prometteuse pour F-CPminer, est de produire une localisation dite *adaptative*. Sachant que F-CPminer retourne un classement des instructions, contenant potentiellement des plages d'instructions équivalentes, en termes de suspicion. L'idée est d'utiliser ces groupes d'instructions, pour produire des cas de test, ciblant uniquement l'exécution d'un sous ensemble d'instructions de cette plage, permettant ainsi, de faire ressortir une partie particulière de ces instructions. Idéalement, si de tels cas de test sont faisables, le but serait de retourner un classement distinct de chaque instruction.

Table des figures

Liste des tableaux

Liste des algorithmes

Bibliographie

[Abreu et al., 2007] Abreu, R., Zoeteweij, P., and van Gemund, A. J. C. (2007). On the accuracy of spectrum-based fault localization. In *Testing : Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007*, pages 89–98. 11, 47, 49, 50, 75, 91

[Agrawal et al., 1993] Agrawal, H., DeMillo, R. A., and Spafford, E. H. (1993). Debugging with dynamic slicing and backtracking. *Softw., Pract. Exper.*, 23(6) :589–616. 53

[Agrawal et al., 1995] Agrawal, H., Horgan, J. R., London, S., and Wong, W. E. (1995). Fault localization using execution slices and dataflow tests. In *Sixth International Symposium on Software Reliability Engineering, ISSRE 1995, Toulouse, France, October 24-27, 1995*, pages 143–151. 48

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. 20

[Apt, 2003] Apt, K. (2003). *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA. 36

[Avizienis et al., 2004] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1) :11–33. 45

[Beeri et al., 1983] Beeri, C., Fagin, R., Maier, D., and Yannakakis, M. (1983). On the desirability of acyclic database schemes. *J. ACM*, 30(3) :479–513. 37

[Beldiceanu et al., 2007] Beldiceanu, N., Carlsson, M., Demassey, S., and Petit, T. (2007). Global constraint catalogue : Past, present and future. *Constraints*, 12(1) :21–62. 38

[Bessiere and Cordier, 1993] Bessiere, C. and Cordier, M. (1993). Arc-consistency and arc-consistency again. In *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, DC, USA, July 11-15, 1993.*, pages 108–113. 34

[Bessiere and Régin, 1996] Bessiere, C. and Régin, J. (1996). MAC and combined heuristics : Two reasons to forsake FC (and cbj ?) on hard problems. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996*, pages 61–75. 36

[Bessiere and Régin, 2001] Bessiere, C. and Régin, J. (2001). Refining the basic constraint propagation algorithm. In *Proceedings of the Seventeenth International*

Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001, pages 309–315. 34, 37

[Borgelt, 2012] Borgelt, C. (2012). Frequent item set mining. *Wiley Interdisc. Rew. : Data Mining and Knowledge Discovery*, 2(6) :437–456. 20, 21, 109

[Burdick et al., 2005] Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., and Yiu, T. (2005). MAFIA : A maximal frequent itemset algorithm. *IEEE Trans. Knowl. Data Eng.*, 17(11) :1490–1504. 21

[Cellier et al., 2008] Cellier, P., Ducassé, M., Ferré, S., and Ridoux, O. (2008). Formal concept analysis enhances fault localization in software. In *Formal Concept Analysis*, pages 273–288. Springer. 53, 75

[Cellier et al., 2009] Cellier, P., Ducassé, M., Ferré, S., and Ridoux, O. (2009). Dellis : A data mining process for fault localization. In *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*, pages 432–437. 10, 11, 53

[Clark, 1977] Clark, K. L. (1977). Negation as failure. In *Logic and Data Bases*, pages 293–322. 37

[Cleve and Zeller, 2005] Cleve, H. and Zeller, A. (2005). Locating causes of program failures. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 342–351. 52, 75

[Cohen and Jeavons, 2016] Cohen, D. A. and Jeavons, P. G. (2016). The power of propagation : when gac is enough. *Constraints*, pages 1–21. 38

[Cousot and Cousot, 1977] Cousot, P. and Cousot, R. (1977). Automatic synthesis of optimal invariant assertions : Mathematical foundations. *SIGART Newsletter*, 64 :1–12. 44

[Crémilleux and Soulet, 2008] Crémilleux, B. and Soulet, A. (2008). Discovering knowledge from local patterns with global constraints. In *ICCSA (2)*, pages 1242–1257. 25

[De Raedt et al., 2008] De Raedt, L., Guns, T., and Nijssen, S. (2008). Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–212. ACM. 9, 10, 11, 20, 40, 58, 108

[De Raedt and Zimmermann, 2007] De Raedt, L. and Zimmermann, A. (2007). Constraint-based pattern set mining. In *Proceedings of the Seventh SIAM International Conference on Data Mining*, Minneapolis, Minnesota, USA. SIAM. 25

[Debruyne and Bessiere, 2001] Debruyne, R. and Bessiere, C. (2001). Domain filtering consistencies. *J. Artif. Intell. Res. (JAIR)*, 14 :205–230. 34

[Dechter, 1990] Dechter, R. (1990). Enhancement schemes for constraint processing : Backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3) :273–312. 36

[DeMillo et al., 1978] DeMillo, R. A., Lipton, R. J., and Sayward, F. G. (1978). Hints on test data selection : Help for the practicing programmer. *IEEE Computer*, 11(4) :34–41. 47

[Denmat et al., 2005] Denmat, T., Ducassé, M., and Ridoux, O. (2005). Data mining and cross-checking of execution traces : a re-interpretation of jones, harrold and stasko test information. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*, pages 396–399. 53

[Dong and Li, 1999] Dong, G. and Li, J. (1999). Efficient mining of emerging patterns : Discovering trends and differences. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*, pages 43–52. 19

[Eric Wong et al., 2010] Eric Wong, W., Debroy, V., and Choi, B. (2010). A family of code coverage-based heuristics for effective fault localization. *Journal of Systems and Software*, 83(2) :188–208. 10, 47, 54, 80

[Geng and Hamilton, 2006] Geng, L. and Hamilton, H. J. (2006). Interestingness measures for data mining : A survey. *ACM Comput. Surv.*, 38(3). 17

[Golomb and Baumert, 1965a] Golomb, S. W. and Baumert, L. D. (1965a). Backtrack programming. *J. ACM*, 12(4) :516–524. 35

[Golomb and Baumert, 1965b] Golomb, S. W. and Baumert, L. D. (1965b). Backtrack programming. *J. ACM*, 12(4) :516–524. 36

[Gonzalez-Sanchez et al., 2011] Gonzalez-Sanchez, A., Abreu, R., Gross, H.-G., and van Gemund, A. J. (2011). Prioritizing tests for fault localization through ambiguity group reduction. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, pages 83–92. IEEE. 86

[Grahne and Zhu, 2005] Grahne, G. and Zhu, J. (2005). Fast algorithms for frequent itemset mining using FP-Trees. *IEEE Trans. Knowl. Data Eng.*, 17(10) :1347–1362. 67

[Guns, 2012] Guns, T. (2012). Declarative pattern mining using constraint programming. 10, 11, 20, 40, 108

[Guns et al., 2011] Guns, T., Nijssen, S., and De Raedt, L. (2011). Itemset mining : A constraint programming perspective. *Artificial Intelligence*, 175(12) :1951–1983. 9, 40, 58, 75

[Guns et al., 2013] Guns, T., Nijssen, S., and Raedt, L. D. (2013). k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25(2) :402–418. 70

[Han et al., 2000] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 1–12. 21

[Hentenryck and Carillon, 1988] Hentenryck, P. V. and Carillon, J. (1988). Generality versus specificity : An experience with AI and OR techniques. In *Proceedings of the 7th National Conference on Artificial Intelligence. St. Paul, MN, August 21-26, 1988.*, pages 660–664. 38

[Hutchins et al., 1994] Hutchins, M., Foster, H., Goradia, T., and Ostrand, T. (1994). Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria. In *Proceedings of the 16th international conference on Software engineering*, pages 191–200. IEEE Computer Society Press. 92

[Jones, 2008] Jones, J. A. (2008). Semi-automatic fault localization. 49, 51

[Jones et al., 2007] Jones, J. A., Bowring, J. F., and Harrold, M. J. (2007). Debugging in parallel. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ISSTA '07, pages 16–26, New York, NY, USA. ACM. 47

[Jones and Harrold, 2005] Jones, J. A. and Harrold, M. J. (2005). Empirical evaluation of the tarantula automatic fault-localization technique. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*, pages 273–282. 10, 11, 47, 49, 75, 80, 91

[Jones et al., 2002] Jones, J. A., Harrold, M. J., and Stasko, J. T. (2002). Visualization of test information to assist fault localization. In *Proceedings of the 22rd International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, pages 467–477. 47

[Jr., 1998] Jr., R. J. B. (1998). Efficiently mining long patterns from databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 85–93. 26, 28

[Kemmar et al., 2015] Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., and Charnois, T. (2015). PREFIX-PROJECTION global constraint for sequential pattern mining. In *CP 2015*, volume 9255 of *LNCS*, pages 226–243. Springer. 9, 11, 58, 110

[Khiari et al., 2010] Khiari, M., Boizumault, P., and Crémilleux, B. (2010). Constraint programming for mining n-ary patterns. In *CP'10*, volume 6308 of *LNCS*, pages 552–567. Springer. 9, 25, 40, 70, 75

[Laprie et al., 1992] Laprie, J. C., Avizienis, A., and Kopetz, H., editors (1992). *Dependability : Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. 45

[Lazaar et al., 2016] Lazaar, N., Lebbah, Y., Loudni, S., Maamar, M., Lemiere, V., Bessiere, C., and Boizumault, P. (2016). A global constraint for closed itemset mining. In *CP'16*, LNCS (Forthcomming). Springer. 12, 38, 75

[Lhomme, 1993] Lhomme, O. (1993). Consistency techniques for numeric csps. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 232–238. 34

[Li et al., 2000] Li, J., Dong, G., and Ramamohanarao, K. (2000). Making use of the most expressive jumping emerging patterns for classification. In *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PADKK 2000, Kyoto, Japan, April 18-20, 2000, Proceedings*, pages 220–232. 19

[Liu and Han, 2006] Liu, C. and Han, J. (2006). Failure proximity : A fault localization-based approach. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 46–56, New York, NY, USA. ACM. 47

[Maamar et al., 2016a] Maamar, M., Aribi, N., Lazaar, N., Loudni, S., and Lebbah, Y. (2016a). F-cpminer* : A new approach based itemset mining and constraint programming for fault localization. In *CP meets Verification 2016 Workshop*. 12, 91

[Maamar et al., 2015] Maamar, M., Lazaar, N., Loudni, S., and Lebbah, Y. (2015). Localisation de fautes à l'aide de la fouille de données sous contraintes. Colloque sur l'Optimisation et les Systèmes d'Information COSI. 12, 91

[Maamar et al., 2016b] Maamar, M., Lazaar, N., Loudni, S., and Lebbah, Y. (2016b). F-cpminer : Une approche pour la localisation de fautes basée sur l'extraction de motifs ensemblistes sous contraintes. In *Actes des Douzièmes Journées Francophones de Programmation par Contraintes, JFPC 2016, Montpellier, France*. 12

[Maamar et al., 2016c] Maamar, M., Lazaar, N., Loudni, S., and Lebbah, Y. (2016c). Fault localization using itemset mining under constraints. *Automated Software Engineering*, pages 1–28. 12, 58, 91

[Mackworth, 1977] Mackworth, A. K. (1977). Consistency in networks of relations. *Artif. Intell.*, 8(1) :99–118. 34, 37

[Mannila and Toivonen, 1997] Mannila, H. and Toivonen, H. (1997). Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3) :241–258. 17

[Mohr and Henderson, 1986] Mohr, R. and Henderson, T. C. (1986). Arc and path consistency revisited. *Artif. Intell.*, 28(2) :225–233. 34

[Montanari, 1974] Montanari, U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. *Inf. Sci.*, 7 :95–132. 30

[Nessa et al., 2008] Nessa, S., Abedin, M., Wong, W. E., Khan, L., and Qi, Y. (2008). Software fault localization using n-gram analysis. In *Wireless Algorithms, Systems, and Applications*, pages 548–559. Springer. 10, 11, 47, 53, 75, 80

[Novak et al., 2009] Novak, P. K., Lavrac, N., and Webb, G. I. (2009). Supervised descriptive rule discovery : A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10 :377–403. 19

[Parnin and Orso, 2011] Parnin, C. and Orso, A. (2011). Are automated debugging techniques actually helping programmers ? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, pages 199–209, New York, NY, USA. ACM. 51

[Pasquier et al., 1999a] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999a). Discovering frequent closed itemsets for association rules. In *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, pages 398–416. 9, 26

[Pasquier et al., 1999b] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999b). Efficient mining of association rules using closed itemset lattices. *Inf. Syst.*, 24(1) :25–46. 21, 26, 28, 39

[Pei et al., 2000] Pei, J., Han, J., and Mao, R. (2000). CLOSET : an efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30. 26, 58

[Pesant, 2004] Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 482–495. 38

[Podgurski et al., 2003] Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Sun, J., and Wang, B. (2003). Automated support for classifying software failure reports. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 465–475, Washington, DC, USA. IEEE Computer Society. 47

[Queille and Sifakis, 1982] Queille, J. and Sifakis, J. (1982). Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, pages 337–351. 44

[Régin, 1994] Régin, J.-C. (1994). A filtering algorithm for constraints of difference in csps. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pages 362–367, Menlo Park, CA, USA. American Association for Artificial Intelligence. 38

[Renieres and Reiss, 2003] Renieres, M. and Reiss, S. P. (2003). Fault localization with nearest neighbor queries. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 30–39. IEEE. 10, 53, 75, 80

[Richardson and Thompson, 1993] Richardson, D. J. and Thompson, M. C. (1993). An analysis of test data selection criteria using the RELAY model of fault detection. *IEEE Trans. Software Eng.*, 19(6) :533–553. 51

[Rojas et al., 2014] Rojas, W. U., Boizumault, P., Loudni, S., Crémilleux, B., and Lepailleur, A. (2014). Mining (soft-) skypatterns using dynamic CSP. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*, volume 8451 of *Lecture Notes in Computer Science*, pages 71–87. Springer. 25, 82

[Rossi et al., 2006] Rossi, F., Beek, P. v., and Walsh, T. (2006). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA. 30, 31, 32

[Soulet et al., 2011] Soulet, A., Raïssi, C., Plantevit, M., and Crémilleux, B. (2011). Mining dominant patterns in the sky. In *11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011*, pages 655–664. 110

[Ugarte et al., 2015] Ugarte, W., Boizumault, P., Loudni, S., and Crémilleux, B. (2015). Modeling and mining optimal patterns using dynamic CSP. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 33–40. 9, 40

[Uno et al., 2003] Uno, T., Asai, T., Uchida, Y., and Arimura, H. (2003). LCM : an efficient algorithm for enumerating frequent closed item sets. In *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA*. 9, 26, 58

[Uno et al., 2004a] Uno, T., Asai, T., Uchida, Y., and Arimura, H. (2004a). An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science, 7th International Conference, DS 2004, Padova, Italy, October 2-5, 2004, Proceedings*, pages 16–31. 26, 27, 39

[Uno et al., 2004b] Uno, T., Kiyomi, M., and Arimura, H. (2004b). LCM ver. 2 : Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*. 21, 26

[Uno et al., 2005] Uno, T., Kiyomi, M., and Arimura, H. (2005). Lcm ver. 3 : Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining : frequent pattern mining implementations*, pages 77–86. ACM. 26

[Verfaillie and Jussien, 2005] Verfaillie, G. and Jussien, N. (2005). Constraint solving in uncertain and dynamic environments : A survey. *Constraints*, 10(3) :253–281. 38

[Vessey, 1985] Vessey, I. (1985). Expertise in debugging computer programs : A process analysis. *International Journal of Man-Machine Studies*, 23(5) :459–494. 45

[Waltz, 1972] Waltz, D. L. (1972). Generating semantic descriptions from drawings of scenes with shadows. Technical report, Cambridge, MA, USA. 34

[Wang et al., 2003] Wang, J., Han, J., and Pei, J. (2003). CLOSET+ : searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003*, pages 236–245. 23, 60

[Wong and Debroy, 2009] Wong, W. E. and Debroy, V. (2009). A survey of software fault localization. *Department of Computer Science, University of Texas at Dallas, Tech. Rep. UTDCS-45-09*. 48, 54, 93

[Yoo, 2012] Yoo, S. (2012). Evolving human competitive spectra-based fault localisation techniques. In *Search Based Software Engineering - 4th International Symposium, SSBSE 2012, Riva del Garda, Italy, September 28-30, 2012. Proceedings*, pages 244–258. 11, 75

[Zaki and Hsiao, 2002] Zaki, M. J. and Hsiao, C. (2002). CHARM : an efficient algorithm for closed itemset mining. In *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*, pages 457–473. 26, 39, 58, 67

[Zaki et al., 1997] Zaki, M. J., Parthasarathy, S., Ogihara, M., and Li, W. (1997). New algorithms for fast discovery of association rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997*, pages 283–286. 9, 20

[Zeller, 2002] Zeller, A. (2002). Isolating cause-effect chains from computer programs. In *Proceedings of the Tenth ACM SIGSOFT Symposium on Foundations of Software Engineering 2002, Charleston, South Carolina, USA, November 18-22, 2002*, pages 1–10. ACM. 52