Réconciliation fouille de données et Programmation Logique Inductive

Étude et utilisation du logiciel de fouille de données **FACE** (FT R&D) pour l'apprentissage de séquences temporelles

8 février 2005

Mémoire de DEA - Équipe DREAM de l'Irisa

Auteur :
Alexandre VAUTIER

Encadreurs :
Marie-Odile CORDIER
René QUINIOU

Remerciements

Je remercie vivement mes deux encadreurs, Marie-Odile Cordier et René Quiniou, qui m'ont guidé et aidé dans mon travail. J'ai particulièrement apprécié leur rigueur et leur recul, ainsi que leurs conseils qui m'ont été précieux.

Je tiens à remercier Hélène Verrière ainsi que Christophe Dousson, du centre de recherche de France-télécom à Lannion, pour leurs réponses au sujet de l'implémentation du logiciel FACE.

Merci enfin à tous les membres de l'équipe DREAM pour leur accueil et leur bonne humeur.

TABLE DES MATIÈRES

Liste des définitions

Liste des propositions et théorèmes

Introduction

L'intelligence artificielle a donné naissance à de nombreux axes de recherche. L'un des plus importants est l'apprentissage artificiel car tout système dit « intelligent » résout quelque part un problème d'apprentissage. Un tel système doit induire des concepts généralisant au mieux un ensemble d'expériences.

Un problème d'apprentissage dépend, d'une part, des connaissances d'un expert sur le domaine à étudier et, d'autre part, de la quantité d'expériences disponibles. On peut donc distinguer deux types d'apprentissage. Le premier, l'apprentissage supervisé, nécessite une quantité limitée de données réparties en classes par un expert du domaine dans le but de trouver des règles discriminant au mieux les différentes classes. Le second, l'apprentissage non supervisé, consiste, à partir d'une grande quantité de données non classées, à découvrir des connaissances cachées.

La programmation logique inductive (PLI) est une technique d'apprentissage supervisé permettant d'apprendre des concepts sous forme de clauses logiques, à partir d'exemples de ces concepts et de connaissances a priori sur ce que l'on souhaite induire. La PLI est particulièrement adaptée à l'apprentissage de concepts à partir de données fortement structurées, par exemple temporellement ou spatialement. Ainsi, on verra un exemple d'apprentissage de chroniques (une chronique est un ensemble d'évènements contraints par des relations temporelles [DG94]) dans le domaine des arythmies cardiaques.

La fouille de données, quant à elle, vise à exploiter de grandes quantités de données pour en extraire des connaissances. En ne disposant, a priori, d'aucune information particulière sur les données, on souhaite découvrir des connaissances cachées sous forme de motifs. Dans ce cadre, on appelle motif un concept (une loi, une règle, une classe) synthétisant un ensemble de données.

Il manquait à la fouille de données une véritable formalisation. Une voie de recherche s'est donc ouverte, à l'initiative de Imielinski et Manilla [IM96], et a donné naissance aux bases de données inductives. Celles-ci intègrent les bases de données à la fouille de données. Les données et leurs motifs sont alors gérés de la même façon. Ainsi, le processus de fouille de données devient un processus interactif dans lequel l'utilisateur produit des requêtes à la base de données inductive qui lui fournit en résultat des motifs qui sont soit entreposés dans la base de données inductive, soit « induits » par un processus basé sur la recherche de motifs intéressants, par exemple fréquents.

Les bases de données inductives donnent une nouvelle vision de la fouille de données. Ainsi, de nombreux points communs apparaissent entre cette technique et la programmation logique

inductive pourtant issue de problématiques différentes. Par conséquent, nous proposons d'en voir un réel rapprochement par la mise en œuvre d'une base de données inductive dont les motifs intègrent une notion temporelle et sont représentés par des chroniques.

L'introduction de la notion de temps dans les connaissances extraites d'une base de données inductive exige son extension. Cet enrichissement rend la fouille de données beaucoup plus complexe : il faut extraire des motifs beaucoup plus riches mais toujours de manière efficace. C'est pourquoi, on propose d'intégrer au cœur de la base de données inductive étendue un outil de fouille de données, FACE, capable d'extraire les chroniques fréquentes dans un journal d'alarmes de réseau de télécommunications [DD99]. Toutefois, son intégration nécessite son adaptation afin qu'il puisse répondre à des requêtes portant sur la notion de fréquence de chronique.

La méthode est appliquée à la découverte de chroniques caractéristiques de pathologies cardiaques à partir d'électrocardiogrammes. Le travail a déjà été réalisé par des méthodes de PLI [CCQW03]. Une validation de l'approche utilisant les bases de données inductives et notre extension peut ainsi être réalisée.

Ce mémoire est organisé en deux parties, la première est un état de l'art du sujet. Dans un premier temps, elle présente l'apprentissage supervisé qui s'appuie sur la théorie de l'espace des versions et introduit la PLI. Ensuite, la fouille de données est exposée et plus particulièrement l'intégration de la notion de temps en fouille de données. Enfin les bases de données inductives sont présentées.

La deuxième partie détaille le travail effectué pendant le stage. Après avoir étendu la notion de base de données inductive au temps, une description détaillée de FACE permet d'en voir une utilisation et une amélioration pour la découverte de chroniques fréquentes. Enfin, les premiers résultats sont présentés et de nouvelles perspectives de recherche sont exposées en conclusion.

Première partie

État de l'art

CHAPITRE

1

Apprentissage supervisé

L'apprentissage supervisé a pour but d'induire des modèles qui expliquent un ensemble d'exemples classés. Après avoir expliqué la problématique de l'apprentissage, nous allons voir le lien reliant les exemples et leurs modèles grâce à l'introduction de diverses relations et leur utilisation pour définir l'espace des versions.

Ceci nous mènera à la programmation logique inductive. Celle-ci induit des concepts sous forme de clauses à partir d'exemples positifs et négatifs et de connaissances sur le domaine.

1.1 L'apprentissage

L'apprentissage, défini par T.Mitchell (1997), correspond au fait qu'un programme informatique *apprend* la tâche $\mathcal{T}$ à partir de l'expérience $\mathcal{E}$ et de la mesure de performance $\mathcal{P}$, si sa capacité à exécuter la tâche $\mathcal{T}$, mesurée par $\mathcal{P}$, augmente avec $\mathcal{E}$.

La définition de ces trois éléments ($\mathcal{E}$, $\mathcal{T}$, $\mathcal{P}$) conditionne donc complètement l'apprentissage à effectuer.

L'apprentissage, en pratique, a pour but de construire et d'expliquer au travers de modèles, l'ensemble des expériences (appelées exemples dans la suite). Ce processus d'induction (défini dans la suite) doit donc manipuler deux classes d'objets distincts : l'espace des exemples E et l'espace des modèles (ou hypothèses) $\mathcal{H}$. En apprentissage supervisé, on considère souvent le cas d'un ensemble $\mathcal{E}$ d'exemples classés en deux ensembles : les exemples positifs E^+ et les exemples négatifs E^-.

L'**induction** est le processus essentiel de l'apprentissage, elle peut être vu comme l'inverse de la déduction. Ce processus consiste à inférer, à partir de faits (les exemples), les règles (les hypothèses) expliquant ces faits. Par exemple, si on a les faits *"Socrate est mortel"* et *"Socrate est un homme"* alors on dit que l'on infère la règle *"Tous les hommes sont mortels"* par induction.

Une manière de réaliser l'induction, en apprentissage supervisé, consiste à partir de l'ensemble $\mathcal{E}$, d'explorer l'espace $\mathcal{H}$ des hypothèses pour rechercher celles qui *couvrent* (voir 1.2.1) au mieux les exemples positifs et ne couvrent pas les exemples négatifs.

1.2 L'espace des versions

Un des concepts importants en apprentissage est celui de l'espace des versions. C'est pourquoi, nous commençons par sa présentation. L'espace des versions, introduit en 1978 par Tom Mitchell [Mit97], est l'ensemble de toutes les hypothèses cohérentes avec les données d'apprentissage (les exemples). On va voir, dans la suite, que cet espace a une structure de treillis.

1.2.1 Couverture des hypothèses

Pour faire le lien entre l'espace des hypothèses $\mathcal{H}$ et l'espace des exemples E, on définit une relation de *couverture*. Cette relation $couverture : \mathcal{H} \rightarrow 2^E$ indique l'adéquation d'une hypothèse à un ensemble d'exemples.

1.2.2 Relation de généralité

Pour se guider plus facilement dans l'exploration de $\mathcal{H}$, on dote cet espace d'une relation de généralité permettant de comparer deux hypothèses entre elles. Cette relation $\preceq$: $\mathcal{H} \rightarrow \mathcal{H}$ ($h_1 \preceq h_2 \Leftrightarrow h_2 \succeq h_1$), doit être compatible avec la relation de couverture. On a donc

$$h_a \preceq h_b \Leftrightarrow couverture(h_a) \subseteq couverture(h_b)$$

On dit que l'hypothèse h_b est plus générale que l'hypothèse h_a ou encore que l'hypothèse h_a est plus spécifique que l'hypothèse h_b. Sur l'exemple de la figure 1.1, chaque hypothèse (représentée par un triangle) couvre un ensemble d'exemples.

1.2.3 Hypothèse cohérente

On dit qu'une hypothèse est cohérente si elle est complète et correcte, ce qui correspond respectivement au fait que tous les exemples positifs (E^+) sont couverts et au fait qu'aucun exemple négatif (E^-) n'est couvert.

1.2.4 Treillis des hypothèses

La relation de généralité entre les hypothèses est seulement partielle. On peut montrer facilement que cette relation définit un ordre partiel qui induit une structure de treillis sur $\mathcal{H}$. Pour tout couple d'hypothèses il existe donc une hypothèse plus générale (resp. spécifique) qu'il est impossible de spécialiser (resp. généraliser) davantage sans perdre cette propriété de généralité (resp. de spécialité).

Par généralisation à plus de deux hypothèses, pour tout ensemble C d'hypothèses, il existe deux ensembles appelés le spécialisé maximalement général noté $gms(C)$ et le généralisé maximalement spécifique noté $smg(C)$.

1.2.5 Représentation de l'espace des versions par ses bornes

Deux propriétés de l'ensemble de l'espace des versions sont nécessaires pour le représenter par ses bornes : la convexité et l'existence de bornes à cet ensemble.

Convexité : On dit qu'un ensemble C est convexe si et seulement si

$$\forall h_1, h_2, h_3 (h_1, h_3 \in C \wedge h_1 \preceq h_2 \preceq h_3) \Rightarrow h_2 \in C$$

Le théorème de convexité de H. Hirsh cité dans [CM02] indique que l'espace $\mathcal{H}$ des hypothèses est convexe.

Ensemble borné : Un ensemble C d'hypothèses est borné si pour tout $h \in C$ il existe une hypothèse g maximalement générale dans C et une hypothèse s maximalement spécifique dans C telles que $s \preceq h \preceq g$.

Il n'est pas possible de garantir cette propriété pour tout ensemble $\mathcal{H}$ décrit dans un langage quelconque. Il faut donc vérifier cette propriété pour chaque langage.

Si l'ensemble $C \subset \mathcal{H}$ des hypothèses cohérentes est convexe et borné, alors il peut être représenté par sa borne inférieure S et sa borne supérieure G : S (resp. G) est l'ensemble des hypothèses cohérentes telles qu'il n'est pas possible de les spécialiser (resp. généraliser) sans perdre leur cohérence avec les exemples.

L'espace des versions peut donc être représenté de manière économique par ses bornes S et G. De plus, un algorithme d'apprentissage peut opérer en calculant ces deux bornes, et donc en calculant l'ensemble des hypothèses cohérentes les plus spécifiques et les plus générales.

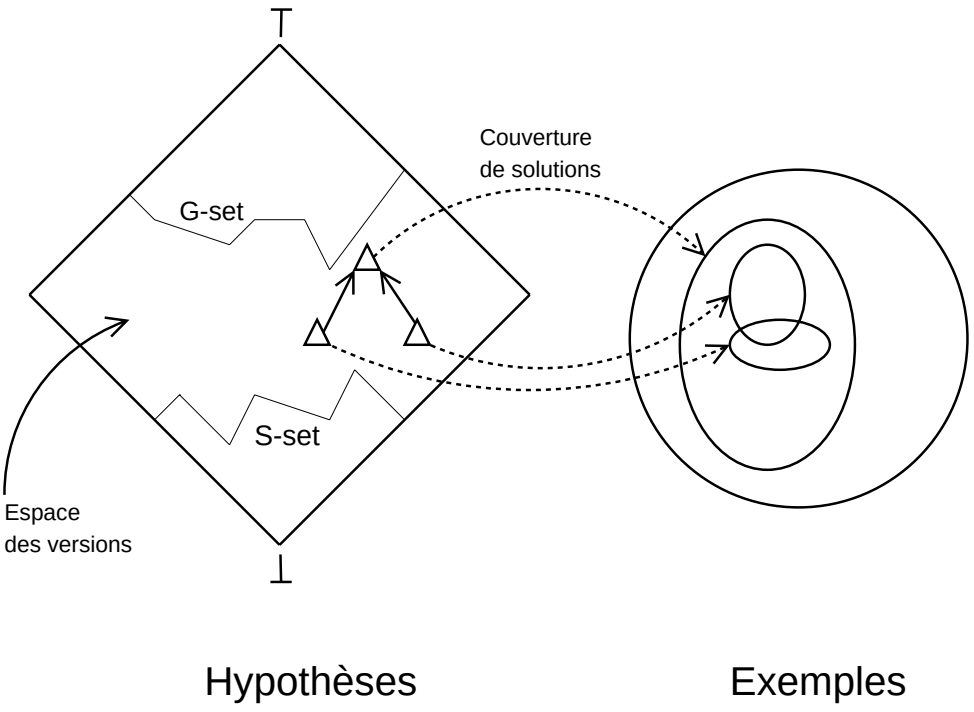


FIG. 1.1 – L'espace des versions représenté par ses bornes et son lien avec les exemples

Conclusion

L'ensemble des hypothèses est en général trop grand pour être manipulé tel quel, mais si on peut définir une relation de généralité entre les hypothèses, alors on peut définir deux ensembles S et G qui ont la propriété de définir implicitement l'ensemble des hypothèses cohérentes avec les exemples.

1.3 Programmation logique inductive

Le principe de la programmation logique inductive (PLI) est un peu plus spécifique qu'une technique d'apprentissage supervisé classique. Aux éléments définissant l'apprentissage supervisé : les exemples (E^+ et E^-), un espace des hypothèses formées dans un langage $\mathcal{L}_h$, et une relation de couverture entre hypothèses et exemples, s'ajoute un ensemble B de connaissances sur le domaine (*background knowledge*). Par ailleurs, le langage $\mathcal{L}_h$ des hypothèses est le langage des clauses qui est dérivé de la logique du premier ordre. Ceci implique que la relation de couverture définie théoriquement précédemment s'appuie sur des bases logiques en PLI.

Ainsi, le problème est de trouver une hypothèse (ou un ensemble d'hypothèses) $H \in \mathcal{L}_h$ cohérente avec les exemples. Dans le cadre de la PLI, H est soit une clause, soit un programme logique, c'est à dire un ensemble de clauses.

Nous verrons, dans un premier temps, la base de la PLI : le langage des clauses. Ensuite la recherche de solutions sera abordée en introduisant une relation de généralité permettant d'utiliser l'espace des versions. On a pu laisser penser en 1.2.2 que celle-ci était simple à définir mais nous verrons qu'il n'en est pas ainsi pour les clauses. De plus, nous introduirons des biais de langage permettant de réduire la taille de l'espace des hypothèses.

1.3.1 Le langage des clauses

En PLI, une hypothèse est un ensemble de clauses du premier ordre. Une clause s'écrit $A \Leftarrow B_1 \wedge ... \wedge B_m$ où A, $B_1 ... B_m$ sont des littéraux.

Les littéraux sont formés à partir de symboles de prédicats appliqués à des termes, eux-mêmes construits à partir de symboles primitifs (les variables, les connecteurs, les fonctions, et les parenthèses) en respectant une syntaxe stricte. Ce langage permet, par l'application de règles d'inférence, de réaliser des démonstrations dans ce système formel, c'est à dire de déduire des théorèmes à partir d'axiomes. La logique des prédicats n'est pas présentée ici mais on peut se reporter au chapitre sur la programmation logique de [CM02] pour plus de détails.

La définition de la relation de généralité sur les clauses fait appel à la notion de substitution :

Définition 1 (Substitution) *Une substitution est une liste finie de couples X_i/t_i, où X_i est une variable et t_i un terme. Si σ est la substitution $\{X_1/t_1, ...X_i/t_i, ...X_n/t_n\}$. L'ensemble des variables $\{X_1, ...X_n\}$ de σ est noté $dom(\sigma)$.*

Une substitution σ s'applique à une clause C en remplaçant chaque occurrence des variables de $dom(\sigma)$ par le terme correspondant, le résultat étant noté $C\sigma$.

1.3.2 Notion de généralité

La problématique de la PLI peut se ramener à une recherche de clauses dans un espace satisfaisant un certain nombre de propriétés. Malheureusement, l'expressivité du langage $\mathcal{L}_h$ employé rend cet espace des hypothèses trop grand. Heureusement, on a vu, en 1.2, que l'espace des versions s'appuyait sur une relation de généralité permettant une recherche plus intelligente.

Dans le cas des logiques dérivées du premier ordre, trouver un tel ordre de généralité est difficile. L'implication logique serait certainement la solution, mais des problèmes d'indécidabilité en empêche l'utilisation dans le cas général. La *θ-Subsomption*, un autre ordre très utilisé en PLI, est définie ci-dessous :

Définition 2 (θ-Subsomption) *Une clause C_1 θ-subsume une clause C_2 ($C_1 \succeq_\theta C_2$) si et seulement si il existe une substitution θ telle que $C_1\theta \subseteq C_2$ (en considérant les clauses comme des ensembles de littéraux).*

On peut noter que si C_1 *θ-subsume* C_2 alors C_1 implique C_2 mais la réciproque n'est pas toujours vraie. Par exemple, $p(Y_1, Y_2) \Leftarrow r(Y_2, Y_1)$ *θ-subsume* $p(a, b) \Leftarrow r(b, a)$ avec $\theta = \{Y_1/a, Y_2/b\}$

Ou encore, en logique,

▷ "Tous les hommes sont mortels" devient $mortel(X) \Leftarrow homme(X)$

▷ "Tout est mortel" devient $mortel(Y) \Leftarrow$

Ainsi on sait que "Tout est mortel" est plus général que "Tous les hommes sont mortels" car ($mortel(Y) \Leftarrow$) *θ-subsume* $mortel(X) \Leftarrow homme(X)$ avec $\theta = \{Y/X\}$

1.3.3 Biais d'apprentissage

En PLI, l'espace de recherche des hypothèses $\mathcal{H}$ est infini. Il est donc important d'imposer des contraintes sur $\mathcal{H}$ pour en réduire la taille et ainsi rendre possible son exploration. Ces restrictions sont appelées biais puisqu'elles vont, en effet, biaiser le résultat en privilégiant certaines hypothèses sur des considérations autres que leur simple adéquation avec les exemples.

Parmi les différents types de biais, les biais déclaratifs s'attachent à définir la forme attendue des hypothèses, c'est-à-dire à préciser leur langage. On distingue principalement deux grandes familles au sein des biais déclaratifs : les biais syntaxiques et les biais sémantiques.

Les biais syntaxiques permettent de définir l'ensemble des clauses envisageables en spécifiant explicitement leur syntaxe. On peut, par exemple, limiter le nombre de variables dans les clauses, ou encore limiter la profondeur de récursion dans les termes. DLAB décrit dans [DR96] est un langage permettant de définir des biais syntaxiques.

Les biais sémantiques permettent de préciser le rôle des littéraux dans une clause. On peut, par exemple, dans une clause, typer une variable qui ne sera utilisable que par des prédicats déclarés comme pouvant fonctionner avec ce type de variable. On peut aussi déclarer des modes. Ils permettent d'imposer des contraintes sur les entrées/sorties des littéraux dans les clauses. Par exemple, on peut imposer que les littéraux d'une clause utilisent en variables d'entrées les variables de sortie des littéraux précédents.

1.3.4 Recherche dans l'espace des solutions

Différentes stratégies sont utilisables pour parcourir l'espace des hypothèses bien formées. La mise en œuvre de ces stratégies est, bien sûr, intimement liée à la notion de généralité entre hypothèses et également au langage $\mathcal{L}_h$ précisé par le biais, c'est à dire la forme des hypothèses attendues. Nous allons succinctement décrire deux stratégies : descendante et ascendante.

Recherche descendante

La stratégie descendante (ou *top-down* en anglais) correspond à une exploration de l'espace des hypothèses de la plus générale à la plus spécifique. À chaque étape de l'exploration, on recherche une clause h couvrant un maximum d'exemples de E^+ et aucun exemple de E^-. Pour effectuer cette recherche, on part de l'hypothèse la plus générale du concept puis on la spécialise à l'aide d'un opérateur (dit de raffinement) qui propose toutes les clauses qui lui sont plus spécifiques (selon la notion de généralité retenue). Le choix des spécialisations effectivement retenues (raffinées à leur tour) se fait à l'aide d'heuristiques, souvent calculées à l'aide des taux de couverture de E^+ et E^-. À la fin, la clause h retenue couvre un maximum d'exemples de E^+ et aucun exemple de E^-. Il suffit, ensuite, d'enlever à E^+ les exemples couverts par h et de recommencer la recherche d'une nouvelle clause. L'algorithme s'arrête quand E^+ est vide, on obtient ainsi l'hypothèse H qui est un ensemble de clauses et qui couvre tous les exemples positifs et aucun exemple négatif.

De nombreuses implémentations utilisent cette approche : on peut citer le précurseur MIS de E. Shapiro (1981), LINUS de Lavrac et Dzeroski (1994), et le très célèbre PROGOL [Mug95] et FOIL de Quinlan et al. [QCJ93].

Recherche ascendante

La stratégie ascendante (*bottom-up*) repose sur une exploration des hypothèses des plus spécifiques aux plus générales. Le point de départ est donc un sous-ensemble des exemples positifs que l'on cherche à généraliser. Cette méthode utilise la notion de moindre généralisé, autrement dit, le plus petit pas inductif dans $\mathcal{H}$ pour explorer cet espace étape par étape. Comme pour la recherche descendante, c'est un opérateur qui a pour fonction de produire les généralisations d'une clause.

On peut citer GOLEM [MF90], un des logiciels pionniers de la PLI.

1.4 Apprentissage de séquences

Un des domaines d'application de l'apprentissage est l'apprentissage de séquences. Quand il s'agit de séquences d'évènements, le concept extrait doit refléter à la fois la nature des évènements et la manière dont ils s'enchaînent. Une séquence d'évènements peut ainsi être modélisée sous la forme d'une clause.

Par exemple, la clause $sequence() \Leftarrow Event(A, t_1) \wedge Event(B, t_2) \wedge (t_2 \geq t_1) \wedge (A <> B)$ reconnaît deux évènements différents et successifs. On voit sur cet exemple que la séquence peut

comporter des contraintes temporelles, exprimant la simple précédence ou des délais numériques (en remplaçant, par exemple, $(t_2 \geq t_1)$ par $(t_2 \geq t_1 + 5) \wedge (t_2 \leq t_1 + 10)$). Ainsi, une clause peut modéliser une chronique qui est un ensemble d'évènements contraints temporellement.

D'autres techniques permettent d'extraire le concept de séquence sur des objets portant ce genre d'information. L'inférence grammaticale en est une, elle extrait à partir d'exemples une grammaire capable d'engendrer un langage, c'est à dire un ensemble de séquences de symboles. Les modèles de Markov cachés donne aussi la possibilité d'induire un concept de nature statistique à partir de séquences d'apprentissage appartenant à ce concept.

Un exemple : recherche de chroniques cardiaques

Carrault et les co-auteurs de [CCQW03] présentent un système permettant le diagnostic d'une arythmie cardiaque, chez un patient, à partir de son électrocardiogramme. Ce système comporte deux parties. La partie en-ligne recherche, dans l'électrocardiogramme d'un patient, un ensemble de motifs caractérisant différents types d'arythmies. Dans ce cadre, un motif est sous la forme d'une chronique qui est un ensemble d'évènements contraints temporellement.

La deuxième partie, hors-ligne, construit la base de chroniques caractérisant des types d'arythmie à partir d'une base de données d'électrocardiogrammes classés en type d'arythmie. Cette partie hors-ligne fonctionne de la façon suivante : un premier module convertit chaque électrocardiogramme (un signal) de la base de données en une suite d'évènements symboliques. Ces suites classées selon le type d'arythmie forment les exemples pour le module d'apprentissage (PLI). Celui-ci recherche le programme logique couvrant la plupart des exemples d'une classe et excluant les autres. Pour chaque type d'arythmie, un programme logique est donc induit.

La connaissance du domaine (*Background knowledge*) permet de définir des prédicats simplifiant l'induction de clauses mais aussi de spécifier le vocabulaire employé par les clauses.

Conclusion

On a vu, dans ce chapitre, une technique d'apprentissage supervisé : la Programmation Logique Inductive. L'apprentissage est ici guidé par une relation de généralité dans l'espace des hypothèses et consiste à trouver une hypothèse cohérente avec les exemples d'apprentissage. Vu la taille de l'espace de recherche, nous avons vu qu'il était nécessaire d'avoir des biais d'apprentissage pour l'élaguer et faciliter ainsi le travail de la recherche de solutions.

CHAPITRE

2

Fouille de données

La *fouille de données*, aussi connue sous le nom de découverte de connaissance, est un domaine de recherche en plein essor. L'enjeu de cette problématique est d'extraire des connaissances cachées à partir d'un tas de données disponibles. Ces connaissances peuvent avoir diverses formes : règles, modèles, concepts, etc.

Une définition assez subjective de ces connaissances que l'on appelle *motifs* peut être trouvée dans [PSF91] et [Sal03] : « un motif est une expression dans un langage qui décrit des relations dans un sous ensemble de données avec une certaine certitude, telle que l'expression est plus simple (en un certain sens) que l'énumération de tous les faits de la base de données ».

Il n'existe pas de définition claire de la fouille de données ainsi, les méthodes utilisées dépendent surtout des données, des types de motif à extraire et de leur mesure d'intérêt choisie. Par exemple, cette mesure peut être la fréquence d'apparition du motif dans les données.

Dans une première partie, nous allons présenter une technique qui a reçu beaucoup d'attention : la découverte de règles d'associations. Enfin, nous verrons diverses façons d'introduire le temps dans les motifs à extraire.

2.1 Découverte de règles d'associations

La découverte de règles d'associations, introduite par [AIS93], est une des techniques les plus courantes en fouille de données. Le problème peut être illustré ainsi : à partir des articles contenus dans les paniers des clients d'un supermarché, on souhaite savoir si pour « beaucoup » de clients la présence de certains articles implique la présence d'autres articles. Par exemple, la règle $Lait \Rightarrow Beurre$ indique que la plupart des clients achetant du lait achète aussi du beurre.

Nous verrons dans un premier temps la forme générale de la résolution de ce problème. Enfin nous verrons comment traiter ce problème non plus avec des items décrits par des propositions mais sous forme attribut-valeur.

2.1.1 Découverte de règles d'associations complexes d'attributs

La découverte d'associations complexes d'attributs cherche à repérer des implications logiques entre attributs ou ensembles d'attributs. Ces attributs sont contenus en très grande

quantité dans des bases de données. Ainsi, celles-ci ont une influence importante en découverte d'associations, c'est pourquoi un exemple sera appelé ici un *enregistrement*. Nous allons illustrer cette courte présentation par l'exemple d'une base de données des ventes d'un magasin.

Les *items* sont les éléments primitifs de cette base de données. Par exemple, chacun d'eux peut représenter un objet en vente dans le magasin. L'ensemble I constitue l'ensemble de tous les items utilisables de la base de données (tous les objets du magasin).

On définit un *itemset* comme un sous-ensemble de l'ensemble I. La liste des achats d'un client du magasin constitue un itemset. L'itemset X est inclus dans l'itemset Y si tous les items de X sont contenus dans Y. Enfin, chaque transaction de la base de données D est un *t-itemset* (de cardinal inférieur à t). La figure 2.1 illustre une telle base de données.

#	Transaction				
1	a		c	d	
2		b	c		e
3	a	b	c		e
4		b			e

FIG. 2.1 – Une base de données de 4 enregistrements, $I = \{a, b, c, d, e\}$

Ainsi, la découverte d'une association $X \Rightarrow Y$ consiste à rechercher si la présence de l'itemset X implique la présence de l'itemset Y dans un grand nombre d'enregistrements de la base de données. Pour cela, on définit le *support* d'un itemset X comme

$$support(X) = \frac{|S_X|}{|D|} \; tel \; que \; S_X = \{E \in D | X \subseteq E\}$$

Le support de $X \bigcup Y$ définit la fréquence d'apparition de l'association $X \Rightarrow Y$ et $\frac{support(X \bigcup Y)}{support(X)}$ définit la $confiance$ en cette association.

On dit qu'une association est valide sur une base de données si sa confiance et sa fréquence dépassent des seuils minimaux fixés par l'utilisateur. Autrement dit, si

▷ $support(X \bigcup Y) \geq MinSupport$, et

▷ $confiance(X \Rightarrow Y) \geq MinConfiance$

alors l'association $X \Rightarrow Y$ est valide.

Il faut par exemple que beaucoup de clients achètent du *lait* et du *beurre* et que peu achètent du *lait* sans le *beurre* pour que la règle $Lait \Rightarrow Beurre$ soit valide.

On voit que le calcul du support est le point clé de la recherche d'associations dans une base de données.

Algorithme A PRIORI

L'algorithme A PRIORI est très utilisé en fouille de données. Il est optimisé de différentes façons en fonction du type des concepts qu'il recherche.

règles $X \Rightarrow Y$	$support(X \bigcup Y)$	$support(X)$	$confiance(X \Rightarrow Y)$
$a \Rightarrow c$	50%	50%	100%
$c \Rightarrow a$	50%	75%	66.7%
$b \Rightarrow c$	50%	75%	66.7%
$c \Rightarrow b$	50%	75%	66.7%
$b \Rightarrow e$	75%	75%	100%
$e \Rightarrow b$	75%	75%	100%
$c \Rightarrow e$	50%	75%	66.7%
$e \Rightarrow c$	50%	75%	66.7%
$b, c \Rightarrow e$	50%	50%	100%
$b, e \Rightarrow c$	50%	75%	66.7%
$c, e \Rightarrow b$	50%	50%	100%
$b \Rightarrow c, e$	50%	75%	66.7%
$c \Rightarrow b, e$	50%	75%	66.7%
$e \Rightarrow b, c$	50%	75%	66.7%

FIG. 2.2 – Les règles d'associations valides pour $MinSupport = 50\%$ et $MinConfiance = 60\%$ de la base de données 2.1

La première étape de la recherche de règles d'associations est de trouver tous les itemsets fréquents, c'est à dire ceux qui ont un support plus grand que $MinSupport$. Cette étape est réalisée à partir de l'algorithme A PRIORI.

A PRIORI travaille de la façon suivante : à la première étape il calcule l'ensemble des *1-itemsets* fréquents. Ensuite à chaque étape k, il produit les *k-itemsets* candidats en utilisant les *(k-1)-itemsets* fréquents calculés à l'étape précédente, puis il ne conserve que les *k-itemsets* fréquents dans la base de données. L'algorithme se termine quand plus aucun *k-itemset* n'est découvert.

L'algorithme A PRIORI utilise une propriété intéressante du support :
$support(i_1, ..., i_n)$ est toujours supérieur à $support(i_1, ..., i_n, i_{n+1})$ puisque le nombre d'occurrences de $i_1, ..., i_n$ est toujours plus grand que le nombre d'occurrences de $i_1, ..., i_n, i_{n+1}$. Cette propriété permet de calculer l'ensemble des *k-itemsets* fréquents, puis à partir de cet ensemble de chercher les *(k+1)-itemsets* fréquents. Cette propriété est qualifiée d'antimonotone (nous verrons la définition de l'antimonotonie dans le chapitre suivant).

Génération des associations fréquentes et de confiance minimum

Pour chaque itemset X fréquent, généré lors de la première étape, on cherche $B \subset X$ et $A = X - B$, tels que la confiance de l'association $A \Rightarrow B$ est supérieure ou égale à la confiance minimum $MinConfiance$. Ainsi l'association $A \Rightarrow B$ est considérée comme valide.

2.1.2 Découverte de règles d'associations sur des données numériques

Les items considérés dans la partie précédente sont des attributs symboliques. Autrement dit, soit un objet (un exemple) a une caractéristique (un item) soit il ne l'a pas. C'est donc une représentation binaire des données. Il est clair que la recherche d'associations sur des attributs numériques est plus complexe, car celle-ci ne dépend plus seulement du nombre d'attributs mais aussi du nombre de valeurs qu'ils peuvent prendre.

Pour traiter cette complexité sur de grands domaines, les données peuvent être regroupées et considérées seulement collectivement. Par exemple, une hiérarchie peut être définie sur un domaine (continent-pays-région-ville). Cette hiérarchie permet de réduire l'espace des règles à considérer.

On peut aussi découper le domaine d'un attribut en intervalles. Chaque intervalle de chaque attribut numérique est considéré comme un item et après conversion des enregistrements numérique en symbolique, il suffit de faire une recherche, avec l'algorithme A Priori par exemple. C'est ce que l'on appelle discrétiser l'espace de recherche.

Srikant et Agrawal [SA96] proposent de découper chaque attribut de façon arbitraire pour qu'ils ne soient « ni trop grands, ni trop petits ». En effet, s'ils sont trop grands, beaucoup d'enregistrements auront les mêmes items et les règles d'association générés seront trop générales. D'un autre côté, si les intervalles sont trop petits alors les enregistrements auront des items différents et peu de règles seront générées.

D'après Miller et Yang [MY97], ce seul critère n'est pas suffisant pour espérer trouver les « bons » intervalles. En effet, il faut que le découpage corresponde à un partitionnement intuitif des données. Ils préfèrent utiliser un algorithme de clustering pour regrouper les données dont les caractéristiques sont assez voisines. L'algorithme regroupe les données en des ensembles denses et ne conserve que les clusters dont le cardinal est supérieur à un seuil. En fait, un indice (*le diamètre*) inversement proportionnel à la densité, est calculé : c'est la moyenne des distances entre les éléments du cluster.

Cet algorithme est exécuté pour chaque attribut ou ensemble d'attributs qui peuvent être traités conjointement (selon l'utilisateur) sur toutes les données. Ainsi, on obtient autant d'ensembles de clusters que d'attributs. Tous ces clusters sont alors combinés avec les clusters des autres attributs pour former des règles basées sur le support et la confiance.

Le problème de cette technique, mis à jour par Barbará et les coauteurs de [BCN04], réside dans le fait que l'étape de clustering considère un à un les attributs, ce qui empêche la découverte de certaines règles dans la deuxième phase qui essaie de les regrouper. Ces auteurs ont proposé d'utiliser la notion de la dimension fractale pour partitionner le plus naturellement possible les attributs en intervalles. Cette technique, selon les auteurs, donne des regroupements plus naturels et des règles intéressantes plus nombreuses.

2.2 Intégration de la notion temporelle

Une information particulière est souvent présente dans les bases de données : la date de chaque enregistrement. Cet attribut temporel ne se gère pas de la même manière que les autres. En effet, le temps donne des informations particulières sur la succession des enregistrements. On peut donc souhaiter découvrir la manière dont ceux-ci s'enchaînent. Cet enchaînement peut

avoir différents degrés de précision, on peut s'intéresser seulement à la notion de succession d'enregistrements, ou bien de manière plus précise au temps écoulé entre deux enregistrements. Ces différents degrés de précision obligent la mise en place de méthodes de recherche différentes.

La première méthode présentée traite de l'évolution dans le temps d'attributs numériques sur une population alors que la deuxième méthode ne traite le problème que par son aspect temporel numérique. Une troisième méthode traite d'une recherche classique de motifs en ajoutant des contraintes de temps. La dernière partie présente un logiciel, FACE, utilisé en supervision de réseaux de télécommunications.

2.2.1 Recherche de l'évolution d'attributs numériques

Wang et les coauteurs de [WYM01] proposent de rechercher l'évolution d'un attribut numérique. On dispose d'attributs sur un ensemble d'individus pour plusieurs dates. À partir de cet ensemble d'instantanés (snapshot : un pour chaque date), leur algorithme permet d'extraire des règles d'évolution des attributs.

Ils utilisent le critère de motif dense pour extraire des informations intéressantes. En effet, l'utilisation d'un seuil de densité est expliqué par le fait qu'il ne suffit pas de regrouper assez d'individus pour qu'un regroupement soit intéressant : il faut aussi que ces individus se ressemblent, d'où l'utilisation de la densité.

Pour ce faire, leur espace de recherche, où chaque dimension est un attribut de chaque instantané, est discrétisé en « petits cubes de base ». Un petit cube de base est associé à un nombre m d'instantanés et à un ensemble d'attributs associés à des intervalles de valeurs.

Un algorithme recherche ceux dont la densité est supérieure à un seuil déterminé à l'avance. La densité d'un tel cube (qui revient à un calcul de support car on connaît son volume) est calculée en comptant le nombre d'individus dont les attributs prennent les valeurs du cube de base pour m instantanés consécutifs. Cela revient à la méthode des fenêtres glissantes de [MTV97].

Les clusters sont ensuite formés à partir des cubes de base proches qui regroupent assez d'éléments selon un seuil de support minimal. Pour chacun des clusters trouvés, on extrait les règles d'association traduisant l'évolution d'un ou plusieurs attributs. Soit X un cluster et $B \subset X$ et $A = X - B$, deux ensembles de cubes de base inclus dans X. Un algorithme recherche les ensembles B et A dont la règle associée $A \Rightarrow B$ a une confiance suffisante.

Cette méthode permet d'extraire des règles du type : $salaire \in [40000, 50000] \rightarrow salaire \in [55000, 57500] \rightarrow salaire \in [60000, 67500]$ dans une base de données des salaires annuels par employés d'une entreprise. Cette règle indique que beaucoup d'employés gagnent entre 40000 et 50000 pendant une année, puis entre 55000 et 57500 l'année suivante, et entre 60000 et 67500 la troisième année.

2.2.2 Découverte d'intervalles maximalement spécifiques et fréquents

Les algorithmes présentés précédemment recherchent les motifs fréquents dans un base de données. Ces motifs sont à base d'intervalles lorsque les données sont numériques. Il n'est donc pas possible de retrouver l'ensemble des motifs (ou intervalles) fréquents. En effet, si un nombre d'enregistrements suffisant possèdent un attribut numérique a tel que $a \in [5, 10]$ alors on dit que

ce motif est fréquent. On peut donc aussi dire que les motifs $a \in [4, 10]$, $a \in [5, 11]$, $a \in [2, 100]$ ... sont aussi fréquents.

On a vu précédemment qu'on introduisait la notion de densité pour rechercher les motifs « intéressants » qui sont donc à la fois fréquents et denses. L'article de Lin [Lin03] propose de rechercher *tous* les intervalles maximalement spécifiques et fréquents dans un ensemble d'intervalles. On souhaite donc trouver la borne $S - set$ de l'espace des versions (voir 1.2).

À partir d'un ensemble d'intervalles de type $[a, b]$, un algorithme classe ces intervalles dans un arbre (*I-Tree*) de façon à les trier selon un ordre de généralité. Ensuite, un parcours de l'arbre et une mise à jour de celui-ci permettent de découvrir l'ensemble complet et correct des intervalles maximalement spécifiques et fréquents.

Cet article ne traite que de la recherche de « mono-intervalles ». Dans le cas d'ensembles de « multi-intervalles », une telle recherche devrait impérativement être biaisée pour espérer obtenir des résultats en un temps acceptable.

2.2.3 Découverte de motifs en incluant les intervalles de temps

Yoshida et les coauteurs de [YISI00] reprennent le problème de découverte d'itemsets fréquents en ajoutant la dimension temporelle. Chaque enregistrement de la base de données est daté et appartient à un groupe d'enregistrements (chaque groupe est par exemple associé à un client). À partir d'un support minimal, leur algorithme permet d'extraire les séquences d'itemsets contraints temporellement. Par exemple, soit a, b et c des items, la séquence fréquente $< \{(a), -, -, -\}\{(b, c), 17, 18, 19\}\{(b), 5, 10, 12\} >$ indique qu'il existe assez de groupes d'enregistrements (de clients) pour lesquels on a un enregistrement contenant l'item a suivi d'un enregistrement contenant (b, c) au moins 17 u.t (unité de temps) après le premier et 19 u.t avant et enfin un enregistrement contenant l'item (b) au moins 5 u.t après le second et 12 u.t avant (les valeurs 18 et 10 représentent les moyennes de temps entre les enregistrements considérés dans la base de données). Leur algorithme est basé sur l'algorithme A Priori qui à partir des séquences fréquentes de taille k génèrent les séquences candidates de taille $k + 1$.

Le passage de l'étape $k = 1$ à l'étape $k = 2$ est la plus importante en terme de coût de calcul car c'est elle qui établit les intervalles entre deux items. À partir de deux items fréquents A et B, on recherche les séquences fréquentes de type $< \{A, -, -, -\}\{B, \Delta_{min}, \Delta_{moy}, \Delta_{max}\} >$. Ainsi, pour chaque groupe d'enregistrements, on recherche les couples d'enregistrements qui incluent respectivement les deux items A et B. On note l'intervalle de temps écoulé entre les deux enregistrement de tous ces couples, ce qui nous donne un ensemble de valeurs. Ensuite, on regroupe ces valeurs en clusters par la méthode de clustering décrite dans [MY97]. Ainsi chaque cluster représente une séquence qui contient les itemsets A et B. On ne conserve que les clusters dont le cardinal dépasse un seuil fixé à l'avance.

2.2.4 Découverte de chroniques

De nombreux systèmes de fouille de données sont adaptés pour l'analyse de données non ordonnées. Cependant, la supervision de systèmes dynamiques consiste à surveiller des séquences d'évènements, autrement dit, des données ordonnées. Dans ce genre de système, de nombreuses

alarmes sont générées, mais beaucoup d'entre elles ne sont pas significatives d'un dysfonctionnement. C'est le cas des journaux d'alarmes d'un réseau de télécommunications.

Ainsi, [DD99] propose de rechercher les motifs fréquents au sein d'un unique journal. Dans ce cas, un motif est un graphe de contraintes où chaque nœud représente un évènement. La figure 2.3 montre ce type de motif que l'on appelle *chronique*.

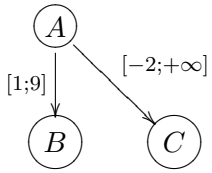


FIG. 2.3 – Exemple de chronique

À partir des chroniques fréquentes d'un journal, un expert peut filtrer celles qui correspondent à un fonctionnement normal du réseau de celles qui traduisent un état inhabituel du système.

Cette recherche de chroniques fréquentes est implémentée dans le logiciel FACE (Frequent Analyser for Chronicle Extraction) de France Télécom. Elle s'appuie sur [MTV97] qui recherche des épisodes fréquents (des chroniques sans contraintes temporelles).

La découverte de chroniques fréquentes s'effectue en deux étapes : leur recherche et leur filtrage.

La première étape est une adaptation de l'algorithme A PRIORI. Pour utiliser cet algorithme, on utilise le fait que les sous-chroniques d'une chronique fréquente sont également fréquentes. À chaque étape k de l'algorithme, le processus génère les chroniques candidates de taille k puis les filtre selon un seuil minimal de fréquence dans le journal. FACE utilise CRS [DG94] qui permet de trouver efficacement toutes les occurrences d'une chronique dans un journal d'évènements.

Le but de la deuxième étape est d'éliminer les sous-chroniques fréquentes lorsqu'elles sont systématiquement incluses dans une chronique plus complexe et de conserver celles qui, bien qu'appartenant parfois à des chroniques plus grosses, sont néanmoins représentatives de phénomènes fréquents.

Conclusion

La fouille de données a été abordée dans ce chapitre de manière à expliquer globalement ses enjeux et quelques unes de ses techniques. On a vu que la notion de temps pouvait être ajoutée de différentes manières et imposait différentes méthodes de recherche. On a aussi vu que la fouille de données recherche les motifs qu'elle qualifie d'intéressants. L'ajout de seuils de densité aux seuils de fréquence permet de mieux qualifier ces motifs intéressants. Cette notion de densité n'est pas pris en compte par FACE. Comme on le verra dans les chapitres suivants, cela pose certains problèmes.

Néanmoins, la découverte de chroniques implémentée dans ce logiciel est un bon exemple de découverte utile car elle a permis de déceler des problèmes insoupçonnées au sein de réseaux de télécommunication. Une intégration élégante et bien formalisée de la notion de fouille de données

et de base de données a fait défaut. La méthode, présentée au chapitre suivant, comble ce vide en considérant la fouille de données comme un processus particulier d'interrogation d'une base de données par extraction de connaissances.

CHAPITRE

3

Base de données inductive

Les bases de données inductives (*BDIs*) sont nées de la nécessité de formaliser la fouille de données et d'établir des liens clairs avec les concepts et les algorithmes utilisés dans le domaine des bases de données. Cette formalisation a été introduite par Mannila et Imielinski [IM96]. Ce domaine pluridisciplinaire se situe entre l'intelligence artificielle, les statistiques et les bases de données. Les bases de données sont le support de toute fouille de données. Leur gestion et leurs structures de données sont aujourd'hui extrêmement poussées. De plus, au vu des énormes quantités de données à traiter, la fouille de données se doit d'être efficace. C'est pourquoi, on espère récupérer l'efficacité des mécanismes de gestion et d'extraction des données des bases de données en les réutilisant dans les bases de données inductives.

En plus des données, une base de données inductive contient leurs modèles (que nous appellerons motifs par la suite). Le processus de fouille de données est considéré comme un processus d'interrogation, au moyen de requêtes, de la base de données inductive. En réponse, celle-ci renvoie un ensemble de motifs.

Le langage de requêtes est donc primordial. La plupart de ceux proposés, dans les implémentations déjà réalisées (MINE RULE, MSQL, DMQL, et XMine [De 02c]), sont des extensions aux langages de requêtes des bases de données (SQL ou XML). Ce travail a été complété par plusieurs approches pour permettre l'expression des contraintes en fouille de données, présentée dans [De 02a]. Celles-ci permettent à l'utilisateur de spécifier, à l'aide de primitives, les motifs intéressants à rechercher.

Cette formalisation de la fouille de données a donné naissance à d'autres axes de réflexion. En base de données classiques l'efficacité de la recherche de données à partir d'une requête est primordiale. Dans une BDI, la recherche de motifs à partir d'une requête doit aussi être efficace malgré l'expressivité du langage de requêtes.

3.1 Présentation

Une base de données inductive $I(\mathcal{D}, \mathcal{P})$ est formée de deux composants : les données $\mathcal{D}$ et les motifs $\mathcal{P}$. Chacun de ces composants est un ensemble d'ensembles.

Chaque ensemble de données $D \in \mathcal{D}$ contient différents enregistrements notés e. Ce qui est

également vrai pour $\mathcal{P}$ où un ensemble $P \in \mathcal{P}$ contient différents motifs notés p.

On introduit une relation de couverture entre motifs et ensemble de données par analogie avec l'espace des versions (en 1.2.1).

Un exemple sur des chaînes de caractères illustre cette définition.

▷ $e_1 = aabbcc$; $e_2 = abbc$; $e_3 = bb$; $e_4 = abc$; $e_5 = bc$; $e_6 = cc$

▷ $D_1 = \{e_1, e_2, e_3\} = \{aabbcc, abbc, bb\}$;
$D_2 = \{e_4, e_5, e_6\} = \{abc, bc, cc\}$;
$D_3 = D_1 \bigcup D_2$

▷ $p_1 = abb$; $p_2 = bb$; $p_3 = cc$

▷ $P_1 = \{p_1, p_2, p_3\}$

▷ $\mathcal{D} = \{D_1, D_2, D_3\}$ et $\mathcal{P} = \{P_1\}$

Dans cet exemple, on définit l'ensemble couvert par un motif p comme l'ensemble des données l'ayant comme sous-chaîne, ainsi p_1 couvre e_1 et e_2.

3.2 Le langage de requêtes

Une des raisons du succès des bases de données relationnelles est la combinaison d'un langage de requêtes à la fois raisonnablement simple et sémantiquement bien défini. La motivation essentielle du concept de BDI est de récupérer ces bonnes propriétés pour un langage de requête étendu.

Beaucoup de problèmes de fouille de données consistent en la recherche d'un ensemble de motifs satisfaisant certaines contraintes. Cet aspect constitue la base du langage de requête sur une base de données inductive : une requête est une conjugaison d'un ensemble de contraintes et son résultat est un ensemble de motifs.

Formellement, la satisfaction d'une requête peut être définie de la manière suivante : $Th(Q, \mathcal{D}, \mathcal{L}) = \{\varphi \in \mathcal{L} | Q(\varphi, \mathcal{D})\}$, c'est à dire trouver l'ensemble des motifs φ en langage $\mathcal{L}$ satisfaisant la requête Q sur la base de données $\mathcal{D}$. Dans le cadre des bases de données inductives, Q est une requête de l'utilisateur.

Le langage présenté ici contient des requêtes booléennes inductives. Celui-ci, d'après [DJLM02], est le langage de requêtes inductives le plus général dans la littérature sur la fouille de données. En effet, ce langage permet d'utiliser des conjonctions de contraintes monotones et anti-monotones (définies dans la suite) alors que d'autres langages permettent soit les unes soit les autres.

Les requêtes étant très dépendantes des données et des motifs qui peuvent être d'un type quelconque (chaînes de caractère, séquences, graphes,...), il est difficile de fournir un langage universel à la manière de SQL. Néanmoins, on peut proposer, comme [DJLM02], un ensemble de requêtes primitives pouvant servir de base.

3.2.1 Requêtes primitives

Nous allons différencier trois types de requêtes primitives, tout d'abord les requêtes primitives utilisant directement une relation de généralité, ensuite celles portant sur la fréquence de motifs dans un ensemble de données, et enfin celles regroupant les opérations ensemblistes.

On impose l'existence d'une relation de généralité entre motifs qui permettra, dans la suite, d'utiliser l'espace des versions présenté en 1.2. Cette relation est utilisable dans une requête de la façon suivante : $p \succeq p'$, $\varphi \succeq p$, $\neg(\varphi \succeq p)$, $p \succeq \varphi$, et $\neg(p \succeq \varphi)$, où p et p' sont des motifs.

Comme dans 1.2, la relation de généralité doit être définie à l'aide de la relation de couverture d'un motif. Une fonction $freq$ permet d'utiliser la fréquence d'apparition des motifs dans un ensemble de données. Celle-ci est définie de la façon suivante : $freq(p, D) = card\{e \in D \mid e \in couverture(p)\}$, elle donne le nombre d'éléments de D contenus dans la couverture de p. Cette fonction est utilisable dans une requête de la façon suivante : $freq(p, D) \geq t$, $freq(p, D) \leq t$, $freq(\varphi, D) \geq t$, et $freq(\varphi, D) \leq t$, où t est un seuil.

De plus, on peut utiliser les opérations ensemblistes classiques : $i \in I$, $i \notin I$, $\varphi \in P$, et $\varphi \notin P$, où i est un élément, et I un ensemble.

Ces requêtes primitives, que l'on nomme contraintes dans la suite, vont servir de base à la construction de requêtes plus complexes. Il ne faut pas oublier qu'on ne peut pas donner une liste exhaustive des contraintes possibles tant elles dépendent du domaine dans lequel on est.

3.2.2 Contraintes monotones et anti-monotones

Lors de la recherche dans l'espace des solutions, la définition de critères de monotonie et d'anti-monotonie sur les contraintes contenues dans les requêtes permet d'utiliser l'espace des versions. Formellement, une contrainte c est monotone (resp. anti-monotone) si et seulement si $\forall motifs \;\; s, g : (s \preceq g) \wedge (g \in sol(c)) \Rightarrow (s \in sol(c))$ (resp. $(s \preceq g) \wedge (s \in sol(c)) \Rightarrow (g \in sol(c))$.

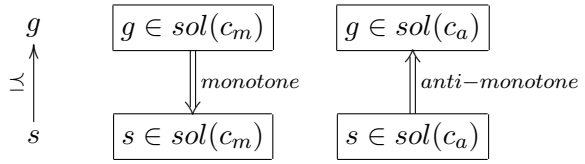


FIG. 3.1 – Contraintes anti-monotones c_a et monotones c_m

Les contraintes basées sur la fréquence et la notion de généralité satisfont soit le critère de monotonie soit le critère d'anti-monotonie. Par exemple, $\varphi \preceq p$ et $freq(\varphi, D) \leq t$ sont monotones et $p \preceq \varphi$, et $freq(\varphi, D) \geq t$ sont anti-monotones. Il est intéressant de noter qu'une contrainte anti-monotone est la négation d'une contrainte monotone. Pour une contrainte qui ne serait ni monotone, ni anti-monotone, il n'existe pas, en fouille de données, d'algorithmes efficaces.

L'algorithme A PRIORI nécessite l'anti-monotonie de la contrainte dont on recherche les solutions. Par exemple, en recherche d'associations présentée en 2.1.1, les itemsets solutions sont contraints d'avoir un support supérieur ou égal à $MinSupport$ ($support(X) \geq MinSupport$). De plus, il existe une relation de généralité entre les itemsets qui est l'inclusion. Ainsi la contrainte d'un support est anti-monotone.

La représentation par l'espace des versions utilise des contraintes monotones et anti-monotones. En effet, l'ensemble des motifs les plus généraux solutions d'une contrainte monotone permettent de déduire, en les spécialisant, tous les motifs solutions de cette contrainte. Il en va de même pour

une contrainte anti-monotone qui autorise le calcul de l'ensemble des motifs solutions les plus spécialisés pour déterminer, en les généralisant, l'ensemble des solutions de cette contrainte. Autrement dit, les bornes S et G de l'espace des versions sont calculables à partir, respectivement, de contraintes anti-monotones et monotones. Cette idée est illustrée par la figure 3.2.

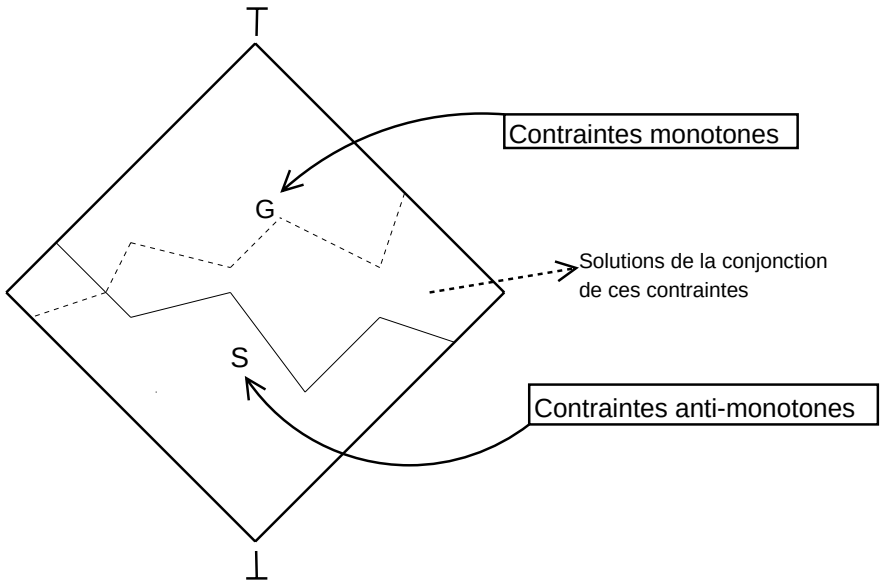


FIG. 3.2 – L'espace des versions avec les contraintes monotones et anti-monotones

Finalement, une requête Q est une expression booléenne utilisant les contraintes précédentes connectées par les opérateurs $\wedge$,$\vee$, et $\neg$. Par exemple, on peut avoir : $Q = (freq(\varphi, D_1) \geq t \vee p_1 \succeq \varphi) \wedge freq(\varphi, D_2) \leq t$.

3.3 Recherche des solutions

La recherche des solutions à requête Q passe par deux étapes. La première consiste en la décomposition de Q en un ensemble de sous-requêtes. L'ensemble des solutions de chacune de ces sous-requêtes peut être représenté par un espace des versions. Le calcul des espaces des versions des solutions de ces sous-requêtes, puis leur union définit la deuxième étape. Le résultat de la requête Q est une union d'espaces des versions.

3.3.1 Décomposition de requêtes

L'approche générale suggérée par [DJLM02] est de décomposer une requête Q en k sous-requêtes Q_i. Cette décomposition assure que Q est équivalente à $Q_1 \vee ... \vee Q_k$.

Pour construire cette décomposition, on réécrit la requête en n'utilisant que des contraintes monotones. Par exemple, on réécrit la requête $Q = (freq(\varphi, D_1) \geq t \vee p_1 \succeq \varphi) \wedge (freq(\varphi, D_2) \leq t \vee p_2 \preceq \varphi)$ en $Q' = (\neg m_1 \vee m_2) \wedge (m_3 \vee \neg m_4)$ où

$m_1 = \neg(freq(\varphi, D_1) \geq t)$
$m_2 = p_1 \succeq \varphi$
$m_3 = freq(\varphi, D_2) \leq t$
$m_4 = \neg(p_2 \preceq \varphi)$

Ainsi Q est décomposée de la façon suivante :

$$Q = Q_1 \vee Q_2 \vee Q_3 \vee Q_4, \text{où } Q_1 = \neg m_1 \wedge m_3,\ Q_2 = \neg m_1 \wedge \neg m_4$$
$$Q_3 = m_2 \wedge m_3,\ \text{et } Q_4 = m_2 \wedge \neg m_4$$

Avant de rechercher les solutions de chaque sous-requête, on enlève celles qui n'ont aucune solution pour n'importe quelle base de données. Dans l'exemple précédent, on suppose que $p_1 \preceq p_2$, or $Q_4 = p_1 \succeq \varphi \succeq p_2$, cette requête n'a donc pas de solutions, et on obtient $Q = Q_1 \vee Q_2 \vee Q_3$.

Un théorème de [DJLM02] indique que $Th(Q_i, \mathcal{D}, \mathcal{L})$ est convexe pour tout $\mathcal{D}$ et $Th(Q, \mathcal{D}, \mathcal{L}) = \bigcup_{h=1}^{k} Th(Q_i, \mathcal{D}, \mathcal{L})$. La convexité de Q_i va permettre dans la suite d'utiliser l'espace des versions pour calculer son résultat.

De plus, il découle une propriété intéressante sur chaque sous-requête Q_i : Q_i peut toujours être reconnue comme une conjonction de deux sous requêtes $Q_{i,M}$ et $Q_{i,A}$, où $Q_{i,M}$ est une conjonction de contraintes monotones et $Q_{i,A}$ est une conjonction de contraintes anti-monotones.

3.3.2 Calcul des solutions d'une sous-requête

Il existe différentes manières de rechercher les solutions d'une sous-requête Q_i (notée Q dans la suite). [DJLM02] propose d'utiliser la théorie des arbres d'espaces de version. Ces arbres sont limités car ils ne sont pas adaptés à la représentation des graphes mais seulement des chaînes de caractères. [DK01] développe une extension de l'algorithme par niveau. Le problème de cette approche est que les algorithmes de résolution des requêtes monotones et anti-monotones sont différents.

Nous allons présenter ici le fonctionnement général de l'algorithme *MineSeqLog* détaillé dans [LD03]. Celui-ci permet le calcul des solutions d'une requête $Q = Q_M \wedge Q_A$, où Q_A est une requête anti-monotone et Q_M une requête monotone, en appelant un autre algorithme déjà décrit dans ce document : l'algorithme A Priori.

Dans l'algorithme de [LD03], A Priori est appelé deux fois. Une première fois pour trouver l'ensemble U des motifs solutions de la requête Q_A et la seconde fois pour trouver l'ensemble L des motifs solutions de la requête $\neg Q_M$. Ceci est possible car la négation d'une requête monotone est une requête anti-monotone.

U représente l'ensemble des solutions de Q_A. Q_A étant anti-monotone, tout élément plus général qu'un élément de U est solution de Q_A. De la même façon pour L qui est l'ensemble des solutions de $\neg Q_M$, tout élément plus général qu'un de ses éléments est solution de $\neg Q_M$. Autrement dit, tout élément qui n'est pas plus général que tout élément de L est solution de Q_M.

L'ensemble des solutions est donc l'ensemble des motifs plus généraux qu'un motif de U et pas plus généraux que tous les motifs de L. Les ensembles U et L forment respectivement les bornes S et G de l'espace des versions d'une requête.

Mais des éléments de U et L ne sont pas utiles, on va donc les enlever de chacun de ces ensembles. On enlève de U l'ensemble des motifs plus généraux qu'un motif de L et on enlève de L l'ensemble des motifs dont il n'existe pas de motif plus spécifique dans U.

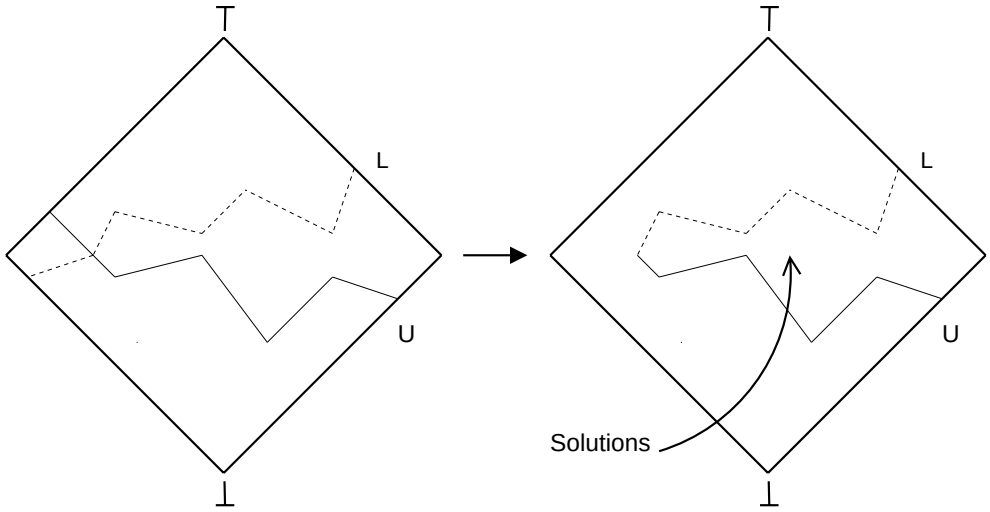


FIG. 3.3 – Suppression des éléments inutiles de L et U, L et U sont les bornes de l'ensemble des motifs solutions d'une requête $Q = Q_M \wedge Q_A$

3.4 Représentation des données et des motifs

Des améliorations des structures de données interviennent surtout dans le cadre de l'optimisation de la recherche de solutions. Ainsi, [De 02c] propose différentes structures de données. Celles-ci doivent donner la possibilité de garder une trace de la relation de sous-ensemble (par l'inclusion $\subset$) entre ensembles. De plus, la relation de généralité ($\preceq$) entre motifs doit, elle aussi, être explicitement représentable. Cela permet de gérer efficacement un ensemble de motifs non convexe ou non représentable par ses bornes. De plus, la notion d'induction entre motifs et données doit être explicitement représentée. Enfin les résultats d'une requête peuvent être conservés en prévision de requêtes futures nécessitant des résultats déjà calculés.

Actuellement, il n'existe pas d'implémentation de ce genre de structures de données, mais [De 02c] indique que leur réalisation devrait permettre une optimisation notable de la recherche de solutions aux requêtes

3.5 Liens avec la PLI

La notion de base de données inductives a été proposée pour formaliser la fouille de données. Pourtant, des présentations de celle-ci et de la programmation logique inductive, on découvre un ensemble de liens entre ces deux techniques.

En effet, une requête sur une BDI peut contraindre la recherche de motifs à généraliser un ensemble de données tout en excluant d'autres ensembles de données. Par exemple, dans la requête $Q = freq(\varphi, D_1) \geq t_1 \wedge freq(\varphi, D_2) \leq t_2$, D_1 peut être considérée comme un ensemble d'exemples positifs et D_2 un ensemble d'exemples négatifs au sens de l'apprentissage supervisé.

Par ailleurs, l'espace de recherche des motifs d'une BDI est muni d'une relation d'ordre. Ainsi, en BDI, le parcours de cet espace de recherche peut s'apparenter à celui de la PLI. Par conséquent, des notions de biais peuvent apparaître dans les requêtes. Par exemple, la requête $Q = freq(\varphi, D_1) \geq t \wedge \varphi \succeq P_3$ impose la recherche de motifs plus généraux qu'un ensemble P_3 de motifs. Ainsi, la relation de généralité existant sur l'espace des requêtes permet d'orienter la recherche comme le font les biais de langage en PLI.

Le langage de requête comprend un ensemble de contraintes qu'il est possible d'étendre suivant les données et les connaissances recherchées. Si ces contraintes sont monotones (resp. anti-monotones) alors elles sont équivalentes au bord G (resp. S) de l'espace des versions.

D'un point de vue théorique, les bases de données inductives permettent, par conséquent, de répondre à des problèmes du même type que ceux résolus par la programmation logique inductive. Les bases de données inductives constituent une réconciliation théorique entre la fouille de données et la programmation logique inductive.

Conclusion de l'état de l'art

Nous avons présenté différentes techniques d'apprentissage issues de problématiques différentes. En apprentissage supervisé, on a vu la programmation logique inductive qui permet d'induire des clauses à partir d'exemples classés. Contrairement à celle-ci, la fouille de données permet de trouver des modèles sans connaissance particulière sur les données.

Enfin, une formalisation de la fouille de données a permis de considérer les bases de données inductives. Celles-ci utilisent des techniques d'apprentissage supervisé, entre autres l'espace des versions. Les bases de données inductives montrent le lien étroit qui relie les deux types d'apprentissage étudiés. Ce lien réconcilie théoriquement la programmation logique inductive et la fouille de données.

De plus, la PLI permet d'apprendre des règles de classification intégrant une notion temporelle. On se propose donc de réaliser ce même travail à partir d'une base de données inductive. Actuellement, les bases de données inductives n'intègrent pas de notion de temps. Par conséquent, l'introduction de la notion temporelle dans les bases de données inductives sera un point essentiel de notre travail.

En outre, on dispose de FACE : un outil de recherche de motifs fréquents temporels ou chroniques. Nous souhaitons adapter cet outil de fouille de données pour mettre en œuvre le concept de base de données inductives étendu à des données présentant des aspects temporels. Une telle réalisation concrétiserait la réconciliation entre apprentissage symbolique et fouille de

données qui n'est encore qu'une notion théorique.

Enfin, on dispose de résultats concernant l'apprentissage de chroniques cardiaques en PLI. Cela rend possible la validation de l'approche proposée.

Deuxième partie

Réconciliation apprentissage symbolique et fouille de données

CHAPITRE

4

Extension du concept de base de données inductives aux séquences temporelles

Les bases de données inductives (*BDIs*) formalisent le processus de fouille de données. Une BDI contient à la fois les données, mais aussi leurs modèles. La fouille de données est donc considérée comme un processus d'interrogation sur ces ensembles au moyen de requêtes.

Les connaissances extraites à partir des bases de données inductives actuelles [LD03, Dze02, IM96, De 02c, De 02b, DJLM02] reposent sur la notion de séquence de symboles. De nombreux problèmes de découverte de connaissances nécessitent d'intégrer d'autres phénomènes. Ainsi, une notion explicite du temps plus riche qu'un ordre sur des évènements, est très souvent nécessaire pour traiter de nombreux domaines : réseau de télécommunications, milieu médical, environnement,etc. Nous proposons donc d'ajouter une dimension temporelle aux motifs extraits dans une base de données inductives. Les motifs que nous extrayons ne sont plus des simples successions de symboles mais des ensembles d'évènements contraints temporellement : des chroniques. Les bases de données ne contiennent plus des séquences de symboles mais des séquences temporelles (ou séquences d'évènements datés).

L'application qui motive cette approche est la découverte de règles de caractérisation d'arythmies en cardiologie. À partir d'électrocardiogrammes convertis en séquences d'évènements (les données), on souhaite apprendre des classes de chroniques cardiaques (les modèles) caractérisant un type d'arythmie. Chaque séquence d'évènements cardiaque correspond à une classe d'arythmie. Par conséquent, les chroniques cardiaques « intéressantes » sont celles que l'on retrouve dans une séquence d'évènements mais pas dans les autres.

Dans une première partie, les principaux concepts liés aux chroniques seront mis en évidence. Ensuite, nous examinerons en détail le type de requête que l'on souhaite poser à une base de données inductives étendue. Enfin, divers axes possibles de recherche seront présentés.

4.1 Les chroniques comme motifs de BDI

Dans cette partie nous allons tout d'abord définir les termes utilisés autour des chroniques. Ensuite, nous verrons les modèles minimaux de chroniques pour aboutir à une définition d'une classe particulière de chroniques : les chroniques simples. Enfin, les opérations importantes sur ces chroniques, le test de sous-chronique et l'union de chroniques, seront présentées et des implémentations seront décrites.

4.1.1 Terminologie

Les chroniques sont des objets complexes dont il faut définir tous les rudiments pour comprendre leur apprentissage. La description d'une séquence d'évènements est tirée de [MTV97]. Cet article décrit la découverte d'épisodes fréquents dans une séquence d'évènements. Le concept de chroniques, celui que nous étudions, est plus général que le concept d'épisode utilisé dans [MTV97].

4.1.1.1 Séquence d'évènements

Nous considérons les données comme des *séquences d'évènements* ordonnés, où chaque évènement est associé à une date d'occurrence. Étant donné un ensemble E de *types d'évènements*, un *évènement* est une paire (A, t), où $A \in E$ est un type d'évènement et t un entier représentant la *date d'occurrence* de l'évènement.

4.1.1.2 Modèle de chronique

Informellement, un modèle de chronique est un ensemble d'évènements contraints temporellement.

Un modèle de chronique $\mathcal{C}$ est un couple $(\mathcal{S}, \mathcal{T})$, où $\mathcal{S}$ est un multi-ensemble d'évènements et $\mathcal{T}$ un graphe de contraintes temporelles. Dans le graphe $\mathcal{T}$, les nœuds représentent les instants des évènements de $\mathcal{S}$, et les arcs représentent les contraintes entre ceux-ci. Un exemple de contrainte est illustré par la figure 4.1.

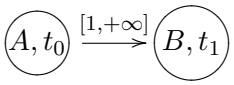


FIG. 4.1 – Une contrainte entre 2 évènements : $1 \leq t_1 - t_0$

Appariement complet Un appariement a entre deux ensembles d'évènements $\mathcal{S}$ et $\mathcal{S}'$ est un ensemble de couples d'évènements. Pour chaque couple, les deux évènements sont de même type, l'un appartient à $\mathcal{S}$ et l'autre à $\mathcal{S}'$. Dans un appariement, les évènements de $\mathcal{S}$ et $\mathcal{S}'$ ne peuvent apparaître qu'une seule fois. Un appariement est complet, c'est à dire aucun couple d'évènements ne peut lui être ajouté. On note $a(\mathcal{S})$ l'ensemble des évènements de $\mathcal{S}$ contenus

dans a (de même pour l'ensemble $\mathcal{S}'$) et $a[e]$ l'évènement associé à l'évènement e dans un couple de a.

Définition 3 (Appariement entre chroniques) *L'ensemble a de couples d'évènements est un appariement entre deux ensembles $\mathcal{S}$ et $\mathcal{S}'$ si et seulement si*

$$\forall c \in a, c = (E_{\mathcal{S}}, E_{\mathcal{S}'}) \Rightarrow E_{\mathcal{S}} \in \mathcal{S} \wedge E_{\mathcal{S}'} \in \mathcal{S}' \wedge type(E_{\mathcal{S}}) = type(E_{\mathcal{S}'}),$$
$$\forall E \in \mathcal{S}, |\{c | c \in a, c = (E, \_)\}| \leq 1,$$
$$\forall E' \in \mathcal{S}', |\{c | c \in a, c = (\_, E')\}| \leq 1$$

L'appariement a est complet si et seulement si il n'existe pas d'appariement entre $(\mathcal{S} \setminus a(\mathcal{S}))$ et $(\mathcal{S}' \setminus a(\mathcal{S}'))$

On pourra parler indifféremment d'appariements entre les évènements de deux chroniques ou bien d'appariements entre ces chroniques.

Ensemble complet d'appariements entre deux chroniques L'ensemble complet de tous les appariements possibles entre deux chroniques $\mathcal{C}$ et $\mathcal{C}'$ est noté $\mathbb{A}^{\mathcal{C}}_{\mathcal{C}'}$. Cet ensemble est symétrique par rapport aux chroniques : $\mathbb{A}^{\mathcal{C}'}_{\mathcal{C}} = \mathbb{A}^{\mathcal{C}}_{\mathcal{C}'}$.

Instance de modèle de chronique Une instance c d'un modèle de chronique $\mathcal{C}$ est un ensemble d'occurrences d'évènements. À chaque occurrence d'évènement de l'instance c est associé un évènement du modèle de la chronique $\mathcal{C}$. Les dates de ces occurrences sont cohérentes avec les contraintes temporelles de $\mathcal{C}$.

4.1.1.3 Ordre

L'utilisation d'un espace de recherche fortement structuré permet une découverte de motifs plus élaborée. C'est pourquoi, une relation d'ordre sur les chroniques est introduite. Celle-ci doit permettre l'utilisation de l'espace des versions introduit en 1.2. Cette relation n'étant pas triviale, elle sera détaillée dans la partie 4.1.4. Nous introduisons deux opérateurs *min* et *max* qui peuvent être utilisés sur des éléments qui possèdent une relation d'ordre. Celle-ci est notée $q \sqsubseteq f$ [1] , on dit alors que q est plus général que f.

Extraction d'éléments maximalement généraux On définit l'opérateur *max* qui permet d'extraire les éléments maximalement généraux d'un ensemble :

$$max(F) = \{f \in F | \neg \exists q \in F : q \sqsubseteq f\}$$

Extraction d'éléments maximalement spécifiques On définit l'opérateur *min* qui permet d'extraire les éléments maximalement spécifiques d'un ensemble :

$$min(F) = \{f \in F | \neg \exists q \in F : f \sqsubseteq q\}$$

[1] Il faut noter que l'opérateur de relation d'ordre de la partie 1.2 n'est pas dans le même sens : $q \succeq f$, q est plus général que f

4.1.2 Modèle minimal de chronique

Dans une chronique $\mathcal{C} = (\mathcal{S}, \mathcal{T})$, le graphe $\mathcal{T}$ est un STP (*Simple Temporal Problem*) décrit par Dechter et les co-auteurs de [DMP91]. D'après le corollaire 3.5 de cet article, il existe un graphe minimal $\mathcal{M}$ équivalent au graphe cohérent $\mathcal{T}$.

On calcule le graphe $\mathcal{M}$ à partir de $\mathcal{T}$ en appliquant un algorithme de propagation de contraintes : *Floyd-Warshall*. La complexité de cet algorithme est en $O(n^3)$, avec n le nombre de nœuds.

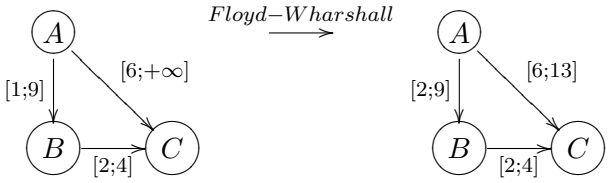


FIG. 4.2 – Graphe minimal d'un modèle de chronique

Le graphe $\mathcal{M}$ d'un modèle de chronique est unique, minimal et complet. Il permet de définir l'unique modèle minimal $(\mathcal{S}, \mathcal{M})$ d'un modèle de chronique $(\mathcal{S}, \mathcal{T})$, tel que $\mathcal{M} = FloydWarshall(\mathcal{T})$. Par la suite on ne travaillera que sur des modèles minimaux de chroniques que nous nommerons chroniques.

4.1.3 Chronique simple

Les chroniques simples $(\mathcal{S}, \mathcal{M})$ sont des chroniques dont les évènements ont tous un type différent. Autrement dit $\mathcal{S}$ n'est plus un multi-ensemble mais un ensemble. Les évènements de chroniques « normales » peuvent avoir des types identiques. Le concept de chronique généralise celui de chronique simple. Un exemple de chronique est donné dans la figure 4.3.

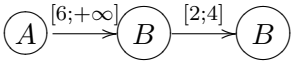


FIG. 4.3 – Une chronique

4.1.4 Relation de sous-chronique

Une chronique $\mathcal{C}$ est une sous-chronique de $\mathcal{C}'$ (noté $\mathcal{C} \sqsubseteq \mathcal{C}'$) si et seulement si à partir de toute instance de $\mathcal{C}'$, on peut extraire une instance de $\mathcal{C}$. Les instances d'une chronique forment sa couverture et la relation de sous-chronique définit un ordre partiel entre chroniques.

4.1.4.1 Cas des chroniques simples

La relation de sous-chronique entre chroniques simples est évaluée de la façon suivante :

Définition 4 (Relation de sous-chronique simple) *Une chronique simple $\mathcal{C} = (\mathcal{S}, \mathcal{M})$ est une sous-chronique de la chronique simple $\mathcal{C}' = (\mathcal{S}', \mathcal{M}')$ si et seulement si $\mathcal{S} \subseteq \mathcal{S}'$ et $\mathcal{M} \supseteq (\mathcal{M}'/\mathcal{S})$.*

L'opérateur $\mathcal{M}/\mathcal{S}$ extrait le sous-graphe de $\mathcal{M}$ dont les nœuds sont dans $\mathcal{S}$. Ainsi les graphes $\mathcal{M}$ et $(\mathcal{M}'/\mathcal{S})$ possèdent les mêmes nœuds. Le test d'inclusion $\subseteq$ de ces sous-graphes se fait en testant l'inclusion de chaque contrainte pour tout couple d'évènements entre les deux graphes. Ce que l'on peut écrire :

$$\mathcal{G} \subseteq \mathcal{G}' \Leftrightarrow noeuds(\mathcal{G}) = noeuds(\mathcal{G}') \\ \wedge \forall (E_1, E_2) \in noeuds(\mathcal{G})^2, contrainte(\mathcal{G}, E_1, E_2) \subseteq contrainte(\mathcal{G}', E_1, E_2)$$

où l'inclusion entre les contraintes $contrainte(\mathcal{G}, E_1, E_2)$ et $contrainte(\mathcal{G}', E_1, E_2)$ représente l'inclusion entre intervalles. Soit $[a, b] = contrainte(\mathcal{G}, E_1, E_2)$ et $[c, d] = contrainte(\mathcal{G}', E_1, E_2)$ alors $[a, b] \subseteq [c, d]$ si et seulement si $c \leq a \leq b \leq d$.

4.1.4.2 Cas des chroniques « générales »

Il peut exister plusieurs appariements possibles entre les évènements d'une chronique $\mathcal{C}$ et ceux d'une chronique $\mathcal{C}'$. Il faut donc compléter la définition 4 :

Définition 5 (Relation de sous-chronique) *Une chronique $\mathcal{C} = (\mathcal{S}, \mathcal{M})$ est une sous-chronique d'une chronique $\mathcal{C}' = (\mathcal{S}', \mathcal{M}')$ si et seulement si $\exists a \in \mathbb{A}_{\mathcal{C}}^{\mathcal{C}'}$ tel que $\mathcal{S} \subseteq \mathcal{S}'$ et $\mathcal{M} \supseteq (\mathcal{M}'/a(\mathcal{S}'))$.*

L'opérateur $\mathcal{M}/_a\mathcal{S}$ permet d'extraire le sous-graphe de $\mathcal{M}$ dont les nœuds sont dans $a(\mathcal{S})$.

La figure 4.4 illustre une relation de sous-chronique entre deux chroniques. C_1 est plus générale que C_2 car il existe un appariement entre les évènements de ces chroniques permettant d'extraire un sous graphe de C_2 plus spécifique que celui de C_1.

L'algorithme 1 teste cette relation de sous-chronique. Il recherche tous les appariements possibles et teste si l'un d'eux exhibe une relation de sous-chronique entre $\mathcal{C}$ et $\mathcal{C}'$. Si aucun appariement ne correspond à une relation de sous-chronique alors $\mathcal{C}$ n'est pas une sous-chronique de $\mathcal{C}'$.

4.1.5 Union de chroniques

L'union de deux chroniques $\mathcal{C}$ et $\mathcal{C}'$ est leur supremum (généralisé maximalement spécifique), on le définit ainsi :

Définition 6 (Supremum de deux chroniques) *Le supremum de deux chroniques $\mathcal{C}$ et $\mathcal{C}'$ est défini par*

$$\mathcal{C} \bigsqcup \mathcal{C}' = min\{\mathcal{D} | \mathcal{D} \sqsubseteq \mathcal{C} \wedge \mathcal{D} \sqsubseteq \mathcal{C}'\}$$

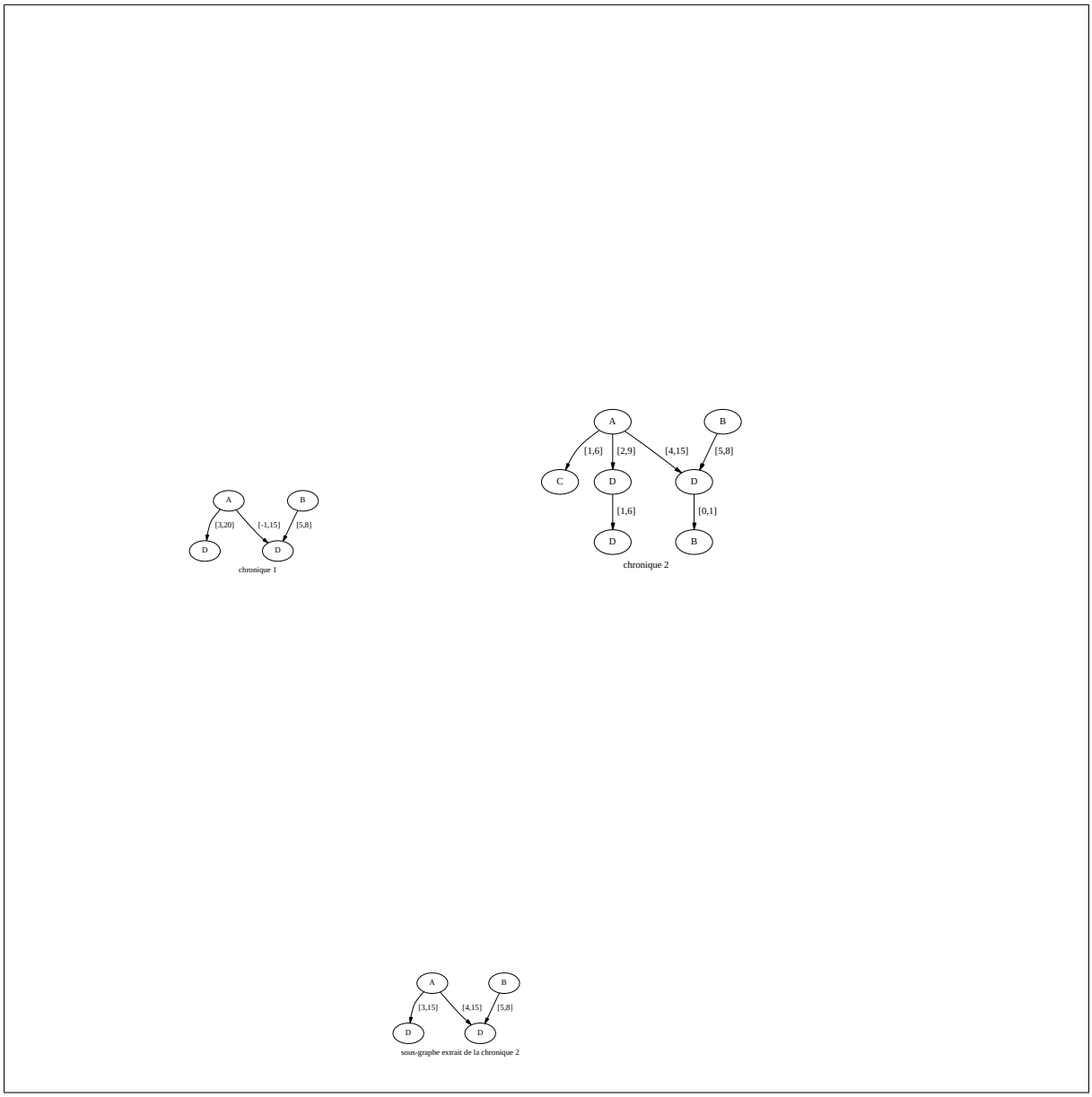


FIG. 4.4 – Une relation de sous-chronique entre deux chroniques : $C_1 \sqsubseteq C_2$

Algorithme 1: Sous-chronique

Entrées : $\mathcal{C} = (\mathcal{S}, \mathcal{M}), \mathcal{C}' = (\mathcal{S}', \mathcal{M}')$
Sorties : si $\mathcal{C} \sqsubseteq \mathcal{C}'$ alors vrai sinon faux
$A = \emptyset$
pour chaque $A = appariement\ entre\ \mathcal{S}\ et\ \mathcal{S}'$ **faire**
  $appariementOk = vrai$
  pour chaque $contrainte \in \mathcal{M} \wedge appariementOk$ **faire**
    $appariementOk = \mathcal{M}'[A[contrainte.NoeudDebut], A[contrainte.NoeudFin]] \subseteq contrainte$
  si $appariementOk$ **alors**
    $retourne\ vrai$
$retourne\ faux$

4.1.5.1 Cas des chroniques simples

L'union de deux chroniques $\mathcal{C}$ et $\mathcal{C}'$ est un ensemble de chroniques qui sont plus générales que ces deux chroniques. Les chroniques de cet ensemble doivent couvrir toutes les instances de $\mathcal{C}$ et $\mathcal{C}'$ et être le plus spécifique possible. Ainsi, leur ensemble d'évènements correspond à l'intersection des ensembles d'évènements de $\mathcal{C}$ et $\mathcal{C}'$ et leur graphe de contrainte à l'union des sous-graphes de $\mathcal{C}$ et $\mathcal{C}'$ par rapport à l'intersection de leurs évènements. Or, il n'existe qu'une chronique satisfaisant ces conditions. L'union de deux chroniques simples est donc un monôme. La proposition 1 détaille formellement ce calcul.

Pour l'expliquer, on peut voir, en (4.1), l'équivalence entre la proposition 1 et la définition 6. On voit que cette équivalence est vraie car il n'existe pas de chronique strictement plus spécifique que $\mathcal{D}$ et plus générale que $\mathcal{C}'$ et $\mathcal{C}$. En effet, si on ajoute un évènements à $\mathcal{S}_u$ ou si l'on réduit un intervalle du graphe $\mathcal{M}_u$, on ne généralise plus $\mathcal{C}'$ ou $\mathcal{C}$. Ainsi, dans le cas de chroniques simples, l'union de deux chroniques est une unique chronique simple.

Proposition 1 (Union de chroniques simples) *Soit deux chroniques simples $\mathcal{C} = (\mathcal{S}, \mathcal{M})$ et $\mathcal{C}' = (\mathcal{S}', \mathcal{M}')$, on définit leur union par*

$$\mathcal{C} \bigcup \mathcal{C}' = \{\mathcal{D}\}$$

$$où\ \mathcal{D} = (\mathcal{S}_u, \mathcal{M}_u), tel\ que\ \mathcal{S}_u = \mathcal{S} \bigcap \mathcal{S}'\ et\ \mathcal{M}_u = (\mathcal{M}/\mathcal{S}_u) \bigcup (\mathcal{M}'/\mathcal{S}_u)$$

$$\begin{aligned} \{\mathcal{D}\} = \mathcal{C} \bigcup \mathcal{C}' \Leftrightarrow & \mathcal{D} \sqsubseteq \mathcal{C} \wedge \\ & \mathcal{D} \sqsubseteq \mathcal{C}' \wedge \\ & (\neg \exists \mathcal{D}' : \mathcal{D} \sqsubset \mathcal{D}' \wedge \mathcal{D}' \sqsubseteq \mathcal{C} \wedge \mathcal{D}' \sqsubseteq \mathcal{C}') \end{aligned} \tag{4.1}$$

L'exemple de la figure 4.5 illustre une union de chroniques simples.

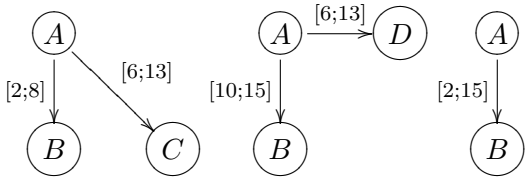


FIG. 4.5 – Union de chroniques (la chronique de droite est le seul élément de l'union des deux autres)

4.1.5.2 Cas des chroniques « générales »

On voit dans la proposition 1 que l'on calcule l'intersection des évènements de $\mathcal{C}$ et $\mathcal{C}'$ pour obtenir $\mathcal{S}_u$. Or, il n'existe qu'une seule manière de faire l'intersection des évènements de deux chroniques. L'intersection de multi-ensembles est définie de la façon suivante :

Définition 7 (Intersection de multi-ensembles) *Les éléments de $S \bigcap T$ sont ceux qui sont à la fois dans les multi-ensembles S et T. Si un élément apparaît plus d'une fois dans S ou T (ou les deux), l'intersection contient m copies de cet élément, où m est le plus petit nombre de fois où cet élément apparaît dans S ou T.*

Ainsi, toutes les chroniques $\mathcal{D} = (\mathcal{S}_u, \mathcal{M}_u)$, union de $\mathcal{C}$ et $\mathcal{C}'$, possèdent les mêmes évènements. Par exemple, si $\mathcal{S} = \{A, B, B, C, D, D, D\}$ et $\mathcal{S}' = \{A, A, A, A, B, D, D\}$ alors $\mathcal{S}_u = \{A, B, D, D\}$.

Par contre, le problème d'appariement d'évènements rencontré précédemment en 4.1.4.2 implique qu'il peut exister différents graphes $\mathcal{M}_u$. En effet, chaque évènement de la chronique union $\mathcal{D}$ doit correspondre à un évènement de la chronique $\mathcal{C}$ et un de la chronique $\mathcal{C}'$. Or, les évènements de la chronique union sont appariables de différentes façons avec, d'une part, les évènements de la chronique $\mathcal{C}$ et, d'autre part, les évènements de $\mathcal{C}'$.

Proposition 2 (Union de chroniques) *Soit deux chroniques $\mathcal{C} = (\mathcal{S}, \mathcal{M})$ et $\mathcal{C}' = (\mathcal{S}', \mathcal{M}')$ et $\mathcal{S}_u = \mathcal{S} \bigcap \mathcal{S}'$. On définit leur union par*

$$\mathcal{C} \bigcup \mathcal{C}' = min\{\mathcal{D} | \exists a_{\mathcal{C}} \in \mathbb{A}^{\mathcal{C}}_{\mathcal{S}_u}, \exists a_{\mathcal{C}'} \in \mathbb{A}^{\mathcal{C}'}_{\mathcal{S}_u} : \mathcal{D} = (\mathcal{S}_u, (\mathcal{M}/_{a_{\mathcal{C}}}\mathcal{S}_u) \bigcup (\mathcal{M}'/_{a_{\mathcal{C}'}}\mathcal{S}_u))\}$$

Certains appariements vont conduire à générer des chroniques union plus générales que d'autres chroniques union. Or, les chroniques union doivent être maximalement spécifiques, il faut donc filtrer certaines chroniques trop générales. C'est ce que fait l'opérateur *min*. Ainsi l'union de deux chroniques sera bien un ensemble de chroniques plus générales que $\mathcal{C}$ et $\mathcal{C}'$ et maximalement spécifiques.

L'exemple de la figure 4.6 montre tous les appariements possibles des évènements de $\mathcal{C}$ et $\mathcal{C}'$ (cités plus haut) avec une chronique union. En particulier, on peut voir que l'évènement A de la chronique union est appariable d'une seule façon avec la chronique $\mathcal{C}$ et de quatre façons différentes avec la chronique $\mathcal{C}'$.

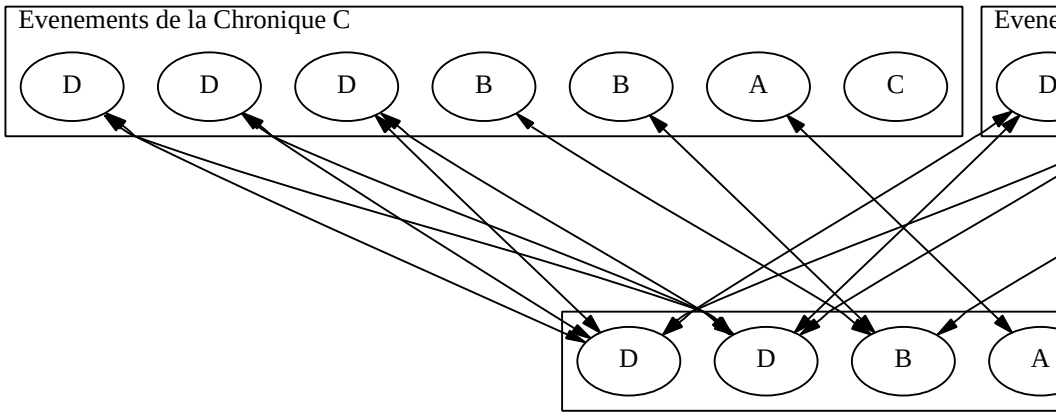


FIG. 4.6 – Appariements des évènements de 2 chroniques pour former une chronique union

$\mathbb{A}_{\mathcal{C}}^{\mathcal{S}_u}$	Évènements de $\mathcal{S}_u$			
T_i : Évènements de $\mathcal{C}$	A	B	D	D
$a_{\mathcal{S}1}$	A_1	B_1	D_1	D_2
$a_{\mathcal{S}2}$	A_1	B_1	D_1	D_3
$a_{\mathcal{S}3}$	A_1	B_1	D_2	D_1
$a_{\mathcal{S}4}$	A_1	B_1	D_2	D_3
$a_{\mathcal{S}5}$	A_1	B_1	D_3	D_1
$a_{\mathcal{S}6}$	A_1	B_1	D_3	D_2

$\mathbb{A}_{\mathcal{C}'}^{\mathcal{S}_u}$	Évènements de $\mathcal{S}_u$			
T_i : Évènements de $\mathcal{C}'$	A	B	D	D
$a_{\mathcal{S}'1}$	A_1	B_1	D_1	D_2
$a_{\mathcal{S}'2}$	A_1	B_1	D_2	D_1
$a_{\mathcal{S}'3}$	A_2	B_1	D_1	D_2
$a_{\mathcal{S}'4}$	A_2	B_1	D_2	D_1
$a_{\mathcal{S}'5}$	A_3	B_1	D_1	D_2
$a_{\mathcal{S}'6}$	A_3	B_1	D_2	D_1
$a_{\mathcal{S}'7}$	A_4	B_1	D_1	D_2
$a_{\mathcal{S}'8}$	A_4	B_1	D_2	D_1

FIG. 4.7 – Listes des appariements de $\mathcal{C}$ et $\mathcal{C}'$ avec $\mathcal{D}$

Un résultat simple est qu'il existe autant de chroniques union potentielles de $\mathcal{C}$ et $\mathcal{C}'$ que de combinaisons d'appariements possibles. Sur la figure 4.7, on a représenté à gauche l'ensemble $\mathbb{A}_{\mathcal{C}}^{\mathcal{S}_u}$ complet des appariements possibles de la chronique $\mathcal{C}$ avec l'union et à droite $\mathbb{A}_{\mathcal{C}'}^{\mathcal{S}_u}$ de $\mathcal{C}'$ avec l'union. Ce qui donne 6 x 8 = 48 chroniques union potentielles pour ce simple exemple.

L'algorithme 2 construit toutes les chroniques union des deux chroniques $\mathcal{C}$ et $\mathcal{C}'$. Tout d'abord, il construit l'ensemble $\mathcal{S}_u$ des évènements contenus dans toutes les chroniques union (1), ensuite pour tous les appariements possibles il construit un nouveau graphe de contraintes $\mathcal{M}_u$ en unifiant (4) les contraintes correspondant aux deux appariements choisis (2 et 3). Puis si la nouvelle chronique est cohérente (5), et si elle n'est pas plus générale qu'une chronique déjà trouvée alors il l'ajoute à l'ensemble des chroniques union (7). De plus si des chroniques sont plus générales que la nouvelle chronique, elles sont effacées (6).

4.1.5.3 Un résultat simple

Si une chronique est plus générale que toutes les chroniques d'un ensemble alors elle est aussi plus générale que l'union des chroniques de cet ensemble. Ce qu'on écrit de la façon suivante :

Proposition 3 *Soit un ensemble E de chroniques et deux chroniques $\mathcal{C}$ et $\mathcal{D}$,*

$$\left(\mathcal{C} = \bigcup_{\mathcal{C}' \in E} \mathcal{C}' \wedge (\forall \mathcal{C}' \in E, \mathcal{D} \sqsubseteq \mathcal{C}')\right) \Rightarrow \mathcal{D} \sqsubseteq \mathcal{C}$$

4.2 Traitement de requêtes sur des séquences temporelles

Une base de données inductive contient à la fois les données et les motifs. Par conséquent la base de données inductive étendue au temps contient des séquences d'évènements et des chroniques. La principale préoccupation de l'application que nous mettrons en œuvre est la recherche de chroniques fréquentes dans certaines séquences et non fréquentes dans d'autres. Par conséquent, nous allons voir dans cette partie la définition d'une requête permettant d'effectuer cette recherche. Ensuite nous verrons diverses propositions pour calculer ses solutions.

Algorithme 2: Union de chroniques

Entrées : $\mathcal{C} = (\mathcal{S}, \mathcal{M}), \mathcal{C}' = (\mathcal{S}', \mathcal{M}')$

Sorties : $UnionSet = \mathcal{C} \bigcup \mathcal{C}'$

$UnionSet = \emptyset$

1 $\mathcal{S}_u = \mathcal{S} \bigcap \mathcal{S}'$

2 **pour chaque** $a_{\mathcal{C}} \in \mathbb{A}^{\mathcal{C}}_{\mathcal{S}_u}$ **faire**

3 **pour chaque** $a_{\mathcal{C}'} \in \mathbb{A}^{\mathcal{C}'}_{\mathcal{S}_u}$ **faire**

$\mathcal{M}_u = \emptyset$

pour chaque $event_X \in \mathcal{S}_u$ **faire**

pour chaque $event_Y \in \mathcal{S}_u$ **faire**

4 $\mathcal{M}_u = \mathcal{M}_u + (\mathcal{M}[a_{\mathcal{C}}[event_X], a_{\mathcal{C}}[event_Y]] \bigcup \mathcal{M}'[a_{\mathcal{C}'}[event_X], a_{\mathcal{C}'}[event_Y]])$

5 **si** $\mathcal{M}_u$ *est cohérent* **alors**

$\mathcal{D} = (\mathcal{S}_u, \mathcal{M}_u)$

$ajoute\mathcal{D} = vrai$

pour chaque $\mathcal{U} \in UnionSet \wedge ajoute\mathcal{D}$ **faire**

si $\mathcal{U} \sqsubseteq \mathcal{D}$ **alors**

6 $Enleve\ \mathcal{U}\ de\ UnionSet$

sinon si $\mathcal{D} \sqsubseteq \mathcal{U}$ **alors**

$ajoute\mathcal{D} = faux$

si $ajoute\mathcal{D}$ **alors**

7 $UnionSet = UnionSet + \mathcal{D}$

retourne UnionSet

4.2.1 Définition de la requête type

On suppose que l'on dispose de deux ensembles de séquences d'évènements. L'un deux, $\mathcal{P}_{\mathcal{L}}$, contient les séquences dont nous souhaitons extraire les chroniques fréquentes tandis que l'autre, $\mathcal{N}_{\mathcal{L}}$, contient les séquences dont on recherche les chroniques non fréquentes. Avant tout, il faut poser le type de requête à laquelle on souhaite que la BDI réponde :

$$R_q = \left(\forall \mathcal{L} \in \mathcal{P}_{\mathcal{L}}, freq(\mathcal{C}, \mathcal{L}) \geq S_{\mathcal{L}}\right) \wedge \left(\forall \overline{\mathcal{L}} \in \mathcal{N}_{\mathcal{L}}, freq(\mathcal{C}, \overline{\mathcal{L}}) \leq S_{\overline{\mathcal{L}}}\right) \tag{4.2}$$

tel que chaque séquence d'évènements $\mathcal{L}$ (resp. $\overline{\mathcal{L}}$) est associée à un seuil $S_{\mathcal{L}}$ (resp. $S_{\overline{\mathcal{L}}}$) fixé en fonction de la requête. En pratique, ces seuils dépendent des données, des chroniques recherchées, des séquences d'évènements, etc. On peut remarquer que $freq(\mathcal{C}, \mathcal{L}) \geq S_{\mathcal{L}}$ est une sous-requête anti-monotone tandis que $freq(\mathcal{C}, \overline{\mathcal{L}}) \leq S_{\overline{\mathcal{L}}}$ est une sous-requête monotone.

4.2.2 Propositions pour un calcul des solutions

L'aspect temporel des motifs recherchés introduit une complexité supplémentaire dans le calcul des solutions aux requêtes. En effet, il ne s'agit plus comme dans [LD03] de rechercher des ensembles de symboles dans une séquence d'évènements mais aussi d'inclure le temps comme attribut *numérique* entre ces symboles.

On peut considérer deux approches pour le calcul des solutions de la requête énoncée précédemment. À partir des résultats des sous-requêtes $freq(\mathcal{C}, \mathcal{L}) \geq S$, il est possible de calculer les solutions. Cette méthode utilisée sur la découverte de séquences dans [LD03] est démontré dans le chapitre suivant. On peut facilement imaginer calculer les solutions de ces sous-requêtes, les enregistrer et les réutiliser en fonction des interrogations de l'utilisateur. Ce principe correspond bien à celui des bases de données inductives.

L'autre façon de faire est d'envisager la requête globalement. En utilisant de manière intelligente les sous-requêtes monotones et anti-monotones, on peut espérer trouver les solutions à la requête considérée plus rapidement qu'en calculant chaque sous-requête séparément puis en combinant leur résultat. Dans l'article [BGKW02], les auteurs utilisent cette solution pour trouver les itemsets solutions d'une conjonction de contraintes monotones et anti-monotones.

Dans un première étape, nous avons préféré porter notre attention sur la première solution car d'une part on souhaite intégrer FACE dans notre BDI et d'autre part on espère déceler et résoudre les problèmes que posent cette solution qui apparaît à première vue plus simple. Les chapitres suivants font état de l'examen de celle-ci. L'examen de la seconde n'est pas réalisé dans ce travail, faute de temps.

4.2.3 Calcul des solutions à partir de l'algorithme de *Mellish*

On souhaite, à partir des chroniques fréquentes de différentes séquences d'évènements, extraire les chroniques fréquentes dans certaines séquences mais non fréquentes dans les autres. La monotonie et l'anti-monotonie des sous-requêtes de la requête générale permettent de calculer les bords de l'espace des solutions car on sait qu'ils induisent l'ensemble des solutions (voir 3.2.2).

Tout d'abord, nous allons voir la définition du bord de l'espace des solutions. Ensuite, nous verrons comment passer de la requête considérée en (4.2) à une autre représentation permettant d'utiliser l'algorithme de Mellish [DK01].

4.2.3.1 Chroniques fréquentes

On travaille à partir d'ensembles de chroniques fréquentes dans différentes séquences d'évènements. On définit cet ensemble de la façon suivante :

Définition 8 (Chroniques fréquentes) *À partir d'une séquence d'évènements $\mathcal{L}$, et d'un seuil S, on définit l'ensemble $Sol(\mathcal{L}, S)$ des chroniques de fréquence supérieure ou égale à un seuil S dans $\mathcal{L}$.*

$$Sol(\mathcal{L}, S) = \{\mathcal{C} : freq(\mathcal{C}, \mathcal{L}) \geq S\}$$

L'ensemble des chroniques de $Sol(\mathcal{L}, S)$ est infini car la requête $freq(\mathcal{C}, \mathcal{L})$ est anti-monotone et ainsi toute chronique plus générale qu'une chronique fréquente est aussi solution. Ainsi le calcul complet de cet ensemble est irréalisable. Parmi les chroniques fréquentes, il existe un ensemble de chroniques maximalement spécifiques, c'est à dire qu'il n'existe pas de chroniques solutions plus spécifiques que l'une d'elle. On arrive ainsi à la définition suivante :

Définition 9 (Chroniques maximalement spécifiques et fréquentes) *L'ensemble $B_{F_{\mathcal{L}} \geq S}$ des chroniques maximalement spécifiques et de fréquence supérieure ou égale à S dans la séquence d'évènements $\mathcal{L}$ est défini relativement à l'ensemble des chroniques solutions de la façon suivante :*

$$\forall \mathcal{C}_S (\mathcal{C}_S \in Sol(\mathcal{L}, S) \Leftrightarrow \exists \mathcal{B} \in B_{F_{\mathcal{L}} \geq S}, \mathcal{C}_S \sqsubseteq \mathcal{B}$$
$$Soit,\ B_{F_{\mathcal{L}} \geq S} = min(Sol(\mathcal{L}, S))$$

L'ensemble $\mathsf{B}_{F_{\mathcal{L}} \geq S}$ est l'ensemble $S - set$ de l'espace des solutions $Sol(\mathcal{L}, S)$.

La plupart des algorithmes [BCG01] de recherche de motifs fréquents dans de grandes bases de données recherchent les motifs fréquents maximalement spécifiques car les motifs fréquents peuvent s'en déduire. Nous posons donc comme hypothèse l'existence d'un tel algorithme pour rechercher les chroniques fréquentes maximalement spécifiques dans une séquence d'évènements. Cet algorithme sera discuté dans le chapitre 5.

4.2.3.2 Réécriture de la requête générale

$\mathcal{C}$ est une chronique solution de la sous-requête anti-monotone $freq(\mathcal{C}, \mathcal{L}) \geq S$ si et seulement si $\exists \mathcal{B} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}, \mathcal{C} \sqsubseteq \mathcal{B}$ (définition 9), ce que l'on peut écrire :

$$\exists \mathcal{B} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}, \mathcal{C} \sqsubseteq \mathcal{B} \Leftrightarrow \mathcal{C} \sqsubseteq \mathcal{B}_1 \vee \mathcal{C} \sqsubseteq \mathcal{B}_2 \vee \ldots \vee \mathcal{C} \sqsubseteq \mathcal{B}_n | \mathcal{B}_i \in \mathsf{B}_{F_{\mathcal{L}} \geq S}$$
$$\Leftrightarrow \bigvee_{\mathcal{B} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}} \mathcal{C} \sqsubseteq \mathcal{B}$$

De même pour une sous-requête monotone $freq(\mathcal{C}, \overline{\mathcal{L}}) \leq S$, $\mathcal{C}$ est une chronique solution si et seulement si $\forall \overline{\mathcal{B}} \in \mathsf{B}_{F_{\overline{\mathcal{L}}} \geq S}, \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}}$. On réécrit cette sous-requête ainsi :

$$\forall \overline{\mathcal{B}} \in \mathsf{B}_{F_{\overline{\mathcal{L}}} \geq S}, \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}} \Leftrightarrow \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}_1} \wedge \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}_2} \wedge \ldots \wedge \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}_n} | \overline{\mathcal{B}_i} \in \mathsf{B}_{F_{\overline{\mathcal{L}}} \geq S}$$
$$\Leftrightarrow \bigwedge_{\overline{\mathcal{B}} \in \mathsf{B}_{F_{\overline{\mathcal{L}}} \geq S}} \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}}$$

On peut maintenant exprimer la requête (4.2) de la façon suivante :

$$R_q = \Big(\bigvee_{\mathcal{B} \in \mathsf{B}_{F \geq S}} \mathcal{C} \sqsubseteq \mathcal{B} \Big) \wedge \Big(\bigwedge_{\overline{\mathcal{B}} \in \overline{\mathsf{B}}_{F \geq S}} \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}} \Big) \tag{4.3}$$

Où $\mathsf{B}_{F \geq S} = \bigcup \mathsf{B}_{F_{\mathcal{L}} \geq S_{\mathcal{L}}}$ et $\overline{\mathsf{B}}_{F \geq S} = \bigcup \mathsf{B}_{F_{\overline{\mathcal{L}}} \geq S_{\overline{\mathcal{L}}}}$.

On voit qu'à partir d'ensembles de chroniques maximalement spécifiques et de fréquence supérieure ou égale à un seuil dans une séquence d'évènements, on peut extraire, par interrogation, des chroniques caractérisant des classes de séquences d'évènements.

4.2.3.3 Calcul des solutions

De Raedt et Kramer [DK01] proposent de calculer les bords de l'ensemble des solutions (l'espace des versions) à partir de conjonctions de contraintes simples telles que $\mathcal{C} \sqsubseteq \mathcal{D}, \mathcal{D} \sqsubseteq \mathcal{C}, \mathcal{C} \not\sqsubseteq \mathcal{D}$, *et* $\mathcal{D} \not\sqsubseteq \mathcal{C}$, où $\mathcal{D}$ est une chronique donnée et $\mathcal{C}$ une chronique recherchée. On peut réécrire la requête (4.3) de façon à obtenir une disjonction de conjonctions de contraintes :

$$R_q = \bigvee_{\mathcal{B} \in \mathsf{B}_{F \geq S}} \Big(\mathcal{C} \sqsubseteq \mathcal{B} \bigwedge_{\overline{\mathcal{B}} \in \overline{\mathsf{B}}_{F \geq S}} \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}} \Big) \tag{4.4}$$

On obtient ainsi $|\mathsf{B}_{F \geq S}|$ calculs d'espaces de versions qu'il suffira d'unifier pour obtenir l'espace des versions correspondants à la requête R_q (4.2) .

Calcul des bords de l'espace des versions pour une conjonction de contraintes L'algorithme de Mellish [DK01] calcule les bords S et G de l'espace des versions d'une conjonction de contraintes. On peut ainsi calculer l'espace des versions de la requête suivante :

$$\mathcal{C} \sqsubseteq \mathsf{B}^i_{F \geq S} \bigwedge_{\overline{\mathcal{B}} \in \overline{\mathsf{B}}_{F \geq S}} \mathcal{C} \not\sqsubseteq \overline{\mathcal{B}}.$$

C'est un algorithme incrémental de construction des bords S et G de l'espace des versions. Dans notre cas, l'algorithme utilise deux opérations de bases : l'union de chroniques déjà expliquée en 4.1.5 et l'opérateur *mgs* « the smallest fragments more specific than f_1 but not more general than f_2 » [DK01]. Formellement on définit cet opérateur de la façon suivante : $mgs(\mathcal{C}_1, \mathcal{C}_2) = max\{\mathcal{C} | \mathcal{C}_1 \sqsubseteq \mathcal{C} \wedge \mathcal{C} \not\sqsubseteq \mathcal{C}_2\}$, où max est l'opérateur défini en 4.1.1.3. L'algorithme 3 reprend l'algorithme de Mellish avec nos notations.

Algorithme 3: Algorithme de Mellish

Entrées : $E_{C_{sr}}$: un ensemble de contraintes

Sorties : (S, G) : l'espace des versions correspondant aux solutions de $\bigwedge_{C_{sr} \in E_{C_{sr}}} C_{sr}$
S : Les chroniques cohérentes maximalement spécifiques
G : Les chroniques cohérentes maximalement générales

$S = \{\bot\}$
$G = \{\top\}$
pour chaque $C_{sr} \in E_{C_{sr}}$ **faire**
  cas où C_{sr} *de type* $\mathcal{C} \sqsubseteq \mathcal{D}$
    $G = \{g \in G | g \sqsubseteq \mathcal{D}\}$
    $S = min\{c | c \in union(\mathcal{D}, s) \wedge s \in S \wedge \exists g \in G : g \sqsubseteq c\}$
  cas où C_{sr} *de type* $\mathcal{C} \not\sqsubseteq \mathcal{D}$
    $S = \{s \in S | s \not\sqsubseteq \mathcal{D}\}$
1   $G = max\{c | \exists g \in G : c \in mgs(g, \mathcal{D}) \wedge \exists s \in S : c \sqsubseteq s\}$

L'opérateur mgs n'est pas instanciable dans le cas des chroniques. En effet, il est nécessaire de disposer d'un opérateur de raffinement sur $\mathcal{C}_1$ qui produit un ensemble de chroniques. À partir de ces chroniques, il faut enlever celles qui sont plus générales que $\mathcal{C}_2$. Or, un tel opérateur de raffinement est, pour les chroniques, extrêmement complexe vu la combinatoire des possibilités de raffinement : ajouter des évènements, réduire les intervalles des contraintes,etc.

Nous allons définir un nouvel opérateur nommé *drc* : *opérateur dirigé de raffinement sous contrainte*. (4.5) reprend la ligne 1 de l'algorithme 3, on réécrit en (4.6) l'opérateur mgs, puis on y introduit l'opération $c \sqsubseteq s$ en (4.7), ce qui définit le nouvel opérateur : *drc*.

$$max\{c | \exists g \in G : c \in mgs(g, \mathcal{D}) \wedge \exists s \in S : c \sqsubseteq s\} \tag{4.5}$$
$$max\{c | \exists g \in G : c \in max\{\mathcal{C} | g \sqsubseteq \mathcal{C} \wedge \mathcal{C} \not\sqsubseteq \mathcal{D}\} \wedge \exists s \in S : c \sqsubseteq s\} \tag{4.6}$$
$$max\{c | \exists g \in G, \exists s \in S : c \in max\{\mathcal{C} | g \sqsubseteq \mathcal{C} \wedge \mathcal{C} \not\sqsubseteq \mathcal{D} \wedge \mathcal{C} \sqsubseteq s\}\} \tag{4.7}$$
$$max\{c | \exists g \in G, \exists s \in S : c \in drc(g, s, \mathcal{D})\},\ avec \tag{4.8}$$
$$drc(g, s, d) = max\{\mathcal{C} | g \sqsubseteq \mathcal{C} \wedge \mathcal{C} \not\sqsubseteq d \wedge \mathcal{C} \sqsubseteq s\} \tag{4.9}$$

À partir d'une chronique g plus générale qu'une chronique s, l'opérateur *drc* spécialise g en « allant » vers s jusqu'à ce que les chroniques trouvées ne soient pas plus générales que d. Un inconvénient de cet opérateur est qu'on doit avoir $s \neq \bot$. En effet, si $s = \bot$ alors g n'a pas de « direction » pour se spécialiser et on revient dans le cas d'un opérateur de raffinement classique trop complexe. On voit sur l'algorithme 3 que $S = \{\bot\}$ si aucune contrainte de type $\mathcal{C} \sqsubseteq \mathcal{D}$ n'a été traitée avant une contrainte de type $\mathcal{C} \not\sqsubseteq \mathcal{D}$. Ce problème est résolu car pour chaque espace de versions à calculer, on dispose d'une contrainte de type $\mathcal{C} \sqsubseteq \mathcal{D}$ qu'il suffira de traiter avant toutes les autres contraintes.

Nous venons de décrire comment à partir des chroniques maximalement fréquentes dans différentes séquences d'évènements, on peut trouver les chroniques caractérisant certaines de

ces séquences. Les chroniques ont été vues comme des objets ponctuels, nous allons voir dans la partie suivante l'espace dans lequel se placent effectivement les chroniques intéressantes.

4.3 Structuration de l'espace de recherche

Structurer l'espace de recherche est une étape importante dans l'élaboration d'un algorithme. Les polytopes, qui sont des figures géométriques, sont largement étudiés pour la résolution de programmes à base de contraintes linéaires. Le livre [Zie98] détaille la plupart des aspects intéressants d'une telle structure. Un second ouvrage [Mar95] traite des compositions de polytopes de manière détaillée.

Nous allons voir dans ce chapitre la relation étroite qui existe entre les chroniques et les polytopes, puis nous aborderons les avantages et les inconvénients des espaces de recherche identifiés.

4.3.1 Chronique sous forme de polytope

Un *polyèdre* est la représentation du domaine des *solutions admissibles* d'un programme de contraintes linéaires. Celui-ci s'énonce de la façon suivante :

$$\begin{aligned} &maximiser\ c\ x \\ &sous\ contrainte\ A\ x \geq b \end{aligned}$$

où c et x sont des vecteurs à n composantes, b un vecteur à m composantes et A une matrice à m lignes et n colonnes. Ainsi, le domaine D des solutions admissibles pour un tel problème est formé par

$$D = \{x \in \mathbb{R}^n | A\ x \geq b\}$$

Ce domaine D est obtenu par intersection d'un nombre fini de demi-espaces de $\mathbb{R}^n$ (chacun de ces demi-espaces correspondant à une des lignes du système matriciel $A\ x \geq b$) et constitue ce que l'on nomme polyèdre. Un polyèdre borné, i.e. un polyèdre pour lequel il existe un nombre B tel que chaque point du polyèdre a des coordonnées comprises entre $-B$ et B, est un *polytope*.

Une chronique est un ensemble d'évènements contraints temporellement, autrement dit pour chaque couple d'événements (A, t_A), (B, t_B), il existe une contrainte $AB_{min} \leq t_B - t_A \leq AB_{max}$. On voit qu'une chronique peut-être interprétée comme le domaine d'un programme de contraintes linéaires.

Pour chacun des couples d'évènements d'une chronique, on pose $X_{AB} = t_B - t_A$, ce qui donne $AB_{min} \leq X_{AB} \leq AB_{max}$. Chaque variable X est donc bornée ce qui permet de dire qu'une chronique est un polytope. De plus, chacune de ces variables obéit à plusieurs égalités triangulaires. Une chronique est donc équivalente au système d'inéquations et d'équations suivant :

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & -1 \end{pmatrix} . \begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_n \end{pmatrix} \geq \begin{pmatrix} 0_{min} \\ -0_{max} \\ 1_{min} \\ -1_{max} \\ \vdots \\ n_{min} \\ -n_{max} \end{pmatrix} \tag{4.10}$$

$$g \text{ équations} \begin{cases} X_0 + X_1 = X_2 \\ X_1 + X_3 = X_4 \\ \cdots \end{cases} \tag{4.11}$$

Dans ces systèmes d'équations et d'inéquations, n caractérise le nombre de couples d'évènements à partir d'une chronique de e évènements. On obtient $n = \frac{e^2-e}{2}$ couples d'évènements pour une telle chronique. Les équations représentent toutes les triangulations linéairement indépendantes qu'il est possible de proposer sur le graphe de la chronique considérée (ABC,ABD,BCD,ACD ....).

D'une part, en examinant le système d'inéquations (4.10), on voit que l'ensemble des points satisfaisant ce système forme un hypercube dans l'espace $\mathbb{R}^n$. Chaque axe de cet espace est associé à une variable X_i bornée entre i_{min} et i_{max}. D'autre part, le système de g équations (4.11) définit un sous-espace vectoriel de dimension $d = n$ - g. Ce sous-espace vectoriel décrit les contraintes que respectent les instances des chroniques possédant les e évènements. Pour en avoir une idée plus claire, il faut définir la chronique associée à une instance.

Une instance c d'une chronique est un ensemble d'occurrences d'évènement. On peut voir cette instance comme un graphe dont les nœuds représentent les évènements associées aux occurrences des évènements de l'instance et les arcs représentent les différences entre les dates de ces occurrences. La figure 4.8 illustre un exemple d'instance de chronique traduite en un graphe.

Définition 10 (Chronique maximalement spécifique associée à une instance) *On définit la chronique $\mathcal{C}_i$ maximalement spécifique d'une instance i par :*

$$\mathcal{C}_i = min\{\mathcal{C} | \mathcal{C} \text{ couvre } i\}$$

Par cette définition, on voit que ce sous-espace vectoriel correspond à l'espace des instances (ou plutôt des chroniques associées à ces instances) de chroniques ayant la même structure (mêmes évènements). Ainsi une chronique peut-être représentée par l'intersection d'un hypercube de dimension n et d'un sous-espace de dimension d. Ce qui prouve bien qu'une chronique est un polytope.

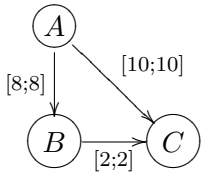


FIG. 4.8 – Graphe de l'instance [(A,12),(B,20),(C,22)]

Exemple Prenons, par exemple, les chroniques composées des trois évènements (A, t_A), (B, t_B), et (C, t_C). Pour chaque couple d'évènements on obtient une variable représentant la durée entre ces deux évènements, soit : $X_{AB} = t_B - t_A$, $X_{AC} = t_C - t_A$ et $X_{BC} = t_C - t_B$. Ce qui nous donne les inéquations et équations suivantes :

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} X_{AB} \\ X_{AC} \\ X_{BC} \end{pmatrix} \geq \begin{pmatrix} AB_{min} \\ -AB_{max} \\ AC_{min} \\ -AC_{max} \\ BC_{min} \\ -BC_{max} \end{pmatrix} \tag{4.12}$$

$$\{X_{AB} + X_{BC} = X_{AC} \tag{4.13}$$

Le plan engendré par l'équation (4.13) a un vecteur normal : $\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$, ce plan est donc engendré par les deux vecteurs $\begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$, et $\begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$.

On voit sur la figure 4.9 l'intersection entre l'hypercube qui représente les bornes de chaque intervalle de la chronique de la figure 4.10 avec l'espace des instances représenté par le plan.

Algorithme de Floyd-Warshall Il est intéressant de noter que l'algorithme de Floyd-Warshall consiste à partir d'un hypercube englobant une certaine surface de l'espace des instances, à trouver le plus petit cube englobant cette même surface. Une chronique est dite incohérente si l'intersection avec l'espace des instances est nulle. La Figure 4.11 reprend les chroniques de la figure 4.2.

4.3.2 Dimensions des espaces

La dimension de l'espace des hypercubes correspond au nombre de couples d'évènements au sein d'une même structure de chronique. Cette dimension est supérieure à celle de l'espace des

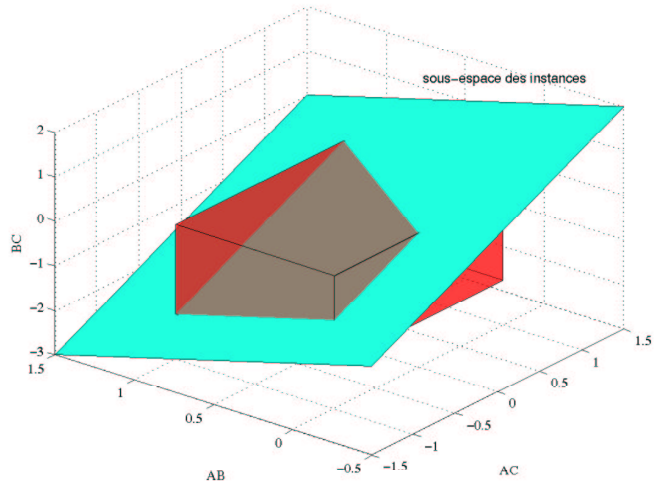


FIG. 4.9 – Une chronique sous forme de polytope

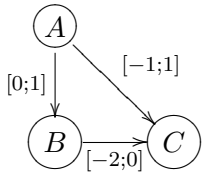


FIG. 4.10 – Une chronique à 3 évènements

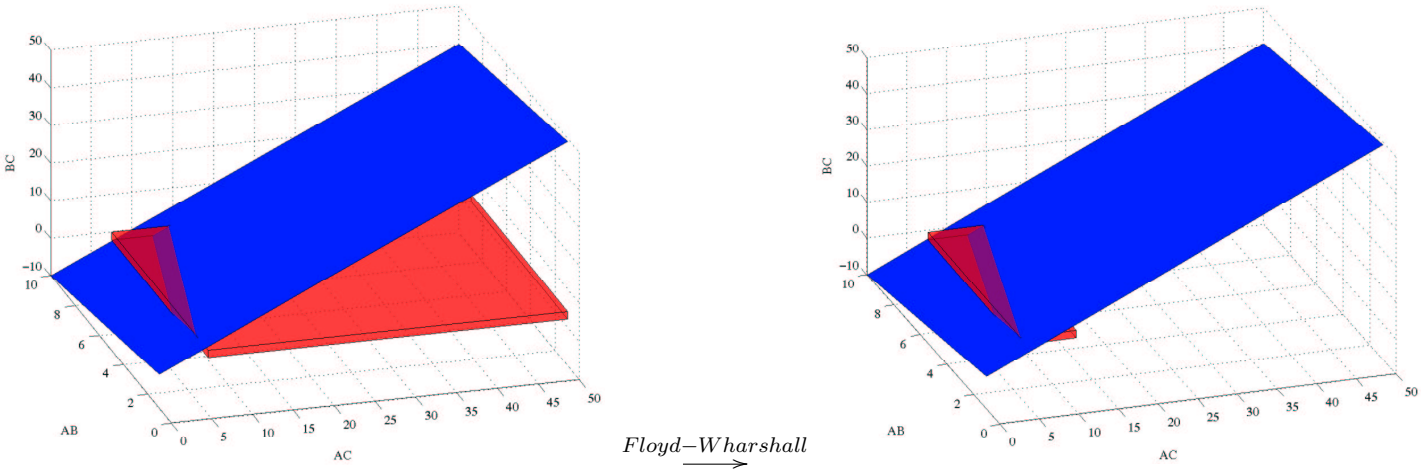


FIG. 4.11 – Graphe minimal d'une chronique pour la même couverture des instances

instances d'une chronique. En effet, le graphe d'une instance i (sa chronique associée $\mathcal{C}_i$) à e occurrences d'évènement possède $e-1$ degrés de liberté. En effet, si l'on prend une occurrence d'évènement de cette instance comme point de référence, la description de cette instance consiste à lister la différence entre toutes les occurrences d'évènement et ce point de référence, soit un vecteur à $d = e-1$ composantes. Ce qui nous donne $n = \frac{d^2+d}{2}$, autrement dit, on peut rechercher les chroniques sous forme d'hypercube dans un espace de dimension $\frac{d^2+d}{2}$ ou sous forme de polytope dans un espace de dimension d.

4.3.3 Avantages et inconvénients

Il semble intéressant de réduire la dimension de l'espace de recherche mais il apparaît très rapidement que les objets recherchés (les polytopes) sont plus complexes que les hypercubes. La réduction de la dimension de l'espace de recherche augmente la complexité des objets recherchés. Ainsi la recherche de chroniques ne se simplifie pas.

Par ailleurs, une instance est plus simple (en terme de taille) à représenter dans l'espace des instances que dans l'espace des chroniques. On dispose donc de deux types de représentation que l'on peut utiliser suivant ce que l'on recherche.

Conclusion

Après avoir détaillé toute la formalisation autour des chroniques, nous avons vu comment traiter une requête portant sur la fréquence dans une base de données inductive étendue. Ce traitement passe par l'utilisation de l'algorithme de Mellish qui travaille à partir des chroniques maximalement spécifiques et fréquentes de chacune des séquences d'évènements. Tout au long de cette partie, nous avons supposé que l'on disposait d'un outil de recherche de ces ensembles

de chroniques. Dans le chapitre suivant, nous allons voir que le fonctionnement d'un tel outil n'est pas trivial et que sa mise en œuvre *complète* reste à réaliser.

CHAPITRE

5

Utilisation de FACE pour la réconciliation

Un réseau de télécommunications produit d'immenses journaux d'alarmes générées par divers évènements du réseau (connexion, déconnexion, requêtes, ...). Il est nécessaire de synthétiser ce flot d'alarmes pour permettre à un expert d'en analyser le contenu. Pour cela, France-télécom dispose d'un outil de fouille de données, FACE, présenté dans 2.2.4. Cet outil est capable de rechercher un ensemble de chroniques fréquentes synthétisant « au mieux » un journal d'alarmes. La notion de journal d'alarmes est analogue à celle de séquence d'évènements. Par conséquent, on propose d'intégrer FACE au cœur de la base de données inductive étendue au temps.

Nous allons montrer qu'il est possible de réutiliser le mécanisme mis en place dans FACE pour retrouver *toutes* les chroniques fréquentes au sein d'une séquence d'évènements. Ainsi, les résultats de FACE seront utilisés par l'algorithme de Mellish pour répondre à des requêtes adressées à la nouvelle base de données.

Dans une première partie, FACE est examiné afin d'en voir les points forts et les points faibles. Ensuite, une formalisation de son fonctionnement va permettre de démontrer que la réutilisation de ses résultats permet le calcul des chroniques fréquentes dans une séquence d'évènements.

5.1 Analyse critique de FACE

L'algorithme principal de FACE est une adaptation de l'algorithme bien connu A Priori à des motifs comportant des contraintes temporelles. Cet outil permet de découvrir les chroniques fréquentes au sein d'une séquence d'évènements. Cette découverte comporte trois étapes : la génération de chroniques à partir des chroniques fréquentes, de taille inférieure, déjà trouvées, la reconnaissance des instances de chroniques dans la séquence d'évènements qui permet le calcul du support et le raffinage des chroniques fréquentes à partir de leurs instances. La figure 5.1 illustre ce fonctionnement général.

Dans un premier temps, nous allons voir en quoi la reconnaissance de chroniques au sein d'une séquence d'évènements peut être considérée comme un biais d'apprentissage. Ensuite, nous verrons les fonctionnements de la génération et du raffinage de chroniques. Enfin, une critique de l'ensemble sera faite pour en voir les limites.

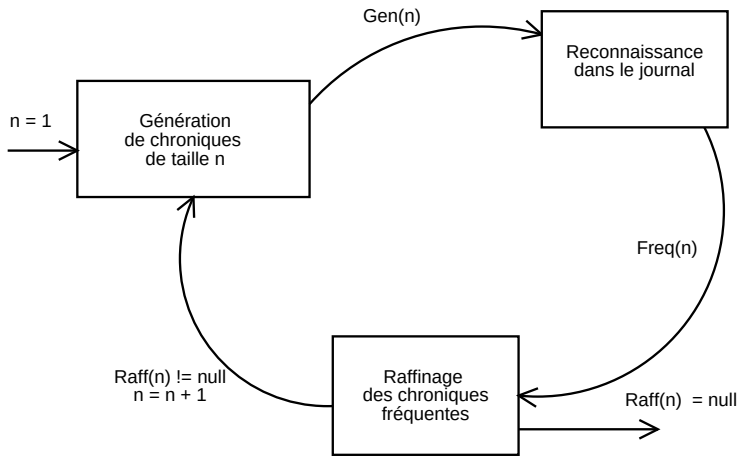


FIG. 5.1 – Fonctionnement général de FACE

5.1.1 Reconnaissance de chroniques

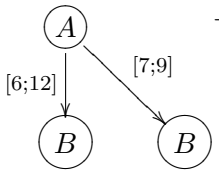
La reconnaissance de chroniques dans FACE est effectuée par CRS [DG94]. À partir d'une séquence d'évènements et d'un ensemble de chroniques, CRS retourne l'ensemble des instances de toutes les chroniques contenues dans la séquence d'évènements. Les instances retournées ne sont pas distinctes (elles peuvent partager des occurrences d'évènement). FACE doit alors sélectionner les instances distinctes d'une chronique pour en calculer la fréquence.

Vu Duong, dans sa thèse [Duo01], indique que le critère de sélection choisi doit permettre à la relation de sous-chronique d'être monotone vis-à-vis de la fréquence. Autrement dit, la fréquence d'une chronique doit être inférieure ou égale à la fréquence de ses sous-chroniques. De plus, seules les instances distinctes sont à considérer dans la plupart des applications. Ainsi, Vu Duong donne le critère d'*instances distinctes au plus tôt* qui conserve la monotonie des sous-modèles de chroniques selon la fréquence.

FACE utilise ce critère d'instances distinctes au plus tôt. Dans le cas de la reconnaissance de chroniques, ce mécanisme empêche la reconnaissance de certaines instances. La figure 5.2 illustre cet effet indésirable. À partir de la chronique de gauche CRS reconnaît dans la séquence d'évènements du milieu les instances de droite. FACE extrait les 2 instances distinctes [(A,0),(B,7),(B,7)] et [(A,12),(B,20),(B,20)] par la reconnaissance au plus tôt. Les deux instances sélectionnées associent la même occurrence aux deux évènements de type B.

On voit donc que par ce biais, si deux évènements d'une chronique de même type peuvent s'instancier à un même instant, alors toutes les instances reconnues de cette chronique contiennent une seule occurrence de ces deux évènements.

Pour certaines applications, la reconnaissance d'une chronique nécessite que chaque occurrence d'évènement de chaque instance corresponde à un unique évènement de la chronique reconnue. Pour ces applications, il faut compléter le critère de reconnaissance au plus tôt par un critère imposant un nombre d'occurrences d'évènement par instance identique au nombre d'évènements dans la chronique à reconnaître. On appellera ce critère *instances distinctes au*



Séquence d'évènements	
type	temps
A	0
B	7
B	10
A	12
B	20
B	21

instances reconnues dans CRS		
A	B[6 ;12]	B[7 ;9]
0	**7**	**7**
0	10	7
12	**20**	**20**
12	21	20
12	20	21
12	21	21

FIG. 5.2 – Une reconnaissance au plus tôt d'une chronique

plus tôt avec occurrences indépendantes. Celui-ci conserve le critère de monotonie de la relation de sous-modèle. Sur l'exemple précédent, FACE, modifié avec le nouveau critère de reconnaissance, reconnaît les instances [(A,0),(B,10),(B,7)] et [(A,12),(B,20),(B,21)].

La reconnaissance faite par CRS, et le critère de sélection d'instances que l'on vient de définir constituent la relation de couverture d'une chronique. Ce nouveau critère semble suffisant pour l'apprentissage de chroniques cardiaques. Mais, on peut penser que cette relation de couverture doit pouvoir être redéfinie en fonction de l'application comme c'est le cas en programmation logique inductive. En effet, en PLI, la relation de couverture des hypothèses sur les exemples est définie par l'utilisateur. La définition de cette relation permet de biaiser l'apprentissage.

5.1.2 Génération

La génération de chroniques s'appuie sur la monotonie de la requête à laquelle répond FACE. Une chronique n'est générée que si ses sous-chroniques sont solutions de cette requête. À chaque étape n la génération construit l'ensemble Φ_c^n des chroniques de taille n susceptibles d'être fréquentes dans la séquence d'évènements. Formellement, on l'écrit :

$$\mathcal{C} \in \Phi_c^n \Leftrightarrow taille(\mathcal{C}) = n \wedge \forall \mathcal{C}' \sqsubseteq \mathcal{C}, \mathcal{C}' \in Sol(\mathcal{L}, S) \tag{5.1}$$

Au lieu de vérifier la présence de toutes les sous-chroniques de $\mathcal{C}$ dans les solutions déjà trouvées, FACE vérifie seulement la présence des sous-chroniques générées à l'étape précédente. Par exemple, la chronique $\mathcal{C} = (\{A, B, B, C\}, \mathcal{T}) \in \Phi_c^4$ si et seulement si les chroniques $(\{A, B, B\}, \mathcal{T}_1)$, $(\{B, B, C\}, \mathcal{T}_2)$ et $(\{A, B, C\}, \mathcal{T}_3)$ appartiennent à Φ^3 (les graphes de contraintes $\mathcal{T}, \mathcal{T}_1, \mathcal{T}_2$ et $\mathcal{T}_3$ doivent aussi permettre la relation de sous-chronique)

Cette étape est très importante car c'est elle qui assure la présence d'une sous-chronique pour chaque chronique solution de la requête $freq(\mathcal{C}, \mathcal{L}) \geq S$ et de taille supérieure ou égale à n. Ce qu'on écrit formellement :

$$\forall \mathcal{C}_S (\mathcal{C}_S \in Sol(\mathcal{L}, S) \wedge taille(\mathcal{C}_S) \geq n \Rightarrow \exists \mathcal{C} \in \Phi_c^n, \mathcal{C} \sqsubseteq \mathcal{C}_S) \tag{5.2}$$

5.1.3 Raffinage de chroniques

Le raffinage de chroniques consiste à spécialiser le plus possible une chronique $\mathcal{C}$ à partir de ses instances trouvées dans une séquence d'évènements. La nouvelle chronique $\mathcal{C}'$, obtenue par spécialisation, doit couvrir toutes les instances de $\mathcal{C}$ trouvées dans la séquence d'évènements.

La définition 10 montre que l'on peut associer à chaque instance i une unique chronique $\mathcal{C}_i$. Si la séquence d'évènements contient des instances (de la chronique $\mathcal{C}$) dont les chroniques associées sont très différentes alors la chronique $\mathcal{C}'$ sera très générale. Inversement, si toutes les chroniques associées aux instances de la chronique $\mathcal{C}$ sont très proches alors $\mathcal{C}'$ sera très spécialisée.

La chronique raffinée $\mathcal{C}'$ est le généralisé maximalement spécifique de toutes les instances de $\mathcal{C}$ trouvées. Ce qui correspond à l'union de toutes les instances de $\mathcal{C}$. Dans le cas de chroniques, l'union est généralement un ensemble de chroniques. Or, toutes les instances de $\mathcal{C}$ possèdent les mêmes occurrences d'évènement, il n'existe donc qu'un seul appariement possible entre la chronique union et toutes les chroniques à unifier. Ainsi, l'ensemble union de ces instances contient une unique chronique $\mathcal{C}'$ définie ainsi :

$$\{\mathcal{C}'\} = \bigcup_{c \in \mathcal{I}_{\mathcal{C}}(\mathcal{L})} \mathcal{C}_c \tag{5.3}$$

où $\mathcal{I}_{\mathcal{C}}(\mathcal{L})$ est l'ensemble des instances de $\mathcal{C}$ dans $\mathcal{L}$ et $\mathcal{C}_c$ la chronique associée à l'instance c. Ce raffinage remplace les étapes de construction de chroniques sans contraintes décrites par Dousson et Vu Duong [DD99].

5.1.4 Incomplétude des résultats

Les résultats que rend FACE sont incomplets pour deux raisons : FACE généralise trop ses résultats pour pouvoir induire l'ensemble complet des solutions et il ne fait qu'une reconnaissance partielle des instances des chroniques. Nous allons détailler ces deux problèmes.

5.1.4.1 Un raffinage trop général

À chaque étape d'apprentissage, la génération, la reconnaissance et le raffinage retournent un ensemble de chroniques. Ces chroniques sont à la fois de plus en plus spécifiques par génération (ajout d'évènements) mais aussi les plus générales possibles par raffinage (unification des contraintes).

Cette généralisation (faite par union des instances) appliquée dans le raffinage produit des résultats incomplets. L'exemple de la figure 5.3 illustre ces propos. Supposons que l'on recherche les chroniques de fréquence supérieure ou égale à 2. À partir d'une chronique générée $A \rightarrow B$ à gauche et d'une séquence d'évènements au milieu, FACE reconnaît les instances de cette chronique dans le tableau de droite. Après raffinage, on obtient la chronique de droite. La chronique suivante : $\mathcal{C}' = A \xrightarrow{[2;2]} B$ a une fréquence de 2 dans la séquence d'évènements de la figure 5.3 mais elle n'est pas plus générale que la chronique $A \xrightarrow{[1;3]} B$, donc elle est rejetée. Dans cette exemple, les résultats de FACE ne sont pas complets au sens de la définition 9.

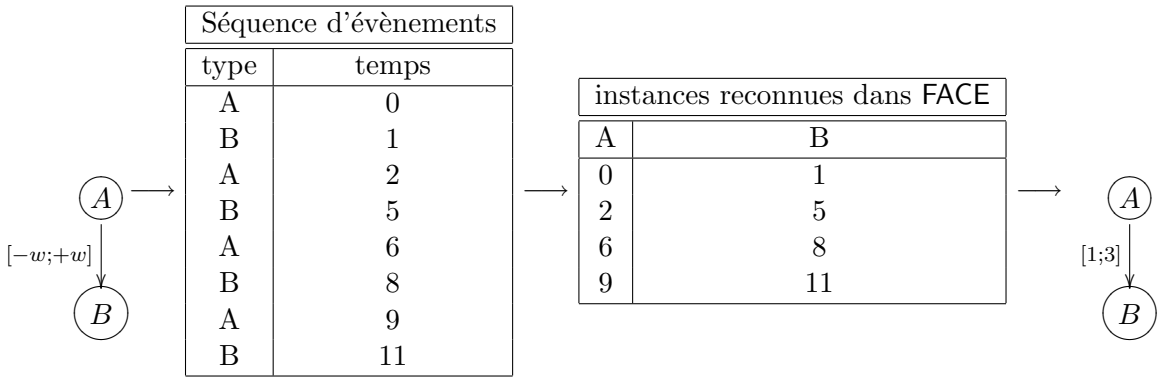


FIG. 5.3 – Un exemple d'incomplétude des résultats

Ainsi l'ensemble des solutions trouvées n'est pas complet selon la requête posée. On obtient :

$$\mathsf{B}_{F_{\mathcal{L}} \geq S} \not\subseteq \mathsf{F}_{F_{\mathcal{L}} \geq S}$$

où $\mathsf{F}_{F_{\mathcal{L}} \geq S}$ est l'ensemble des chroniques fréquentes retourné par FACE à partir d'un seuil S et d'une séquence d'évènements $\mathcal{L}$.

Une spécialisation de chroniques qui réduit les intervalles de contraintes peut être envisagée. Ce procédé augmentera très significativement le nombres de chroniques générées pour de grandes séquences d'évènements. De plus, la recherche de chroniques fréquentes se satisfait de ces résultats incomplets. En effet, le plus souvent on s'intéresse aux graphes des chroniques sans contraintes sur les arcs. Vu la complexité algorithmique d'un tel apprentissage, sa nécessité doit être évaluée avant tout changement dans FACE.

5.1.4.2 Une reconnaissance partielle

La reconnaissance au plus tôt par instances disjointes biaise l'apprentissage des chroniques fréquentes. En effet, lors du raffinage, FACE utilise ces instances pour calculer la chronique couvrant toutes les instances possédant les mêmes types d'évènements. Ainsi, si l'on omet certaines instances pourtant présentes dans la séquence d'évènements, on modifie la chronique raffinée. Ce qui donne un résultat incomplet.

Exemple Il est aisé de comprendre cette affirmation en s'aidant de la figure 5.3, on voit que les instances reconnues de la chronique candidate $A \xrightarrow{[-w;+w]} B$ ont toutes l'occurrence d'évènement A avant l'occurrence d'évènement B, alors que ce modèle candidat ne l'oblige en rien. Ceci vient de la reconnaissance au plus tôt par instances disjointes qui ne retient que certaines instances. Dans notre exemple 5.3, pour un seuil minimal de 2, toutes les chroniques fréquentes de la figure 5.4 ont été perdues et ne sont pas induites par l'unique chronique trouvée.

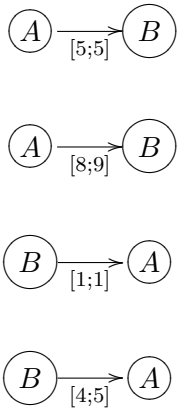


FIG. 5.4 – Chroniques de fréquence ≥ 2 non trouvées par FACE dans la séquence d'évènements de la figure 5.3

5.1.5 Intérêt

On voit que FACE donne des résultats volontairement incomplets pour essayer de faire ressortir rapidement les éléments pouvant synthétiser au mieux une séquence d'évènements. Cela se justifie en fouille de données où la grande quantité de données impose une synthétise efficace des connaissances cachées.

L'algorithme de Mellish, énoncé dans la partie 4.2.3, nécessite l'ensemble complet des chroniques maximalement spécifiques et fréquentes au sein d'une séquence d'évènements pour retrouver les solutions au type de requête proposée.

Les résultats de FACE sont donc inutilisables tels quels et doivent être retravaillés. Ainsi, nous verrons dans les parties suivantes que l'intégration de FACE dans une base de données inductive requiert quelques changements.

5.2 Formalisation et amélioration de FACE

Un des problèmes que pose FACE se situe au niveau du raffinage. En effet, ce raffinage est à double emploi, d'une part il fournit à l'étape suivante des chroniques fréquentes de base, et d'autre part ces chroniques sont enregistrées comme le résultat de la requête des chroniques de la taille correspondant à l'étape en cours. Ce faisant, le raffinage généralise les instances de chroniques pour l'étape suivante alors que l'on souhaite, en résultat, des chroniques les plus spécifiques possibles.

Nous proposons donc d'effectuer un raffinage supplémentaire destiné à fournir les chroniques fréquentes de la taille correspondante à celle de l'étape en cours. La figure 5.5 illustre ce nouveau fonctionnement.

Le deuxième problème se situe au niveau de la reconnaissance. Le critère de choix des instances, inclus dans celle-ci, implique une recherche partielle des instances des chroniques candidates. Un nouveau type de reconnaissance va donc être intégré dans FACE.

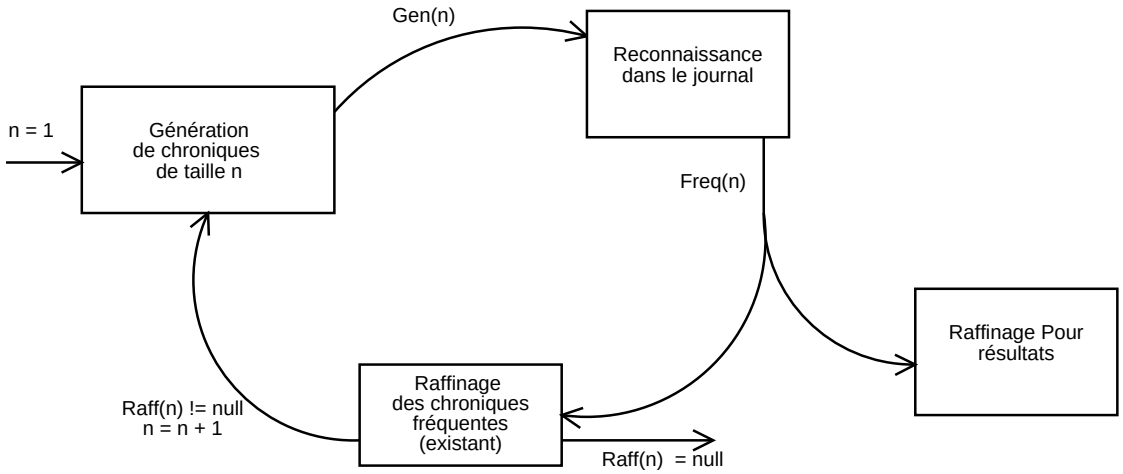


FIG. 5.5 – Nouveau fonctionnement général de FACE

Après avoir défini formellement l'ensemble des termes liés à la reconnaissance de chroniques et à leur raffinage, nous présentons une formalisation de FACE permettant d'en déduire la véritable nature des chroniques qu'il retourne. Enfin, nous proposons un nouveau fonctionnement général permettant de découvrir l'ensemble des chroniques maximalement fréquentes dans une séquence d'évènements.

5.2.1 Définitions

5.2.1.1 Opérateurs de raffinement

Nous allons définir deux opérateurs de raffinement sur les chroniques. Ces opérateurs sont difficilement instanciables mais ils permettent d'expliquer et de démontrer formellement l'apprentissage de chroniques fréquentes au sein d'une séquence d'évènements. De plus, on définit l'opérateur inverse d'un opérateur de raffinement.

Opérateur par restriction de contraintes L'opérateur de raffinement par restriction de contraintes (noté R_T) permet d'obtenir l'ensemble des chroniques plus spécifiques que $\mathcal{C}$ et ayant les mêmes évènements que celle-ci.

Définition 11 (R_T : opérateur par restriction de contraintes) *Soit la chronique $\mathcal{C} = (\mathcal{S}, \mathcal{T})$, on définit l'ensemble des chroniques spécialisées par l'opérateur de raffinement par restriction de contraintes sur $\mathcal{C}$:*

$$R_T(\mathcal{C}) = \{\mathcal{C}' | \mathcal{C}' = (\mathcal{S}', \mathcal{T}'), \mathcal{C} \sqsubseteq \mathcal{C}' \wedge \mathcal{S}' = \mathcal{S}\}$$

Opérateur par ajout d'évènements L'opérateur de raffinement par ajout d'évènements (noté R_S) permet d'obtenir l'ensemble des chroniques plus spécifiques que $\mathcal{C}$ telles que le graphe de contraintes de $\mathcal{C}$ est un sous-graphe du graphe de contraintes de chacune de ces chroniques.

Définition 12 (R_S : opérateur par ajout d'évènements) *Soit la chronique $\mathcal{C} = (\mathcal{S}, \mathcal{T})$, on définit l'ensemble des chroniques spécialisées par l'opérateur de raffinement par ajout d'évènements sur $\mathcal{C}$:*

$$R_S(\mathcal{C}) = \{\mathcal{C}' | \mathcal{C}' = (\mathcal{S}', \mathcal{T}'), \mathcal{C} \sqsubseteq \mathcal{C}' \wedge \mathcal{T} = (\mathcal{T}'/\mathcal{S})\}$$

Opérateur de généralisation À chaque opérateur R de raffinement correspond un opérateur G de généralisation. On le définit ainsi :

Définition 13 (Opérateur de généralisation) *Soit un opérateur de raffinement R, on définit l'opérateur G de généralisation associé par :*

$$G(\mathcal{C}) = \{\mathcal{C}' | \mathcal{C} \in R(\mathcal{C}')\}$$

5.2.1.2 Ensembles d'instances

Une instance est un ensemble d'occurrences d'évènement. On sait qu'une instance est couverte par une chronique. De plus, une instance peut être incluse dans une séquence d'évènements qui est un ensemble ordonné d'occurrences d'évènements. Ainsi, on se propose d'établir le lien entre une séquence d'évènements et une chronique.

Étant donné une séquence d'évènements $\mathcal{L}$ et une chronique $\mathcal{C}$, on note $\mathcal{I}_{\mathcal{C}}^{Q}(\mathcal{L})$ l'ensemble des instances de $\mathcal{C}$ reconnues dans $\mathcal{L}$ selon un critère de reconnaissance Q (voir 5.1.1).

Définition 14 (Instances d'une chronique dans une séquence d'évènements) *On définit $\mathcal{I}_{\mathcal{C}}^{Q}(\mathcal{L})$ l'ensemble des instances d'une chronique dans une séquence d'évènements de la façon suivante :*

$$\mathcal{I}_{\mathcal{C}}^{Q}(\mathcal{L}) = \{i | \mathcal{C}_i \in R_T(\mathcal{C}) \wedge i \subseteq \mathcal{L}\}$$

De plus $\mathcal{I}_{\mathcal{C}}^{Q}(\mathcal{L})$ doit vérifier le critère Q :

$$Q(\mathcal{I}_{\mathcal{C}}^{Q}(\mathcal{L}))$$

Reconnaissance sans critère de reconnaissance Il peut être utile de définir un ensemble d'instances extraites d'une séquence d'évènements sans aucun critère de reconnaissance, nous verrons dans la suite qu'il est même nécessaire pour l'apprentissage des chroniques fréquentes. On note donc $\mathcal{I}_{\mathcal{C}}(\mathcal{L})$ l'ensemble des instances d'une chronique dans une séquence d'évènements.

Ceci permet de définir $\mathcal{I}(\mathcal{L})$, l'ensemble des *instances potentielles* d'une séquence d'évènements, ou encore l'ensemble des parties de $\mathcal{L}$:

$$\mathcal{I}(\mathcal{L}) = 2^{\mathcal{L}}$$

5.2.1.3 Fréquence de chronique

La fréquence d'une chronique $\mathcal{C}$ dans une séquence d'évènements $\mathcal{L}$ selon un critère de reconnaissance Q est définie par :

$$freq^Q(\mathcal{C}, \mathcal{L}) = |\mathcal{I}_\mathcal{C}^Q(\mathcal{L})|$$

Critère de reconnaissance Le critère de reconnaissance Q, énoncé précédemment, doit avoir certaines propriétés pour pouvoir définir la fréquence :

- Pour toute séquence d'évènements, et pour toute chronique, $\mathcal{I}_\mathcal{C}^Q(\mathcal{L})$ doit être unique. Ainsi la fréquence de la chronique $\mathcal{C}$ dans $\mathcal{L}$ est également unique.

$$\forall \mathcal{C}, \mathcal{L} : \mathcal{I}_\mathcal{C}^Q(\mathcal{L}) \textit{ est unique.}$$

- Le critère Q doit conserver la relation de monotonie entre chronique et sous-chronique. La fréquence d'une sous-chronique de $\mathcal{C}$ doit être supérieure ou égale à la fréquence de $\mathcal{C}$ dans une même séquence d'évènements $\mathcal{L}$.

$$\forall \mathcal{C} \sqsubseteq \mathcal{C}' : freq^Q(\mathcal{C}, \mathcal{L}) \geq freq^Q(\mathcal{C}', \mathcal{L})$$

Instances distinctes avec occurrences d'évènements indépendantes On peut définir le critère d'instances distinctes avec occurrences d'évènements indépendantes :

$$\begin{gathered} Q_{dis\&indep}(\mathcal{I}_\mathcal{C}(\mathcal{L})) = Q_{dis}(\mathcal{I}_\mathcal{C}(\mathcal{L})) \wedge Q_{indep}(\mathcal{I}_\mathcal{C}(\mathcal{L})) \\ Q_{dis}(\mathcal{I}_\mathcal{C}(\mathcal{L})) = \forall (i, i') \in \mathcal{I}_\mathcal{C}(\mathcal{L})^2, i \neq i' \Rightarrow i \cap i' = \varnothing \\ Q_{indep}(\mathcal{I}_\mathcal{C}(\mathcal{L})) = \forall i \in \mathcal{I}_\mathcal{C}(\mathcal{L}), \exists! i' \in \mathcal{I}_\mathcal{C}(\mathcal{L}), i = i' \end{gathered}$$

5.2.1.4 Union contrôlée par fréquence

À partir d'un ensemble $\mathbb{E}$ d'instances, l'union contrôlée par fréquence (notée $\Cup_F^Q(\mathbb{E})$) effectue l'union des chroniques correspondant aux instances pour obtenir toutes les chroniques de fréquence supérieure ou égale à un seuil S. Les instances unifiées doivent respecter un critère Q.

Définition 15 (Union contrôlée par fréquence) *Soit un ensemble $\mathbb{E}$ d'instances, on définit l'ensemble des chroniques obtenues par union de ces instances sous contrôle de fréquence par :*

$$\Cup_S^Q(\mathbb{E}) = \{\mathcal{C} | \exists \mathbb{E}_\mathcal{C} \subseteq \mathbb{E} \wedge |\mathbb{E}_\mathcal{C}| \geq S \wedge \mathcal{C} = \bigcup_{i \in \mathbb{E}_\mathcal{C}} \mathcal{C}_i \wedge \forall i \in \mathbb{E}_\mathcal{C} : \mathcal{C}_i \in R_T(\mathcal{C}) \wedge Q(\mathbb{E}_\mathcal{C})\}$$

La définition 9 définit l'ensemble $\mathsf{B}_{F_\mathcal{L} \geq S}$ des chroniques maximalement spécifiques et de fréquence supérieure ou égale à S dans la séquence d'évènements $\mathcal{L}$. On reprend cette définition en y incluant la nouvelle définition de la fréquence :

$$\forall \mathcal{C}_\mathsf{S} (\mathcal{C}_\mathsf{S} \in Sol(\mathcal{L}, S) \Leftrightarrow \exists \mathcal{B} \in \mathsf{B}_{F_\mathcal{L} \geq S}, \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{B})$$

Théorème 1 (Induction des chroniques fréquentes par union contrôlée) *L'union contrôlée par fréquence de l'ensemble $\mathcal{I}(\mathcal{L})$ de toutes les instances potentielles d'une séquence d'évènements $\mathcal{L}$ induit l'ensemble des chroniques fréquentes de cette séquence.*

$$B_{F_{\mathcal{L}} \geq S} = min(\Cup\!\!\!\!\Cup_S^Q(\mathcal{I}(\mathcal{L})))$$

La preuve de ce théorème est donnée en annexe A.1.

5.2.2 Formalisation d'une étape d'apprentissage de FACE

On a montré qu'à partir de l'ensemble des instances potentielles d'une séquence d'évènements, on peut calculer l'ensemble des chroniques fréquentes de cette séquence (définition 9 et théorème 1). Or, le calcul de ces instances est très complexe et selon la fréquence choisie, beaucoup d'entre elles ne seront pas unies. On a donc besoin d'un mécanisme d'extraction d'instances pour calculer l'ensemble des solutions.

Avant d'examiner ce mécanisme, nous allons formaliser le fonctionnement de FACE pour voir en quoi il peut être utile. La reconnaissance a déjà été étudiée en détail dans la partie 5.2.1.2 contrairement à la génération de chroniques qui n'a pas été complètement décrite en (5.1). On y introduit l'opérateur de généralisation G_S (le dual de R_S) :

$$\Phi_c^{n+1} = \{\mathcal{C} | taille(\mathcal{C}) = n+1, \forall \mathcal{C}' \in G_S(\mathcal{C}), taille(\mathcal{C}') = n : \mathcal{C}' \in \Phi^n\} \tag{5.4}$$

On rappelle que Φ_c^n est l'ensemble des chroniques candidates générées à l'étape n et Φ^n est l'ensemble des chroniques résultats à l'étape n.

Le raffinage, abordé rapidement en 5.1.3, nécessite une plus grande attention. Celui-ci consiste à réaliser l'union des instances reconnues. Autrement dit, à chaque étape n, à partir de l'ensemble des instances reconnues, FACE effectue l'union de toutes celles qui proviennent d'une même chronique. On peut donc montrer que le raffinage consiste en :

$$\Phi^n = max(\Cup\!\!\!\!\Cup_S^Q(\mathcal{I}^Q_{\Phi_c^n}(\mathcal{L}))) \tag{5.5}$$

où $\mathcal{I}^Q_{\Phi_c^n}(\mathcal{L})$ est l'ensemble des instances des chroniques de Φ_c^n dans $\mathcal{L}$.

À partir du raffinage vu dans FACE, on prouve, en annexe A.2, qu'il est équivalent à la composition des deux opérateurs $\Cup\!\!\!\!\Cup_S^Q$ et max.

5.2.3 Complétude de FACE par rapport à l'opérateur de raffinement par restriction de contraintes (R_T)

Nous venons donc de voir une formalisation d'une étape d'apprentissage avec FACE :

$$\Phi^0 = \{(\varnothing, \varnothing)\}$$
$$(5.4)\ \Phi_c^{n+1} = \{\mathcal{C} | taille(\mathcal{C}) = n+1, \forall \mathcal{C}' \in G_S(\mathcal{C}), taille(\mathcal{C}') = n : \mathcal{C}' \in \Phi^n\} \tag{5.6}$$
$$(5.5)\ \Phi^{n+1} = max(\Cup\!\!\!\!\Cup_S^Q(\mathcal{I}^Q_{\Phi_c^{n+1}}(\mathcal{L}))) \tag{5.7}$$

On a vu, dans la partie 5.1.4.2, que le critère de reconnaissance d'instances disjointes au plus tôt empêche FACE de découvrir toutes les chroniques fréquentes. Ce critère est donc, en partie, à l'origine de l'incomplétude des résultats. On se propose de ne pas l'utiliser lors de la reconnaissance mais uniquement lors du raffinage. L'ensemble des instances retournées lors de la phase de reconnaissance n'a donc plus de propriétés particulières (instances disjointes, au plus tôt,...). La reconnaissance (5.7) se définit maintenant par :

$$\Phi^{n+1} = max(\uplus_S^Q(\mathcal{I}_{\Phi_c^{n+1}}(\mathcal{L}))) \tag{5.8}$$

À partir de la formalisation que nous venons de présenter, on souhaite qualifier les résultats que rend FACE. Il retourne un ensemble de chroniques $\mathsf{F}_{F_{\mathcal{L}}\geq S} = \Phi = \bigcup_{i\in 0..n} \Phi^i$. On a montré l'incomplétude de cet ensemble mais il est possible de le qualifier par rapport à l'opérateur de raffinement R_T.

Proposition 4 (Complétude de FACE par rapport à l'opérateur de raffinement R_T) *Toute chronique maximalement spécifique et solution de la requête est un raffinement par restriction de contraintes (R_T) d'une chronique que rend FACE. Ce qui s'écrit :*

$$\forall \mathcal{C}_B \in B_{F_{\mathcal{L}}\geq S} \Rightarrow \exists \mathcal{C}_\Phi \in \Phi, \mathcal{C}_B \in R_T(\mathcal{C}_\Phi)$$

Cette démonstration se fait par récurrence sur la taille n des chroniques $\mathcal{C}_\mathsf{B}$. On pourra trouver cette démonstration en annexe A.3.

La figure 5.6 représente les chroniques retournées par FACE dans l'espace des versions. On voit que l'ensemble des chroniques maximalement spécifiques est couvert par les chroniques que retourne FACE.

5.2.4 FACE : un extracteur d'instances

On a vu en 4.2.3.2 que l'algorithme de *Mellish* nécessitait la connaissance de **toutes les chroniques solutions maximalement spécifiques**. Il est donc primordial de pouvoir retrouver ces solutions avec FACE.

On vient de montrer qu'un raffinement R_T des chroniques que rend FACE permet de trouver toutes les chroniques que l'algorithme de *Mellish* nécessite. Une implémentation de cette méthode « top-down » nécessiterait de raffiner au fur et à mesure les chroniques de FACE pour arriver aux chroniques recherchées. Malheureusement, la combinatoire de l'opérateur R_T étant beaucoup trop importante, cette méthode paraît difficile à mettre en œuvre.

On va donc voir qu'à partir des instances des chroniques que FACE retourne, il est possible de retrouver les chroniques attendues. Autrement dit, on veut prouver que FACE extrait toutes les instances des chroniques maximalement spécifiques et fréquentes d'une séquence d'évènements. Par la suite, cela permettra de mettre en œuvre une méthode « bottom-up ». Ainsi, si toutes les instances des chroniques de l'ensemble $\mathsf{B}_{F_{\mathcal{L}}\geq S}$ sont extraites par FACE, alors on peut dire qu'il est un extracteur complet et correct d'instances.

On définit l'ensemble des instances retournées par FACE de cette façon :

$$\mathcal{I}_\Phi(\mathcal{L}) = \bigcup_{i\in 0..n} \mathcal{I}_{\Phi^i}(\mathcal{L})$$

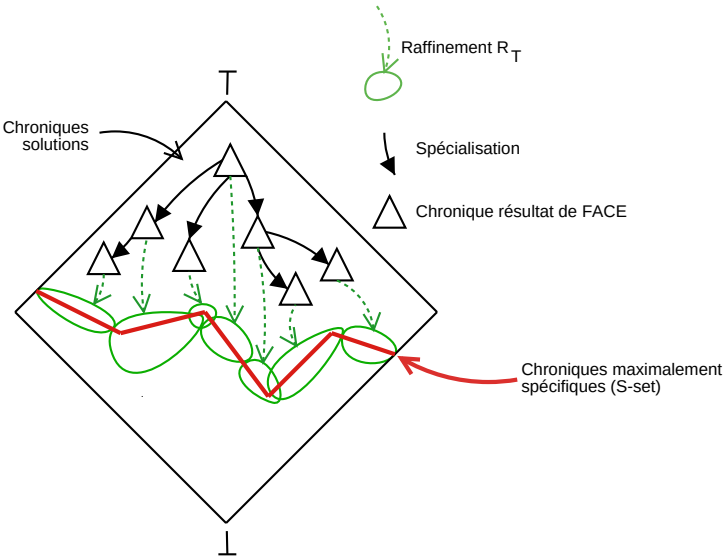


FIG. 5.6 – La complétude de FACE par l'opérateur de restriction de contraintes dans l'espace des versions

FACE est un extracteur complet et correct d'instances par rapport à la requête type si les instances qu'il retourne permettent de calculer l'ensemble $\mathsf{B}_{F_{\mathcal{L}\geq S}}$.

Théorème 2 (FACE : un extracteur complet et correct d'instances) *FACE est un extracteur complet et correct d'instances dans une séquence d'évènements $\mathcal{L}$ selon un seuil de fréquence S. Ce qui s'écrit :*

$$B_{F_{\mathcal{L}\geq S}} = min(\Cup_S^Q(\mathcal{I}_\Phi(\mathcal{L})))$$

La preuve est donné dans l'annexe A.4.

5.2.5 Nouveau fonctionnement général

Nous venons de voir que l'algorithme de FACE pouvait être largement conservé même si le critère de reconnaissance n'affectait plus la reconnaissance mais l'union contrôlée et qu'un nouveau raffinage produisant les résultats de l'apprentissage devait être implémenté pour espérer obtenir un résultat utilisable par la suite (algorithme de Mellish). Nous allons donc détailler ici ces changements dans FACE.

Le raffinage pour résultat correspond au calcul d'une union contrôlée par fréquence d'un sous-ensemble des instances de la séquence d'évènements. Autrement dit, à chaque étape on effectue l'union contrôlée par fréquence des instances des chroniques fréquentes. On peut réaliser le raffinage de cette façon grâce à l'égalité suivante :

$$min(\mathbb{U}_S^Q(\mathcal{I}_\Phi(\mathcal{L}))) = min(\bigcup_{i \in 0..n} min(\mathbb{U}_S^Q(\mathcal{I}_{\Phi^i}(\mathcal{L}))))$$

L'algorithme 4 détaille le nouveau fonctionnement général de FACE. Seul le raffinage est changé. Le calcul des chroniques maximalement spécifiques de la ligne 1 constitue la part du travail la plus importante. Ensuite, l'opérateur de raffinage $raff$ fait l'union des chroniques de $\mathsf{B}_\mathcal{C}$ (ligne 2). Il correspond à l'ancien opérateur de raffinage. Enfin, en résultat, on ne garde à chaque étape que les chroniques maximalement spécifiques (ligne 3).

Algorithme 4: nouvel algorithme général de FACE

Entrées : Une séquence d'évènements : $\mathcal{L}$
Un seuil : S
Sorties : L'ensemble des chroniques de fréquence supérieur ou égale au seuil S dans la séquence d'évènements $\mathcal{L}$ et maximalement spécifiques : $\mathsf{B}_{F_\mathcal{L} \geq S}$

$\mathsf{B}_{F_\mathcal{L} \geq S} = \emptyset$
$\Phi^0 = \{(\emptyset, \emptyset)\}$
$n = 1$
tant que $\Phi^{n-1} \neq \emptyset$ **faire**
  étape classique de FACE
  $\Phi_c^n = generation(\Phi^{n-1})$
  Raffinage
  $\Phi^n = \emptyset$
  pour chaque $\mathcal{C} \in \Phi_c^n$ **faire**
1 $\mathsf{B}_\mathcal{C} = min(\mathbb{U}_S^Q(\mathcal{I}_\mathcal{C}(\mathcal{L})))$
2 $\Phi^n = \Phi^n \bigcup \{raff(\mathsf{B}_\mathcal{C})\}$
3 $\mathsf{B}_{F_\mathcal{L} \geq S} = min(\mathsf{B}_{F_\mathcal{L} \geq S} \bigcup \mathsf{B}_\mathcal{C})$
  $n = n + 1$
retourne $\mathsf{B}_{F_\mathcal{L} \geq S}$

Conclusion

Nous avons présenté FACE en détails, afin d'en voir les limites. Son intérêt a été démontré de façon à s'en servir comme un extracteur d'instances. Nous allons voir dans le chapitre suivant que le problème de trouver les chroniques maximalement spécifiques à partir des instances d'une chronique est assez complexe mais que, heureusement, des solutions existent.

CHAPITRE

6

Propositions pour un algorithme efficace

L'étape principale de la découverte des chroniques maximalement spécifiques et fréquentes (*CMFs*) repose sur l'usage de l'opérateur d'union contrôlée par fréquence. D'une part, nous allons voir que ce problème est complexe mais que certaines hypothèses vont permettre de réduire cette complexité. D'autre part, la prise en compte d'un facteur d'intérêt autre que la fréquence va permettre d'accentuer la recherche sur la découverte de connaissances plus intéressantes.

6.1 Définition du problème

Le but est de trouver une implémentation du raffinage pour résultats dans FACE. Ceci correspond à l'opération :

$$\mathsf{B}_{\mathcal{C}} = min(\Cup_S^Q(\mathcal{I}_{\mathcal{C}}(\mathcal{L}))) \tag{6.1}$$

À partir d'un ensemble $\mathbb{E}$ d'instances, l'opérateur d'union contrôlée par fréquence consiste à unir certaines des instances satisfaisant des conditions particulières. Ces instances doivent respecter un critère Q, elles doivent correspondre au raffinement par restriction de contraintes d'une même chronique et être en nombre suffisant.

Rappel de la définition 15 :

$$\Cup_S^Q(\mathbb{E}) = \{\mathcal{C} | \exists \mathbb{E}_{\mathcal{C}} \subseteq \mathbb{E} \wedge |\mathbb{E}_{\mathcal{C}}| \geq S \wedge \mathcal{C} = \bigcup_{i \in \mathbb{E}_{\mathcal{C}}} \mathcal{C}_i \wedge \forall i \in \mathbb{E}_{\mathcal{C}} : \mathcal{C}_i \in R_T(\mathcal{C}) \wedge Q(\mathbb{E}_{\mathcal{C}})\}$$

Les chroniques construites dans le cas de l'opération 6.1 proviennent de la reconnaissance d'une chronique $\mathcal{C}$ dans une séquence d'évènements. Elles appartiennent donc, par définition, au raffinement par restriction de contraintes de $\mathcal{C}$. Ainsi, cette condition n'est pas à vérifier. Il faut seulement vérifier le critère Q et le cardinal des ensembles d'instances à unir. L'opérateur *min* ne conserve que les chroniques maximalement spécifiques. Par conséquent, pour minimiser le nombre des chroniques construites, il faut seulement construire celles qui sont maximalement spécifiques.

Ce problème peut être défini plus largement. On a vu dans la partie 4.3.3, qu'il existait un espace des instances. On peut donc considérer les instances comme des points dans un espace. L'union de ces instances a pour résultat une chronique qui forme un polytope dans ce même espace ou un hypercube dans un espace de plus grande dimension. Le problème peut donc être décrit de la façon suivante :

Trouver les hypercubes, englobant au moins S points, qui n'englobent aucun hypercube solution, et qui satisfont un critère Q.

Les hypercubes solutions peuvent se chevaucher mais pas s'inclure.

6.2 Algorithme

Énumérer tous les sous-ensembles de $\mathcal{I}_{\mathcal{C}}(\mathcal{L})$ n'est pas envisageable vu leur nombre. Il faut utiliser la notion de continuité sur chacun des axes de l'espace de recherche pour trouver les hypercubes minimaux. Chaque axe correspond à une contrainte entre deux évènements (AB, AC, BC, BD ...).

On peut décomposer le problème en une recherche axe par axe des sous-ensembles d'instances satisfaisant Q et de cardinal supérieur ou égal à S. Ce qui donne l'algorithme 5. Cet algorithme ne donne pas l'ensemble complet des solutions mais son implémentation a dévoilé un problème qui serait empiré par un algorithme complet : **le nombre de CMFs est extrêmement grand**.

Algorithme 5: Recherche des CMFs à partir des instances d'une chronique

Entrées : $\mathcal{I}_{\mathcal{C}}(\mathcal{L})$: l'ensemble des instances de $\mathcal{C}$ dans $\mathcal{L}$
S : le seuil de fréquence minimale
Q : critère de reconnaissance

Sorties : $\mathsf{B}_{\mathcal{C}} = min(\biguplus_S^Q(\mathcal{I}_{\mathcal{C}}(\mathcal{L})))$

$\mathsf{B}_{\mathcal{C}} = \emptyset$
pour chaque *axe* $\mathbb{A}$ *de l'espace engendré par* $\mathcal{C}$ **faire**
  $\mathcal{I}_{\mathbb{A}} = trierAxe(\mathcal{I}_{\mathcal{C}}(\mathcal{L}), \mathbb{A})$
  for $i = 0$ **à** $|\mathcal{I}_{\mathbb{A}}| - S$ **do**
    $\mathbb{E}_{\mathcal{C}'} = \{\mathcal{I}_{\mathbb{A}}[j] : j \in [i..i+S]\}$
    si $Q(\mathbb{E}_{\mathcal{C}'})$ **alors**
      $\mathcal{C}' = \bigcup_{c \in \mathbb{E}_{\mathcal{C}'}} \mathcal{C}_c$
      $\mathsf{B}_{\mathcal{C}} = min(\mathsf{B}_{\mathcal{C}} \bigcup \{\mathcal{C}'\})$
retourne $\mathsf{B}_{\mathcal{C}}$

Lors de ces tests, Q était basé sur 3 critères : instances au plut tôt, disjointes, et dans une fenêtre de 3 secondes (l'intervalle de temps entre la première occurrence d'évènement et la dernière était au maximum de 3 secondes). Ce critère étant encore intégré à la reconnaissance, aucune vérification de Q n'était effectuée dans l'algorithme 5 qui s'en trouvait simplifié.

Nous disposons de quatre séquences d'évènements, qui caractérisent, chacune, un type d'arythmie et contiennent entre 3500 et 4000 évènements. Chaque séquence représente environ une demi

heure d'enregistrements d'électrocardiogrammes. Chaque séquence a été traitée par la nouvelle version de FACE où l'algorithme 5 a été intégré. Tous les algorithmes décrits dans ce document ont été programmé en JAVA et intégrés à FACE. De plus, le programme source fourni par France-télécom a permis de disposer d'un grand nombre de fonctionnalités préexistantes dans FACE.

Pour un seuil de 7%, nous avons obtenu les résultats « incomplets » du tableau de la figure 6.2. La taille maximale obtenue indique le nombre d'évènements des plus grandes chroniques fréquentes trouvées. La figure 6.1 illustre une CMF trouvée dans un électrocardiogramme d'un patient atteint d'arythmie de type LBBB. Les nœuds indiquent le type d'onde cardiaque. Un nœud (Q) correspond à une onde complexe QRS anormale qui concorde avec une contraction anormale des ventricules, tandis qu'un nœud (p) équivaut à une onde P qui concorde avec une contraction des oreillettes du cœur. Les valeurs des intervalles de contraintes sont exprimées en millisecondes.

Ces résultats montrent qu'un nombre important de CMFs sont trouvées dans les séquences d'évènements cardiaques. On a vu que la requête (4.2) à poser à la base de données nécessitait l'utilisation des CMFs des quatre séquences d'évènements. Ce qui fait un total de 4327+8528+4622+4817 = 22294 chroniques. Il n'est pas envisageable de lancer un tel calcul par l'algorithme de Mellish sans risquer de sortir des limites de temps et d'espace mémoire maximum.

Au vu des résultats, on peut légitimement se poser une question : les CMFs sont-elles indispensables ?

Un exemple des instances d'une chronique cardiaque montre que les CMFs ne sont pas forcément adaptées. La figure 6.3 montre l'histogramme des instances disjointes reconnues au plus tôt de la chronique $Q \xrightarrow{[-3000;+3000]} Q$ dans la séquence 119. On voit qu'il existe deux « paquets » d'instances, l'un centré sur 900ms et l'autre sur 1850ms. Si on prend comme seuil de fréquence $S = 50$, la chronique $Q \xrightarrow{[1000;1840]} Q$ est une CMF dans la séquence d'évènements 119. Mais cette chronique ne traduit aucun phénomène particulier, elle est juste le résultat des effets de bord des deux paquets d'instances. Il faut introduire un autre seuil que la fréquence pour décider de l'intérêt d'une chronique.

6.3 Utilisation de la densité

Intuitivement, la densité d'une chronique est fonction du nombre d'instances couvertes par cette chronique dans un journal, par conséquent de la fréquence, et du volume occupé par cette chronique dans l'espace des instances. Par exemple, on peut donner une mesure de la densité de la chronique $Q \xrightarrow{[1000;1840]} Q$ de fréquence 50 dans la séquence 119, avec : $\frac{50}{1840-1000} = \frac{5}{84}$. Tandis que la CMF $Q \xrightarrow{[900;940]} Q$ de la séquence 119 a une densité plus élevée de $\frac{50}{40} = \frac{5}{4}$. Ainsi, on voit que la densité peut être une autre mesure d'intérêt que la fréquence. Plus la densité d'une chronique est faible moins elle est intéressante.

Un algorithme de clustering La recherche ne doit plus seulement porter sur la fréquence mais aussi sur la densité des chroniques. À partir des instances, on doit trouver des chroniques

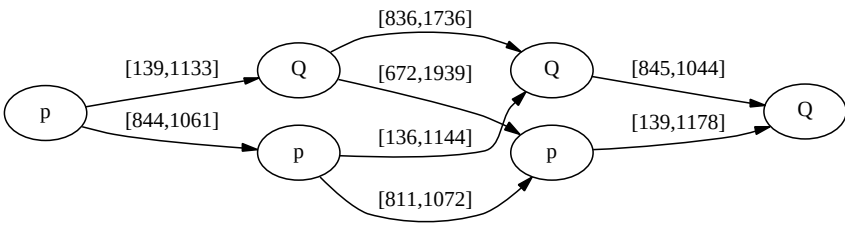


FIG. 6.1 – Exemple de CMF dans la séquence 214

Type d'arythmie	Séquence d'évènements cardiaques	Temps(min)	Taille maximale	$\|CMFs\|$
Bigeminy	L119	90	6	4327
LBBB	L214	80	6	8528
Mobitz Type 2	L231	40	7	4622
Normal	L100	120	6	4817

FIG. 6.2 – Tableau des résultats de FACE sur des séquence d'évènements cardiaques

maximalement spécifiques, fréquentes et **denses**.

La technique générale de regroupement des données en paquets denses est appelé *clustering*. Le but de celui-ci est de découvrir des groupes de données partageant les mêmes propriétés. On a vu, en 2.1.2, que la fouille de données utilise des techniques de clustering pour rechercher des associations entre attributs numériques. Il faut donc voir dans quelle mesure un algorithme de clustering peut être adapté pour répondre à notre problème de recherche de chroniques maximalement spécifiques, fréquentes, et denses. Cette partie ne sera pas plus développée pour des raisons de temps.

Conclusion

Nous avons esquissé une notion de densité qui apparaît indispensable dans la recherche des chroniques intéressantes. D'une part, l'intérêt d'une chronique est mieux qualifié et d'autre part le nombre de CMFs va diminuer et ainsi permettre de trouver les solutions à des requêtes sur une base de données inductive étendue.

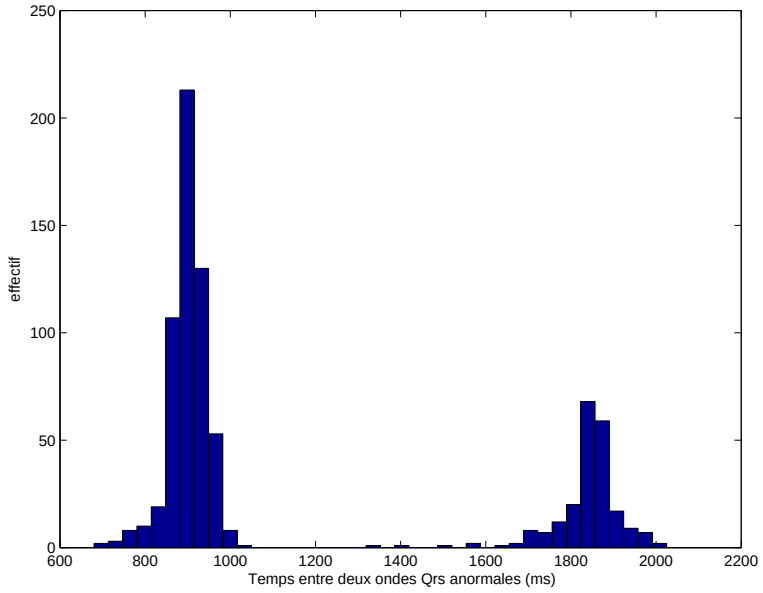


FIG. 6.3 – Histogramme d'instances disjointes de la chronique $Q \rightarrow Q$ dans la séquence d'évènements 119

Conclusion

Les applications d'une base de données inductive sur les séquences d'évènements et les chroniques sont très larges. Les données utilisables vont des électrocardiogrammes jusqu'aux journaux d'alarmes d'un réseau de télécommunications. La détection d'intrusion, de pannes ou, plus communément, de dysfonctionnements dans un tel réseau constitue une des applications prioritaires d'une telle base de données inductive. Plus généralement, toute donnée liée à une notion de distance dans l'espace ou de délai dans le temps peut, en principe, être représentée par une source de connaissances dans une telle BDI. Ainsi le travail présenté dans ce mémoire répond à une motivation croissante pour le sujet.

Nous nous sommes d'abord intéressés à l'apprentissage de connaissances sur des séquences temporelles à travers deux techniques : la programmation logique inductive et la fouille de données. Ces deux techniques font partie, respectivement, de l'apprentissage supervisé et de l'apprentissage non supervisé. Nous avons montré qu'un lien étroit existait entre ces deux domaines par l'introduction des bases de données inductives initialement créées pour formaliser la fouille de données. Ce lien réconcilie la PLI et la fouille de données sur un plan théorique.

L'approche qui consistait à réconcilier par une véritable application la PLI et la fouille de données constitue le second volet de ce travail. Le point central de cette réconciliation est l'extension du concept de base de données inductive à la découverte de motifs intégrant le temps. Ces motifs, représentés sous forme de chroniques, ont été largement décrits et leur recherche dans une base de données inductive a été détaillée.

À partir des résultats théoriques, la mise en œuvre d'une telle base de données inductive étendue a été réalisée. Elle a comporté deux volets. Le premier a consisté en l'implémentation de tous les algorithmes des opérations sur les chroniques ainsi que l'algorithme de Mellish. Ce dernier nécessite de disposer des chroniques maximalement spécifiques et fréquentes dans chaque séquence d'évènements. Le second volet a été consacré à la recherche de ces ensembles de chroniques. Celui-ci a été facilité par l'utilisation et l'amélioration de l'outil FACE. Malgré le fait que celui-ci soit plus adapté à effectuer une synthèse rapide de journaux d'alarmes de réseaux de télécommunications, on a vu qu'il n'en était pas moins très utile pour la recherche des chroniques fréquentes dans une séquence d'évènements. Sa formalisation a d'abord permis d'en voir les limites puis à partir de celle-ci, son amélioration a pu être effectuée. Elle comporte deux aspects : le passage de la recherche de chroniques simples aux chroniques générales et l'intégration d'algorithmes de recherche de chroniques maximalement spécifiques et fréquentes.

Le passage à l'échelle de tels algorithmes a montré quelques faiblesses. Ce problème n'est pas encore résolu mais quelques pistes, comme le clustering, permettent d'entrevoir des solutions.

Perspectives

Ce travail ouvre la voie vers une recherche plus approfondie en fouille de données dans les séquences d'évènements. Le processus d'extraction des connaissances cachées dans ces séquences d'évènements peut prendre différentes formes. C'est pourquoi l'utilisation du concept des bases de données inductives parait très prometteur. Divers axes de recherche peuvent être abordés.

Pour la plupart des méthodes de fouille de données, la fréquence est le critère principal indiquant la recherche à effectuer. Il est intéressant de voir que dans notre approche ce critère n'est pas unique. On a introduit celui de la densité et à un niveau supérieur celui du facteur discriminant entre séquences d'évènements. Il faut persévérer dans cette voie pour espérer extraire les informations intéressantes à partir d'une grande quantité de données réelles.

L'optimisation du traitement des requêtes est un point important dans toute base de données. La qualité et la précision des motifs extraits doivent dépendre des souhaits de l'utilisateur. Cela peut aller d'une recherche très focalisée et lente à une recherche très large et rapide. Ce degré de liberté dans l'exactitude et la complétude des réponses est un élément déterminant en fouille de données. De telles mesures doivent être intégrées à toute requête sur une BDI.

L'extraction de connaissances sur des données numériques en est à ses débuts. Une mauvaise discrétisation introduit des erreurs et les connaissances d'un expert sont souvent mises à contribution pour discrétiser au mieux les attributs numériques. Cette discrétisation biaise inévitablement l'extraction des connaissances. Par conséquent, de nouvelles méthodes doivent être proposées pour éviter ces écueils.

ANNEXE

A

Preuves

A.1 Preuve du théorème 1

On veut prouver le théorème 1, soit :

$$\forall \mathcal{C}_\mathsf{S}(\mathcal{C}_\mathsf{S} \in Sol(\mathcal{L}, S) \Leftrightarrow \exists \mathcal{C}_\mathsf{B} \in min(\uplus_S^Q(\mathcal{I}(\mathcal{L}))), \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_\mathsf{B})$$

Tout d'abord, on prouve

$$\forall \mathcal{C}_\mathsf{S} \in Sol(\mathcal{L}, S) \Rightarrow \exists \mathcal{C}_\mathsf{B} \in min(\uplus_S^Q(\mathcal{I}(\mathcal{L}))), \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_\mathsf{B} \tag{A.1}$$

Soit $\mathcal{C}_\mathsf{S} \in Sol(\mathcal{L}, S)$, on a $freq^Q(\mathcal{C}_\mathsf{S}, \mathcal{L}) \geq S$, ou encore

$$|\mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L})| \geq S \tag{A.2}$$

De plus, par définition, on a :

$$Q(\mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L})) \tag{A.3}$$
$$\forall i \in \mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L}) : \mathcal{C}_i \in R_T(\mathcal{C}_\mathsf{S}) \tag{A.4}$$
$$\forall i \in \mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L}) : \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_i \tag{A.5}$$

Les instances de $\mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L})$ sont tirées de la séquence d'évènements $\mathcal{L}$, on obtient : $\mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L}) \subseteq \mathcal{I}(\mathcal{L})$. On peut donc poser $\mathbb{E}_\mathcal{C} = \mathcal{I}^Q_{\mathcal{C}_\mathsf{S}}(\mathcal{L})$, ce qui donne d'après (A.2), (A.3), (A.4) et (A.5) :

$$|\mathbb{E}_\mathcal{C}| \geq S \tag{A.6}$$
$$Q(\mathbb{E}_\mathcal{C}) \tag{A.7}$$
$$\forall i \in \mathbb{E}_\mathcal{C} : \mathcal{C}_i \in R_T(\mathcal{C}) \tag{A.8}$$
$$\forall i \in \mathbb{E}_\mathcal{C} : \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_i \tag{A.9}$$

D'après (A.6), (A.7), (A.8) et la définition 15, on a

$$\mathcal{C}' = \bigcup_{i \in \mathbb{E}_{\mathcal{C}}} \mathcal{C}_i \tag{A.10}$$

tel que, $\mathcal{C}' \in \Cup_S^Q(\mathcal{I}(\mathcal{L}))$ et donc :

$$\exists \mathcal{C}_\mathsf{B} \in min(\Cup_S^Q(\mathcal{I}(\mathcal{L}))) : \mathcal{C}' \sqsubseteq \mathcal{C}_\mathsf{B} \tag{A.11}$$

De plus, d'après (A.9), (A.10) et la proposition 3

$$\mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}' \tag{A.12}$$

et ainsi, on déduit d'après (A.11), (A.12) :

$$\exists \mathcal{C}_\mathsf{B} \in min(\Cup_S^Q(\mathcal{I}(\mathcal{L}))) : \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_\mathsf{B}$$

Ce qui prouve (A.1).
Ensuite, on prouve

$$\forall \mathcal{C}_\mathsf{S}(\exists \mathcal{C}_\mathsf{B} \in min(\Cup_S^Q(\mathcal{I}(\mathcal{L}))), \mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_\mathsf{B} \Rightarrow \mathcal{C}_\mathsf{S} \in Sol(\mathcal{L}, S)) \tag{A.13}$$

Soit $\mathcal{C}_\mathsf{B} \in \Cup_S^Q(\mathcal{I}(\mathcal{L}))$, $\mathcal{C}_\mathsf{B}$ vérifie donc :

$$|\mathbb{E}_{\mathcal{C}_\mathsf{B}}| \geq S \tag{A.14}$$
$$Q(\mathbb{E}_{\mathcal{C}_\mathsf{B}}) \tag{A.15}$$
$$\forall i \in \mathbb{E}_{\mathcal{C}_\mathsf{B}} : \mathcal{C}_i \in R_T(\mathcal{C}_\mathsf{B}) \tag{A.16}$$
$$\forall i \in \mathbb{E}_{\mathcal{C}_\mathsf{B}} : \mathcal{C}_\mathsf{B} \sqsubseteq \mathcal{C}_i \tag{A.17}$$

Ainsi, $\mathcal{C}_\mathsf{B}$ est solution de $freq^Q(\mathcal{C}, \mathcal{L}) \geq S$.
D'après le monotonie de cette requête, si $\mathcal{C} \sqsubseteq \mathcal{C}'$ et $\mathcal{C}'$ est une solution alors $\mathcal{C}$ est aussi une solution. On a $\mathcal{C}_\mathsf{S} \sqsubseteq \mathcal{C}_\mathsf{B}$ et $\mathcal{C}_\mathsf{B}$ est solution la requête donc $\mathcal{C}_\mathsf{S}$ est aussi solution. Ce qui prouve (A.13).
D'après (A.1) et (A.13), on prouve le théorème 1.

A.2 Détails du raffinage de FACE

On sait que FACE effectue un raffinage de toute chronique $\mathcal{C} \in \Phi_c^n$ fréquente. Son raffinage correspond à la nouvelle chronique $\mathcal{C}' \in \Phi^n$ (voir partie 5.1.3) :

$$\mathcal{C}' = raff(\mathcal{C}) = \bigcup_{i \in \mathcal{I}_\mathcal{C}^Q(\mathcal{L})} \mathcal{C}_i \tag{A.18}$$

On veut montrer que ce raffinage correspond à l'expression suivante :

$$(5.5) \quad \Phi^n = max(\Cup_S^Q(\mathcal{I}_{\Phi_c^n}^Q(\mathcal{L})))$$

On démontre tout d'abord que

$$\forall \mathcal{C} \in \Phi_c^n \Rightarrow \mathcal{C}' \in max(\Cup_S^Q(\mathcal{I}_{\Phi_c^n}^Q(\mathcal{L}))) \tag{A.19}$$

Le raffinage de $\mathcal{C}$ est effectué si et seulement si $\mathcal{C}$ est fréquent, autrement dit

$$|\mathcal{I}_{\mathcal{C}}^Q(\mathcal{L})| \geq S \tag{A.20}$$

De plus, par la définition 14, on a :

$$Q(\mathcal{I}_{\mathcal{C}}^Q(\mathcal{L})) \tag{A.21}$$

D'après (A.18), on a :

$$\forall i \in \mathcal{I}_{\mathcal{C}}^Q(\mathcal{L}), \mathcal{C}_i \in R_T(\mathcal{C}') \tag{A.22}$$

Ainsi, d'après (A.18), (A.20),(A.21), (A.22) et la définition 15, on a

$$\mathcal{C}' \in \Cup_S^Q(\mathcal{I}_{\Phi_c^n}^Q(\mathcal{L}))$$

De plus, on voit qu'il n'existe pas de chroniques plus générales que $\mathcal{C}'$ dans $\Cup_S^Q(\mathcal{I}_{\Phi^n}^Q(\mathcal{L}))$ car celles-ci seraient générées par les instances de $\mathcal{C}$ et d'autres instances. Or c'est impossible, car ces dernières sont bloquées par l'opérateur R_T. On a donc démontré (A.19).

Ensuite on démontre que :

$$\forall \mathcal{C}' \in max(\Cup_S^Q(\mathcal{I}_{\Phi_c^n}^Q(\mathcal{L}))) \Rightarrow \mathcal{C} \in \Phi_c^n \tag{A.23}$$

Ce qui veut dire qu'il n'y pas de chronique créée par l'union qui ne corresponde à un raffinage de Φ_c^n. Cela se démontre facilement avec l'opérateur R_T qui bloque la construction de telles chroniques. Cette démonstration n'est pas effectuée ici. Enfin les expressions (A.23) et (A.19) prouvent (5.5).

A.3 Preuve de la complétude de FACE selon l'opérateur R_T

On va prouver la proposition 4 : toute chronique de $\mathsf{B}_{F_{\mathcal{L}} \geq S}$ est un raffinement par restriction de contraintes d'au moins une chronique de $\mathsf{F}_{F_{\mathcal{L}} \geq S} = \Phi$. Cette démonstration se fait par récurrence sur la taille des chroniques. Nous allons définir l'ensemble des chroniques maximalement spécifiques de taille inférieure ou égale à n. Soit :

$$\mathsf{B}_{F_{\mathcal{L}} \geq S}^n = min(\{\mathcal{C} : \mathcal{C} \in Sol(\mathcal{L}, S) \wedge taille(\mathcal{C}) \leq n\}) \tag{A.24}$$

Étape 0 Pour $n = 0$, $\mathcal{C}_{\mathsf{B}}^0 = (\varnothing, \varnothing) \in \Phi^0$, la proposition 4 est donc vraie.

Étape n Supposons

$$\forall \mathcal{C}_{\mathsf{B}}^{n} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n} \Rightarrow \exists \mathcal{C}_{\Phi^n} \in \Phi^n, \mathcal{C}_{\mathsf{B}}^{n} \in R_T(\mathcal{C}_{\Phi^n}) \tag{A.25}$$

On veut démontrer

$$\forall \mathcal{C}_{\mathsf{B}}^{n+1} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n} \Rightarrow \exists \mathcal{C}_{\Phi^{n+1}} \in \Phi^{n+1}, \mathcal{C}_{\mathsf{B}}^{n+1} \in R_T(\mathcal{C}_{\Phi^{n+1}}) \tag{A.26}$$

Soit $\mathcal{C}_{\mathsf{B}}^{n+1} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n}$, d'après l'anti-monotonie de la requête, ses sous-chroniques (par généralisation G_S) de taille n sont aussi solutions :

$$\forall \mathcal{C}' \in G_S(\mathcal{C}_{\mathsf{B}}^{n+1}) \wedge taille(\mathcal{C}') = n \Rightarrow \mathcal{C}' \in Sol(\mathcal{L}, S) \tag{A.27}$$

Ainsi, d'après la définition (A.24) :

$$\forall \mathcal{C}' \in G_S(\mathcal{C}_{\mathsf{B}}^{n+1}) \wedge taille(\mathcal{C}') = n \Rightarrow \exists \mathcal{C}_{\mathsf{B}}^{n} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n}, \mathcal{C}' \sqsubseteq \mathcal{C}_{\mathsf{B}}^{n}$$

On peut aussi écrire que $\mathcal{C}_{\mathsf{B}}^{n}$ est plus spécifique que $\mathcal{C}'$ par l'opérateur de raffinement R_T car ces chroniques ont la même structure :

$$\forall \mathcal{C}' \in G_S(\mathcal{C}_{\mathsf{B}}^{n+1}) \wedge taille(\mathcal{C}') = n \Rightarrow \exists \mathcal{C}_{\mathsf{B}}^{n} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n}, \mathcal{C}_{\mathsf{B}}^{n} \in R_T(\mathcal{C}')$$

ce qui donne d'après (A.25) :

$$\forall \mathcal{C}' \in G_S(\mathcal{C}_{\mathsf{B}}^{n+1}) \wedge taille(\mathcal{C}') = n \Rightarrow \exists \mathcal{C}_{\mathsf{B}}^{n} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n}, \mathcal{C}_{\mathsf{B}}^{n} \in R_T(\mathcal{C}') \wedge \exists \mathcal{C}_{\Phi^n} \in \Phi^n, \mathcal{C}_{\mathsf{B}}^{n} \in R_T(\mathcal{C}_{\Phi^n})$$

Or comme $\mathcal{C}_{\mathsf{B}}^{n+1}$ est maximalement spécifique et, d'après l'opérateur *max* de l'expression (5.8), la chronique $\mathcal{C}_{\Phi^n}$ est la plus générale possible donc $\mathcal{C}'$ est plus spécifique que $\mathcal{C}_{\Phi^n}$ par l'opérateur R_T. Et on obtient :

$$\forall \mathcal{C}' \in G_S(\mathcal{C}_{\mathsf{B}}^{n+1}) \wedge taille(\mathcal{C}') = n \Rightarrow \exists \mathcal{C}_{\Phi^n} \in \Phi^n, \mathcal{C}' \in R_T(\mathcal{C}_{\Phi^n})$$

Il existe une chronique $\mathcal{C}_R$, telle que

$$\mathcal{C}_R \in G_T(\mathcal{C}_{\mathsf{B}}^{n+1}) \; et$$
$$\forall \mathcal{C}' \in G_S(\mathcal{C}_{\mathsf{B}}^{n+1}) \wedge taille(\mathcal{C}') = n \wedge \exists \mathcal{C}_{\Phi^n} \in \Phi^n, \mathcal{C}' \in R_T(\mathcal{C}_{\Phi^n}) \Rightarrow \mathcal{C}_R \in R_S(\mathcal{C}_{\Phi^n})$$

Ce que l'on peut écrire :

$$\exists \mathcal{C}_R \in G_T(\mathcal{C}_{\mathsf{B}}^{n+1}), \forall \mathcal{C} \in G_S(\mathcal{C}_R) \wedge taille(\mathcal{C}) = n \Rightarrow \mathcal{C} \in \Phi^n \tag{A.28}$$

D'après (5.6) et (A.28), on a :

$$\mathcal{C}_R \in \Phi_c^{n+1} \tag{A.29}$$

de plus, d'après la définition de G_T :

$$\mathcal{C}_{\mathsf{B}}^{n+1} \in R_T(\mathcal{C}_R)$$

Ainsi, on a :

$$\forall \mathcal{C}_{\mathsf{B}}^{n+1} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n} \Rightarrow \exists \mathcal{C}_R \in \Phi_c^{n+1}, \mathcal{C}_{\mathsf{B}}^{n+1} \in R_T(\mathcal{C}_R) \tag{A.30}$$

Lors de la phase de reconnaissance (5.8), les instances de $\mathcal{C}_R$ telles que $\mathcal{I}_{\mathcal{C}_R}(\mathcal{L}) \subseteq \mathcal{I}_{\Phi_c^{n+1}}(\mathcal{L})$ sont extraites. Or, puisque la chronique $\mathcal{C}_{\mathsf{B}}^{n+1}$ est fréquente dans la séquence d'évènements $\mathcal{L}$ d'après (A.30), on a $freq^Q(\mathcal{C}_{\mathsf{B}}^{n+1}, \mathcal{L}) \geq S$ et donc il existe au moins S instances de cette chronique dans $\mathcal{I}_{\mathcal{C}_R}(\mathcal{L})$ (ce qui n'était pas forcément vrai avec le critère de reconnaissance Q). Ainsi, on peut écrire :

$$\forall \mathcal{C}_{\mathsf{B}}^{n+1} \in \mathsf{B}_{F_{\mathcal{L}} \geq S}^{n} \Rightarrow |\{i : i \in \mathcal{I}_{\Phi_c^{n+1}}(\mathcal{L}) \wedge \mathcal{C}_i \in R_T(\mathcal{C}_{\mathsf{B}}^{n+1})\}| \geq S \tag{A.31}$$

et on a :

$$\mathcal{I}^Q_{\mathcal{C}_{\mathsf{B}}^{n+1}}(\mathcal{L}) \subseteq \mathcal{I}_{\Phi_c^{n+1}}(\mathcal{L}) \tag{A.32}$$

$$\mathcal{I}^Q_{\mathcal{C}_{\mathsf{B}}^{n+1}}(\mathcal{L}) \geq S \tag{A.33}$$

$$\forall i \in \mathcal{I}^Q_{\mathcal{C}_{\mathsf{B}}^{n+1}}(\mathcal{L}) : \mathcal{C}_i \in R_T(\mathcal{C}_{\mathsf{B}}^{n+1}) \tag{A.34}$$

$$\mathcal{C}_{\mathsf{B}}^{n+1} = \bigcup_{i \in \mathcal{I}^Q_{\mathcal{C}_{\mathsf{B}}^{n+1}}(\mathcal{L})} \mathcal{C}_i \tag{A.35}$$

$$Q(\mathcal{I}^Q_{\mathcal{C}_{\mathsf{B}}^{n+1}}(\mathcal{L})) \tag{A.36}$$

D'après la définition 15 de l'union contrôlée par fréquence et toutes les conditions remplies, (A.32), (A.33), (A.34), (A.35) et (A.36) : $\mathcal{C}_{\mathsf{B}}^{n+1} \in \mathbb{U}^Q_S(\mathcal{I}_{\Phi_c^{n+1}}(\mathcal{L}))$. Or FACE extrait de cet ensemble les chroniques maximalement générales (opérateur max) on a donc : $\exists \mathcal{C}_{\Phi^{n+1}} \in max(\mathbb{U}^Q_S(\mathcal{I}_{\Phi_c^{n+1}}(\mathcal{L}))) : \mathcal{C}_{\Phi^{n+1}} \sqsubseteq \mathcal{C}_{\mathsf{B}}^{n+1}$. Les chroniques $\mathcal{C}_{\Phi^{n+1}}$ et $\mathcal{C}_{\mathsf{B}}^{n+1}$ ont la même taille $n+1$, elles possèdent donc la même structure, ainsi on peut dire que $\mathcal{C}_{\mathsf{B}}^{n+1} \in R_T(\mathcal{C}_{\Phi^{n+1}})$. Ce qui prouve (A.26). La propostion 4 est donc démontrée par récurrence.

A.4 Preuve que FACE est un extracteur complet et correct d'instances

Pour prouver que FACE est un extracteur complet et correct d'instances, on prouve le théorème 2, soit : $\mathsf{B}_{F_{\mathcal{L}} \geq S} = min(\mathbb{U}^Q_S(\mathcal{I}_\Phi(\mathcal{L})))$.

On sait que $\mathcal{I}_\Phi(\mathcal{L}) \subseteq \mathcal{I}(\mathcal{L})$. Ainsi, par construction de $\mathbb{U}^Q_S$ on a

$$min(\mathbb{U}^Q_S(\mathcal{I}_\Phi(\mathcal{L}))) \subseteq min(\mathbb{U}^Q_S(\mathcal{I}(\mathcal{L}))) \tag{A.37}$$

Ainsi, d'après le théorème 1 et (A.37), on a

$$min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}_\Phi(\mathcal{L}))) \subseteq \mathsf{B}_{F_\mathcal{L}} \tag{A.38}$$

D'après l'expression (4) et la définition 14 des instances d'une chronique :

$$\forall \mathcal{C}_\mathsf{B} \in \mathsf{B}_{F_\mathcal{L} \geq S} \Rightarrow \exists \mathcal{C}_\Phi \in \Phi, \mathcal{C}_\mathsf{B} \in R_T(\mathcal{C}_\Phi) \Rightarrow \mathcal{I}_{\mathcal{C}_\mathsf{B}}(\mathcal{L}) \subseteq \mathcal{I}_{\mathcal{C}_\Phi}(\mathcal{L})$$

ce que l'on écrit aussi :

$$\mathcal{I}_\mathsf{B}(\mathcal{L}) \subseteq \mathcal{I}_\Phi(\mathcal{L}) \tag{A.39}$$

Par construction de $\Cup\!\!\!\!\!\!\Cup_S^Q$, on obtient

$$min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}_\mathsf{B}(\mathcal{L}))) \subseteq min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}_\Phi(\mathcal{L}))) \tag{A.40}$$

Or, il est évident que :

$$min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}_\mathsf{B}(\mathcal{L})) = min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}(\mathcal{L}))) = \mathsf{B}_{F_\mathcal{L} \geq S} \tag{A.41}$$

On a donc d'après (A.40) et (A.41) :

$$\mathsf{B}_{F_\mathcal{L}} \subseteq min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}_\Phi(\mathcal{L}))) \tag{A.42}$$

(A.38) et (A.42) prouvent $\mathsf{B}_{F_\mathcal{L} \geq S} = min(\Cup\!\!\!\!\!\!\Cup_S^Q(\mathcal{I}_\Phi(\mathcal{L})))$, soit FACE est un extracteur complet et correct d'instances.

BIBLIOGRAPHIE

[AIS93] Rakesh Agrawal, Tomasz Imielinski et Arun Swami. « Mining association rules between sets of items in large databases ». Dans *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216. ACM Press, 1993.

[BCG01] Douglas Burdick, Manuel Calimlim et Johannes Gehrke. « MAFIA : A Maximal Frequent Itemset Algorithm for Transactional Databases ». Dans *Proceedings of the 17th International Conference on Data Engineering*, pages 443 – 452, Washington, DC, 2001. IEEE Computer Society.

[BCN04] Daniel Barbará, Ping Chen et Zohreh Nazeri. Self-Similar Mining of Time Association Rules. Dans *Procedings of Advances in Knowledge Discovery and Data Mining : 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia*, volume 3056, pages 86–95. LNAI Springer-Verlag, 2004.

[BGKW02] Cristian Bucila, Johannes Gehrke, Daniel Kifer et Walker White. « DualMiner : a dual-pruning algorithm for itemsets with constraints ». Dans *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 42–51. ACM Press, 2002.

[CCQW03] Guy Carrault, Marie-Odile Cordier, René Quiniou et Feng Wang. « Temporal abstraction and Inductive Logic Programming for arrhythmia recognition from electrocardiograms ». *Artificial Intelligence in Medicine*, 28(231-263), 2003.

[CM02] Antoine Cornuéjols et Laurent Miclet. *Apprentissage Artificiel*. Eyrolles, 2002.

[DD99] Christophe Dousson et Thang Vu Duong. « Découverte de chroniques avec contraintes temporelles à partir de journaux d'alarmes pour la supervision de réseaux de télécommunications ». Dans *Actes de la Conférence d'Apprentissage (CAP-99)*, pages 620–626, 1999.

[De 02a] Luc De Raedt. « *Data Mining as Constraint Logic Programming* », volume 2048 de *LNAI*, pages 526–547. Springer-Verlag, 2002.

[De 02b] Luc De Raedt. « Inductive databases : a declarative data mining approach ». Dans *From Machine Learning to Data Mining*, 2002.

[De 02c] Luc De Raedt. « A perspective on inductive databases ». *SIGKDD Explor. Newsl.*, 4(2) :69–77, 2002.

[DG94] Christophe Dousson et Malik Ghallab. « Suivi et reconnaissance de chroniques ». *Revue d'intelligence artificielle*, 8 :29–61, 1994.

[DJLM02] Luc De Raedt, Manfred JAEGER, Sau Dan LEE et Heikki MANNILA. « A Theory of Inductive Query Answering ». Dans *Proceedings of the International Conference on Data Mining (ICDM'02)*, Maebashi City, Japan, 2002. IEEE Computer Society.

[DK01] Luc De Raedt et Stefan Kramer. « The levelwise Version Space Algorithm and its Application to Molecular Fragment Finding ». Dans *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI 2001*, pages 853–862, Seattle, USA, août 2001. Morgan Kaufmann.

[DMP91] Rina Dechter, Itay Meiri et Judea Pearl. « Temporal constraint networks ». *Artificial Intelligence*, 49(1-3) :61–95, 1991.

[DR96] Luc Dehaspe et Luc De Raedt. « DLAB : A Declarative Language Bias Formalism ». Dans *International Symposium on Methodologies for Intelligent Systems*, pages 613–622, 1996.

[Duo01] Thang Vu Duong. « *Découverte de chroniques à partir de journaux d'alarmes, application à la supervision de réseaux de télécommunications* ». Thèse de doctorat, France Télécom R&D, DTL/DLI, 2001.

[Dze02] Saso Dzeroski. « Computational scientific discovery and inductive databases ». Dans *International Workshop on Active Mining (AM-2002)*, pages 4–15, Maebashi City, Japan, 2002. IEEE Computer Society.

[IM96] Tomasz Imielinski et Heikki Mannila. « A database perspective on knowledge discovery ». *Comm. of The ACM*, 39 :58–64, 1996.

[LD03] Sau Dan Lee et Luc De Raedt. « *Database Support for Data Mining Applications* », volume 2682 de *LNCS*, Chapitre Constraint Based Mining of First Order Sequences in SeqLog. Springer-Verlag, 2003.

[Lin03] Jun-Lin Lin. « Mining maximal frequent intervals ». Dans *Proceedings of the 2003 ACM symposium on Applied computing*, pages 426–431. ACM Press, 2003.

[Mar95] François Margot. *Composition de polytopes combinatoires*, volume 4 de *Cahier Mathématiques de l'EPFL*. Presses polytechniques et universitaires romandes, Lausanne, 1995.

[MF90] Stephen Muggleton et Cao Feng. « Efficient induction of logic programs ». Dans *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[MTV97] Heikki Mannila, Hannu Toivonen et A. Inkeri Verkamo. « Discovery of Frequent Episodes in Event Sequences ». *Data Mining and Knowledge Discovery*, 1(3) :259–289, 1997.

[Mug95] Stephen Muggleton. « Inverse Entailment and Progol ». *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4) :245–286, 1995.

[MY97] Renée J. Miller et Yuping Yang. « Association rules over interval data ». Dans *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 452–461. ACM Press, 1997.

[PSF91] Gregory Piatetsky-Shapiro et William J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, Cambridge, MA, 1991.

[QCJ93] John Ross Quinlan et Mike Cameron-Jones. « FOIL : A Midterm Report ». Dans *Machine Learning : ECML-93, European Conference on Machine Learning, Proceedings*, volume 667, pages 3–20. Springer-Verlag, 1993.

[SA96] Ramakrishnan Srikant et Rakesh Agrawal. « Mining quantitative association rules in large relational tables ». Dans *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 1–12. ACM Press, 1996.

[Sal03] Ansaf Salleb. « *Recherche de motifs fréquents pour l'extraction de règles d'association et de caractérisation* ». Thèse de doctorat, Université d'Orléans, 2003.

[WYM01] Wei Wang, Jiong Yang et Richard R. Muntz. « TAR : Temporal Association Rules on Evolving Numerical Attributes ». Dans *Seventeenth International Conference on Data Engineering, ICDE 2001*, pages 283–292, Heidelberg, Germany, 2001. IEEE Computer Society.

[YISI00] Mariko Yoshida, Tetsuya Iizuka, Hisako Shiohara et Masanori Ishiguro. « Mining sequential patterns including time intervals ». Dans *Proc. SPIE Vol. 4057, p. 213-220, Data Mining and Knowledge Discovery : Theory, Tools, and Technology II, Belur V. Dasarathy ; Ed.*, pages 213–220, avril 2000.

[Zie98] Günter M. Ziegler. *Lectures on Polytopes*, volume 152 de *Graduate Texts in Mathematics*. Springer, New York, 1995, revised edition 1998.


IRISA
INSA
RENNES