Caractérisation logique de données : application aux données biologiques

Arthur Chambon

UNIVERSITE BRETAGNE LOIRE

université angers

Thèse de Doctorat

Arthur CHAMBON

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique, section CNU 27
Unité de recherche : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)
Institut de Recherche en Horticulture et Semences (IRHS)

Soutenue le 13 décembre 2017

Caractérisation logique de données
Application aux données biologiques

JURY

Rapporteurs :	**M^me^ Laetitia JOURDAN**, Professeur des universités, Université Lille
	M. Lakhdar SAIS, Professeur des universités, Université d'Artois
Examinateurs :	**M. Tristan BOUREAU**, Maître de conférences-HDR, Université d'Angers
	M. Arnaud LALLOUET, Professeur des universités, Université de Caen Basse-Normandie
	M. Bruno ZANUTTINI, Maître de conférences-HDR, Université de Caen Basse-Normandie
Directeur de thèse :	**M. Frédéric SAUBION**, Professeur des universités, Université d'Angers
Co-directeur de thèse :	**M. Frédéric LARDEUX**, Maître de conférences-HDR, Université d'Angers

Remerciements

J'aimerais tout d'abord remercier les professeurs Laetitia Jourdan et Lakhdar Sais pour avoir eu le courage de relire cette thèse, d'en être rapporteurs, et pour le temps consacré à cette tâche. Je tiens également à remercier Arnaud Lallouet et Bruno Zanuttini pour avoir accepté d'être membre du jury et s'être intéressé à mon travail.

Je voudrais chaleureusement remercier Tristan Boureau, Frédéric Lardeux et Frédéric Saubion pour m'avoir dirigé tout au long de cette thèse. Je suis particulièrement reconnaissant du temps et de la patience qu'ils m'ont consacrés. Leurs précieux conseils, aussi bien scientifiques que pédagogiques, m'ont non seulement permis de m'intégrer dans le monde de la recherche et de l'enseignement, mais également d'apprécier grandement ces années de doctorat. Nul doute que je leur dois beaucoup et que j'ai beaucoup appris à leurs cotés. Soyez certain de mon estime et ma profonde gratitude.

Ce travail n'aurait pas été possible sans le soutien du projet régional GRIOTE et Angers Loire Métropole qui m'ont permis, grâce à une allocation de recherches et diverses aides financières, de me consacrer sereinement à l'élaboration de ma thèse.

Je remercie bien évidemment tous les membres du département informatique de l'UFR Sciences de l'université d'Angers et du laboratoire LERIA pour leur sympathie et leurs conseils. Je souhaite particulièrement remercier Matthieu Basseur, Benoit Da Mota, Béatrice Duval, Adrien Goëffon et André Rossi pour les nombreux conseils qu'ils m'ont apportés et m'avoir accordé une oreille attentive. Je souhaite également remercier les secrétaires, Christine Bardaine et Catherine Pawlonski pour leur sympathie et leur efficacité.

Je remercie également les membres de l'équipe Bio-Informatique de l'IRHS pour leur accueil chaleureux et leur sympathie. Il a été particulièrement simple de se sentir intégré au sein d'une équipe aussi conviviale.

Je voudrais également remercier les autres thésards qui m'ont supporté tous les jours durant ces trois dernières années. Nous avons passé de bons moments ensemble et c'est aussi grâce à eux que j'ai apprécié ces années de thèse.

Finalement, je souhaite remercier mes proches pour leur soutien et leur encouragement, et en particulier Géraldine pour son soutien moral et sa patience, notamment durant les derniers mois de rédaction qui n'ont pas été les plus simples. Merci beaucoup.

Table des matières

Liste des tableaux

Table des figures

1

Introduction

Contexte général

La collecte de données semble exister depuis les premières structures sociales et les premiers textes écrits retrouvés étaient des recensements de bétail[1]. On sait que des recensements étaient faits en Chine vingt-trois siècles avant J.C.. Sous l'empire romain, Cicéron (106 av. J.C. à 43 av. J.C.) insistait sur l'importance des statistiques descriptives. Le recensement romain permettait à la fois de connaître les ressources en hommes mobilisables et en biens, et de classer les citoyens afin de répartir charges et avantages. Le recensement était également une démonstration de puissance, permettant de proclamer publiquement l'ampleur de la domination romaine.

Il faudra toutefois attendre le milieu du XXème siècle avant d'observer une prise de conscience de l'importante quantité d'informations que l'on peut tirer des grands jeux de données[2] ainsi que les capacités technologiques pour stocker ces grands jeux de données. Mais ces données deviennent tellement massives et volumineuses qu'elles dépassent aujourd'hui l'intuition humaine.

La détection d'informations à partir de ces jeux de données est devenue rapidement un enjeu majeur. Par des méthodes statistiques et grâce à la puissance de calcul des ordinateurs, de nombreuses méthodes d'extractions de connaissances se sont développées pour décrire ces données mais également pour "prédire" des résultats. Ces méthodes sont souvent constituées de processus systématiques et s'exécutent au moyen d'algorithmes. L'ensemble de ces méthodes est désigné par le terme *analyse de données* mais relèvent aussi du domaine de l'apprentissage automatique.

[1] http ://www.math93.com/index.php/histoire-des-maths/les-developpements/564-histoire-des-statistiques
[2] https ://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/#6dc380d265a1

Peter L. Hammer (1936 à 2006) a proposé une alternative originale aux méthodes statistiques. Chercheur influent en recherche opérationnelle et mathématiques discrètes appliquées, il est considéré comme le fondateur de la "Théorie des Fonctions Pseudo-Booléennes". En étendant ses travaux sur les fonctions booléennes (fonctions définies sur $\{0,1\}^n$) à l'analyse de données, il développe *l'analyse logique de données*. Considérant initialement deux groupes de données booléennes (l'un positif et l'autre négatif) cette méthodologie a pour objectif, entre autres, de déterminer des motifs présents dans les données d'un groupe, mais jamais dans l'autre, et ce afin de les discriminer. Cette approche a eu un certain succès pour l'aide au diagnostic et au pronostic en médecine.

Dans le cadre de cette thèse, nous nous intéresserons à une extension de l'analyse logique de données à plus de deux ensembles initiaux, où la discrimination de tous les ensembles doit être simultanée et se baser sur les mêmes descripteurs. L'objectif de cette caractérisation multiple est donc d'obtenir un ensemble de critères permettant de distinguer chaque groupe de données.

Contexte biologique

Les problématiques liées à l'accumulation de grands jeux de données et leurs difficultés d'interprétation se posent dans le domaine de la biologie. Par exemple, les évolutions technologiques permettent aujourd'hui l'acquisition importante de données génomiques. Bien évidemment, ces données sont riches en informations plus ou moins pertinentes et l'extraction de l'information la plus pertinente devient un enjeu important. Isoler l'information pertinente demande ainsi des connaissances biologiques mais également mathématiques, statistiques et informatiques. C'est pourquoi, des interactions entre ces domaines scientifiques se sont développées, générant de nouveaux domaines de recherche comme la bio-informatique.

Depuis ces évolutions technologiques, et depuis que nous sommes capables de "séquencer" un génome, c'est à dire déterminer la séquence des gènes d'un génome, l'étude du vivant au travers son génome s'est considérablement développée. En particulier, la construction et la compréhension de "groupes" d'espèces vivantes n'est plus uniquement phénotypique, mais également génotypique.

Toutefois, la ressemblance génomique ne permet pas toujours d'expliquer les similarités entre les espèces vivantes que l'on pourrait regrouper de façon arbitraire. Par exemple, le dauphin est d'un point de vue génomique très proche de l'être humain, pourtant il est classé dans le groupe des animaux marins (il serait absurde de le considérer comme animal terrestre parce qu'il est génétiquement plus proche de l'homme que du poisson). À partir des données génomiques, la caractérisation va mettre en évidence les gènes pouvant jouer un rôle dans la classification arbitraire étudiée (dans notre exemple, quels sont les gènes nécessaires pour pouvoir vivre dans l'eau et être considéré comme un animal marin).

Initialement, dans le cadre de cette thèse, nous nous sommes intéressés à la caractérisation

de souches bactériennes du genre *Xanthomonas*. Ces souches sont classées par groupes nommés *pathovars*, c'est à dire en fonction de la pathologie observée et de l'hôte affecté. Toutefois, des souches bactériennes peuvent être proches d'un point de vue génomique et pourtant appartenir à des pathovars distincts, ou bien être différentes et appartenir au même pathovar. L'objectif de la caractérisation est donc de déterminer quels sont les descripteurs jouant un rôle dans l'appartenance à tel ou tel pathovar, dans un but descriptif (comprendre comment les souches appartenant à un pathovar donné provoquent une certaine pathologie chez l'hôte), ou prédictif (déterminer le pathovar dont une souche donnée fait partie).

Nous nous sommes aussi intéressés aux données de patients atteints de leucémie. La leucémie est un cancer des cellules de la moelle osseuse qui produisent les cellules sanguines. Les cellules leucémiques se comportent de manière anormale en raison d'une modification de leur génome avec une accumulation, au niveau de leur ADN, de mutations acquises qui permettent la transformation de la cellule. Ici, la caractérisation permet de mettre en évidence les gènes mutés pouvant jouer un rôle dans les différents types de leucémie et expliquer ou prévoir les rémissions/rechutes du patient.

Contributions

Différentes contributions ont été apportées dans le domaine de la caractérisation multiple de données au cours de cette thèse. Tout d'abord nous avons proposé une nouvelle formulation du problème de caractérisation multiple (par rapport à des travaux précédents (Chhel et al. [2012])) orientée sur les contraintes du problème et non sur les données.

Ensuite, nous avons proposé de nouveaux algorithmes permettant de résoudre le problème de caractérisation multiple et de déterminer l'ensemble des solutions optimales de ce problème. Cela nous permet d'extraire toute l'information non-redondante liée à la caractérisation.

Nous avons également établi des liens entre l'analyse logique de données et la caractérisation multiple, ce qui nous a permis de proposer une méthode efficace de génération de motifs optimaux (patterns) dans le contexte de l'analyse logique de données. À partir de ces résultats, nous avons pu étendre l'information recueillie et développer un nouvel axe d'optimisation de l'information récupérée lors de la caractérisation de données.

Enfin, nous avons étudié des données biologiques et comparé l'information obtenue par nos méthodes de caractérisation avec l'information obtenue par des méthodes statistiques d'analyse de données. Les résultats nous permettent de constater que la caractérisation multiple apporte des informations clairement différentes des méthodes statistiques usuelles.

Organisation du manuscrit

Tout au long de ce manuscrit, nous présentons les aspects algorithmiques et appliqués de la caractérisation de données. Ce manuscrit se scinde en trois chapitres :

- Un premier chapitre décrit le contexte biologique ainsi que l'état de l'art de la caractérisation multiple de données et des domaines connexes. Nous introduisons en premier lieu l'analyse logique de données et les diverses avancées qui ont été faites dans le domaine. Puis nous abordons le problème de caractérisation de données. Ensuite, nous présentons le problème de génération d'ensembles intersectants lié au problème de caractérisation multiple. Enfin, nous discutons d'analyse de données afin de présenter d'autres approches statistiques d'étude des données.

- Un second chapitre est consacré aux contributions algorithmiques pour le problème de caractérisation multiple et à l'analyse logique de données en terme d'algorithmes.

- Un troisième chapitre présente les différentes expérimentations et études que nous avons réalisées sur des données réelles. Cette étude consiste en premier lieu à comparer la performance de nos méthodes de caractérisation avec celles déjà existantes, puis à comparer l'information obtenue par nos méthodes de caractérisation avec celle obtenue par des méthodes statistiques et enfin à étudier précisément les solutions.

Nous terminons par une conclusion et des perspectives de travaux futurs.

2

État de l'art

2.1 Contexte biologique

Dans cette section, nous parlerons du contexte biologique initial à l'origine de cette thèse. Chhel et al. [2013] présente une application du problème de caractérisation multiple pour identifier des souches bactériennes appartenant au genre *Xanthomonas* à partir de gènes de virulences. Ces souches bactériennes sont responsables de pathologies sur une large gamme de cultures et engendrent des pertes de rendement importantes. Afin de prévenir de la destruction des récoltes infectées, des tests peuvent être effectués en amont de la mise en culture. Ces tests consistent à isoler des bactéries provenant de la surface des graines puis à effectuer une étude biochimique, sérologique (réaction d'anticorps des bactéries lorsqu'elles sont attaquées) (Engvall and Perlmann [1971]) ou simplement une observation d'apparition de signes d'infection d'une plante cible sur laquelle la bactérie suspectée a été introduite.

L'approche taxonomique regroupe les souches bactériennes au sein d'espèces. Les espèces définies doivent en théorie être représentatives des liens de parenté entre les souches. L'approche phylogénétique permet de délimiter des groupes de souches apparentées qui pourraient correspondre à une espèce bactérienne (cette approche est une classification ascendante hiérarchique basée sur la proximité génétique des organismes qui sera présentée dans la sous-section 2.5.1.1).
Cependant d'un point de vue pratique pour les pathologistes, le regroupement en espèce peut présenter certaines limitations :

- La délimitation de certaines espèces fait encore débat,

- Une même espèce bactérienne peut contenir des souches pathogènes sur des hôtes différents,

- La position phylogénétique n'est pas forcément corrélée à l'hôte attaqué par les souches, comme on peut le voir dans la Figure 2.1.

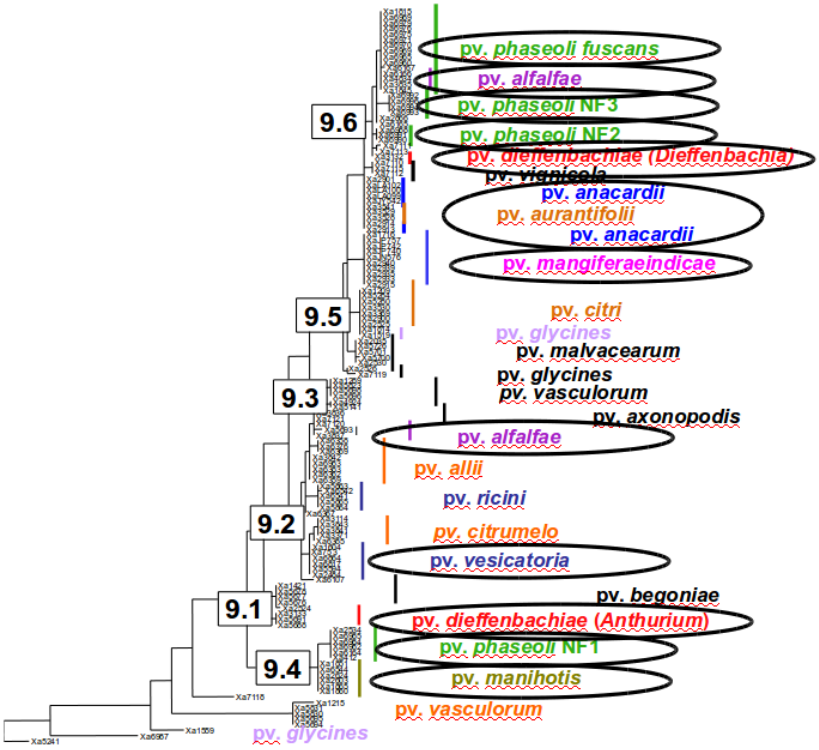


FIGURE 2.1 – Phylogénie de l'espèce *Xanthomonas*.
On constate que plusieurs souches d'un même pathovar sont très éloignées (*glycine, alfalfae, phaseoli, . . .*)

Au niveau pratique, les pathologistes utilisent un classement par pathovars basé sur les symptômes et les caractéristiques de pathogénicité. Par exemple, chez les souches bactériennes appartenant à *Xanthonomas axonopodis*, celles du pathovar *citri* sont responsables du chancre citrique des agrumes, celles du pathovar *phaseoli* de la graisse commune du haricot, celles du pathovar *vesicatoria* de la gale bactérienne du poivron, etc (Figure. 2.2). Un tel classement n'implique ni considération génétique ni description physique de la bactérie, et n'a pas de valeur taxonomique.
Il est donc difficile d'établir une relation entre le génotype des bactéries (i.e. l'information contenue dans l'ADN) et leur pouvoir pathogène. Il est ainsi important d'extraire les caractéristiques génétiques les plus pertinentes. Dès lors que dans un même pathovar, deux bactéries éloignées dans la classification phylogénétique peuvent quand même partager des gènes com-

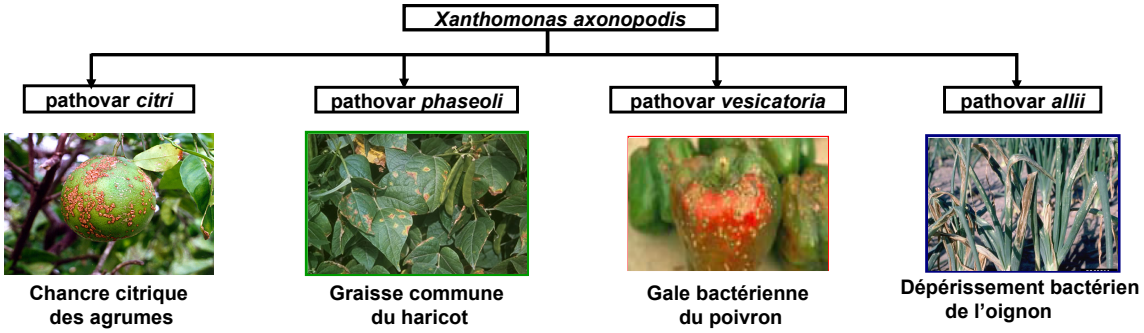


FIGURE 2.2 – Maladies provoquées par différentes souches bactériennes provenant de différents pathovars du genre *Xanthomonas*.

muns, ces derniers peuvent s'avérer induire un comportement pathogène similaire des deux bactéries.

Hajri et al. [2009] ont observé une corrélation entre les pathovars et la composition des répertoires de gènes de virulence chez les bactéries du genre Xanthomonas. Ainsi les souches bactériennes peuvent être vues comme un vecteur de valeurs binaires qui reflète la présence ou l'absence de caractères particuliers. En déterminant le plus petit ensemble de gènes de virulence caractérisant chaque pathovar, il est possible de concevoir un test PCR multiplex pour l'identification de chaque souche :
Pour identifier les pathologies inoculées par les bactéries, la technique décrite dans Boureau et al. [2013] consiste à utiliser une PCR multiplex (*Polymerase chain reaction*) indiquant la présence ou l'absence de gène de virulence. La PCR est une méthode qui consiste à multiplier l'ADN d'un échantillon grâce à l'ajout d'enzyme polymérase. Des marqueurs sont ensuite ajoutés et permettent la détection de l'expression ou non d'un gène donné en fonction d'un seuil de concentration. Par abus de langage, nous parlerons de présence/absence d'un gène pour l'expression ou non dudit gène. L'objectif de la caractérisation de données est ainsi de calculer l'ensemble de gènes à étudier pour évaluer le pathovar auquel appartient la bactérie testée.

De plus, le diagnostic étant basé sur des puces à ADN, la minimisation du nombre de spots pour la détection d'un seul pathogène peut permettre de multiplexeur à la détection de plusieurs pathogènes en simultanée, ce qui permettrait de construire des outils plus génériques et moins couteux. Ceci revient à chercher une solution de taille minimale du problème de caractérisation multiple (que nous décrirons dans la section 2.3).

2.2 Analyse logique de données

L'analyse logique de données est une méthodologie basée sur la détermination de combinaisons de valeurs binaires, nommées patterns, discriminant un sous-ensemble d'un groupe de données

de manière exacte. Il s'agit d'une alternative originale à l'analyse de données classique qui, elle, est constituée de méthodes statistiques d'étude des données, mais dont la finalité consiste également à étudier un jeu de données dans un but descriptif et/ou prédictif.

Dans cette section, nous présenterons l'analyse logique de données au travers d'exemples et de définitions formelles, puis nous présenterons diverses méthodes de calcul des patterns discriminants avant de finir sur quelques exemples d'application.

2.2.1 Présentation

L'analyse logique de données, introduite pour la première fois par Peter Hammer (Hammer [1986]) est une méthode d'analyse s'appuyant sur l'optimisation combinatoire et le concept de fonctions booléennes partiellement définies. Son objectif est la caractérisation de données au moyen de motifs. Ces motifs, nommés patterns, sont des ensembles de valeurs présentes dans des observations appartenant à un même ensemble de données (groupe) et permettant ainsi d'identifier des caractéristiques communes. L'idée est de se concentrer sur la justification explicite des groupes de données et constitue en cela une alternative originale aux approches de classification traditionnelles de l'apprentissage artificiel qui vont, elles, se concentrer sur la construction ou l'identification de ces groupes. Plus concrètement, l'objectif de l'analyse logique de données est :

- De déterminer des similarités chez certains individus appartenant à un même ensemble de données,
- De discriminer les individus n'appartenant pas à un ensemble de données.
- De déduire des règles logiques propres à chacun des ensembles de données.

La figure 2.3 représente d'une façon naïve un pattern d'un ensemble de points (noirs) par une zone hachurée : cette zone recouvre uniquement des points noirs et met donc en évidence une similarité/proximité entre ces points. Notons qu'un pattern ne recouvre pas nécessairement l'ensemble des observations d'un groupe. En effet, dans notre représentation, nous ne pouvons pas élargir notre zone afin de recouvrir les deux points noirs restant sans recouvrir un point blanc.
Commençons avec un exemple pratique :

Exemple 1. *Un joueur d'échec étudie 7 parties (numérotées de* 1 *à* 7*) avec 8 stratégies (numérotées de* a *à* h*) qu'il a tenté ou non d'appliquer. Parmi ces 7 parties, il en a perdu 3. Le joueur reporte donc les résultats ainsi que les stratégies jouées à chaque partie dans le tableau suivant (*$x_{ij} = 1$ *s'il a utilisé la stratégie* j *dans la partie* i*,* 0 *sinon).*

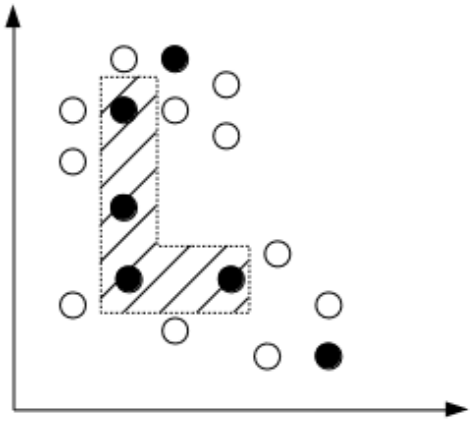


FIGURE 2.3 – Représentation naïve d'un pattern

Parties	*Résultat*	*Stratégies*							
		a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
1	*Défaite*	*0*	*1*	*0*	*1*	*0*	*1*	*1*	*0*
2		*1*	*1*	*0*	*1*	*1*	*0*	*0*	*1*
3		*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*
4	*Victoire*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*1*
5		*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
6		*1*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
7		*0*	*0*	*1*	*0*	*1*	*0*	*1*	*0*

Une étude rapide de ce tableau permet d'en tirer quelques remarques :

- Dans les parties gagnées, la stratégie b n'a jamais été utilisée sans la stratégie a, contrairement à certaines parties perdues. Utiliser la stratégie b sans utiliser la stratégie a n'est peut être pas très pertinent. Nous dirons qu'utiliser b et ne pas utiliser a est un "pattern" du groupe des parties perdues.

- De même, dans les parties gagnées, soit la stratégie f a été utilisée, soit la g. Or, dans les parties perdues soit les deux stratégies ont été utilisées, soit aucune. Il y a donc peut-être une incompatibilité entre ces deux stratégies. Utiliser f et g est donc un "pattern" du groupe des parties perdues ainsi que n'utiliser ni f ni g.

Ces trois "patterns" ne sont bien évidemment pas les seuls et nous pouvons imaginer des combinaisons créées à partir de plus de deux stratégies. L'analyse logique de données permet de générer les patterns pour identifier s'il y aura victoire ou non et les listes des stratégies suffisantes pour expliquer ces résultats.

Dans notre exemple, nous ne considérons que deux états possibles pour chaque stratégie : la stratégie est jouée ou non. Nous travaillons donc sur des données binaires. Toutefois, en pratique, ce n'est pas toujours le cas. On pourrait par exemple prendre en compte une stratégie commencée mais incomplète (donc une nouvelle modalité), voir le pourcentage de réalisation de la stratégie (donc des données continues au lieu de binaires). Dans Boros et al. [1997], la "binarisation" de données est introduite, permettant de passer de données réelles (i.e. à valeur dans $\mathbb{R}$) en données binaires. L'idée de base de la binarisation est simple : chaque donnée réelle est associée à un seuil puis on considère que la donnée binaire vaut 1 (respectivement 0) si la donnée réelle est au dessus de ce seuil (respectivement en dessous). Un unique seuil étant trop discriminant, il est important d'étudier plusieurs combinaisons de seuil. Un des problèmes lié à la binarisation est donc de trouver un nombre minimal de ces seuils qui préserverait l'essentiel de l'information contenue dans un jeu de données (Boros et al. [1997]; Hammer and Bonates [2006]).

2.2.2 Terminologie et notation

Nous reprenons les principaux concepts de l'analyse logique de données (LAD par la suite) (Boros et al. [2011]; Chikalov et al. [2013]; Hammer and Bonates [2006]; Hammer et al. [2004]). Nous utilisons en particulier ceux des fonctions booléennes partiellement définies.

Définition 1. *Une fonction booléenne f de n variables, $n \in \mathbb{N}$, est une fonction $f : \mathbb{B}^n \to \mathbb{B}$, où $\mathbb{B}$ est l'ensemble $\{0, 1\}$.*

Les n variables sont donc dans l'exemple précédent les stratégies utilisées ou non par le joueur.

Définition 2. *Un vecteur $x \in \mathbb{B}^n$ est un vecteur positif (resp. vecteur négatif) de la fonction booléenne f si $f(x) = 1$ (resp. $f(x) = 0$). $T(f)$ (resp. $F(f)$) est l'ensemble des vecteurs positifs (resp. vecteurs négatifs) de la fonction booléenne f.*

Dans l'exemple précédent, chaque partie perdue est donc représentée par un vecteur booléen positif, et chaque partie gagnée par un vecteur négatif. Par la suite, nous nommerons chacune de ces parties une observation. L'ensemble des observations sera l'ensemble Ω.

Définition 3. *Une fonction booléenne partiellement définie (pdBf) sur $\mathbb{B}^n$ est un couple (P,N) tel que $P,N \subseteq \mathbb{B}^n$ et $P \cap N = \emptyset$.*

Dans une pdBf, les observations sont donc représentées par des vecteurs booléens et classées en deux groupes P (groupe positif) et N (groupe négatif), représentant par exemple le groupe des parties perdues et gagnées.

Un *littéral* est soit une variable binaire x_i soit sa négation $\bar{x}_i$. Un terme est ainsi une pdBf représentée par une conjonction de littéraux distincts qui ne contient pas à la fois une variable et sa négation.

Définition 4. *Étant donné :* $\sigma^+, \sigma^- \subseteq \{1, 2, ..., n\}$, $\sigma^+ \cap \sigma^- = \emptyset$, *un terme* t_{σ^+,σ^-} *est une fonction booléenne dont l'ensemble positif* $T(t_{\sigma^+,\sigma^-})$ *est de la forme :*

$$T(t_{\sigma^+,\sigma^-}) = \{x \in \mathbb{B}^n | x_i = 1 \; \forall i \in \sigma^+ \textit{ et } x_j = 0 \; \forall j \in \sigma^-\}$$

Un terme t_{σ^+,σ^-} peut être représenté par une *conjonction élémentaire*, i.e., une expression booléenne de la forme :

$$t_{\sigma^+,\sigma^-}(x) = (\bigwedge_{i \in \sigma^+} x_i) \wedge (\bigwedge_{j \in \sigma^-} \bar{x}_j)$$

Un vecteur booléen satisfait (ou vérifie) un terme s'il possède les mêmes valeurs binaires que le terme sur les variables en question, quelque soit la valeur de ses autres variables.

L'ensemble des littéraux d'un terme t se note $Lit(t)$. Le *degré* d'un terme t est le nombre de littéraux qui compose ce terme, c'est à dire $|Lit(t)|$.

Définition 5. *Un pattern d'une pdBf* (*P*,*N*) *est un terme* t_{σ^+,σ^-} *tel que* $|P \cap T(t_{\sigma^+,\sigma^-})| > 0$ *et* $|N \cap T(t_{\sigma^+,\sigma^-})| = 0$.

Un pattern est donc un terme satisfait par au moins un vecteur positif et aucun vecteur négatif. Bien évidemment, les patterns sont associés au groupe positif. Il est également possible de concevoir des patterns associés au groupe négatif en considérant la pdBf (N,P) au lieu de la pdBf (P,N).

Définition 6. *La couverture d'un pattern* p, *noté* $Cov(p)$ *est l'ensemble* $Cov(p) = P \cap T(p)$.

Autrement dit, la couverture d'un pattern p est l'ensemble des vecteurs booléens positif satisfaisant p. Pour simplifier la compréhension, nous représenterons par la suite chaque vecteur booléen par le numéro de l'observation qui lui est associée. Le cas particulier où deux observations sont représentées par le même vecteur booléen sera étudié dans la sous-section 2.3.2.

Exemple 2. *Reprenons l'exemple 1 formalisé comme un problème d'analyse logique de données :*

Observations	*Groupes*	*Variables*							
		a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
1	*P*	*0*	*1*	*0*	*1*	*0*	*1*	*1*	*0*
2		*1*	*1*	*0*	*1*	*1*	*0*	*0*	*1*
3		*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*
4	*N*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*1*
5		*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
6		*1*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
7		*0*	*0*	*1*	*0*	*1*	*0*	*1*	*0*

$p_1 = \bar{a} \wedge b$ et $p_2 = \bar{f} \wedge \bar{g}$ sont donc deux patterns couvrant les observations 1 et 3 pour p_1 et 2 et 3 pour p_2.

Un terme peut également avoir une représentation géométrique sous forme de sous-cube d'un cube de n dimensions $\{0,1\}^n$ représentant tous les vecteurs booléens possibles vérifiant le terme. Soit un terme t, l'ensemble des points de ce sous-cube, i.e. l'ensemble des vecteurs booléens respectant ce pattern, se note $S(t)$. Reprenons l'exemple 2 et considérons le pattern $p = \bar{a} \wedge \bar{c} \wedge d \wedge f \wedge g \wedge \bar{h}$ (ou de manière équivalente $a = 0, c = 0, d = 1, f = 1, g = 1, h = 0$). Nous avons donc $S(p) = \{(00010110), (00011110), (01010110), (01011110)\}$.

Les patterns peuvent être très nombreux, c'est pourquoi il est important d'introduire des relations d'ordre pour les comparer. Avec les trois ensembles $Lit(p)$, $S(p)$ et $Cov(p)$ nous pouvons définir 3 types de préordres partiels : *la préférence de simplicité, la préférence sélective* et *la préférence évidentielle*.

Définition 7. *La préférence de simplicité σ, notée $\succcurlyeq_\sigma$, est une relation binaire sur un ensemble de pattern $\mathcal{P}$ tel que pour un couple $(p_1, p_2) \in \mathcal{P}^2$, on a $p_1 \succcurlyeq_\sigma p_2$ si et seulement si $Lit(p_1) \subseteq Lit(p_2)$.*

Définition 8. *La préférence sélective Σ, notée $\succcurlyeq_\Sigma$, est une relation binaire sur un ensemble de pattern $\mathcal{P}$ tel que pour un couple $(p_1, p_2) \in \mathcal{P}^2$, on a $p_1 \succcurlyeq_\Sigma p_2$ si et seulement si $S(p_1) \subseteq S(p_2)$.*

Définition 9. *La préférence évidentielle $\mathcal{E}$, notée $\succcurlyeq_\mathcal{E}$, est une relation binaire sur un ensemble de pattern $\mathcal{P}$ tel que pour un couple $(p_1, p_2) \in \mathcal{P}^2$, on a $p_1 \succcurlyeq_\mathcal{E} p_2$ si et seulement si $Cov(p_2) \subseteq Cov(p_1)$.*

Pour une préférence $\succcurlyeq_\pi$ ($\pi \in \{\sigma, \Sigma, \mathcal{E}\}$) et deux patterns p_1 et p_2, nous noterons la double relation $p_1 \succcurlyeq_\pi p_2$ et $p_2 \succcurlyeq_\pi p_1$ par $p_1 \approx_\pi p_2$.

Exemple 3. *Reprenons l'exemple 2 et considérons les patterns* $p_1 = \bar{a} \wedge \bar{c} \wedge d \wedge f \wedge g \wedge \bar{h}$, $p_2 = f \wedge g$ *et* $p_3 = \bar{a} \wedge b$. *Nous avons* $p_2 \succcurlyeq_\sigma p_1$ *car* $Lit(p_2) \subseteq Lit(p_2)$, $p_1 \succcurlyeq_\Sigma p_2$ *car* $S(p_1) \subseteq S(p_2)$ *et* $p_3 \succcurlyeq_\mathcal{E} p_2$ *car les deux patterns couvre l'observation 1 mais seul* p_3 *couvre l'observation 3, donc* $Cov(p_2) \subseteq Cov(p_3)$.

À partir de ces définitions, nous pouvons noter plusieurs propriétés :

Propriété 1.

- *(1)* $p_1 \succcurlyeq_\sigma p_2$ *si et seulement si* $p_2 \succcurlyeq_\Sigma p_1$,
- *(2) Si* $p_1 \succcurlyeq_\sigma p_2$ *alors* $p_1 \succcurlyeq_\mathcal{E} p_2$,
- *(3) Si* $p_1 \succcurlyeq_\Sigma p_2$ *alors* $p_2 \succcurlyeq_\mathcal{E} p_1$,
- *(4) Si* $p_1 \approx_\sigma p_2$ *alors* $p_1 = p_2$,
- *(5) Si* $p_1 \approx_\Sigma p_2$ *alors* $p_1 = p_2$.

Afin d'affiner la comparaison des patterns, nous allons considérer des combinaisons de préférences.

Définition 10. *Soit deux préférences* π *et* ρ *sur un ensemble de patterns* $\mathcal{P}$, *et soit* $(p_1, p_2) \in \mathcal{P}^2$,

- *le pattern* p_1 *est préféré au pattern* p_2 *au sens de l'intersection* $\pi \cap \rho$, *noté* $p_1 \succcurlyeq_{\pi\cap\rho} p_2$, *si et seulement si* $p_1 \succcurlyeq_\pi p_2$ *et* $p_1 \succcurlyeq_\rho p_2$.
- *le pattern* p_1 *est préféré au pattern* p_2 *au sens du raffinement lexicographique* $\pi|\rho$, *noté* $p_1 \succcurlyeq_{\pi|\rho} p_2$, *si et seulement si* $p_1 \succcurlyeq_\pi p_2$ *ou* $p_1 \approx_\pi p_2$ *et* $p_1 \succcurlyeq_\rho p_2$.

Notons qu'à partir des propriétés présentées ci-dessus (propriété 1) nous remarquons que :

- À partir de la propriété (1), nous avons une équivalence entre les préférences σ et Σ. Donc l'étude des préférences $\sigma \cap \Sigma$, $\sigma|\Sigma$ et $\Sigma|\sigma$ n'ont pas de sens,
- À partir de la propriété (2), la préférence au sens de σ implique celle au sens de $\mathcal{E}$, donc la préférence $\sigma \cap \mathcal{E}$ est équivalente à σ,
- À partir de la propriété (4), on voit que la préférence σ offre un ordre total. Donc la préférence $\sigma|\mathcal{E}$ est équivalente à σ,
- À partir de la propriété (5), on voit que la préférence Σ offre un ordre total. Donc la préférence $\Sigma|\mathcal{E}$ est équivalente à Σ.

Au final, les préférences intéressantes à étudier sont σ, Σ, $\mathcal{E}$, $\Sigma \cap \mathcal{E}$, $\mathcal{E}|\sigma$ et $\mathcal{E}|\Sigma$.

Définition 11. *Soit une préférence* $\succcurlyeq_\pi$ *sur un ensemble de patterns* $\mathcal{P}$, *un pattern* $p_1 \in \mathcal{P}$ *est Pareto-optimal au sens de* π *si et seulement si* $\nexists p_2 \in \mathcal{P}\backslash\{p_1\}$ *tel que* $p_2 \succ_\pi p_1$ *(i.e* $p_2 \succcurlyeq_\pi p_1$ *et* $p_2 \not\approx_\pi p_1$*).*

Nous pouvons ainsi définir six types de patterns Pareto-optimaux, en fonction des six préférences :

Préférences	Patterns Pareto-optimaux
σ	Prime pattern
Σ	Minterm
$\mathcal{E}$	Strong pattern
$\Sigma \cap \mathcal{E}$	Spanned pattern
$\mathcal{E}\|\sigma$	Strong prime pattern
$\mathcal{E}\|\Sigma$	Strong spanned pattern

Remarquons les deux propriétés suivantes démontrées dans Hammer et al. [2004].

Propriété 2. *Un pattern est pareto-optimal au sens de :*

- $\mathcal{E}|\sigma$ *si et seulement s'il est strong et prime.*
- $\mathcal{E}|\Sigma$ *si et seulement s'il est strong et spanned.*

Exemple 4. *Reprenons l'exemple 2 pour illustrer les différents types de patterns :*

- *Le pattern* $a \wedge d \wedge e$ *est un prime pattern car si on retire un littéral, ce n'est plus un pattern.*
- *Le pattern* $\bar{a} \wedge b \wedge \bar{c} \wedge d \wedge \bar{e} \wedge f \wedge g \wedge \bar{h}$ *est un minterm car on ne peut lui rajouter de littéraux.*
- *Le pattern* $e \wedge \bar{f} \wedge \bar{g}$ *est un strong pattern car il n'existe pas de pattern couvrant les mêmes observations plus une.*
- *Le pattern* $b \wedge e \wedge \bar{f} \wedge \bar{g} \wedge h$ *est un spanned pattern car on ne peut pas lui rajouter un littéral sans diminuer sa couverture. Remarquons que ce pattern est également strong, c'est donc un strong spanned pattern.*
- *Le pattern* $\bar{f} \wedge \bar{g}$ *est à la fois strong et prime. Il s'agit donc d'un strong prime pattern.*

Dans Bonates et al. [2008], un nouveau type de pattern est introduit :

Définition 12. *Soit* p_1 *un pattern appartenant à un ensemble de patterns* $\mathcal{P}$*, nous dirons que* p_1 *est un pattern maximal si et seulement si* $\nexists p_2 \in \mathcal{P}$ *tel que* $|Cov(p_2)| > |Cov(p_1)|$.

Dans l'exemple précédent, le pattern $e \wedge \bar{g} \wedge h$ est maximal car il couvre deux observations du groupe positif et il n'existe aucun pattern couvrant les trois observations de ce groupe.

Les patterns maximaux sont donc les patterns les plus couvrants. Ainsi, un pattern maximal est toujours strong, mais tous les patterns strong ne sont pas maximaux.

Dans Boros et al. [2000] est présenté une autre vision d'optimisation : les *support sets*. Un support set est un sous-ensemble de variables tel que, en ne travaillant que sur les variables sélectionnées, tous les vecteurs booléens positifs sont différents des vecteurs booléens négatifs. Autrement dit, nous cherchons un ensemble de variables discriminant tout le groupe. Un support set est dit *irréductible* si le retrait d'une variable, quelle qu'elle soit, donne un ensemble de variables qui n'est plus un support set.

La recherche de patterns maximaux va donc être un problème de maximisation sur la couverture tandis que la recherche de support set irréductible va être un travail de minimisation sur les variables.

2.2.3 Génération de patterns

Nous allons présenter dans cette sous-section plusieurs méthodes pour générer différents types de patterns. En premier lieu, nous présenterons l'approche de générations de patterns par la programmation linéaire, puis nous présenterons un algorithme permettant de générer l'ensemble des primes patterns d'une instance et un autre déterminant l'ensemble des spanned patterns. Enfin nous présenterons des méthodes permettant de transformer des patterns en patterns Pareto-optimaux.

2.2.3.1 Programme linéaire

Le problème de génération de patterns optimaux peut se résoudre par un programme linéaire. Dans Ryoo and Jang [2009] sont proposés des programmes linéaires en nombres entiers permettant de générer différents types de pattern.

L'idée principale de ces programmes linéaires est d'utiliser un vecteur booléen x de $2n$ valeurs où n est le nombre de variables du problème initial (notées de l_1 à l_n). Un vecteur x sera ainsi associé à un pattern p où la valeur du j-ième terme vaut 1 si $0 < j \leqslant n$ et $l_j \in Lit(p)$ ou si $n < j \leqslant 2n$ et $\bar{l}_{j-n} \in Lit(p)$, 0 sinon. Ainsi, soit l'ensemble $V = \{1, 2, \ldots, n\}$ représentant l'ensemble des indices des variables, un terme t sera généré à partir d'un vecteur x tel que : $t = (\bigwedge_{\substack{x_j=1 \\ j\in V}} l_j) \wedge (\bigwedge_{\substack{x_{n+j}=1 \\ j\in V}} \bar{l}_j)$.

Bien évidemment, pour une même variable j, nous ne pouvons pas avoir les littéraux l_j et $\bar{l}_j$ présents dans le même pattern. Une première contrainte est donc : $\forall j \in V, x_j + x_{n+j} \leqslant 1$. Chaque vecteur booléen x respectant cette contrainte peut être associé à un terme.

Afin de générer un pattern d'un degré d souhaité, la contrainte $\sum_{j=1}^{2n} x_j = d$ est ajoutée.

L'ensemble des vecteurs x réalisables sous ces deux contraintes correspond à l'ensemble des termes de degré d. Il faut maintenant imposer des contraintes permettant de discriminer les patterns parmi ces termes. Pour qu'un terme soit un pattern, deux conditions sont neccessaires :

- (1) Aucune observation du groupe négatif ne doit être couverte par le pattern,

- (2) Au moins une observation du groupe positif doit être couverte par le pattern.

Pour modéliser ces conditions, un vecteur booléen a^o est introduit pour chaque observation o. Ce vecteur possède également $2n$ valeurs, dont les n premières seront identiques au vecteur booléen décrivant l'observation, et $\forall j \in V$, $a^o_{n+j} = 1 - a^o_j$.

Ainsi, pour un vecteur x et le terme t associé à x, $\forall j \in \{1, \ldots, n\}$, $a^0_j x_j = 1$ si et seulement si le vecteur booléen associé à o a pour j-ième valeur 1 et que $l_j \in Lit(t)$ et $\forall j \in \{n+1, \ldots, 2n\}$, $a^0_j x_j = 1$ si et seulement si le vecteur booléen associé à o a pour $(j-n)$-ième valeur 0 et $\bar{l}_j \in Lit(t)$. Si le terme est de degré d, nous devons avoir exactement d valeurs j pour lesquelles $a^o_j x_j = 1$ afin de pouvoir affirmer que le terme couvre l'observation o. Pour vérifier la condition (1) nécessaire à un terme pour être un pattern, nous devons donc vérifier que les observations appartenant au groupe négatif ne sont pas couvertes, d'où les contraintes suivantes :

$$\sum_{j=1}^{2n} a^o_j x_j \leqslant d - 1, \quad \forall o \in N.$$

Un autre vecteur booléen y est introduit pour modéliser la seconde condition. Ce vecteur se compose de $|P|$ valeurs, et y_j vaut 0 si la j-ième observation est couverte par le terme généré par le vecteur x, 1 sinon. Nous avons donc :

$$y_o = \begin{cases} 0 & \text{si } \sum_{j=1}^{2n} a^o_j x_j \geqslant d \qquad (1) \\ 1 & \text{sinon.} \end{cases}$$

Pour chaque observations positives, les auteurs de Ryoo and Jang [2009] impose la contrainte :

$$\sum_{j=1}^{2n} a^o_j x_j + n y_o \geqslant d, \quad \forall o \in P. \quad (a)$$

Notons toutefois que cette contrainte est toujours vérifiée, car $n \geqslant d$, et ne permet que de fixer y_o à 1 si $\sum_{j=1}^{2n} a^o_j x_j < d$. Pour vérifier la seconde condition, il aurait fallu utiliser la contrainte suivante :

$$\sum_{o=1}^{|P|} y_o < |P|. \quad (b)$$

et pour "mettre à jour" les y_o, utiliser les contraintes

$$\sum_{j=1}^{2n} a^o_j x_j \geqslant d(1 - y_o), \quad \forall o \in P. \quad (c)$$

et

$$(\sum_{j=1}^{2n} a_j^o x_j) \times y_o < d, \quad \forall o \in P. \quad (d)$$

La contrainte (c) permet de fixer y_o à 1 si l'observation o n'est pas couverte ($\sum_{i=1}^{2n} a_i^o x_i < d$) et la contrainte (d) permet de fixer y_o à 0 si l'observation o est couverte ($\sum_{i=1}^{2n} a_i^o x_i \geqslant d$). En fait, (c) et (d) permettent de modéliser la valeur conditionnelle de y comme dans l'équation (1).

Maintenant, pour maximiser la couverture, nous chercherons une solution (x, y, d) qui couvrira le plus d'observations possible, et donc minimisera la somme des y_o. Les auteurs de Ryoo and Jang [2009] proposent donc le programme linéaire suivant :

$$Z = min_{x,y,d} \sum_{o=1}^{|P|} y_o$$

$$s.t. :$$

$$\sum_{j=1}^{2n} a_j^o x_j + n y_o \geqslant d, \quad \forall o \in P \quad (a)$$

$$\sum_{j=1}^{2n} a_j^o x_j \leqslant d - 1, \quad \forall o \in N$$

$$x_j + x_{n+j} \leqslant 1, \quad \forall j \in \{1, \ldots, n\}$$

$$\sum_{j=1}^{2n} x_j = d$$

$$1 \leqslant d \leqslant n$$

$$x \in \{0, 1\}^{2n}$$

$$y \in \{0, 1\}^{|P|}$$

$$d \in \mathbb{N}$$

Remarque 1. *Une preuve est apportée pour montrer que si une solution (x, y, d) est réalisable, alors le terme généré par x est un pattern. Toutefois, la preuve est incomplète et ce théorème est faux. En particulier, une solution (x, y, d) pour laquelle $y_o = 1 \; \forall o \in P$ peut être réalisable alors que, par construction, elle ne couvre aucune observation positive.*

Comme indiqué ci-dessus, la contrainte (a) est insuffisante. En la remplaçant par les contraintes (b), (c) et (d), cette fois-ci, nous pouvons dire qu'une solution (x, y, z) est réalisable si et seulement si le terme généré par x est un pattern. Toutefois, avec la contrainte (a) ou les contraintes (b), (c) et (d), le second théorème de Ryoo and Jang [2009] est vérifié dès lors qu'il existe un pattern :

Théorème 1. *Soit (x, y, d) une solution optimale, le terme p généré par x est un strong pattern*

de degré d.

Notons toutefois que la réciproque n'est pas vraie. En particulier, une solution (x, y, d) est optimale si et seulement si le pattern p généré par x est un pattern maximal. Mais si tous les patterns maximaux sont strong, tous les strong patterns ne sont pas maximaux.

En remplaçant la fonction objectif par

$$Z = min_{x,y,d} \sum_{i=1}^{|P|} y_i + cd$$

nous obtenons un nouveau programme linéaire dont une solution optimale (x, y, d) permettra de générer un strong prime pattern si $c > 0$ ou un strong spanned pattern si $c < 0$. Notons qu'en fonction de c, la solution changera. Notons également que quel que soit c, certains strong prime patterns et strong spanned patterns pourraient n'être représentés par aucune solution optimale du programme linéaire.

Exemple 5. *Soit les données suivantes :*

Observations	*Groupes*	*Variables*			
		a	*b*	*c*	*d*
1		*1*	*1*	*1*	*1*
2	*P*	*1*	*1*	*0*	*0*
3		*0*	*0*	*0*	*0*
4		*1*	*0*	*1*	*1*
5	*N*	*1*	*1*	*0*	*1*
6		*0*	*1*	*1*	*1*

$p_1 = a \wedge b \wedge c$ *et* $p_2 = \bar{d}$ *sont les deux seuls strong prime patterns couvrant l'observation 1 pour* p_1 *et 2 et 3 pour* p_2. *Or* p_1 *est associé à la solution* $s_1 = (x, y, d) = ((1\,1\,1\,0\,0\,0\,0\,0), (0\,1\,1), 3)$ *et* p_2 *est associé à la solution* $s_2 = (x', y', d') = ((0\,0\,0\,0\,0\,0\,0\,1), (1\,0\,0), 1)$. *On a donc* $\sum_{o \in P} y_o > \sum_{o \in P} y'_o$ *et* $\forall c > 0$, $cd > cd'$ *et donc,* $\forall c \geqslant 0$, s_2 *est la solution optimale. La solution associée au pattern* p_1 *ne sera jamais retournée par le programme linéaire.*

Si l'approche par programmation linéaire ne nous permet pas de générer tous les patterns optimaux, d'autres algorithmes le font. Dans Boros et al. [2000] est proposé un algorithme générant l'ensemble des prime patterns tandis que dans Alexe and Hammer [2006] est défini un algorithme générant l'ensemble des spanned patterns. Notons également que dans Hammer et al. [2004] sont décrits trois algorithmes transformant un pattern en prime pattern, en spanned pattern et en strong pattern.

2.2.3.2 Génération de prime patterns

Concernant la génération des prime patterns, l'algorithme de Boros et al. [2000] propose la génération de tous les prime patterns inférieurs ou égaux à un degré d. En choisissant le degré d_{max} le plus élevé des prime patterns, on peut tous les générer, mais nous ne savons pas calculer ce degré sans générer l'ensemble des patterns. Nous choisirons donc une borne supérieure à d_{max} afin d'être certains de notre résultat. En choisissant $d = n$ le nombre de variables, nous sommes ainsi certain que $d \geqslant d_{max}$ et ainsi, l'algorithme retournera l'ensemble des prime pattterns.

Le fonctionnement de l'algorithme de Boros est très simple. Il génère en premier lieu les prime patterns de degré 1, puis ceux de degré 2 et ainsi de suite jusqu'au degré d. À l'étape 1, l'algorithme teste tous les termes de degré 1 et les classe en fonction de leur intérêt. Tous les termes qui sont des patterns seront dans un ensemble P, et dans un ensemble C_1 les termes de taille 1 couvrant des observations positives et négatives. Aux étapes suivantes (E) ne seront considérés que les termes de l'ensemble C_{E-1}, termes de degré $E-1$, pour lesquels on essaye de rajouter un littéral pour le transformer en terme de degré E. Si ce terme est un pattern, il rejoint l'ensemble P et s'il couvre toujours à la fois des observations positives et négatives, il rejoint l'ensemble C_E. Une fois terminée l'étape $E = d$, nous aurons l'ensemble P qui contiendra tous les prime patterns de degré inférieur ou égal à d.

L'algorithme 1 présente le pseudo-code de cet algorithme.

Comme nous testons toutes les combinaisons, nous sommes certain de générer l'ensemble des prime patterns. De plus, pour être généré, un terme doit être dans l'ensemble P et donc vérifier les conditions pour être un pattern. Comme avant d'être dans l'ensemble P, le terme était dans un ensemble C_i, si on lui retire un littéral, le terme n'est plus un pattern. L'ensemble P contient donc tous les prime patterns et ne contient que des prime patterns.

2.2.3.3 Algorithme de génération des spanned patterns

Concernant les spanned patterns, l'algorithme `SPIC`, présenté dans Alexe and Hammer [2006], va générer l'ensemble de ces patterns grâce à une nouvelle notion d'intervalle appelé *consensus*.

Soit deux vecteurs booléens α et β de n variables, l'intervalle généré par ces vecteurs booléens est un couple de vecteurs booléens $[\alpha', \beta']$ tel que $\forall i \in \{1, \ldots, n\}$, $\alpha'_i = min(\alpha_i, \beta_i)$[1] et $\beta'_i = max(\alpha_i, \beta_i)$[2]. Par exemple, si $\alpha = (0, 1, 1)$ et $\beta = (1, 0, 1)$, l'intervalle généré par ces vecteurs booléens sera $[(0, 0, 1), (1, 1, 1)]$. Un vecteur booléen γ sera couvert par un intervalle $[\alpha, \beta]$ si $\forall i \in \{1, \ldots, n\}$, $\alpha_i \leqslant \gamma_i \leqslant \beta_i$. Par exemple $\gamma = (1, 1, 1)$ sera couvert par $[\alpha', \beta']$ tandis que $\gamma' = (1, 1, 0)$ ne le sera pas.

[1] Dans le cas général, nous pouvons travailler sur des données réelles, mais dans notre contexte nous travaillons sur des données booléennes et donc $min(\alpha_i, \beta_i) = \alpha_i \wedge \beta_i$.

[2] Dans notre contexte $max(\alpha_i, \beta_i) = \alpha_i \vee \beta_i$ car nous travaillons sur des données booléennes.

Données : D le degré maximal des patterns qui seront générés.
Résultat : P_D l'ensemble des prime patterns de taille plus petite ou égale à D.
$P_0 = \emptyset$
//C_i est l'ensemble des termes de taille i pouvant devenir des patterns, qui couvrent à la fois des observations positives et négatives.
$C_0 = \{\emptyset\}$
pour $i = 1$ *à* D **faire**
  $P_i = P_{i-1}$
  $C_i = \emptyset$
  pour tous les $t \in C_{i-1}$ **faire**
    p=index maximal des variables dans t
    pour $s = p + 1$ *à* n **faire**
      pour tous les $l \in \{x_s, \bar{x}_s\}$ **faire**
        $T = t \wedge l$
        pour $j = 1$ *à* $i - 1$ **faire**
          $t' = T$ avec la j-ème variable retirée
          si $t' \notin C_{i-1}$ **alors**
            aller à $\Diamond$
        si T *couvre un vecteur positif mais pas de vecteur négatif* **alors**
          $P_i = P_i \cup \{T\}$
        si T *couvre à la fois un vecteur positif et négatif* **alors**
          $C_i = C_i \cup \{T\}$
        $\Diamond$
retourner P_D

Algorithme 1 : Calcul des prime patterns d'après Boros et al. [2000]

À partir d'un intervalle $[\alpha, \beta]$ un terme t peut être défini de la façon suivante :

$$t = \Big(\bigwedge_{i \text{ t.q. } \alpha_i=\beta_i=1} l_i\Big) \wedge \Big(\bigwedge_{j \text{ t.q. } \alpha_j=\beta_j=0} \bar{l}_j\Big)$$

où l_i est la i-ième variable du jeu de données.

De même, à partir d'un terme t, nous pouvons reconstruire son intervalle correspondant $[\alpha, \beta]$ avec $\forall i \in \{1, \ldots, n\}$, $\alpha_i = 1$ si $l_i \in Lit(t)$ et 0 sinon, et $\beta_i = 0$ si $\bar{l}_i \in Lit(t)$, 1 sinon.

Il y a donc équivalence entre le terme et son intervalle, et nous pouvons remarquer que la notion de couverture de l'intervalle est identique à la notion de couverture du terme. Ainsi, comme le terme, un intervalle définira un pattern s'il couvre au moins une observation positive et aucune observation négative.

Exemple 6. *Reprenons notre exemple avec* $\alpha = (0, 1, 1)$ *et* $\beta = (1, 0, 1)$ *et notons les variables* $l_1 = a, l_2 = b, l_3 = c$. *Le terme généré par l'intervalle* $[\alpha, \beta]$ *est donc* $t = c$ *et couvre bien l'observation* $\gamma = (1, 1, 1)$ *et pas* $\gamma' = (1, 1, 0)$.

Un *consensus* entre deux patterns générés par les intervalles $[\alpha, \beta]$ et $[\alpha', \beta']$ est un intervalle

$[\alpha'', \beta'']$ avec $\forall i \in \{1, \ldots, n\}$, $\alpha''_i = min(\alpha_i, \alpha'_i)$ et $\beta''_i = max(\beta_i, \beta'_i)$. Le consensus étant un intervalle, il est donc associé à un terme qui peut être un pattern. Un consensus entre 2 patterns p_1 et p_2 peut donc être défini comme un terme t tel que $Lit(t) = Lit(p_1) \cap Lit(p_2)$. Notons qu'avec un consensus entre deux patterns p_1 et p_2, où p_2 est déjà un consensus entre p_1 et un troisième pattern, alors ce nouveau consensus sera identique à p_2. Notons également qu'un pattern p_1 réalisé par un consensus entre deux patterns p_2 et p_3 sera nécessairement de degré inférieur à p_2 et p_3, ou égal si $p_1 = p_2$ ou $p_1 = p_3$.

L'algorithme `SPIC` va partir d'un ensemble de spanned patterns facilement générable, de degré maximal n et de couverture minimale (de taille 1). Puis, chaque consensus réalisé par un couple de spanned patterns (p_1, p_2) proposera un terme couvrant l'ensemble $Cov(p_1) \cup Cov(p_2)$ en retirant le moins de littéraux possible. En considérant l'ensemble des consensus possibles, nous aurons déterminé tous les spanned patterns couvrant chaque combinaison d'observations possible. Cet ensemble de spanned patterns de départ sera tout simplement les patterns de degré n couvrant chaque observation positive une à une.

Nous présentons l'algorithme `SPIC` (algorithme 2) ci-dessous puis le décrirons en suivant un exemple.

Données : C_0 L'ensemble des spanned patterns couvrant chaque observation positive une à une.
Résultat : C l'ensemble des spanned patterns.
$i = 0$, $C = C_0$, $W_0 = C_0$
Faire
  $W_{i+1} = \emptyset$
  pour tous les $p_1 \in C_0$ *et* $p_2 \in W_i$ **faire**
    p'= le consensus de p_1 et p_2
    si p' *est un pattern et* $p' \notin C$ **alors**
      $C = C \cup \{p'\}$
      $W_{i+1} = W_{i+1} \cup \{p'\}$
  i=i+1
Tant que $W_i = \emptyset$;
retourner C

Algorithme 2 : Calcul des spanned patterns : `SPIC` d'après Alexe and Hammer [2006]

Exemple 7. *Illustrons le déroulement de l'algorithme 2 au travers l'exemple suivant :*

Observations	*Groupes*	*Variables*		
		a	*b*	*c*
1	*P*	*1*	*0*	*0*
2		*0*	*1*	*0*
3		*0*	*1*	*1*
4	*N*	*1*	*1*	*1*
5		*1*	*1*	*0*

C_0 est donc composé de 3 spanned patterns de degré $n = 3$ couvrant respectivement les observations 1, 2 et 3. Donc $C_0 = \{p_1, p_2, p_3\} = \{a \wedge \bar{b} \wedge \bar{c},\ \bar{a} \wedge b \wedge \bar{c},\ \bar{a} \wedge b \wedge c\}$.
Nous pouvons transformer les patterns en intervalles correspondant :

- $p_1 = [(1\ 0\ 0), (1\ 0\ 0)]$,
- $p_2 = [(0\ 1\ 0), (0\ 1\ 0)]$,
- $p_3 = [(0\ 1\ 1), (0\ 1\ 1)]$.

On peut remarquer que pour chaque intervalle $[\alpha, \beta]$, on a ici $\alpha = \beta$ car les patterns sont de degré n.

$C = C_0 = \{p_1, p_2, p_3\}$ et idem pour $W_0 = \{p_1, p_2, p_3\}$. Puis i=0.

Nous entrons dans la boucle et commençons par initialiser $W_1 = \emptyset$. Il y a 3 éléments dans C_0 et 3 dans W_0 mais le consensus de deux patterns identiques donnent toujours ce pattern et le consensus de deux patterns p et p' est identique au consensus de p' et p. Il n'y a donc que 3 consensus qui nous intéressent à cet étape :

- *Le consensus $p_{1,2}$ entre p_1 et p_2 (ou entre p_2 et p_1),*
- *Le consensus $p_{1,3}$ entre p_1 et p_3 (ou entre p_3 et p_1),*
- *Le consensus $p_{2,3}$ entre p_2 et p_3 (ou entre p_3 et p_2).*

On a donc :

- $p_{1,2} = [(0\ 0\ 0), (1\ 1\ 0)]$ *correspondant à $\bar{c}$ mais couvre l'observation 5. Ce n'est donc pas un pattern.*
- $p_{1,3} = [(0\ 0\ 0), (1\ 1\ 1)]$, *couvre toutes les observations ("terme vide"), ce n'est pas un pattern.*
- $p_{2,3} = [(0\ 1\ 0), (0\ 1\ 1)]$ *correspond à $\bar{a} \wedge b$ et ne couvre aucune observation négative. C'est un pattern qui, de plus, n'appartient pas à C.*

Nous avons déterminé un pattern, donc $C = C \cup \{p_{2,3}\}$ et $W_1 = W_1 \cup \{p_{2,3}\}$.
On incrémente i à 1.

On a bien $W_1 = \{p_{2,3}\}$ non vide, on reprend la boucle.

$W_2 = \emptyset$. Nous avons 1 élément dans W_1 et toujours 3 dans C_0. Il y a donc 3 combinaisons réalisables. Toutefois, $p_{2,3}$ est le consensus entre p_2 et p_3, donc le consensus entre $p_{2,3}$ et p_2 sera égal à $p_{2,3}$. Idem pour le consensus entre $p_{2,3}$ et p_3 et donc, ces deux consensus seront déjà contenu dans C puisque $p_{2,3} \in C$. Il n'y a donc que le consensus $p_{1,2,3}$ entre p_1 et $p_{2,3}$ qui nous intéresse.

$P_{1,2,3} = [(0\ 0\ 0), (1\ 1\ 1)]$ *est identique à* $p_{1,3}$ *et couvre toutes les observations, dont les négatives. Ce n'est pas un pattern.*

$i = 2$ *et cette fois-ci,* $W_2 = \emptyset$*. On sort de la boucle et on retourne l'ensemble* C *correspondant à l'ensemble des spanned patterns. Nous avons donc 4 spanned patterns :*

- $p_1 = a \wedge \bar{b} \wedge \bar{c}$,
- $p_2 = \bar{a} \wedge b \wedge \bar{c}$,
- $p_3 = \bar{a} \wedge b \wedge c$,
- $p_{2,3} = \bar{a} \wedge b$.

2.2.3.4 Transformation de pattern en pattern optimal

Nous présenterons ici 3 algorithmes issus de Hammer et al. [2004] transformant respectivement un pattern en prime pattern, spanned pattern et strong pattern.

Concernant la transformation d'un pattern en prime pattern, l'idée est tout simplement de retirer le plus de littéraux possible jusqu'à ce qu'on ne le puisse plus.

Données : p un pattern constitué de t litéraux $l_1, \ldots, l_t$.
Résultat : p' un prime pattern.
$p' = p$
pour i *de* 1 *à* t **faire**
  //On retire le i-ème littéral de p'
  $C = p' \backslash \{l_i\}$
  si C *est un pattern* **alors**
    $p' = C$
retourner p' ;

Algorithme 3 : Transformation d'un pattern en prime pattern

Notons que cet algorithme va diminuer le nombre de littéraux du pattern initial, mais peut également augmenter la couverture de ce dernier.

Concernant l'algorithme de transformation d'un pattern en spanned pattern, l'idée est de récupérer l'ensemble des variables sur lesquelles les observations couvertes par p sont identiques. Pour cela, on définit la notion d'enveloppe convexe $[S]$ d'un sous ensemble d'observations S comme étant le plus petit sous-cube contenant S. $[S]$ est donc un terme constitué de l'ensemble des variables de valeurs identiques chez toutes les observations de S.
Par exemple, considérons les observations suivantes :

Observations	Groupes	Variables		
		a	b	c
1	P	0	1	0
2		0	1	1

Alors, soit S l'ensemble de ces 2 observations, nous avons $[S] = \bar{a} \wedge b$.

Ainsi, l'algorithme de transformation d'un pattern p en spanned pattern va déterminer l'enveloppe convexe de la couverture de p.

Données : p un pattern.
Résultat : p' un spanned pattern.
$S = Cov(p)$
$p' = [S]$
retourner p' ;

Algorithme 4 : Transformation d'un pattern en spanned pattern

Notons que si nous avons augmenté le nombre de littéraux, la couverture n'a pas changé.

Concernant l'algorithme de transformation en strong pattern, l'idée est à peu près similaire sauf que l'on cherche à agrandir la couverture lors du calcul de l'enveloppe convexe :

Données : p un pattern, P l'ensemble des observations positives
Résultat : p' un strong pattern.
$p' = [Cov(p)]$
$K = P \backslash Cov(p)$
Faire
  Soit $X \in K$
  $C = [Cov(p') \cup X]$
  $K = K \backslash \{X\}$
  si C *est un pattern* **alors**
    $p' = C$
    $K = K \backslash Cov(p')$
Tant que $K \neq \emptyset$;
retourner p' ;

Algorithme 5 : Transformation d'un pattern en strong pattern

2.2.4 Applications

De nombreuses applications de l'analyse logique de données sont présentées dans plusieurs publications. Ces applications peuvent aborder de nombreux domaines comme la médecine ou l'économie.

Dans Reddy et al. [2008] une étude sur les AVC est présentée. L'objectif est d'utiliser l'analyse logique de données afin d'obtenir des biomarqueurs pour dianostiquer un AVC ischémique aigu en utilisant un échantillon de sang. Le modèle de classification obtenu à une précision de 75% avec seulement 3 biomarqueurs.

De la même manière, mais dans un autre contexte, Mortada et al. [2012] cherche à détecter les pièces défectueuses dans l'industrie aéronautique au moyen de prime patterns. Cela permet de considérablement réduire le coût et d'augmenter la sécurité et la performance de l'industrie.

Toujours dans le domaine de l'industrie aéronautique, Dupuis et al. [2012] utilise l'analyse logique de données pour estimer le taux de clients d'Air Canada qui se présenteront lors d'un vol. Cela permet de surréserver un vol et d'éviter un manque à gagner financier lorsque les clients ne se présentent pas. À partir de la base de données d'Air Canada, 30 variables sont extraites, dont seulement 16 apparaissent dans les patterns. Une fois ces patterns déterminés, un système de score est appliqué sur chaque observation (passager) pour prédire s'il se présentera ou non, en fonction du nombre de patterns associés au groupe positif et au groupe négatif qu'il respecte.

Dans le domaine de la finance, Hammer et al. [2012] présente une application de l'analyse logique de données pour évaluer les risques de crédit. En effet, les agences de notation comme Standard&Poor's ou Fitch ont un impact important dans le marché financier, et les notations qu'elles donnent peuvent influencer le monde économique. Toutefois, depuis 2008, leurs travaux font l'objet de critiques sérieuses dû au fait de leur incapacité d'avoir pu prévoir les plus grands effondrements financiers de ces dernières années, que certains attribuent à la présence de conflits d'intérêts induits par un modèle émetteur-payeur. Un rapport de la Securities and Exchange Commission (2003)[3] indique que 90% des revenus de Moody's et de Fitch proviennent de commissions payées par les émetteurs. Ceci croisé avec les accords de Bâle III (des accords de réglementation bancaire pour renforcer le système financier, dont les 3èmes ont été mis en place en 2012 à la suite de la crise des subprimes) pousse les banques à développer des modèles internes d'évaluation de risque plus précis. L'article montre que les patterns distinguant les banques bien et mal notées permettent de proposer un modèle supérieur aux méthodes statistiques classiques et des résultats très proches de ceux obtenus par une méthode de regression logistique ordonnée. Toutefois, la méthode par analyse logique de données propose des notes plus proches des notes de Fitch en n'utilisant qu'un sous-ensemble des variables à disposition.

L'analyse logique de données est donc une alternative aux méthodes de classification statistiques classiques et est aujourd'hui utilisée dans de nombreux domaines.

2.3 Caractérisation multiple de données

La caractérisation multiple de données est une extension de l'analyse logique de données où l'on considère plusieurs groupes de données. L'objectif est de calculer un ensemble de caractéristiques, nommé solution, suffisant pour discriminer chaque groupe de données des autres, de façon simultanée. Dans cette section, nous présenterons la caractérisation multiple et ses problématiques avant de présenter les définitions et notations liées au domaine. Puis nous présenterons la problématique de génération de formules caractéristiques de groupe avant de présenter les différentes méthodes existantes pour générer des solutions.

[3] https ://www.sec.gov/news/studies/credratingreport0103.pdf

2.3.1 Présentation

Tout comme l'analyse logique de données, la caractérisation multiple de données va chercher à discriminer les observations de groupes différents. Proposé dans Chhel et al. [2012] en réponse à une problématique de biologie végétale, il s'agit d'une extension à l'analyse logique de données dont ces dernières sont réparties en un nombre de groupes pouvant être important. L'objectif de ce problème est de déterminer un ensemble de variables (que nous nommerons attributs) discriminant tous les groupes simultanément, c'est à dire que pour chaque couple d'observations issus de groupes différents, ces observations doivent être distinctes. Il s'agit donc là d'une approche similaire à la détermination des support sets (vu dans la sous-section 2.2.2) mais où le nombre de groupes à considérer peut être plus important que 2.

Illustrons avec un nouvel exemple :

Exemple 8. *L'image d'un homme politique peut influencer l'opinion*[4]*, nous allons donc ici présenter un candidat à une élection qui cherche sur quel point il doit s'appliquer en se basant sur un discours présenté lors de six meetings dans plusieurs villes. Pour cela, il évalue plusieurs variables (ou attributs) :*

- *Son rythme de parole (on codera 0 pour lent et 1 pour soutenu)*
- *La couleur de son costume (on codera 0 pour sombre et 1 pour coloré),*
- *Sa coupe de cheveux (on codera 0 pour long et 1 pour court),*
- *Lunettes ou lentilles (on codera 0 pour lentilles et 1 pour lunettes),*
- *Sa montre (on codera 0 pour discrète et 1 pour visible).*

Puis, il évaluera le succès chacun des meetings par rapport aux dons obtenus dans chaque ville, qu'il séparera en 4 groupes :

- *Peu satisfaisant si les dons sont inférieurs à 5k€,*
- *Passable s'ils sont compris entre 5 et 10k€,*
- *Bon s'ils sont entre 10 et 20k€,*
- *Excellent s'ils sont au dessus de 20k€.*

Au final, il obtient le tableau de données suivant :

[4] http ://archives.lesechos.fr/archives/cercle/2012/03/14/cercle_44542.htm

Attributs / *Succès*	*Rythme*	*Costume*	*Cheveux*	*Lunettes*	*Montre*
Peu satisfaisant	*0*	*1*	*1*	*0*	*0*
Passable	*1*	*1*	*1*	*1*	*1*
	0	*1*	*0*	*1*	*1*
Bon	*0*	*0*	*0*	*1*	*0*
Excellent	*1*	*0*	*1*	*0*	*0*
	1	*1*	*0*	*0*	*1*

Nous pouvons remarquer qu'avoir un rythme de parole soutenu et un costume sombre est un "pattern du 4ème groupe" (au sens où il est valide chez une observation du 4ème groupe et chez aucune observation des 3 autres groupes), mais il n'explique pas tout le groupe. De plus le rythme et la couleur du costume ne sont pas deux attributs suffisants pour distinguer tous les groupes, puisqu'un rythme soutenu et un costume clair a provoqué une fois un succès passable et une fois un succès excellent.

En revanche, les attributs costume, lunettes et montre permettent à eux trois de discriminer tous les groupes. En effet à chaque meeting ayant un succès différent, la combinaison de ces 3 attributs est différente. Pour les 2 meetings du 4ème groupe, la combinaison est différente, mais chacun se différencie de tous les autres meetings sur ces 3 attributs, ils sont donc suffisant pour discriminer chaque groupe.

Au final, notre candidat pourrait penser qu'il est important pour avoir du succès de bien assortir son costume à ses lunettes et sa montre, indépendamment du rythme de parole et de la coupe de cheveux.

Il est important de noter que la caractérisation multiple de données est différente de la sélection de variables/attributs (que nous présenterons dans la section 2.5.2) car il ne s'agit pas de déterminer les attributs les plus informatifs statistiquement mais bien d'avoir une combinaison d'attributs discriminant les groupes de manière exacte. Dans l'exemple précédent nous avons vu qu'une solution au problème de caractérisation multiple serait de retenir les attributs costume, lunettes et montre. Toutefois, l'attribut rythme est plutôt bien corrélé au groupe puisqu'à l'exception d'un meeting, il évolue de la même manière que les groupes au sens où le rythme est soutenu presque uniquement que dans les meetings d'un seul groupe.

À partir de là, deux aspects d'optimisation peuvent être intéressants à étudier. En premier lieu, la minimisation du nombre d'attributs retenus. Il est en effet intéressant pour notre candidat de travailler sur le moins d'attributs possibles, surtout si cela représente un coût important ou pose des contraintes difficile à résoudre. En second lieu, la maximisation de la ressemblance des observations d'un même groupe par rapport aux attributs. En effet dans notre exemple, les deux meetings au succès excellent proposent des combinaisons différentes sur les trois variables

retenues. Or la justification de l'appartenance à ce groupe serait plus forte si tous les meetings excellents s'étaient fait avec le même costume, sans lunettes et avec la même montre.

2.3.2 Terminologie et notations

Nous utiliserons ici une approche plus matricielle pour définir le problème de caractérisation multiple.

Ainsi, les observations étudiées, appartenant à un groupe Ω, seront toujours exprimées sur des variables booléennes, que nous avons nommées attributs, et appartenant à l'ensemble $\mathcal{A}$. Ces observations sont réparties en plusieurs groupes appartenant à l'ensemble des groupes $\mathcal{G}$. Toutes ces données seront représentées par une matrice D.

Commençons par définir une instance du problème de caractérisation multiple (PCM) :

Définition 13. *Une instance du PCM est un quadruplet $(\Omega, \mathcal{A}, D, G)$ défini par un ensemble d'observations Ω dont les éléments sont exprimés sur un ensemble d'attributs A, et sont représentés par une matrice de données booléennes $D_{|\Omega|\times|\mathcal{A}|}$ et une fonction $G : \Omega \to \mathcal{G}$, telle que $G(o)$ est le groupe auquel appartient l'observation $o \in \Omega$.*

La matrice de données est définie comme suit :

- La valeur $D[o, a]$ représente la présence/absence de l'attribut a pour l'observation o.
- Une ligne $D[o, .]$ représente le vecteur booléen de présence/absence des attributs pour l'observation o.
- Une colonne $D[., a]$ représente le vecteur booléen de présence/absence de l'attribut a pour chaque observation.

Ainsi, deux observations $o, o' \in \Omega$ peuvent être représentées par le même vecteur booléen (donc $D[o, .] = D[o', .]$) et pourtant être considérées comme deux observations distinctes.

Par la suite, nous ne considérerons que les instances réalisables, c'est à dire pour lesquelles il existe une solution.

Propriété 3. *Une instance $(\Omega, \mathcal{A}, D, G)$ du PCM est réalisable si et seulement si :*

$$\nexists (o, o') \in \Omega^2 \text{ tel que } D[o, .] = D[o', .] \text{ et } G(o) \neq G(o')$$

Définition 14. *Soit $A \subset \mathcal{A}$, D^A est la matrice de données réduite au sous-ensemble d'attributs A.*

Définition 15. *Pour une instance $(\Omega, \mathcal{A}, D, G)$, la matrice $D_{\succ}$ est la matrice D où les observations de même groupe et identiques ont été supprimées.*

Prenons un exemple pour illustrer ces deux dernières définitions :

Exemple 9. *Reprenons l'exemple 8, et considérons le sous-ensemble d'attributs* $A = \{$*Costume,Lunettes,Montre*$\}$.
La matrice de données D *peut être représentée comme suit :*

$D =$

Groupes \ *Attributs*	*Rythme*	*Costume*	*Cheveux*	*Lunettes*	*Montre*
Peu satisfaisant	*0*	*1*	*1*	*0*	*0*
Passable	*1*	*1*	*1*	*1*	*1*
	0	*1*	*0*	*1*	*1*
Bon	*0*	*0*	*0*	*1*	*0*
Excellent	*1*	*0*	*1*	*0*	*0*
	1	*1*	*0*	*0*	*1*

Ainsi, la matrice réduite D^A *ne comportera que les données sur les attributs du sous-ensemble* A *:*

$D^A =$

Groupes \ *Attributs*	*Costume*	*Lunettes*	*Montre*
Peu satisfaisant	*1*	*0*	*0*
Passable	*1*	*1*	*1*
	1	*1*	*1*
Bon	*0*	*1*	*0*
Excellent	*0*	*0*	*0*
	1	*0*	*1*

Nous pouvons voir que deux observations du même groupe passable sont identiques sur A*. La matrice* $D^A_{\subset}$ *n'en conservera qu'une (même s'il s'agit de deux observations différentes) :*

$D^A_{\subset} =$

Groupes \ *Attributs*	*Costume*	*Lunettes*	*Montre*
Peu satisfaisant	*1*	*0*	*0*
Passable	*1*	*1*	*1*
Bon	*0*	*1*	*0*
Excellent	*0*	*0*	*0*
	1	*0*	*1*

Nous allons maintenant introduire la notion de solution du PCM :

Définition 16. *Soit une instance* $(\Omega, \mathcal{A}, D, G)$*, un sous-ensemble d'attributs* $S \subseteq \mathcal{A}$ *est une solution si et seulement si* $\forall (o, o') \in \Omega^2$*,* $G(o) \neq G(o') \Rightarrow D^S[o, .] \neq D^S[o', .]$.

Autrement dit, S est une solution si deux observations, issus de deux groupes différents, sont différents sur au moins un attribut $a \in S$.

Tout comme les patterns, les solutions du PCM peuvent être très nombreuses. En particulier, pour une solution S donnée, ajouter un attribut à cette solution donne une nouvelle solution S'. Nous dirons que S' est dominée par S.

Définition 17. *Une solution S est non-dominée si et seulement si $\forall s \in S$, $\exists (o, o') \in \Omega^2$ tel que $G(o) \neq G(o')$ et $D^{S\setminus\{s\}}[o, .] = D^{S\setminus\{s\}}[o', .]$ (i.e. $S\setminus\{s\}$ n'est pas une solution).*

La recherche de solutions non-dominées permet ainsi d'éviter de rechercher de l'information redondante tout en limitant le nombre de solutions. Nous pouvons également affiner encore plus la recherche en se concentrant uniquement sur les solutions comptant le moins d'attributs. En représentant le nombre d'éléments d'un ensemble E par la fonction $|E|$, nous définissons ci-dessous la notion de solution minimale.

Définition 18. *Une solution S est minimale si et seulement si $\nexists S'$ avec $|S'| < |S|$ tel que S' est une solution.*

Notons qu'une solution minimale est toujours non-dominée.

À partir de ces définitions, pour une instance $\mathcal{I} = (\Omega, \mathcal{A}, D, G)$ du PCM, nous pouvons identifier plusieurs catégories de problèmes :

- *Sol-PCM* : calcul d'une solution pour $\mathcal{I}$,
- *Min-PCM* : calcul d'une solution minimale pour $\mathcal{I}$,
- *1Min-PCM* : calcul d'une solution minimale pour un groupe contre tous les autres (support set de taille minimale d'un groupe),
- *NonDom-PCM* : calcul de l'ensemble des solutions non-dominées pour $\mathcal{I}$,
- *MinAll-PCM* : calcul de l'ensemble des solutions minimales pour $\mathcal{I}$.

2.3.3 Formulation du PCM

Dans Chhel et al. [2012], l'objectif n'est pas d'obtenir l'ensemble des attributs discriminant chaque groupe, mais une formule logique associée à chacun des groupes, de telle sorte que chaque observation respecte la formule de son groupe et jamais celle d'un autre.

Cette formule logique s'écrit comme la présence/absence de certains attributs liés par les opérateurs logiques "ET" ($\wedge$) et "OU" ($\vee$).

Exemple 10. *Nous avons vu dans l'exemple précédent la solution* $S = \{$*Costume,Lunettes,Montre*$\}$ *définie par la matrice suivante :*

$D^S =$

Attributs / *Groupes*	*Costume (c)*	*Lunettes (l)*	*Montre (m)*
Peu satisfaisant (1)	*1*	*0*	*0*
Passable (2)	*1*	*1*	*1*
	1	*1*	*1*
Bon (3)	*0*	*1*	*0*
Excellent (4)	*0*	*0*	*0*
	1	*0*	*1*

où on représente chaque attribut par la lettre entre parenthèse et chaque groupe par le chiffre entre parenthèse.

Soit l'ensemble de formules $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$*, avec :*

- $\phi_1 = c \wedge \bar{l} \wedge \bar{m}$,
- $\phi_2 = c \wedge l \wedge m$,
- $\phi_3 = \bar{c} \wedge l \wedge \bar{m}$,
- $\phi_4 = \bar{l} \wedge ((\bar{c} \wedge \bar{m}) \vee (c \wedge m))$.

Chaque observation de chaque groupe respecte bien la formule logique associée au groupe. Par exemple, pour que le meeting soit excellent (groupe 4), il ne faut pas de lunettes et soit un costume sombre avec une montre discrète, soit un costume clair avec une montre visible, ce qui est bien le cas pour chacun des meetings classés dans ce groupe. Φ *est donc la formulation logique de l'instance* $\{\Omega, \mathcal{A}, D, G\}$ *du PCM.*

Nous voyons par cet exemple que le lien entre la solution d'un PCM et sa formulation logique est très étroit. En effet, l'ensemble des variables d'une formulation logique sera une solution et à partir d'une solution il est facile de déterminer une formulation logique par la méthode suivante :

- Partir de l'ensemble solution S de l'instance $\{\Omega, \mathcal{A}, D, G\}$,
- Pour chaque observation o du groupe $G(o)$, construire le pattern

$$p_o = \Big(\bigwedge_{a \in S \text{ t.q. } D[o,a]=1} x_a\Big) \wedge \Big(\bigwedge_{a \in S \text{ t.q. } D[o,a]=0} \bar{x}_a\Big)$$

- Construire la formule logique du groupe g tel que :

$$\phi_g = \bigvee_{o \text{ t.q. } G(o)=g} p_o$$

- Construire une formulation logique de l'instance à partir des formules de chaque groupe $g \in \{1, ..., gr\}$:

$$\Phi = \bigcup_{g=1}^{gr} \{\phi_g\}$$

Notons que cette formulation est sous la forme dite "DNF" (forme normale disjonctive), c'est à dire qu'elle s'écrit comme une disjonction de conjonctions : $\phi_g = \bigvee_{o \in g} \bigwedge_{a \in S} l_a$.

Nous verrons ainsi les méthodes de résolution par calcul de solutions dans la sous-section 2.3.4. Nous allons pour la suite de cette sous-section regarder comment déterminer directement une formulation.

Dans Chhel et al. [2012] est présentée une méthode par génération de formules. En considérant chaque groupe contre tous les autres, et en essayant toutes les formules, on finira par déterminer une formule pour chaque groupe. Bien évidemment, le nombre de formules possibles est bien trop important pour que chacune soit testée, et il est difficile d'éviter la redondance. Dans Quinlan [1990] est proposé un algorithme permettant de générer une formule en un temps polynomial en essayant de séparer les exemples positifs et négatifs. Toutefois, avec un tel algorithme, les solutions associées aux formules trouvées sont régulièrement d'une taille proche de $|\mathcal{A}|$. C'est pourquoi dans Chhel et al. [2012], il est proposé d'élargir le champ des formules générées en implémentant un générateur se basant sur la construction récursive et exhaustive des formules de forme normale conjonctive (CNF), c'est à dire des conjonctions de disjonctions. Grâce à un ordonnancement des clauses (i.e. des disjonctions de littéraux), le nombre de redondances est réduit de manière conséquente. Ainsi le solveur *Exact-Form-Car* a été développé pour construire l'ensemble des formules dont les solutions associées sont de taille k avant de passer à celles de taille $k + 1$.

Toutefois plusieurs problématiques majeures apparaissent avec cette méthode. Non seulement le nombre de formules reste très important (et augmente de façon exponentiel avec le nombre d'attributs), mais de plus, il est difficile d'éviter les formules redondantes en un temps et une complexité raisonnable (la génération de CNF sans redondance est un problème coNP-complet). Mais le problème majeur est qu'il est difficile de raisonner sur l'instance complète en se basant sur les groupes un à un. En effet, trouver une formule pour chaque groupe tout en minimisant le nombre d'attributs utilisé pour l'ensemble de ces formules est très coûteux. Il est donc très difficile de déterminer une formulation logique dont la solution associée soit minimale.

2.3.4 Calcul de solutions

Dans cette sous-section, nous présentons différentes méthodes de calcul de solutions minimales du PCM. En premier lieu, nous présentons une méthode proche de *Exact-Form-Car* décrit pré-

cédemment mais adaptée à la détection d'attributs discriminants plutôt qu'à la détection de formules. Ensuite nous présentons l'approche par programmation linéaire permettant de calculer l'ensemble des solutions pour une instance. Enfin, nous présentons des algorithmes gloutons et de recherche locale permettant d'obtenir des solutions de taille proche de la taille minimale.

2.3.4.1 *Exact-Proj-Car*

Toujours dans Chhel et al. [2012] le solveur *Exact-Proj-Car* est présenté. Son fonctionnement est assez similaire à celui de *Exact-Form-Car*, mais va se baser sur la construction de solutions plutôt que de formules.

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Résultat : Une solution S
// $\mathcal{G}$ est l'ensemble des groupes retourné par la fonction G.
pour k *de* 1 *à* $|\mathcal{A}|$ **faire**
  pour tous les *sous-ensembles* $S \in \mathcal{A}$ *de* k *attributs* **faire**
    $consistent = true$
    pour tous les $g \in \mathcal{G}$ **faire**
      G' : une fonction telle que $\forall o \in \Omega$, $G'(0) = P$ si $G(o) = g$, $G'(o) = N$ sinon.
      si (Ω, S, D^S, G') *est une instance non réalisable* **alors**
        $consistent = false$
        Break
    si *consistent=true* **alors**
      retourner S ;

Algorithme 6 : *Exact-Proj-Car*

L'algorithme va donc tester toutes les combinaisons de k attributs avant de passer à celles de taille $k+1$ si elle n'en trouve pas. Ainsi, la première solution trouvée est forcément de taille minimale.

Une autre approche est de partir d'une taille $k = |\mathcal{A}|$ et de diminuer k dès qu'une solution est trouvée. Lorsque toutes les combinaisons d'une taille k_{min} sont testées sans trouver de solution, nous pourront en déduire que la solution trouvée de taille $k_{min} + 1$ sera minimale. Cette approche se trouve être plus efficace pour deux raisons. En premier lieu, si l'algorithme doit s'arrêter (temps de calcul trop important, problème technique, ...), il peut retourner la dernière solution trouvée (non optimale nécessairement). De plus, le nombre de solutions étant proportionnellement plus important (par rapport aux nombre possible de combinaisons) à cause de la redondance des solutions dominées, trouver une solution de taille k pour passer à l'étape $k-1$ sera plus rapide (nous verrons la répartition des solutions par rapport à leurs tailles dans la section 4.5.1).

Toutefois, tester toutes les combinaisons reste une tâche difficile (il y a $\frac{|\mathcal{A}|!}{k!(|\mathcal{A}|-k)!}$ combinaisons de taille k différentes).

2.3.4.2 Programmation linéaire

Dans Boros et al. [2000] est présenté un modèle de programmation linéaire pour calculer les support sets (voir la sous-section 2.2.2), en transformant le problème de calcul de support sets par un problème de couverture par ensembles. Cette version a été étendue dans Chhel et al. [2013] afin de transformer le PCM en un problème de couverture minimale par ensembles.

Le problème de couverture minimale par ensembles est un des 21 problèmes NP-complet de Karp (Karp [1972]). Soit un univers $\mathscr{E}$ et une liste d'éléments e_i appartenant à un ou plusieurs ensembles $E_j \in \mathscr{E}$, le problème consiste à calculer le plus petit sous-ensemble $S \subseteq \mathscr{E}$ tel que :

$$\forall i, \exists j \text{ tel que } e_i \in E_j \in S$$

Autrement dit, il s'agit de trouver le plus petit nombre d'ensembles contenant tous les éléments de l'univers $\mathscr{E}$.

Dans le *Min-PCM*, l'objectif est de trouver le plus petit ensemble d'attributs tel que deux observations issus de deux groupes différents soient au moins différents sur un attribut. En modélisant chacune de ces comparaisons par un élément e_i du problème de couverture minimale par ensembles, et chaque attribut par un ensemble E_j contenant les couples d'observations qui se distinguent sur cet attribut, notre modèle se modélise bien par un problème de couverture minimale par ensemble.

En premier lieux, nous devons définir l'ensemble des éléments, c'est à dire l'ensemble des couples d'observations issus de deux groupes différents.

Définition 19. *Soit une instance $(\Omega, \mathcal{A}, D, G)$, l'ensemble Θ est l'ensemble des couples d'observations $(o, o') \in \Omega^2$ tel que $G(o) \neq G(o')$.*

Nous pouvons ainsi représenter pour chaque couple $(o, o') \in \Theta$ et pour chaque attribut $A \in \mathcal{A}$ si les deux observations sont distinctes ou non. Pour cela, on utilise une matrice de contraintes C.

Définition 20. *Soit une instance $(\Omega, \mathcal{A}, D, G)$ et Θ l'ensemble des couples d'observations issus de deux groupes différents, la matrice de contraintes est une matrice booléenne C de taille $|\Theta| \times |\mathcal{A}|$ construite comme suit :*

- *Chaque ligne est numérotée par un couple d'observations $(o, o') \in \Theta$,*
- *Chaque colonne représente un attribut,*
- *$C[(o, o'), a] = 1$ si $D[o, a] \neq D[o', a]$, $C[(o, o'), a]$ sinon,*
- *On note $C[(o, o'), .]$ le vecteur booléen représentant les différences entre les observations o et o' sur chaque attributs.*

Exemple 11. *Reprenons l'exemple 8 :*

Meeting	*Att* / *Gr*	*Rythme(r)*	*Costume(c)*	*Cheveux(ch)*	*Lunettes(l)*	*Montre(m)*
1	*Peu satisf.*	*0*	*1*	*1*	*0*	*0*
2	*Pass.*	*1*	*1*	*1*	*1*	*1*
3		*0*	*1*	*0*	*1*	*1*
4	*Bon*	*0*	*0*	*0*	*1*	*0*
5	*Excel.*	*1*	*0*	*1*	*0*	*0*
6		*1*	*1*	*0*	*0*	*1*

On constate que lors du meeting 1*, le candidat ne portait pas de montre, mais lors du meeting* 2*, si. Donc l'attribut* `montre` *différencie le meeting* 1 *du meeting* 2*. On a donc* $C[(1,2),m] = 1$*. Toutefois, il portait le même costume lors des deux meeting, ce qui donne* $C[(1,2),c] = 0$*. En remplissant de cette manière la matrice* C*, nous obtenons la matrice de contraintes suivante :*

Couples \ *Attributs*	*Rythme (r)*	*Costume (c)*	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(1,2)	*1*	*0*	*0*	*1*	*1*
(1,3)	*0*	*0*	*1*	*1*	*1*
(1,4)	*0*	*1*	*1*	*1*	*0*
(1,5)	*1*	*1*	*0*	*0*	*0*
(1,6)	*1*	*0*	*1*	*0*	*1*
(2,4)	*1*	*1*	*1*	*0*	*1*
(2,5)	*0*	*1*	*0*	*1*	*1*
(2,6)	*0*	*0*	*1*	*1*	*0*
(3,4)	*0*	*1*	*0*	*0*	*1*
(3,5)	*1*	*1*	*1*	*1*	*1*
(3,6)	*1*	*0*	*0*	*1*	*0*
(4,5)	*1*	*0*	*1*	*1*	*0*
(4,6)	*1*	*1*	*0*	*1*	*1*

Maintenant que chaque couple d'observations de Θ est représenté par C, nous devons, pour chaque ligne, c'est à dire chaque couple, avoir au moins un attribut différenciant le couple dans notre solution. Soit $Y \in \{0,1\}^{|\mathcal{A}|}, Y = [y_1, \cdots, y_{|\mathcal{A}|}]$ tel que $y_i = 1$ si le i-ième attribut fait partie de la solution, 0 sinon, on a la contrainte du programme linéaire suivante :

$$C.Y^t \geqslant \mathbb{1}^t$$

où $\mathbb{1}$ représente un vecteur composé de 1 et V^t est la transposée du vecteur V.

Comme nous cherchons à minimiser le nombre d'attributs dans notre solution, la fonction objectif sera simplement :

$$min : \sum_{i=1}^{|\mathcal{A}|} y_i$$

Au final nous obtenons le programme linéaire (Karp [1972]) suivant :

$$\begin{aligned} & min : \sum_{i=1}^{|\mathcal{A}|} y_i \\ & s.t. : \\ & C \,.\, Y^t \geqslant \mathbb{1}^t \\ & Y \in \{0,1\}^{|\mathcal{A}|}, Y = [y_1, ..., y_{|\mathcal{A}|}] \end{aligned}$$

Notons que par construction, nous pouvons obtenir l'ensemble des solutions minimales par programmation linéaire, et donc résoudre le *MinAll-PCM*.

Toutefois, comme pour tout problème de couverture minimale par ensembles, résoudre ce programme linéaire peut être coûteux en temps et en mémoire.

2.3.4.3 Algorithmes gloutons et recherche locale

Nous venons de voir que le *Min-PCM* peut se modéliser comme un problème de couverture par ensembles. Or il existe de nombreux algorithmes gloutons permettant de rapidement apporter une borne maximale au *Min-PCM* en déterminant une solution du *Sol-PCM*. Le principe de la plupart de ces algorithmes est simple, choisir un attribut, retirer tous les couples de Θ se distinguant par cet attribut, puis recommencer jusqu'à ce qu'il ne reste plus de couple à retirer. Obtenir une bonne solution consistera ainsi à choisir à chaque étape le bon attribut pour minimiser la taille de la solution finale.

Dans Chvatal [1979], il est proposé, à chaque étape, de choisir l'attribut couvrant le plus d'éléments non couverts par les attributs sélectionnés précédemment. Le calcul est rapide mais peut donner une solution de taille assez importante face à la solution minimale.

Dans Chhel et al. [2013] est proposée une méthode par recherche locale, qui va chercher à partir d'une solution (la solution de taille $|\mathcal{A}|$)) à retirer ou échanger ou rajouter un attribut (dans cet ordre) pour obtenir de meilleures solutions et répéter cette opération tant que ces trois actions (ou moins) améliorent un critère objectif basé sur la taille de la solution, mais également basé sur une notion de poids de variables afin de ne pas se retrouver bloqué à la première solution non-dominée trouvée. En lançant l'algorithme plusieurs fois, et en faisant évoluer les poids sur les attributs en fonction du nombre de fois qu'ils ont été sélectionnés précédemment, l'algorithme explore un large espace de recherche. Ainsi, plus on relance l'algorithme, plus la solution sera

d'une taille proche des solutions minimales, mais demandera plus de temps de calcul.

2.4 Problème de génération d'ensembles intersectants

Nous venons de voir que résoudre le PCM (i.e. calculer une solution du PCM) consiste à calculer un ensemble d'attribut discriminant chaque observation de groupe différent. Cet ensemble doit contenir au moins une variable qui différencie ces couples d'observations. Pour chaque couple (o, o') d'observations de groupes différents, nous pouvons considérer $E_{(o,o')}$ l'ensemble des attributs sur lesquels o et o' sont différents. Ainsi, notre ensemble d'attributs discriminants S (i.e. notre solution) doit avoir un attribut commun avec chaque ensemble $E_{(o,o')}$ $\forall (o, o') \in \Theta$ (où Θ est l'ensemble des couples d'observations issus de groupes différents).

Un tel ensemble possédant au moins un élément en commun avec chaque ensemble d'une famille est appelé *ensemble intersectant*. Après une présentation du problème, nous décrivons dans cette section les différentes approches existantes pour générer ces ensembles intersectants.

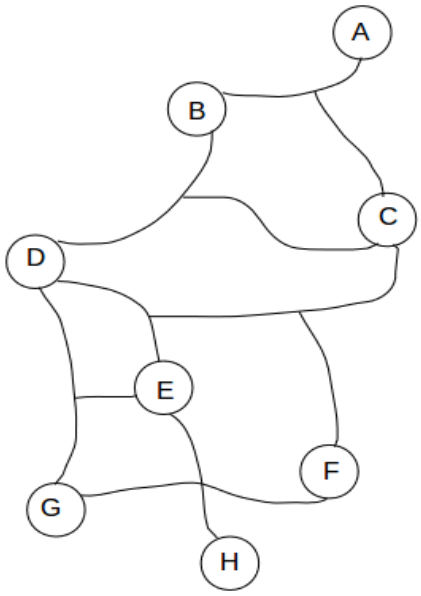
2.4.1 Présentation

Étant donnés un ensemble V d'éléments et une famille d'ensembles $\mathcal{F} = \{F_1, F_2, \cdots\}$ tels que $\forall i$, $F_i \subseteq V$, un ensemble intersectant H (hitting set en anglais) est un sous ensemble de V tel que $\forall i, H \cap F_i \neq \emptyset$. Autrement dit, l'ensemble H doit avoir au moins un élément commun avec chacun des ensembles de $\mathcal{F}$. Ce problème de génération d'ensembles intersectants est l'un des 21 problèmes NP-complet de Karp Karp [1972].

L'ensemble V est un ensemble intersectant, toutefois il n'est pas le plus intéressant à étudier. En particulier nous cherchons à construire généralement des ensembles minimaux. Un ensemble intersectant minimal H est un ensemble intersectant tel qu'il n'existe pas de sous-ensemble intersectant $H' \subset H$. Nous pouvons donc avoir des ensembles intersectants minimaux constitués de plus d'éléments que d'autres. Par la suite, nous appellerons les ensembles intersectants minimaux MHS (Minimal Hitting Set).

Exemple 12. *Exemple de couverture par sommets d'un hypergraphe.*
Un hypergraphe est un graphe dont les arcs peuvent avoir plus de deux extrémités (ces arcs sont nommés hyperarcs). Le problème de couverture par sommet consiste à trouver un sous-ensemble de sommets tel que tous les arcs soient adjacents à au moins un des sommets du sous-ensemble. Soit l'hypergraphe $H = (V, \mathcal{F})$ ci-dessous, avec :

- $V = \{A, B, C, D, E, F, G, H\}$
- $\mathcal{F} = \{\{A, B, C\}, \{B, C, D\}, \{C, D, E, F\}, \{D, E, G\}, \{E, F, G, H\}\}$



Trouver une couverture par sommet consiste à déterminer un sous-ensemble $S \subseteq V$ tel que chaque hyperarc $\mathcal{F}_i \in \mathcal{F}$ soit adjacent à au moins un sommet de S, i.e. $\forall \mathcal{F}_i \in \mathcal{F}$, $\exists s \in S$ tel que $s \in \mathcal{F}_i$.

Si on considère chaque sommet comme un élément et l'ensemble $\mathcal{F}$ comme une famille d'ensembles (où chaque hyperarc représente un ensemble de $\mathcal{F}$), le problème de couverture par sommets est équivalent à un problème de génération d'ensembles intersectants.

Remarque 2. *Le problème de génération d'ensembles intersectants est équivalent au problème de couverture par ensembles (voir la sous-section 2.3.4.2).*

2.4.2 L'algorithme de Berge

Pour résoudre ce problème de nombreux algorithmes ont été proposés. L'algorithme de Berge (Berge [1984]) est probablement le plus important puisque la plupart des autres méthodes sont des améliorations de ce dernier.

L'algorithme de Berge est un algorithme permettant de générer la totalité des MHS. Soit une famille de n ensembles $\mathcal{F} = \{F_1, F_2, \cdots, F_n\}$ telle que $\forall i$, $F_i \subseteq V$, l'algorithme consiste à trouver tous les MHS pour la "sous-famille" d'ensembles $\mathcal{F}' \subset \mathcal{F} = \{F_1, F_2, \cdots, F_j\}$, avec $j < n$ (nous nommerons MHS_j la totalité des MHS pour la sous-famille d'ensembles $\{F_1, F_2, \cdots, F_j\}$). Une fois l'ensemble MHS_j déterminé, pour générer l'ensemble MHS_{j+1}, il suffit de construire toutes les combinaisons entre les éléments de MHS_j et les éléments de F_{j+1}, puis, parmi les ensembles intersectants obtenus, de retirer ceux qui ne sont pas minimaux.

Soit T un ensemble d'ensembles intersectants, on notera $min(T)$ la fonction retirant de T les ensembles intersectants non minimaux : $min(T) = \{t \in T | \nexists t' \in T : t' \subset t\}$. Ainsi, le pseudo-code de l'algorithme de Berge est donné par l'algorithme 7.

Données : Une famille de n ensembles $\mathcal{F} = \{F_1, F_2, \cdots, F_n\}$
Résultat : L'ensemble T des MHS de $\mathcal{F}$
// Création de MHS_1
$T = \{\{e\} | e \in F_1\}$
// Incrémentation de T pour toutes les familles de F
pour s *de* 2 *à* n **faire**
  $T = \{t \cup \{e\} | t \in T, e \in F_s\}$
  $T = min(T)$
retourner T ;

Algorithme 7 : *Algorithme de Berge*

2.4.3 Autres algorithmes

Dans Gainer-Dewar and Vera-Licona [2017], les auteurs présentent de nombreux algorithmes existants capables de calculer l'ensemble des MHS. Les auteurs divisent les algorithmes en 4 catégories :

- Approches par itérations d'ensembles : Cette méthode consiste, tout comme l'algorithme de Berge, à construire l'ensemble MHS_j des j premiers ensembles de $\mathcal{F}$, puis de mettre à jour cet ensemble en étudiant le $(j+1)$-ième ensemble de $\mathcal{F}$. La très grande majorité des algorithmes de ce type sont des améliorations de l'algorithme de Berge (Reiter [1987],Wotawa [2001],Kavvadias and Stavropoulos [2005],Dong and Li [2005]).
  Dans Reiter [1987], il est proposé, pour chaque élément $t \in T$, de ne considérer que les ensembles F_i tels que $t \cap F_i = \emptyset$. Par conséquent, soit $F_1 \in \mathcal{F}$ et $F_2 \in \mathcal{F}$ avec $F_1 \subset F_2$, alors l'ensemble des MHS de $\mathcal{F}$ sera strictement identique à l'ensemble des MHS de $\mathcal{F} \backslash F_2$. Nous pouvons donc supprimer de notre étude la famille F_2.

- Approches par divisions et conquêtes : Cette approche va consister à diviser la famille d'ensembles $\mathcal{F}$ en plusieurs sous-familles qui vont être résolues séparément, de façon à ce qu'en réunissant l'ensemble des résultats, on obtienne l'ensemble des MHS. Par exemple, Lin and Jiang [2003] propose un algorithme se basant sur 3 étapes :
  1. S'il existe un élément $e \in V$ dans tous les ensembles de $\mathcal{F}$, alors l'élément est un MHS. Il ne reste qu'à calculer les MHS obtenus avec $V' = V \backslash \{e\}$.
  2. S'il existe un ensemble F_i constitué d'un unique élément $e \in V$, alors tous les MHS contiendront e. Il suffit de réduire notre problème à $V \backslash \{e\}$ et de rajouter e à chaque MHS déterminé.
  3. Sinon, il faut choisir un élément de $e \in V$ et diviser notre famille d'éléments $\mathcal{F}$ en deux sous-familles $\mathcal{F}_1 = \{F_i \backslash \{e\} | F_i \in \mathcal{F}, e \in F_i\}$ et $\mathcal{F}_2 = \{F_i | F_i \in \mathcal{F}, e \notin F_i\}$, de calculer l'ensemble des MHS sur chacune des deux sous-familles et d'ajouter e à chaque MHS déterminé avec $\mathcal{F}_2$.

Dans Knuth [2011] et Toda [2013], les auteurs utilisent des BDD (*binary decision diagrams*), ce qui permet de gagner un temps précieux dans la réalisation d'algorithmes de ce type. En particulier, l'algorithme présenté dans Toda [2013] permet de calculer les MHS très rapidement grâce à une variation des BDD, malgré un coût en mémoire important.

- Approches par accumulation : L'objectif ici est de calculer des sous-ensembles d'éléments $S \subset V$ qui sont des sous-ensembles d'au moins un MHS, puis rajouter des éléments jusqu'à obtenir un MHS. Nous décrirons les algorithmes `MMCS` et `RS` présentés dans Murakami and Uno [2014] dans la sous-section 2.4.4.

- Approches par couverture complète : Cette approche consiste à calculer plusieurs ensembles $S_i \subset V$ tels que $\forall F_j \in \mathcal{F},\ \exists S_i$ tel que $F_j \subseteq S_i$. Soit $\mathcal{F}_{S_i} = \{F_j \in \mathcal{F} | F_j \subseteq S_i\}$, calculer l'ensemble des MHS sur la famille d'ensembles $\mathcal{F}_{S_i}$ est donc plus efficace dès lors que $\mathcal{F}_{S_i} \subset \mathcal{F}$. Soit la famille d'ensembles de MHS sur chaque solution $\mathcal{H} = \{H_{S_i}\}$ où H_{S_i} est l'ensemble des MHS sur S_i, l'ensemble des MHS sur $\mathcal{H}$ est identique à l'ensemble des MHS sur $\mathcal{F}$. L'intérêt de cette approche est donc de déterminer la famille d'ensembles $S = \{S_i\}$ telle qu'il soit plus efficace de résoudre l'ensemble des sous-problèmes que le problème général. Notons que la résolution du problème sur un ensemble S_i est indépendante de la résolution sur un autre ensemble S_j, ce qui permet de résoudre le problème sur chaque ensemble en parallèle.

2.4.4 Les algorithmes MMCS et RS

Les expérimentations présentées dans Gainer-Dewar and Vera-Licona [2017] tendent à montrer que les algorithmes `MMCS` et `RS` (Murakami and Uno [2014]) sont les plus efficaces de la littérature.

L'algorithme de Berge est souvent peu efficace car le calcul de $min(T)$ demande la comparaison de tous les couples d'ensembles intersectants appartenant à T, ce qui est très couteux en terme de temps. Les algorithmes `MMCS` et `RS` sont basés sur une propriété qui permet de déterminer si un ensemble intersectant est minimal sans le comparer aux autres. Cette propriété utilise un concept particulier : l'ensemble des hyperarcs critiques.

Soit une famille d'ensembles F, un ensemble intersectant $S \subseteq V$ et un élément $v \in S$, l'ensemble des hyperarcs critiques de v est l'ensemble $crit(v, S) = \{F_i | F_i \in \mathcal{F}, S \cap F_i = \{v\}\}$. En d'autre terme, $crit(v, S)$ est l'ensemble des éléments de $\mathcal{F}$ qui ne possèdent que v en commun avec l'ensemble critique S.

Soit $uncov(S) = \{F_i | F_i \in \mathcal{F}, F_i \cap S = \emptyset\}$ l'ensemble des éléments de $\mathcal{F}$ n'ayant pas d'élément en commun avec S, i.e. S est un ensemble intersectant de $\mathcal{F} \backslash uncov(S)$, la propriété clé des algorithmes est la suivante :

Propriété 4. *$S \subseteq V$ est un MHS si et seulement si $uncov(S) = \emptyset$ (i.e. S est un ensemble intersectant) et $crit(v, S) \neq \emptyset \; \forall v \in S$.*

L'idée de ces algorithmes est de se souvenir de chaque ensemble $crit(v, S) \; \forall v \in S$ et de construire S de façon incrémentale, en suivant l'algorithme de Berge. Toutefois, on incrémente les ensembles S si et seulement si chaque élément $v \in S$ est tel que $crit(v, S) \neq \emptyset$. Ainsi, le calcul de $min(T)$ est inutile, puisqu'on ne génère que des MHS d'après la propriété 4.

Il est utile de remarquer également que les auteurs choisissent de ne pas explorer l'ensemble des éléments de V lors de la construction des MHS lorsqu'ils sont persuadés qu'étudier un de ces éléments ne permettra pas la génération d'un nouvel MHS. En particulier, s'ils lancent l'algorithme en partant d'un élément v, ils trouveront l'ensemble des MHS contenant v. Dans ce cas, pour calculer les MHS restant, ils peuvent ne lancer l'algorithme que sur $V \backslash \{v\}$. Dans cette optique, ils utilisent un sous-ensemble $\Upsilon \subseteq V$ représentant l'ensemble des éléments de V sur lesquels ils lancent leur génération.

L'algorithme `MMCS` est un algorithme récursif, démarrant avec un ensemble $S = \{\emptyset\}$, $crit(v, S) = \emptyset \; \forall v \in V$, $uncov(S) = \mathcal{F}$ et $\Upsilon = V$.

Le pseudo-code de la fonction récursif `MMCS` est donné par l'algorithme 8.

Pour une étape correspondant à l'étude de l'ensemble $F_i \in \mathcal{F}$, on commence à diminuer au maximum l'ensemble Υ, puis on lui ajoute progressivement chaque $v \in F_i$ si et seulement si $crit(s, S \cup \{v\}) \neq \emptyset \; \forall s \in S$, c'est à dire si $S \cup \{v\}$ n'est pas un ensemble intersectant non minimal. En effet, si $S \cup \{v\}$ est un ensemble intersectant non minimal, alors $\forall S'$ tel que $S \subset S'$, $S' \cup \{v\}$ sera nécessairement un ensemble intersectant non minimal puisqu'il contiendra $S \cup \{v\}$. Pour toutes les itérations à partir de S, il est donc inutile de considérer l'élément v. Nous dirons que v est un élément *transgressant*.

L'algorithme `RS`, quant à lui, utilise une autre approche basée sur la même propriété 4. L'ensemble MHS_j représentant les MHS obtenus par la famille d'ensembles $\mathcal{F}_j = \{F_1, \cdots, F_j\} \subseteq \mathcal{F}$, nous pouvons dire que chaque élément $S_j \in MHS_j$ respecte la propriété $crit(v, S_j) \; \forall v \in S$ sur la famille $\mathcal{F}_j$. On a donc : $\forall v \in S_j$, $min(i | F_i \in crit(v, S)) < j + 1$. De façon général, en étudiant à chaque fois l'ensemble $F_k \in \mathcal{F}$ tel que $k = min(t | F_t \in uncov(S))$, on aura $min(i | F_i \in crit(v, S)) < k \;\; \forall v \in S$.

Le pseudo-code de la fonction récursif `RS` est donné par l'algorithme 9.

2.5 Analyse de données et apprentissage

Nous diffusons tous les jours de nombreuses informations, en connectant notre téléphone, en remplissant des formulaires, en discutant sur des réseaux sociaux ou en passant notre carte de fidélité à la caisse d'un supermarché. Lorsque les jeux de données sont de tailles élevées, les interpréter devient difficile. En effet, expliquer le comportement d'un million d'individus, ou

Données : Un sous ensemble d'attributs S
Un ensemble Υ d'attributs
Un ensemble de MHS T
Une famille d'ensembles $uncov$
l'ensemble $crit$ regroupant les ensembles d'éléments $crit[v]$
Résultat : L'ensemble T des MHS de $\mathcal{F}$
Fonction $MMCS(S, \Upsilon, T, uncov, crit)$
  si $uncov = \emptyset$ **alors**
    $T = T \cup \{S\}$
  sinon
    Choisir un $F_i \in uncov$
    $C = \Upsilon \cap F_i$
    $\Upsilon = \Upsilon \backslash C$
    pour tous les $v \in C$ **faire**
      // Faire une copie des $crit()$ et de $uncov$.
      $crit2(s) = crit(s)\ \forall s \in S$
      $uncov2 = uncov$
      //Mise à jour des $crit()$ et $uncov$ pour l'ajout de v dans S.
      pour tous les $F \in \mathcal{F}$ *tel que* $F \cap v \neq \emptyset$ **faire**
        si $F \in crit(u)$ *pour des* $u \in S$ **alors**
          $crit(u) = crit(u) \backslash \{F\}$
        si $F \in uncov$ **alors**
          $uncov = uncov \backslash \{F\}$
          $crit(v) = crit(v) \cup \{F\}$
      //Récursivité avec l'ajout de v à S.
      si $crit(s, S \cup \{v\}) \neq \emptyset\ \forall s \in S$ **alors**
        $T = MMCS(S \cup \{v\}, \Upsilon, T, uncov, crit)$
        $\Upsilon = \Upsilon \cup \{v\}$
      $crit(s) = crit2(s)\ \forall s \in S$
      $uncov = uncov2$
  retourner T ;

Algorithme 8 : *Fonction* `MMCS`
Initialisation : $S = \emptyset, \Upsilon = V, T = \emptyset, uncov = \mathcal{F}, crit[v] = \emptyset\ \forall v \in V$.

alors le comportement d'un individu décrit sur une centaine de variables, parait être une tache longue et contre productive. C'est pourquoi des méthodes exploratoires ont été développées ces dernières années pour tenter d'interpréter ces grands jeux de données et d'en tirer une information statistique de qualité. Les applications sont nombreuses dans notre société actuelle, que ce soit pour développer des modèles de détection de fraude, d'action terroriste, pour analyser des comportements de consommateurs et faire des suggestions de produits ou pour créer un modèle de tarification adapté.

Ces différentes méthodes statistiques, descriptives ou prédictives, sont regroupées pour former une famille de méthodes nommée *analyse de données*. Plus formellement, l'analyse de

Données : Un sous ensemble d'attributs S
Un ensemble de MHS T
Une famille d'ensembles $uncov$
l'ensemble $crit$ regroupant les ensembles d'éléments $crit[v]$
Résultat : L'ensemble T des MHS de $\mathcal{F}$
Fonction $RS(S, T, uncov, crit)$
  si $uncov = \emptyset$ **alors**
    $T = T \cup \{S\}$
  sinon
    $Transg := \emptyset$
    $i = min(j|F_j \in uncov)$
    pour tous les $v \in F_i$ **faire**
      si v *est un élément transgressant* **alors**
        $Transg = Transg \cup \{v\}$
    pour tous les $v \in F_i \backslash Transg$ **faire**
      // Faire une copie des $crit()$ et de $uncov$.
      $crit2(s) = crit(s)\ \forall s \in S$
      $uncov2 = uncov$
      //Mise à jour des $crit()$ et $uncov$ pour l'ajout de v dans S.
      pour tous les $F \in \mathcal{F}$ *tel que* $F \cap v \neq \emptyset$ **faire**
        si $F \in crit(u)$ *pour des* $u \in S$ **alors**
          $crit(u) = crit(u) \backslash \{F\}$
        si $F \in uncov$ **alors**
          $uncov = uncov \backslash \{F\}$
          $crit(v) = crit(v) \cup \{F\}$
      //Récursivité avec l'ajout de v à S.
      si $min(t|F_t \in crit(s)\ \forall s \in S$ **alors**
        $T = RS(S \cup \{v\}, T, uncov, crit)$
      $crit(s) = crit2(s)\ \forall s \in S$
      $uncov = uncov2$
  retourner T ;

Algorithme 9 : *Fonction* `RS`
Initialisation : $S = \{\emptyset\}, T = \emptyset, uncov = \mathcal{F}, crit[v] = \emptyset\ \forall v \in V$.

données est une famille de méthodes statistiques, descriptives et multidimensionnelles, dont l'objectif est l'étude de l'homogénéité des données et les relations pouvant exister entre elles. Dans cet ensemble de méthode, nous pouvons notamment distinguer :

- L'analyse par classification : Approche consistant à déterminer des règles statistiques permettant de regrouper les données les plus similaires. Nous présenterons cet aspect dans la sous-section 2.5.1,

- L'analyse par réduction de dimension : Approche consistant à réduire le jeu de données

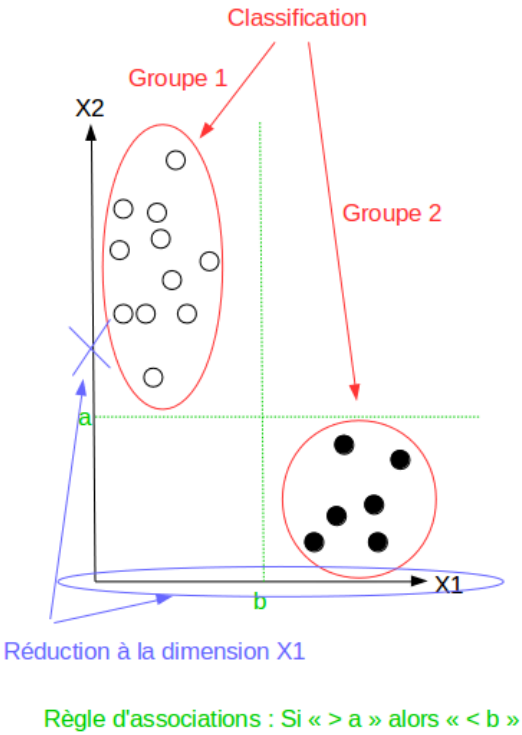


FIGURE 2.4 – Représentation naïve de différents aspects de l'analyse de données

afin de le simplifier ou mettre en évidence les dimensions ou données qui paraissent statistiquement les plus descriptives. Nous présenterons cet aspect dans la sous-section 2.5.2,

- L'analyse par règle d'associations : Approche qui consiste à rechercher des règles statistiques sur les liens entre les données. Nous présenterons cet aspect dans la sous-section 2.5.3,

La figure 2.4 présente (de façon naïve) ces différents aspects : la classification va réunir les données qui se ressemblent, la réduction de dimension va diminuer le nombre de dimensions sur lesquelles observer les données afin de conserver au mieux leur groupement et la recherche de règle d'associations va déterminer des liens communs entre les dimensions. Dans notre exemple, à chaque fois qu'une donnée est supérieure à a sur l'axe $X2$, elle sera inférieure à b sur l'axe $X1$.

Certaines de ces méthodes correspondent également à des processus d'apprentissage automatique. L'apprentissage automatique est un champs d'étude de l'intelligence artificielle servant par exemple à prédire l'appartenance d'un objet à une classe ou à sélectionner la meilleure action à effectuer. Par exemple, lors d'achat sur internet, certains sites proposent des "suggestions" grâce à des méthodes de classification. À partir d'observations sur les achats d'un ensemble de clients, il est possible d'extraire des "profils types" et de classifier les produits et/ou les acheteurs. Si le comportement d'un acheteur ressemble à ceux d'une même classe, on

peut lui suggérer les produits régulièrement achetés par les clients associés à cette classe.

L'analyse logique de données et la caractérisation multiple sont également des méthodologies basées sur des ensembles de données dont les objectifs sont descriptifs et prédictifs. Dans cet aspect, la caractérisation va discriminer le groupe de manière exacte. Toutefois, extraire de l'information de manière statistique peut être également intéressant pour :

- Réaliser un prétraitement sur les données : avant d'étudier les données, retirer celles dont on n'aura probablement pas besoin, voir émettre des suppositions sur celles qui seront les plus utiles,
- Mettre en évidence des facteurs qui paraissent statistiquement discriminants,
- Anticiper de potentielles erreurs lors de la constitution du jeu de données.

2.5.1 Classification

La classification consiste à regrouper en *classes* des données décrivant des *objets*. Ces classes doivent être les plus homogènes possible, c'est à dire partager le plus de caractéristiques communes. Concrètement, l'objectif de la classification est de regrouper les objets similaires pour travailler ensuite sur les classes. Par exemple, en marketing, il peut être intéressant de découper la clientèle en segments pour faire une communication spécifique à chaque segment, et ainsi cibler davantage la clientèle. L'objectif n'est donc pas nécessairement de comprendre ces classes et les étudier, mais bien de les construire. La classification est également utilisée en apprentissage. En effet, l'induction (émettre des hypothèses de lois à partir d'observations), et donc la capacité d'associer de nouveaux objets à des classes par l'observation de jeux de données permet d'apprendre sur un sous-ensemble pour une exploitation à grande échelle.

Pour mesurer la ressemblance des objets et ainsi construire des classes homogènes, il est important de choisir une distance entre les objets. Dans les cas simples où toutes les variables sont quantitatives et exprimées dans la même unité, on choisit en général la distance euclidienne. Dans le cadre des données binaires, la distance euclidienne est équivalente au coefficient d'appariement simple, c'est à dire le ratio du nombre de variables différentes sur le nombre de variables totales. Pour les données binaires, si les variables sont symétriques, c'est-à-dire que la valeur "1" a autant d'impact que la valeur "0", alors le coefficient d'appariement est généralement utilisé. Par exemple, une variable "genre" est une variable symétrique car coder le genre féminin par 1 et le genre masculin par 0 ou l'inverse est équivalent.
Les variables peuvent également être asymétriques, c'est à dire qu'une similarité positive (i.e. la valeur de l'attribut est à "1" pour chaque objet) est plus importante qu'une similarité négative (i.e. la valeur de l'attribut est à "0" pour chaque objet). Par exemple, la présence de symptômes d'une maladie peut avoir plus d'impact que l'absence de ces symptômes.

Il existe plusieurs autres mesures de distances/similarités dont parmi elles :

- L'index de Russel et Rao (Rao [1948]) : compte le nombre de similarités positives sur le nombre de variables globales.
- Le coefficient de Sokal et Michener (Sokal [1958]) : qui est équivalent au coefficient d'appariement simple (il mesure les similarités au lieu des différences).
- Le coefficient de Jaccard (Sneath [1957]) : compte le nombre de similarités positives mais contrairement à l'index de Russel et Rao, ne divise que par le nombre de variables différentes et de similarités positives (on ignore les similarités négatives lors de la division).
- Le coefficient de Dice (ou Sorensen) (Dice [1945]) : identique au coefficient de Jaccard si ce n'est qu'il pondère par deux le nombre de similarités positives (au numérateur et au dénominateur).
- Le coefficient de Haman (Holley and Guilford [1964]) : il s'agit du nombre de similarités auquel on soustrait le nombre de différences, puis on divise par le nombre de variables. Cela nous donne une mesure comprise entre -1 et 1.

Une fois la distance $d(i, j)$ entre deux objets i et j choisie, il existe de multiples méthodes de partitionnement des données dont les plus courantes sont :

- les méthodes de regroupement hiérarchique (cluster ascendant hiérarchique),
- les méthodes basées centroïdes,
- l'algorithme *Expectation-Maximization* (`EM`) (Dempster et al. [1977]).

2.5.1.1 Méthodes de regroupement hiérarchique

Bien qu'il existe différentes méthodes de regroupement hiérarchique, seule la classification ascendante hiérarchique reste réellement utilisée aujourd'hui. La classification ascendante hiérarchique est une méthode qui va initialement considérer que chaque objet appartient à une classe différente et va progressivement réunir les différentes classes en fonction d'une distance de classe déterminée à partir de la distance des objets qui les composent. Nous obtenons donc une hiérarchie de classes que l'on peut représenter sous la forme d'un dendrogramme.

Exemple 13. *La figure 2.5 est un dendrogramme qui représente une hiérarchie de classes de 6 objets notés de $x1$ à $x6$.*

Nous avons initialement considéré que chaque objet constitue une classe. La distance entre les classes constituées respectivement de l'objet $x1$ et $x2$ étant la plus faible, nous les avons réuni dans une même classe au niveau 1. Puis, la distance entre cette nouvelle classe et la classe

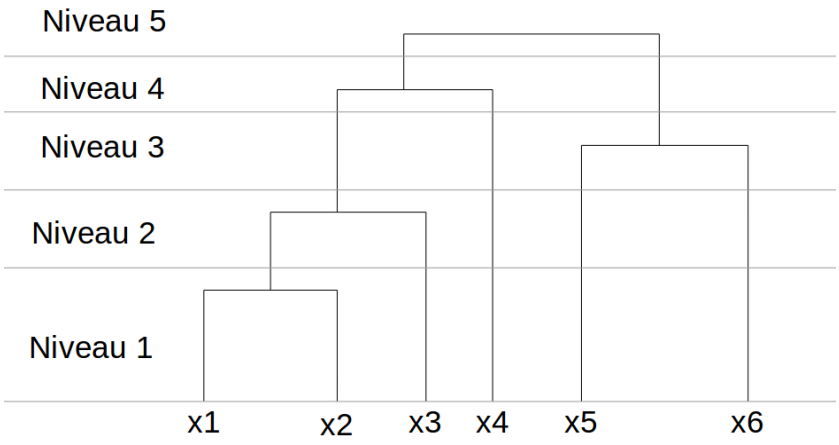


FIGURE 2.5 – Exemple de dendrogramme.

constituée de $x3$ est la nouvelle distance la plus faible. Nous avons réuni ces deux classes au niveau 2. Puis nous avons continué de cette façon jusqu'à obtenir une unique classe composée de tous les objets au niveau 5. Le dendrogramme nous indique donc une priorité de groupement à considérer. Par exemple, si nous décidons de choisir les classes au niveau 3, nous aurons trois classes G_1, G_2 et G_3 avec $G_1 = \{x1, x2, x3\}$, $G_2 = \{x4\}$ et $G_3 = \{x5, x6\}$.

Soit n le niveau choisi et m le nombre d'objets, il est évident que le nombre de classes sera $m - n$. Ainsi, si nous devons générer un nombre de classes précis, il suffit de prendre le niveau adéquate. Si le nombre de classes n'est pas déterminé, on choisit en général le niveau qui offre la plus grande inertie inter-classe B avec :

$$B = \frac{\sum_{k=1}^{K} m_k \times (\bar{x}_k - \bar{x})^2}{m}$$

où m représente le nombre d'objets, K le nombre de classes, m_k le nombre d'objets dans la classe k, $\bar{x}_k$ l'objet moyen de la classe k et $\bar{x}$ l'objet moyen global.

En choisissant le niveau maximisant B, nous obtenons des classes bien différentiées. Par convention, la hauteur de chaque niveau du dendrogramme représente la valeur de B et ainsi, nous choisissons le niveau de plus grande hauteur.

Concernant la mesure de distance $d(G_1, G_2)$ entre les classes G_1 et G_2, tout comme la mesure de distance $d(a, b)$ entre les objets a et b, il en existe plusieurs :

- Le lien minimum : $d(G_1, G_2) = min(d(a, b)|a \in G_1, b \in G_2)$.
- Le lien maximum : $d(G_1, G_2) = max(d(a, b)|a \in G_1, b \in G_2)$.
- La distance moyenne : $d(G_1, G_2) = \frac{\sum_{a \in G_1} \sum_{b \in G_2} d(a,b)}{m_{G_1} m_{G_2}}$, où m_G représente le nombre d'objets dans la classe G.

- La distance entre les moyennes : $d(\bar{x}_{G_1}, \bar{x}_{G_2})$, où $\bar{x}_G$ représente l'objet moyen de la classe G.
- La distance de Ward : $\frac{d(\bar{x}_{G_1}, \bar{x}_{G_2})^2}{\frac{1}{m_{G_1}} + \frac{1}{m_{G_2}}}$, où m_G représente le nombre d'objets dans la classe G et $\bar{x}_G$ représente l'objet moyen de la classe G.

La distance de Ward est une distance peu coûteuse et moins sensible aux valeurs extrêmes que le lien maximum, tout en proposant un bon compromis entre les autres distances. Toutefois, en fonction de la "forme géométrique" des classes que l'on souhaite avoir, on peut être enclin à choisir d'autres distances.

Remarque 3. *La phylogénie présentée dans la section 2.1 est une méthode de classification ascendante hiérarchique, cherchant à classer les souches bactériennes à partir de leurs génomes. La figure 2.1 représente le dendrogramme obtenu à partir d'un certain niveau.*

2.5.1.2 Méthodes basées centroïdes

Les méthodes basées centroïdes ont pour objectif de réunir un ensemble objets semblables autour d'un objet, réel ou théorique, représentatif de cet ensemble. Cet objet représentatif de l'ensemble est nommé centroïde. Si nous voulons répartir nos objets en k classes, l'objectif est de créer k centroïdes et de réunir chaque objet autour du centroïde qui lui ressemble le plus.

Parmi les nombreux algorithmes qui peuvent être utilisés dans cette méthodologie (voir Aggarwal and Reddy [2013]), la méthode la plus populaire est probablement la méthode des *K-moyennes* (*K-means* en anglais) (Hartigan and Wong [1979]; MacQueen et al. [1967]) où le centroïde d'une classe est l'objet moyen de cette classe. Il s'agit donc d'un processus dynamique où le centroïde évolue en fonction des objets qui composent la classe, et où la composition de la classe dépend du centroïde. L'objectif étant d'obtenir des classes similaires, et surtout ressemblant à leur centroïde, la méthode cherchera à optimiser la fonction suivante :

$$min \sum_{i=1}^{k} \sum_{o \in S_i} d(o, \mu_i)$$

où S_i représente l'ensemble des objets de la classe i et μ_i l'objet moyen de la classe S_i. Autrement dit, la méthode k-means cherchera à classer les objets de façon à minimiser la somme des écarts entre les observations et le centroïde de leur classe.

La classification k-means étant un problème d'optimisation NP-difficile (Dasgupta [2008]), on utilise régulièrement des algorithmes heuristiques. Par la suite nous utiliserons l'algorithme heuristique classique présenté dans Hartigan and Wong [1979] car nous ne cherchons pas la performance. Les centroïdes peuvent également être fixés et les classes sont ainsi construites autour de ces centroïdes donnés (approche supervisée).

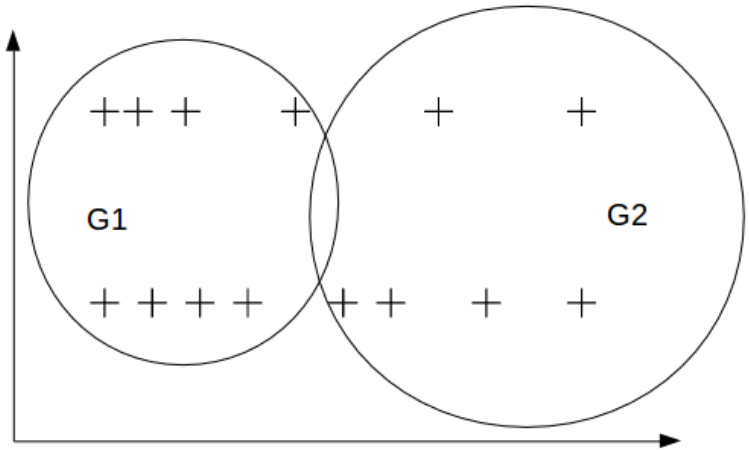


FIGURE 2.6 – Exemple de “mauvaise” classification k-means.

Dans la méthode k-means, les centroïdes sont des objets moyens, inexistants dans notre ensemble d'objets. La méthode des *K-médoïdes* (Kaufman and Rousseeuw [1990]) est une méthode proche des k-means, mais où le centroïde n'est plus l'objet moyen de la classe, mais son médoïde, c'est à dire l'objet le plus central. Ainsi, le centroïde n'est plus un objet théorique mais bien réel et le plus représentatif de la classe.

Notons que les algorithmes heuristiques de ce type sont très connus et très peu coûteux en calcul, mais sont très sensibles à l'initialisation des centroïdes et ne détecte que les “classes sphériques” (par rapport à la distance utilisée).

La figure 2.6 nous présente une répartition d'objets sur deux critères (en abscisses et ordonnées). Il parait simple de voir qu'on peut les séparer en deux classes bien distinctes : ceux avec une ordonnée élevée et ceux avec une ordonnée faible. Cependant la classification de type k-means ne détecte que les classes sphériques et fera un autre classement où l'inertie entre les classes sera plus faible. Toutefois, avec des données binaires et une distance euclidienne, les objets se comparent sur le nombre de variables similaires. Les classes seront donc sphériques et le modèle k-means valable.

Remarquons que contrairement à la classification ascendante hiérarchique, la méthode k-means ne peut être réalisée qu'en connaissant le nombre de centroïdes à générer, et donc il est nécessaire de connaître le nombre de classes que nous souhaitons, ce qui n'est pas toujours le cas en pratique.

2.5.1.3 L'algorithme *Expectation-Maximization*

L'algorithme espérance-maximisation (Expectation-Maximization en anglais et souvent abrégé en EM) a été proposé par Dempster et al. [1977] pour faire de l'inférence statistique de modèle

"à variables cachées", c'est à dire estimer des densités de probabilités conditionnelles à partir de densités jointes. Par exemple, la longueur des ailes des passereaux suit une loi normale, mais de paramètres différents chez les mâles et les femelles. Si on mesure la taille des ailes chez une sous-population de passereaux mâles et femelles réunis, nous obtiendrons une densité jointe qui n'aura plus la "forme" de celle d'une loi normale.

L'utilisation du théorème de Bayes (Bayes et al. [1763]) va permettre de calculer les probabilités à posteriori des objets d'appartenir aux différentes classes.

L'algorithme fonctionne en deux étapes que l'on réitère :

- Étape E : Évaluation de l'espérance où l'espérance de vraisemblance des classes est calculée en fonction des paramètres estimés à l'étape M précédente.
- Étape M : Maximisation de la vraisemblance des paramètres en fonction des espérances de vraisemblance trouvées à l'étape E précédente.

Il s'arrête quand il converge.

Si le coût calculatoire de l'algorithme est raisonnable et que l'on n'a pas d'influence liée à l'initialisation, l'algorithme fonctionne mal sur des instances avec beaucoup de variables.

2.5.2 Réduction de dimension

Si le clustering permet de regrouper les objets, la réduction de dimension permet de travailler sur moins de caractéristiques. Par exemple, dans la sous-section 2.5.1.2, la figure 2.6 présente une classification par les k-means moins intéressante que celle que nous observons instinctivement (à savoir une classe pour les objets avec une ordonnée élevée, et un second pour ceux avec une ordonnée faible). Toutefois, en supprimant de l'information et en ne considérant plus que la variable représentée en ordonnée, les deux classes deviennent évidentes et seront les mêmes quelque soit la méthode utilisée (puisqu'il ne reste plus que deux valeurs différentes pour ces objets). De façon générale, il est souvent utile de sélectionner l'information en fonction des classes dont on dispose ou que l'on souhaite obtenir non seulement pour éviter l'information redondante ou inutile, mais également pour travailler sur de moins grandes dimensions. Deux approches existent :

- La sélection de variables, qui consiste à simplement supprimer des variables pour ne conserver que les plus intéressantes. Malheureusement, sur le plan statistique, cela peut entraîner beaucoup de pertes d'informations.
- L'extraction de variables, qui va construire de nouvelles variables à partir des variables initiales (par combinaison linéaire) de sorte qu'elles expliquent au mieux le phénomène étudié. Par exemple, soient v_1, v_2, v_3 trois variables, on peut former une nouvelle variable

$v = f(v_1, v_2, v_3)$ et ainsi travailler sur deux variables de moins. S'il n'y a que peu de pertes d'informations, ces variables n'ont malheureusement aucune signification et il peut être difficile d'interpréter le résultat.

2.5.2.1 Méthodes d'extraction de variables

Concernant les méthodes d'extraction de variables, l'analyse en composantes principales (Person [1901]) et l'analyse discriminante de Fisher (Fisher [1936]) sont les principales méthodes utilisées pour les données quantitatives.

L'**analyse en composante principale** (ACP) va quantifier la part d'information conservée par la part de variance conservée. C'est pourquoi à partir d'une matrice de données D de n objets et m variables, l'ACP va travailler sur la matrice de variance-covariance $V = \frac{1}{n}\bar{D}^t\bar{D}$ où $\bar{D}$ est la matrice centrée des données et D^t est la transposée de la matrice D. Le premier axe u_1 de l'ACP est le vecteur propre associé à la plus grande valeur propre de V et les vecteurs principaux suivants sont les vecteurs propres de V associés aux plus grandes valeurs propres suivantes.

On détermine ces nouveaux axes en suivant les étapes suivantes :

- Calcul de V,
- Décomposition spectrale de V : $V = U^t \Delta U$ où U est la matrice des vecteurs propres ordonnés,
- Choisir $U_1, \ldots, U_{m'}$ les m' premières colonnes de U.

Si m' (le nombre de variables à retenir) n'est pas déterminé, on choisit en général $m' = min_{m'}$ tel que $\frac{\sum_{j=1}^{m'} \lambda_j}{\sum_{j=1}^{m} \lambda_j} > 0.9$ où λ_j est la j-ième valeur propre des valeurs propres ordonnées.

Contrairement à l'ACP, l'**analyse discriminante de Fisher** (ADF) prend en entrée le jeu de données ainsi que les classes des objets étudiés. L'objectif est de trouver des axes tels que la projection des données sur ces axes sépare au mieux les classes et que chaque classe soit le plus homogène possible. Il s'agit donc :

- de maximiser l'inertie inter-classe, c'est à dire l'éloignement entre les classes, traduit par la matrice de variance-covariance inter-classe B avec :

$$B = \frac{1}{m} \sum_{k=1}^{K} m_k \times (\bar{x}_k - \bar{x})^2$$

où m représente le nombre d'objets, K le nombre de classes, m_k le nombre d'objets dans la classe k, $\bar{x}_k$ l'objet moyen de la classe k et $\bar{x}$ l'objet moyen global,

- de minimiser la dispertion à l'intérieur des classes, donnée par la matrice de dispersion intra-classe W avec :

$$W = \frac{1}{m} \sum_{k=1}^{K} m_k \times W_k$$

où W_k est la matrice de variance-covariance de la classe k.

Notons également que la dispersion totale est donnée par la matrice de variance-covariance totale $V = B + W$ d'après le théorème d'Huyghens. Ainsi, les axes retenus seront les vecteurs propres associés aux valeurs propres non-nulles de la matrice $V^{-1}B$. Nous obtenons au maximum $K - 1$ nouveaux axes dont nous choisirons en priorité ceux associés aux valeurs propres les plus élevées.

Notons que ces méthodes ne fonctionnent que pour un jeu de données quantitatives, c'est à dire des données qui peuvent être mesurées (par exemple le poids ou la taille). Concernant les variables qualitatives, c'est à dire qu'on ne peut mesurer (par exemple le sexe ou un groupe social), on utilise généralement l'**analyse des correspondances multiples** (ACM).

L'objectif est identique à l'ACP : visualiser les proximités entre les catégories des variables et les objets. Pour cela, il est nécessaire de créer un tableau disjonctif complet, c'est à dire un tableau où chaque ligne représente un objet et chaque colonne une modalité (et non une variable). La valeur x_{ij} de ce tableau est 1 si l'objet i possède la modalité j, 0 sinon. À partir de ce tableau disjonctif complet, l'ACM construira un tableau de Burt, c'est-à-dire un tableau similaire au tableau de contingence, regroupant l'ensemble des modalités en ligne et en colonne et comptant le nombre d'objets ayant simultanément la modalité en ligne et en colonne. Il s'agit d'une matrice regroupant tous les tableaux de contingence associés à tous les couples de variables.
Il est ainsi possible de construire des matrices P_L des *profils lignes* et P_C des *profils colonnes* telles que :

- La l-ième ligne de P_L est $(\frac{\eta_{l1}}{\eta_{l+}}, \ldots, \frac{\eta_{lh}}{\eta_{l+}}, \ldots, \frac{\eta_{lc}}{\eta_{l+}})$ avec η_{lh} la valeur de la l-ième ligne et h-ième colonne du tableau de Burt, η_{l+} la somme des valeurs de la l-ième ligne du tableau de Burt et c le nombre de modalités totales,

- La h-ième ligne de P_C est $(\frac{\eta_{1h}}{\eta_{+h}}, \ldots, \frac{\eta_{lh}}{\eta_{+h}}, \ldots, \frac{\eta_{ch}}{\eta_{+h}})$, avec η_{+h} la somme des valeurs de la h-ième colonne du tableau de Burt.

Le principe de l'ACM est d'effectuer une ACP sur la matrice P_L ou P_C en prenant une distance du χ^2, c'est à dire :

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

.

Remarque 4. *Les matrices P_C et P_L sont identiques car le tableau de Burt est symétrique. Toutefois, si on ne compare que deux variables, alors il est intéressant de partir du tableau de contingence et non du tableau de Burt pour faire les profils lignes et colonnes, et obtenir des matrices P_C et P_L différentes, mais où les ACP se correspondent puisque les données sont extraites de la même table de contingence. Il est donc dans ce cas intéressant de superposer les deux représentations graphiques. Ce cadre particulier s'appelle l'analyse factorielle des correspondances (AFC).*

Nous obtenons ainsi un nouveau repère où l'on peut placer les objets et modalités tels que à un coefficient près, pour un axe donné :

- Un objet est au barycentre des modalités qu'il possède,
- Une modalité est au barycentre des objets qui la possèdent.

Toutefois, si les analyses en composantes principales et analyses discriminantes de Fisher sont efficaces, ce n'est généralement pas le cas de l'analyse des correspondances multiples et on ne l'utilise généralement que pour les petits tableaux de données. Nous parlerons donc dans cette sous-section principalement des méthodes de sélection de variables.

2.5.2.2 Méthodes de sélection de variables

Les méthodes de sélection de variables recherchent le sous-ensemble de variables le plus prédictif, c'est-à-dire des variables expliquant au mieux une variable cible (souvent la classe). Il s'agit donc d'éliminer les variables n'apportant que peu d'informations à l'explication de cette variable cible.

Ces méthodes d'élimination de variables peuvent se classer en trois types (Kohavi and John [1997]) :

- Les méthodes de filtrage (filter methods) qui consistent à classer les variables et ne conserver que celles en tête de liste,
- Les méthodes enveloppantes (wrapper methods) qui consistent à trouver un sous-ensemble de variables donnant la meilleure performance de prédiction,
- Les méthodes intégrées (embedded methods) qui combinent sélection de variables et estimation du modèle.

Leur objectif est double : le sous-ensemble de variables doit être pertinent, c'est-à-dire que les variables doivent être fortement corrélées avec la variable cible et elles ne doivent pas être redondantes, c'est-à-dire qu'elles doivent n'être que faiblement corrélées entre elles.

Méthodes de filtrage

Le principe de ces méthodes est de classer les variables selon un critère de sélection approprié. Un seuil est utilisé pour éliminer les variables en dessous de ce seuil. En général, le critère de sélection choisi évalue la pertinence d'une variable, c'est pourquoi ces méthodes sont plutôt utilisées comme étape de pré-traitement.

L'un des critères les plus simples est le **coefficient de corrélation de Pearson** (Battiti [1994]; Guyon and Elisseeff [2003]). Le coefficient de corrélation entre la variable cible Y et la i-ième variable X_i est défini ainsi :

$$R(i) = \frac{cov(X_i, Y)}{\sigma_{X_i}\sigma_Y}$$

où $cov(X, Y)$ désigne la covariance entre les variables X et Y et σ_x l'écart type de la variable X. On obtient $R(i) = 0$ si la variable X_i est indépendante de Y (et donc non pertinente), $R(i) = 1$ si on a une relation linéaire positive et $R(i) = -1$ si on a une relation linéaire négative.

Notons que le coefficient de corrélation mesure la dépendance dans laquelle la relation entre les deux variables est linéaire. Dans le cadre général, nous utilisons l'**information mutuelle** (MI) comme mesure de dépendance entre ces deux variables (Battiti [1994]; Guyon and Elisseeff [2003]; Kwak and Choi [2002]; Lazar et al. [2012]). Plus particulièrement, MI mesure la quantité d'information apportée par une variable X sur les probabilités de la variable cible Y. En mesurant l'absence d'information par l'entropie, on a la valeur d'information mutuelle $I(X, Y) = H(X) - H(X|Y)$ avec $H(X)$ l'entropie de la variable X et $H(X|Y)$ l'entropie conditionnelle de X sachant Y. $H(X)$ et $H(X|Y)$ se calculent comme suit :

$$H(X) = -\sum_i P(X = x_i) log(P(X = x_i))$$

et

$$H(X|Y) = -\sum_i \sum_j P(X = x_i, Y = y_j) log(P(Y = y_j|X = x_i))$$

où les x_i représentent les différentes valeurs de la variable X et les y_j les différentes classes. Les probabilités sont estimées par les fréquences des différentes valeurs possibles.

On obtient ainsi le résultat suivant :

$$I(X, Y) = \sum_i \sum_j P(X = x_i, Y = y_j) log(\frac{P(X = x_i, Y = y_j)}{P(X = x_i)P(Y = y_j)})$$

L'information mutuelle $I(X, Y) = 0$ si et seulement si X et Y sont indépendantes et donc si X n'est pas pertinente.

Notons qu'encore une fois, MI est utilisée avec des variables quantitatives. Concernant les variables qualitatives, une approche similaire est l'**incertitude symétrique** (Witten et al. [2016]). Assez proche de la corrélation, cette valeur va calculer le ratio entre l'information mutuelle à partir des fréquences conjointes et la somme des entropies. À partir du tableau de contingence entre la variable cible Y et une variable X, la fréquence conjointe entre la modalité x_l et la classe y_k sera $p_{kl} = \frac{\eta_{kl}}{\eta}$ avec η le nombre d'objets (ou la somme des valeurs du tableau de contingence) et η_{kl} le nombre d'objets appartenant à la classe y_k et présentant la modalité x_l. On notera $p_{k.} = \frac{\sum_l \eta_{kl}}{\eta}$ et $p_{.l} = \frac{\sum_k \eta_{kl}}{\eta}$ les fréquences marginales lignes et colonnes. Ainsi, comme précédemment, l'information mutuelle se calcule comme suit :

$$I(Y,X) = \sum_k \sum_l p_{kl} log(\frac{p_{kl}}{p_{k.} \times p_{.l}})$$

De la même façon, la valeur des entropies vaut

$$H(X) = -\sum_l p_{.l} log(p_{.l})$$

et

$$H(Y) = -\sum_k p_{k.} log(p_{k.})$$

La valeur de l'incertitude symétrique vaut alors :

$$\rho_{Y,X} = 2 \times \frac{I(Y,X)}{H(Y) + H(X)}$$

Notons que la valeur est symétrique ($\rho_{Y,X} = \rho_{X,Y}$) et est comprise entre 0 et 1. Ainsi, plus $\rho_{Y,X}$ est proche de 1, plus la variable X est pertinente.

Étudier la dépendance d'une variable par rapport à une autre peut avoir rapidement ses limites. En effet, une variable X peut être fortement dépendante d'une autre variable Y, mais moins que la "réunion" d'un ensemble de variables $\{X_1, \ldots, X_n\}$ même si chacune de celles-ci, prise individuellement, offre une faible dépendance avec Y.

C'est pourquoi, toujours dans les variables qualitatives, la méthode **CFS** (Correlation based Feature Selection) (Hall and Smith [1997]) va se baser sur une mesure μ évaluant un ensemble de variables M non seulement face à la valeur cible, mais également entre elles. Cette mesure évaluera donc à la fois la pertinence et la redondance. Cette mesure se définie comme suit :

$$\mu_M = \frac{m \times \bar{\rho}_{Y,X}}{\sqrt{m + m \times (m-1) \times \bar{\rho}_{X,X}}}$$

avec m le nombre de variables sélectionnées. $\bar{\rho}_{Y,X}$ est la moyenne des corrélations entre les

variables choisies et la cible :

$$\bar{\rho}_{Y,X} = \frac{1}{m} \sum_{i} \rho_{Y,X_i}$$

et $\bar{\rho}_{X,X}$ est la moyenne des corrélations croisées entre les variables choisies :

$$\bar{\rho}_{X,X} = \frac{2}{m \times (m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \rho_{X_i,X_j}$$

Il s'agit donc de déterminer le sous-ensemble M offrant la meilleure valeur μ_M.

Méthodes enveloppantes

Si les méthodes de filtrage permettent un pré-traitement efficace, elles ne se basent que sur la pertinence et, quelques fois, la redondance mais sont indépendantes vis à vis de la technique d'apprentissage supervisé, c'est à dire à la technique de production de règle à partir de la base de données. Les méthodes enveloppantes (Kohavi and John [1997]), quant à elles, vont évaluer et optimiser un critère de performance, basé régulièrement sur le taux d'erreur, de la méthode d'apprentissage avec les variables sélectionnées. Bien évidemment, évaluer 2^N sous-ensembles de variables (où N est le nombre global de variables) est un problème NP-difficile et donc très coûteux en calcul. C'est pourquoi les méthodes de recherche sont généralement heuristiques ou aléatoires.

La base de données est divisée en deux parties, une première pour choisir le modèle d'apprentissage et les variables sélectionnées (base d'apprentissage) et une seconde pour valider le sous-ensemble de variables sélectionnées (base de validation).

L'algorithme "**Sequential Forward Selection**" (`SFS`) (Marill and Green [1963]) est un algorithme naïf de sélection séquentielle qui part d'un ensemble de variables vide et ajoute à chaque étape la variable augmentant au mieux le critère de performance jusqu'à atteindre un nombre de variables fixé (algorithme 10).

Similairement à `SFS`, l'algorithme "**Sequential Backward Selection**" (`SBS`) a été proposé dans Whitney [1971] et part de l'ensemble de toutes les variables pour les retirer une à une (algorithme 11).

`SBS` donne de meilleurs résultats car l'évaluation est faite avec un ensemble plus grand et gère donc mieux la redondance. Par contre une variable éliminée ne sera plus prise en compte et donc plus l'algorithme évolue, plus nous limiterons le potentiel du sous-ensemble final. Pour rendre l'approche plus souple, l'article Pudil et al. [1994] propose l'algorithme "**Sequential Floating Forward Selection**" (`SFFS`) qui consiste à utiliser successivement les algorithmes `SFS` et `SBS` jusqu'à l'obtention d'un critère d'arrêt.

La génération de sous-ensemble de variables augmentant le critère de performance étant coûteux en temps de calcul, une approche aléatoire a également été proposée dans Liu and

Données : $F = \{f_1, \ldots, f_N\}$ un ensemble de variables
N' nombre de variables désiré
Résultat : $F' \subset F$ l'ensemble de variables sélectionnées
$F' = \emptyset$
pour *i de* 1 *à* N' **faire**
  $f_{max} = f_1$
  pour tous les $f \in F \backslash \{f_1\}$ **faire**
    si $F' \cup \{f_i\}$ *est mieux évalué par le critère de performance que* $F' \cup \{f_{max}\}$ **alors**
      $f_{max} = f_i$
  $F' = F' \cup \{f_{max}\}$
  $F = F \backslash \{f_{max}\}$
retourner(F')

Algorithme 10 : `SFS`

Données : $F = \{f_1, \ldots, f_N\}$ un ensemble de variables
N' nombre de variables désiré
Résultat : $F' \subset F$ l'ensemble de variables sélectionnées
pour *i de* 1 *à* $(N - N')$ **faire**
  $f_{min} = f_1$
  pour tous les $f \in F \backslash \{f_1\}$ **faire**
    si $F' \backslash \{f_i\}$ *est mieux évalué par le critère de performance que* $F' \backslash \{f_{min}\}$ **alors**
      $f_{min} = f_i$
  $F = F \backslash \{f_{min}\}$
retourner(F)

Algorithme 11 : `SBS`

Setiono [1997] avec l'algorithme "**Las Vegas Wrapper**" (`LVW`). Cette méthode génère simplement un sous-ensemble de variables de façon aléatoire et l'évalue. Puis, en génère de nouveaux sous-ensembles jusqu'à ce que l'un d'eux augmente le critère de performance. On réitère les générations jusqu'à avoir un certain nombre d'essais consécutifs infructueux. Bien évidemment, plus le nombre d'essais infructueux pour arrêter l'algorithme est élevé, plus les chances d'avoir un bon résultat seront élevées, mais plus l'algorithme sera coûteux.

Par soucis de performance, les **algorithmes génétiques** (Ferri et al. [1993]; Goldberg [1989]; Kudo and Sklansky [2000]) sont également utilisés afin de générer des sous-ensembles de variables. Ces algorithmes s'inspirent des mécanismes d'évolution darwinienne et ont été utilisés historiquement par des biologistes pour simuler l'évolution des organismes. Au cours de la reproduction d'organismes, leurs chromosomes (i.e. structures composées d'ADN et supports des gènes) fusionnent en créant un nouvel organisme. La théorie de l'évolution propose qu'au fils du temps, seul ceux qui sont le plus adaptés à l'environnement sont conservés. Les algorithmes génétiques sont une analogie avec cette théorie et "imitent" le "brassage génétique" de cette phase de reproduction. Chouaib [2011] présente le fonctionnement des algorithmes génétiques de la façon suivante :

- Étape 1 - Initialisation : génération aléatoire d'une population de N chromosomes,
- Étape 2 - Évaluation : évalue chaque objet de la population par une fonction appropriée,
- Étape 3 - Reproduction : modifie la population par des méthodes de *sélection*, de *croisement* et de *mutation*,
- Étape 4 - Validation : retourne à l'étape d'Évaluation tant que la condition d'arrêt n'est pas satisfaite.

Les chromosomes sont représentés sous forme d'une chaîne binaire de gènes. Dans notre contexte, il ne s'agit pas de traiter des chromosomes mais des sous-ensembles de variables dont chaque variable sera modélisée par un gène de chromosome, donc une valeur binaire qui définira la présence ou l'absence de la variable. La fonction d'évaluation peut être mono-objectif ou multi-objectif et donc potentiellement proposer plusieurs critères de performance (minimiser l'erreur, minimiser le nombre de variable, . . .).

Dans l'étape de la reproduction, nous avons parlé de 3 méthodes de modification :

- Méthode de sélection : la sélection consiste à choisir les objets qui "survivront", c'est à dire qui généreront de nouveaux objets par croisement ou mutation. Cette sélection se fait par association d'une probabilité de "survie" par objet en fonction de leur évaluation à l'étape précédente.
- Méthode de croisement : le croisement consiste à prendre plusieurs objets (parents) pour en construire des nouveaux (enfants) qui soient le "mélange" des parents. Les chromosomes (ou sous-ensemble de variables dans notre cas) échangent des séquences de gènes entre eux (ou séquences de présence/absence de variables dans notre cas).
- Méthode de mutation : la mutation va tout simplement changer aléatoirement la valeur d'un gène (ou la présence/absence d'une variable dans notre cas) dans un sous-ensemble. Ceci permet d'explorer un espace de recherche qui n'est plus limité par l'initialisation et nous pouvons ainsi potentiellement atteindre l'optimum global.

Si les algorithmes génétiques nous donnent des solutions plus performantes, notamment grâce à l'introduction de critères multi-objectifs, Kudo and Sklansky [2000] les recommande également pour les instances avec un nombre de variables initiales supérieur à 100.

Méthodes intégrées

Les méthodes intégrées sont des méthodes de sélection de variables intégrées à un processus d'apprentissage. Contrairement aux méthodes enveloppantes qui cherchent à diminuer le nombre de variables pour tester la classification, les méthodes intégrées cherchent en premier lieu à classifier, et vont diminuer le sous-ensemble de variables au cours de l'apprentissage.

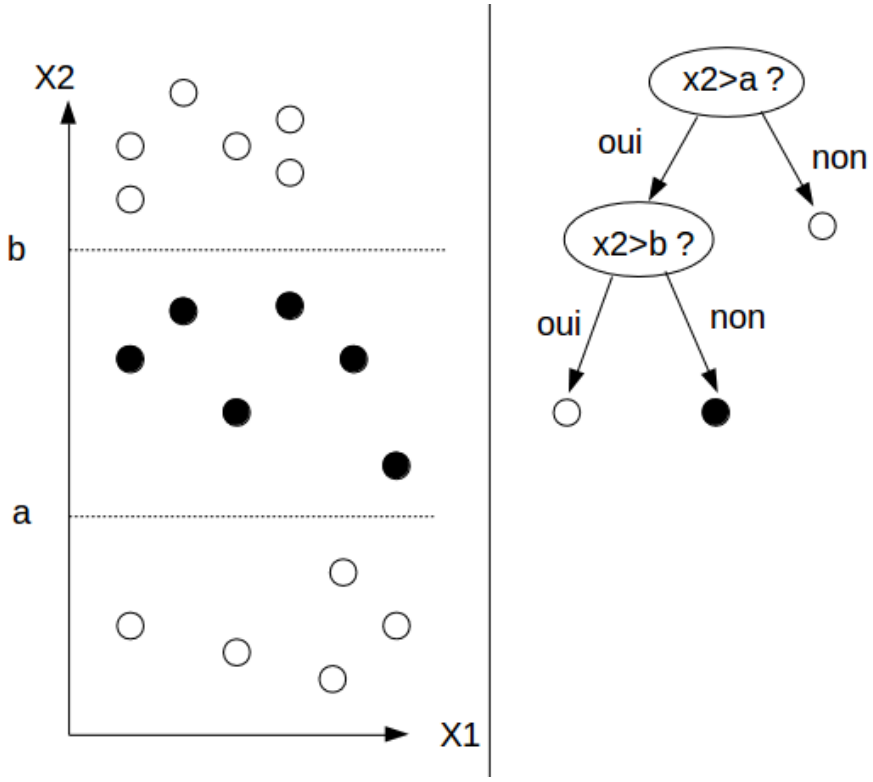


FIGURE 2.7 – Arbre de décisions sur deux dimensions

Les **arbres de décisions** peuvent ainsi intégrer un tel mécanisme. Le principe de l'arbre de décision est de tester les variables d'un objet les unes à la suites des autres. Toutefois, il peut être possible de classifier n'importe quel objet sans tester toutes les variables. Par exemple, sur la figure 2.7, nous pouvons voir un modèle de classification qui n'utilise qu'une seule des deux variables. En effet, bien que nous soyons en deux dimensions, seule la variable $X2$ est utilisée dans l'arbre pour différencier la classe des points blancs de la classe des points noirs.

Il existe plusieurs algorithmes permettant la création d'arbres de décisions, tout comme il existe régulièrement plusieurs arbres de décisions possibles, comme par exemple l'algorithme `CART` (Breiman et al. [1984]), l'algorithme `ID3` (Quinlan [1986]) ou l'algorithme basé sur le principe du *minimum description length* (Quinlan and Rivest [1989]). Ces algorithmes se distinguent notamment par la sélection du critère de décision à chaque niveau de l'arbre.

Les séparateurs à vastes marges (ou support vector machine), couramment désignés par l'abréviation **SVM** (Boser et al. [1992]; Vapnik [2013]) sont des techniques répandues en apprentissage statistique (Ghattas and Ben Ishak [2008]). Ici, l'objectif est de déterminer un hyperplan séparant les objets des deux classes tout en maximisant la *marge*, c'est-à-dire la plus petite distance séparant un point de l'espace des objets à cet hyperplan (ces points sont appelés les vecteurs supports). Si, par exemple, l'hyperplan est d'équation $f(x) = 0$, alors, un objet x_i sera dans une classe si $f(x_i) > 0$ et dans l'autre si $f(x_i) < 0$. Si les données ne sont pas

linéairement séparables, c'est-à-dire qu'il n'existe pas d'hyperplan linéaire séparateur dans le repère, alors l'espace de représentation des données d'entrées est transformé en un espace de plus grande dimension dans lequel il existera probablement un séparateur linéaire. Par exemple, si les données ne peuvent être séparées que par un cercle, alors le cercle est un séparateur linéaire en coordonnées polaires.

Outre le fait que l'hyperplan séparateur peut éventuellement s'exprimer dans une dimension plus petite que l'espace d'entré, se qui permettrait de réduire les dimensions, il est également possible de calculer par cette méthode un score pour définir le rang d'importance des variables (Ghattas and Ben Ishak [2008]). Ainsi, le score peut se baser sur plusieurs critères dépendant des vecteurs supports, comme par exemple la valeur de la marge. Le *score d'ordre zero* d'une variable est la valeur de ce critère lorsque l'on retire cette variable et similairement, le *score par différences* d'une variable est la différence entre la valeur du critère avec toutes les variables et la valeur du critère sans cette variable. Il est ainsi possible de donner un score à chaque variable et de les classer.

Une autre approche pour réduire les dimensions est de les réunir en fonction de leur similarité. Li [2005] présente un modèle de classification pour les données binaires qui va réaliser simultanément une classification des objets et une classification des variables. Pour un nombre K de classes d'objets et un nombre C de classes de variables, l'auteur utilise les notations suivantes :

- $W_{n\times m}$: la matrice des données contenant n objets et m variables,
- P_k : l'ensemble des objets classés dans la classe d'objets k,
- p_k : la taille de la classe d'objets k,
- Q_c : l'ensemble des variables classées dans la classe de variables c,
- q_c : la taille de la classe de variables c,
- $A_{n\times K}$: une matrice dont chaque terme a_{ik} désigne si l'objet i appartient à la classe d'objets k,
- $B_{m\times C}$: une matrice dont chaque terme b_{jc} désigne si la variable j appartient à la classe de variables c,
- $X_{K\times C}$: une matrice représentant les liens entre les classes d'objets et de variables.

Les matrices A et B représentent ainsi les associations de chaque objet et variable à leur classe. Comme chacun n'est associé qu'à une seule classe, les matrices sont binaires et la somme de chaque ligne de chaque matrice vaut 1. Supposons que A et B sont deux matrices carrées identité (i.e. chaque objet et chaque variable est seul dans sa classe), alors nous avons l'égalité

suivante : $W = AWB^t$ où B^t est la transposée de B. De façon plus général, pour une classification A des objets donnés et une classification B des variables données, nous pouvons déterminer X et E tel que $W = AXB^t + E$, où E est une matrice permettant d'équilibrer les erreurs liées à X. En cherchant la matrice X minimisant l'erreur (c'est-à-dire la somme des carrés des valeurs de E la plus faible possible), nous obtenons la matrice optimale X telle que :

$$x_{kc} = \frac{1}{p_k q_c} \sum_{i \in P_k} \sum_{j \in Q_c} w_{ij} \quad (1)$$

où x_{kc} est la valeur de la k-ième ligne et c-ième colonne de X et w_{ij} la valeur de la i-ième ligne et j-ième colonne de W.

X peut donc être décrit comme la matrice des centroïdes pour le double problème de classification et représente les associations entre les deux classifications. De plus, à partir de X et d'une des classifications A ou B données, il est possible de modifier l'autre classification afin de minimiser l'erreur :

- Soit X et B fixe, la matrice A minimisant l'erreur est donnée par

$$a_{ik} = \begin{cases} 1 & \text{si } \sum_{c=1}^{C} \sum_{j \in Q_c} (w_{ij} - x_{kj})^2 < \sum_{c=1}^{C} \sum_{j \in Q_c} (w_{ij} - x_{lj})^2 \\ & \text{pour } l = 1, \ldots, K, l \neq k \\ 0 & \text{sinon.} \end{cases} \quad (2)$$

- Soit X et A fixe, la matrice B minimisant l'erreur est donnée par

$$b_{jc} = \begin{cases} 1 & \text{si } \sum_{k=1}^{K} \sum_{i \in P_k} (w_{ij} - x_{ic})^2 < \sum_{k=1}^{K} \sum_{i \in P_k} (w_{ij} - x_{il})^2 \\ & \text{pour } l = 1, \ldots, C, l \neq c \\ 0 & \text{sinon.} \end{cases} \quad (3)$$

À partir de la matrice des données W, du nombre de classes des objets K et du nombre de classes des variables C, l'auteur présente ainsi l'algorithme **General Clustering Procedure** (algorithme 12).

2.5.3 Itemsets et règles d'association

La génération d'itemsets et de règles d'association est une recherche et étude des régularités dans les jeux de données. Ces régularités peuvent permettre de détecter des liens et mécanismes pour mieux comprendre et appréhender les données, voir les simplifier. Dans cette sous-section nous présenterons dans un premier temps les itemsets et des méthodes pour les générer, puis dans un second temps, les règles d'associations et leurs générations à partir des itemsets.

Données : W, K, C
Résultat : A la matrice de classification des objets
B la matrice de classification des variables
Initialiser A et B
Calculer X avec l'équation (1)
tant que *Le critère d'arrêt n'est pas obtenu* **faire**
  Calculer le nouveau A à partir de l'équation (2)
  Calculer le nouveau B à partir de l'équation (3)
  Calculer X à partir de l'équation (1)
retourner(A,B)

Algorithme 12 : `General Clustering Procedure`

2.5.3.1 Détermination d'itemsets fréquents

La recherche d'itemsets fréquents est une recherche d'association d'objets fréquents. En déterminant ces associations d'objets, nous pouvons en extraire des *règles d'association*. En 1993, la recherche d'itemsets fréquents et des règles d'association est développée dans Agrawal et al. [1993] pour répondre à un besoin d'analyse du comportement des clients de supermarché face à l'achat des produits.

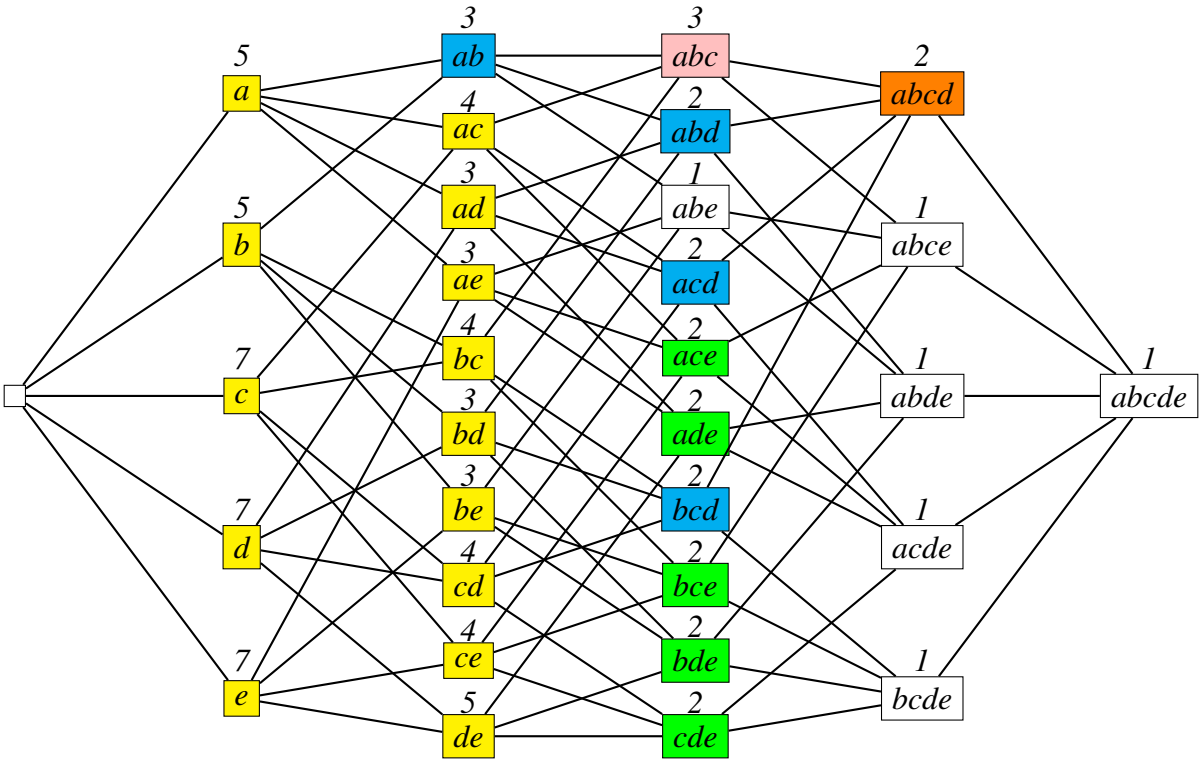
Exemple 14. *Un magasin étudie la consommation de 10 clients (numérotés de* 1 *à* 10*) sur 5 produits (numérotés de* a *à* e*). Le magasin récupère les informations, à partir des tickets de caisse par exemple, et entre les résultats dans le tableau suivant. La valeur* x_{ij} *vaut* 1 *si le client* i *a acheté le produit* j*,* 0 *sinon.*

Clients	*Produits*				
	a	*b*	*c*	*d*	*e*
1	*0*	*1*	*0*	*1*	*1*
2	*1*	*1*	*1*	*1*	*1*
3	*0*	*1*	*1*	*0*	*1*
4	*1*	*0*	*1*	*0*	*1*
5	*0*	*0*	*0*	*1*	*1*
6	*1*	*1*	*1*	*1*	*0*
7	*0*	*0*	*1*	*1*	*1*
8	*0*	*0*	*1*	*1*	*0*
9	*1*	*1*	*1*	*0*	*0*
10	*1*	*0*	*0*	*1*	*1*

Le magasin s'intéresse à trouver les produits (ou items) que les consommateurs ont tendance à acheter ensemble. Par exemple, seulement 30% des clients ont acheté le produit a *et le produit* b *ensemble tandis que 50% des clients ont acheté les produits* d *et* e *en même temps. Le magasin a donc peut être un intérêt à rapprocher le rayon des produits* d *de celui des produits* e*. De*

même, les clients achetant les produits a et b achètent également le produit c. Une promotion sur les produits a et b devraient donc augmenter les ventes du produit c.

Afin de lire plus facilement les itemsets (i.e. les associations d'items), nous pouvons les représenter sous la forme d'un graphe où chaque noeud est étiqueté par la fréquence d'apparition de l'itemset.



Pour une instance $I = \{T, S\}$ d'écrivant un ensemble S d'items et un ensemble T de transactions sur ces items, le *support* d'un itemset $X \subseteq S$ est sa fréquence d'apparition, c'est-à-dire le nombre de transactions contenant X. Ce nombre peut être indiqué sous sa forme absolue, ou relative (i.e. $\frac{\text{nombre de transaction contenant } X}{\text{nombre de transaction totale}}$).

Un itemset est dit fréquent si son support est supérieur à un seuil donné.

Un itemset A est un superset de l'itemset B si et seulement si $B \subset A$ c'est-à-dire que l'on retrouve tous les items de B dans A.

Parmi les itemsets fréquents, nous pouvons affiner notre recherche :

- Un itemset fréquent est fermé si aucun superset n'a de support identique (Pasquier et al. [1999]).
- Un itemset est dit maximal si aucun de ses supersets n'est fréquent (Bayardo Jr [1998]; Lin and Kedem [1998]).
- Un itemset A est dit générateur si $\forall B \subset A$, le support de B est strictement supérieur à celui de A (Bastide et al. [2000]).

Nous pouvons remarquer deux propriétés évidentes :

- Un itemset a toujours un support supérieur ou égal à ses supersets.
- Un itemset maximal est toujours fermé.

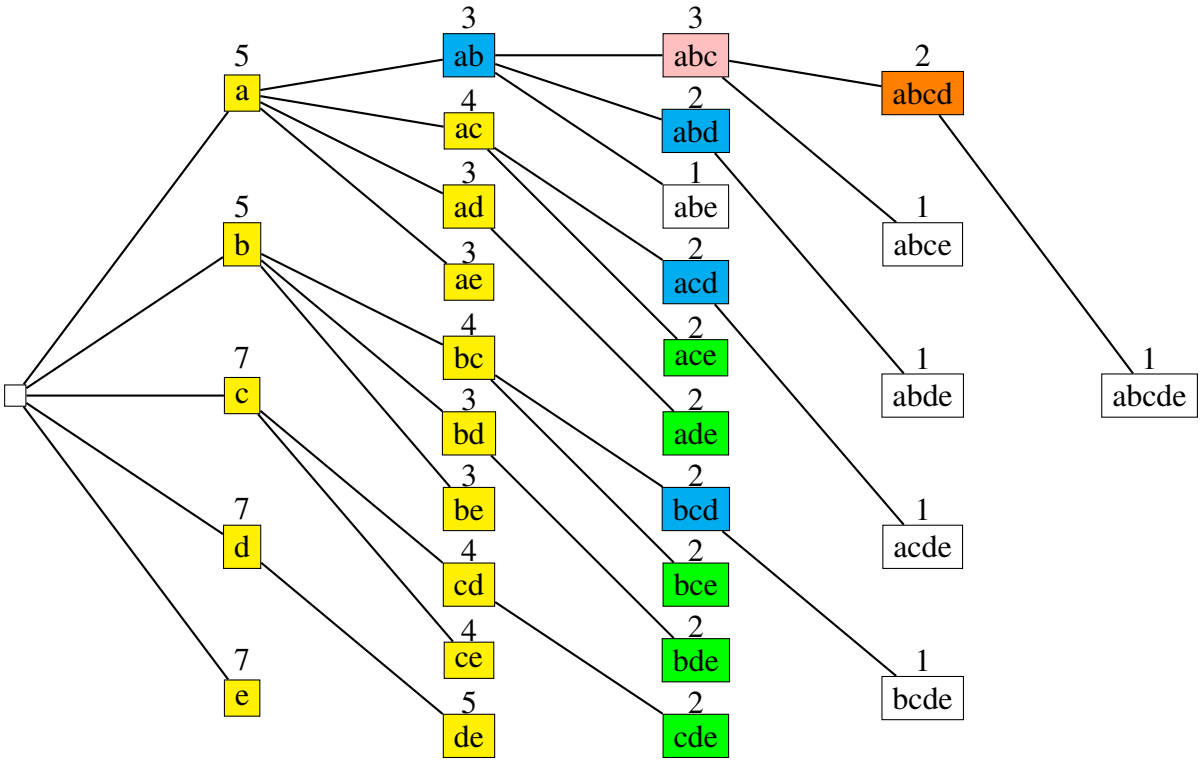
Dans notre exemple 14, nous avons mis plusieurs couleurs sur nos itemsets fréquents, par rapport à un seuil de 20% :

- Vert pour les itemsets maximaux et générateurs,
- Orange pour les itemsets maximaux non générateurs,
- Cyan pour les itemsets générateurs non fermés,
- Rose pour les itemsets fermés non générateur et non maximaux,
- Jaune pour les itemsets fermés et générateurs mais non maximaux.

Notons également que pour un seuil de 30%, l'itemset $\{a, b, c\}$ est maximal mais pas générateur.

Pour déterminer l'ensemble des itemsets fréquents, une première idée est simplement de parcourir l'arbre en largeur et de supprimer les itemsets non fréquents rencontrés ainsi que tous leurs supersets. En effet, comme un itemset a toujours un support supérieur à ses supersets, les supersets d'un itemset non fréquent ne le seront pas non plus.

Afin d'éviter la redondance, l'algorithme `Apriori` (Agrawal et al. [1994]) travaillera sur un arbre particulier fait sous la forme d'un diagramme de Hasse où les items sont ordonnés. Dans cet arbre les successeurs d'un itemset X seront les itemsets $Y = X \cup \{x\}$ où x est un item d'ordre supérieur à tous les items de X. Si nous reprenons l'exemple 14 et que nous considérons l'ordre des items tel que $a < b < c < d < e$, nous aurons un arbre comme suit :



Nous voyons ainsi que le graphe est un arbre, il n'y a qu'un seul chemin pour construire chaque itemset. Nous évitons ainsi la redondance. En évitant de générer un nœud non fréquent, nous ne générons aucun de ses supersets et ainsi, nous obtenons l'ensemble des itemsets fréquents. De plus, comme nous évitons la redondance, il est également possible de générer l'arbre via un parcours en largeur ou en profondeur (Kosters and Pijls [2003]).

Avec ce nouvel arbre, nous pouvons générer l'ensemble des itemsets fréquents plutôt rapidement. Toutefois, cette forme induit une complication pour déterminer quels sont les itemsets X maximaux, fermés ou générateurs, puisque ces définitions dépendent des supersets de ces itemsets (ou des itemsets tels que X soit un superset dans le cas où X est générateur). Or, nous avons retiré la plupart des relations d'inclusions entre les itemsets. Afin de déterminer les itemsets fréquents fermés, l'algorithme `LCM` (Uno et al. [2003]) va, pour chaque itemset, retourner au tableau de données et ne conserver que les transactions possédant cet itemset. Si chacune de ces transactions possèdent un autre item commun, alors l'itemset n'est pas fermé. L'algorithme `FP-Growth` (Grahne and Zhu [2003, 2004]), va lui, entre autre, utiliser un répertoire d'itemsets fermés déjà trouvés, et pour chaque itemset, il va vérifier qu'il n'existe pas de superset de même support dans le répertoire. En déterminant tous les itemsets fréquents fermés de taille k avant d'étudier ceux de taille $k-1$, l'algorithme déterminera tous les itemset fréquents fermés.

Les itemsets maximaux étant des itemsets fermés, les techniques pour les déterminer sont proches de celles pour déterminer les itemsets fermés (Borgelt [2012]).

2.5.3.2 Règle d'association

À partir des itemsets, il est possible de générer des règles d'association. Soit deux itemsets X et Y tel que $X \cap Y = \emptyset$, une règle d'association est une application de la forme $X \rightarrow Y$ suggérant une relation forte et causale de X vers Y. Soit la fonction $SUP(X)$ désignant le support de l'itemset X, on évalue la force d'une règle d'association à l'aide de deux indices :

- Son support $SUP(X \rightarrow Y) = SUP(X \cup Y)$ i.e. la proportion des transactions contenant à la fois X et Y,
- Sa confiance $CONF(X \rightarrow Y) = \frac{SUP(X \cup Y)}{SUP(X)}$ i.e. parmi les transactions contenant X, combien contiennent aussi Y.

Exemple 15. *Un magasin vend plusieurs produits que l'on notera dans un ensemble I avec :*

$$I = \{barbecue, charbon, parasol, table, chaises\}$$

Ce magasin a répertorié les transactions des clients dans le tableau ci-dessous :

Clients	*Produits*				
	barbecue	*charbon*	*parasol*	*table*	*chaises*
1	*0*	*1*	*0*	*1*	*1*
2	*1*	*1*	*1*	*1*	*1*
3	*1*	*1*	*1*	*0*	*1*
4	*1*	*0*	*1*	*0*	*1*
5	*0*	*1*	*0*	*1*	*1*
6	*1*	*1*	*0*	*1*	*1*
7	*0*	*0*	*1*	*1*	*1*
8	*0*	*1*	*1*	*1*	*1*
9	*1*	*1*	*1*	*0*	*0*
10	*1*	*1*	*0*	*0*	*1*

Nous nous demandons si les clients achetant un barbecue ont tendance à acheter également du charbon : $SUP(barbecue \rightarrow charbon) = 5/10 = 50\%$, il est donc fréquent que les clients achètent les deux. $CONF(barbecue \rightarrow charbon) = 5/6 = 83\%$, donc 83% des clients qui achètent un barbecue achètent également du charbon. Nous pouvons donc soupçonner un lien de cause à effet entre acheter un barbecue et du charbon. De plus, $CONF(charbon \rightarrow barbecue) = 5/8 = 62\%$ donc beaucoup moins de personne achetant du charbon achètent également un barbecue. Nous pouvons donc plus fortement soupçonner l'achat du barbecue être la cause de l'achat du charbon que l'inverse (bien évidement $SUP(charbon \rightarrow barbecue) = SUP(barbecue \rightarrow charbon)$).

Nous nous demandons alors si l'achat simultané d'un barbecue et d'une table entraîne l'achat d'un parasol et de chaises. $SUP(\{barbecue, table\} \rightarrow \{parasol, chaises\}) = 10\%$*, il est donc peu fréquent que les clients achètent l'ensemble de ces produits. Notre règle d'association n'est pas fréquente. Par contre,* $CONF(\{barbecue, table\} \rightarrow \{parasol, chaises\}) = 1/2 = 50\%$*, donc malgré le faible support, la confiance est relativement élevée. La règle d'association est donc plutôt pertinente mais peut être dû au hasard.*

Lorsque que l'ont recherche des règles d'associations, toutes ne sont pas intéressantes. En particulier, nous souhaitons ne conserver que celles ayant un support au dessus d'un seuil précis $minsup$ et une confiance au dessus d'un autre seuil précis $minconf$. Il est donc intéressant de commencer par générer les itemsets fréquents pour un seuil de $minsup$ avant de générer les règles d'associations.

Dans Agrawal et al. [1994] est présenté l'algorithme `genrules` qui consiste, une fois l'ensemble des itemsets fréquents $(X \cup Y)$ générés (avec un seuil de $minsup$), de générer chaque règle possible $X \rightarrow Y$ tel que $|Y| = 1$. Si cette règle possède une confiance supérieure à $minconf$, on la garde et on génère toutes les règles $(X \backslash X') \rightarrow (Y \cup X') = X_1 \rightarrow Y_1$ tel que $|X'| = 1$, puis on répète l'opération sur $X_1 \rightarrow Y_1$.
En d'autres termes, si une règle $X \rightarrow Y$ n'a pas une confiance suffisante, alors $\forall X' \subset X$, $(X \backslash X') \rightarrow (Y \cup X')$ n'aura pas de confiance suffisante non plus. L'algorithme génère donc toutes les règles possibles sauf celles-là.

Toutefois, l'algorithme n'est pas très efficace en terme de temps. C'est pourquoi les auteurs proposent l'algorithme `ap-genrules`, plus rapide, utilisant une démarche "inverse" en réécrivant la propriété précédente : soit une règle $X \rightarrow Y$ ayant une confiance suffisante, alors $\forall Y' \subset Y$, $(X \cup Y') \rightarrow (Y \backslash Y')$ a également une confiance suffisante.
L'algorithme va générer l'ensemble des règles $X_1 \rightarrow Y_1$ tel que $|Y_1| = 1$, supprimer les règles de confiance insuffisante, puis générer les règles $X_2 \rightarrow Y_2$ telles que $|Y_2| = 2$ si et seulement si $\forall Y' \subset Y_2$ avec $|Y'| = 1$, $(X_2 \cup Y') \rightarrow (Y_2 \backslash Y')$ est une règle qui n'a pas été supprimée à l'étape précédente (afin de respecter la propriété précédente).

L'ensemble de règles d'association peut être de taille très important et certaines règles sont redondantes. Afin que l'utilisateur puisse récupérer facilement des informations pertinentes, beaucoup d'études sont faites pour déterminer des "règles non-redondantes minimales".

Nous dirons qu'une règle d'association $X \rightarrow Y$ est minimale non-redondante si et seulement si il n'existe pas de règle $X' \rightarrow Y'$ différente de $X \rightarrow Y$ telle que :

- $SUP(X \rightarrow Y) = SUP(X' \rightarrow Y')$,
- $CONF(X \rightarrow Y) = CONF(X' \rightarrow Y')$,
- $X' \subseteq X$,

- $Y \subseteq Y'$.

Exemple 16. *Prenons 3 règles d'association d'une instance* $\{a\} \rightarrow \{c, d\}$, $\{a\} \rightarrow \{c\}$ *et* $\{a, b\} \rightarrow \{c, d\}$ *de sorte que chacune ait le même support et la même confiance.* $\{a\} \rightarrow \{c\}$ *et* $\{a, b\} \rightarrow \{c, d\}$ *sont redondantes car l'information qu'elles apportent est déjà apportée par* $\{a\} \rightarrow \{c, d\}$. *Elles ne sont pas minimales car* $\{a\} \rightarrow \{c, d\}$ *apporte plus d'information que chacune d'elles.*

Dans Bastide et al. [2000] a été proposé une caractérisation d'itemsets X générant des règles minimales non-redondantes $X \rightarrow Y$, appelées itemsets générateurs minimaux. Un itemset X est générateur d'un de ses supersets fermés X' s'il est générateur et que $SUP(X') = SUP(X)$ (i.e. X est générateur et non fermé).

Dans Boudane et al. [2016] est présenté une approche pour la fouille des règles d'association via la satisfiabilité propositionnelle (SAT). Soit une formule booléenne sous sa forme normale conjonctive (conjonction de disjonction de littéraux), le problème SAT consiste à décider si la formule peut être satisfaite, c'est à dire qu'il existe une combinaison de littéraux pour laquelle la formule booléenne vaut 1. Par exemple, la formule $a \wedge \neg a$ ne peut pas être satisfaite (où $\neg a$ représente la négation de a, i.e. $a \in \{0, 1\}$ et $\neg a = 1 - a$).

Puisque nous allons manipuler des formules booléennes, nous devons définir les connecteurs logiques classiques en algèbre de Boole :

- $\wedge$ est la conjonction : $[0, 0, 1, 1] \wedge [0, 1, 0, 1] = [0, 0, 0, 1]$,
- $\vee$ est la disjonction : $[0, 0, 1, 1] \vee [0, 1, 0, 1] = [0, 1, 1, 1]$,
- $\oplus$ est la disjonction exclusive (Xor) : $[0, 0, 1, 1] \oplus [0, 1, 0, 1] = [0, 1, 1, 0]$,
- $\rightarrow$ est l'opérateur logique de l'implication : $[0, 0, 1, 1] \rightarrow [0, 1, 0, 1] = [1, 1, 0, 1]$
- $\leftrightarrow$ (ou $\odot$) est l'opérateur logique de l'équivalence (XNor) : $[0, 0, 1, 1] \leftrightarrow [0, 1, 0, 1] = [1, 0, 0, 1]$,
- $\neg$ est la négation logique : $\neg[0, 0, 1, 1] = [1, 1, 0, 0]$

Soit ω l'ensemble des items, $\mathcal{D}$ l'ensemble des transactions et pour $i \in \mathcal{D}$, I_i l'ensemble des items présent dans la transaction i. L'encodage SAT présenté dans l'article consiste en un ensemble de contraintes permettant :

- De déterminer que les itemsets X et Y de la règle $X \rightarrow Y$ ne sont pas vides grâce à des variables booléennes x_a (resp. y_a) représentant la présence/absence de chaque item a dans l'itemset X (resp. Y) :

$$(1) \quad (\bigvee_{a \in \omega} x_a) \wedge (\bigvee_{a \in \omega} y_a)$$

- De déterminer que $X \cap Y = \emptyset$:

$$(2) \quad \bigwedge_{a \in \omega} (\neg x_a \vee \neg y_a)$$

- De représenter la couverture de l'itemset X (resp. $X \cup Y$) grâce aux variables booléennes p_i (resp. q_i), i.e. $p_i = 1$ (resp. $q_i = 1$) si tous les items $a \in X$ (resp. $a \in X \cup Y$) sont présent dans la transaction i :

$$(3) \quad \bigwedge_{i \in \mathcal{D}} \neg p_i \leftrightarrow \bigvee_{a \in \omega \setminus I_i} x_a$$

$$(4) \quad \bigwedge_{i \in \mathcal{D}} \neg q_i \leftrightarrow \neg p_i \vee (\bigvee_{a \in \omega \setminus I_i} y_a)$$

- De vérifier que le support et la confiance sont suffisant :

$$(5) \quad \sum_{i \in \mathcal{D}} q_i \geqslant |\mathcal{D}| \times minsup$$

$$(6) \quad \frac{\sum_{i \in \mathcal{D}} q_i}{\sum_{i \in \mathcal{D}} p_i} \geqslant minconf$$

- De vérifier si $X \cup Y$ est un itemset fermé (facultatif) :

$$(7) \quad \bigwedge_{a \in \omega} ((\bigwedge_{i \in \mathcal{D}} q_i \rightarrow a \in I_i) \rightarrow x_a \vee y_a)$$

Boudane et al. [2017] propose une extension afin de déterminer uniquement les règles d'associations minimales non-redondantes en rajoutant une contrainte permettant de déterminer si $\forall a \in X$, on a bien $SUP(X) < SUP(X \backslash \{a\})$ (i.e. X est générateur) :

$$(8) \quad (\bigwedge_{a \in \omega} x_a \rightarrow \bigvee i \in \mathcal{D}, a \notin I_i (\bigwedge_{b \notin I_i \cup \{a\}} \neg x_b)) \vee (\sum_{b \in \omega} x_b = 1))$$

Une proposition est formulée dans l'article : La règle $X \rightarrow Y$, est une règle minimale non redondante si et seulement si $X \cup Y$ est fermé et X est générateur ($|X| = 1$ ou $\forall a \in X$, $SUP(X) < SUP(X \backslash \{a\})$. De ce fait, en utilisant cette contrainte (8) et la contrainte facultative (7), nous nous assurons que $X \rightarrow Y$ est bien une règle minimale non redondante. Ainsi, l'encodage $(1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6) \wedge (7) \wedge (8)$ est valide.

3

Résolution du problème de caractérisation multiple

Dans ce chapitre, nous nous intéresserons au calcul des solutions du problème de caractérisation multiple (PCM) présenté dans la section 2.3, et plus particulièrement à la recherche des solutions de plus petites tailles. Ce chapitre se divisera en 5 sections :

- Reformulation du PCM afin de définir nos méthodes de résolution du problème.
- Présentation de nos méthodes pour résoudre le *NonDom-PCM*, c'est à dire le problème de calcul de l'ensemble des solutions non-dominées.
- Présentation de nos méthodes pour résoudre le *MinAll-PCM*, c'est à dire le problème de calcul de l'ensemble des solutions minimales.
- Calcul de patterns en analyse logique de données (LAD) grâce aux méthodes de résolution du PCM (nous rappellerons les principaux concepts de LAD dans la section 3.4).
- Présentation d'un nouvel axe d'optimisation des solutions du PCM et calcul de l'ensemble de ces solutions.

3.0 Rappels sur le problème de caractérisation multiple

Rappelons les principales définitions de la section 2.3 :

- **Définition 13 :** Instance du problème de caractérisation multiple (PCM)
  Une instance du PCM est un quadruplet $(\Omega, \mathcal{A}, D, G)$ défini par un ensemble d'observations Ω dont les éléments sont exprimés sur un ensemble d'attributs A, et sont représentés par une matrice de données booléennes $D_{|\Omega|\times|\mathcal{A}|}$ et une fonction $G : \Omega \to \mathcal{G}$, telle que $G(o)$ est le groupe auquel appartient l'observation $o \in \Omega$.

- **Définition 16 :** Solution du PCM
  Soit une instance $(\Omega, \mathcal{A}, D, G)$, un sous-ensemble d'attributs $S \subseteq \mathcal{A}$ est une solution si et seulement si $\forall (o, o') \in \Omega^2, G(o) \neq G(o') \Rightarrow D^S[o, .] \neq D^S[o', .]$.

Exemple 17. *Reprenons l'exemple 8 :*
Soit l'ensemble $\Omega = \{1, 2, 3, 4, 5, 6\}$ *représentant 6 meetings, l'ensemble* $\mathcal{A} = \{Rythme, Costume, Cheveux, Lunettes, Montre\}$ *représentant les 5 attributs étudiés et 4 groupes* $\{Peu\ satisfait, Passable, Bon, Excellent\}$ *distribués par la fonction de groupes G, le tableau suivant présente la matrice de données D ainsi que les résultats de la fonction G :*

Meetings	*Attributs* / *Groupes*	*Rythme*	*Costume*	*Cheveux*	*Lunettes*	*Montre*
1	*Peu satisfaisant*	*0*	*1*	*1*	*0*	*0*
2	*Passable*	*1*	*1*	*1*	*1*	*1*
3		*0*	*1*	*0*	*1*	*1*
4	*Bon*	*0*	*0*	*0*	*1*	*0*
5	*Excellent*	*1*	*0*	*1*	*0*	*0*
6		*1*	*1*	*0*	*0*	*1*

Une solution du PCM, comme définie dans la définition 16, sera par exemple le sous-ensemble d'attributs $\{Rythme, Costume, Cheveux\}$ *car il n'existe pas deux meetings appartenant à deux groupes différents qui soient identiques sur ces trois attributs.*

Ces solutions sont nombreuses et souvent redondantes. Nous avons donc identifié deux catégories de solutions intéressantes :

- Les solutions non-dominées (définition 17) : Une solution S est non-dominée si et seulement si $\forall s \in S, \exists (o, o') \in \Omega^2$ tel que $G(o) \neq G(o')$ et $D^{S\setminus\{s\}}[o, .] = D^{S\setminus\{s\}}[o', .]$ (i.e. $S\setminus\{s\}$ n'est pas une solution).

  Nous pouvons visualiser chacune de ces solutions comme un ensemble suffisant pour discriminer les groupes d'observations. Ce sont des solutions "non-redondantes".

- Les solutions minimales (définitions 18) : Une solution S est minimale si et seulement si $\nexists S'$ avec $|S'| < |S|$ tel que S' est une solution. Ce sont les solutions les plus petites.

Soit une instance $\mathcal{I}$ du PCM, nous avons ainsi identifié plusieurs catégories de problèmes, en particulier :

- *Sol-PCM* : calcul d'une solution du PCM pour l'instance $\mathcal{I}$,
- *NonDom-PCM* : calcul de l'ensemble des solutions non-dominées du PCM pour $\mathcal{I}$,
- *MinAll-PCM* : calcul de l'ensemble des solutions minimales du PCM pour $\mathcal{I}$.

3.1 Reformulation du PCM

Comme vu dans la sous-section 2.3.4.2, le problème de caractérisation multiple minimale peut être formulé comme un programme linéaire en nombre entier, et plus particulièrement comme un problème de couverture minimale (par ensembles). Nous avons également vu dans la section 2.4 que le problème de couverture par ensembles est équivalent au problème de génération d'ensembles intersectants. Le PCM peut donc être formulé comme un problème de génération d'ensembles intersectants. Une solution non-dominée correspond, avec cette approche, à un ensemble intersectant minimal.

Le programme linéaire en nombre entier permettant de déterminer les solutions minimales du PCM, présenté dans la sous-section 2.3.4.2, est de la forme :

$$
\begin{aligned}
& min : \sum_{i=1}^{|\mathcal{A}|} y_i \\
& s.t. : \\
& C \,.\, Y^t \geqslant \mathbb{1}^t \\
& Y \in \{0,1\}^{|\mathcal{A}|}, Y = [y_1, ..., y_{|\mathcal{A}|}]
\end{aligned}
$$

où $\mathbb{1}$ représente un vecteur composé de 1 et Y^t est la transposée du vecteur Y. La matrice C, que nous avons nommée matrice de contraintes, est la seule variable représentant l'instance $(\Omega, \mathcal{A}, D, G)$ dans le programme linéaire et peut être obtenue à partir de la matrice de données D et de la fonction de groupes G.

La matrice de contraintes C est donc une entrée importante pour la résolution du PCM. Rapellons qu'elle est construite comme suit : soit Θ l'ensemble des couples d'observations $(o, o') \in \Omega^2$ appartenant à des groupes différents,

- Chaque ligne est numérotée par un couple d'observations $(o, o') \in \Theta \subseteq \Omega^2$,

- Chaque colonne représente un attribut,
- $C[(o, o'), a] = 1$ si $D[o, a] \neq D[o', a]$, $C[(o, o'), a] = 0$ sinon,
- On note $C[(o, o'), .] \in \mathbb{B}^n$ le vecteur booléen (également appelé contrainte) représentant les différences entre les observations o et o' sur chaque attribut.

Rappelons les notations des connecteurs logiques en algèbre de Boole :

- $\wedge$ est la conjonction : $[0, 0, 1, 1] \wedge [0, 1, 0, 1] = [0, 0, 0, 1]$,
- $\vee$ est la disjonction : $[0, 0, 1, 1] \vee [0, 1, 0, 1] = [0, 1, 1, 1]$,
- $\oplus$ est la disjonction exclusive (Xor) : $[0, 0, 1, 1] \oplus [0, 1, 0, 1] = [0, 1, 1, 0]$,
- $\odot$ (ou $\leftrightarrow$) est l'opérateur logique de l'équivalence (XNor) : $[0, 0, 1, 1] \odot [0, 1, 0, 1] = [1, 0, 0, 1]$,
- $\neg$ est la négation logique : $\neg[0, 0, 1, 1] = [1, 1, 0, 0]$.

Tout comme la matrice de données D, nous définissons une projection de la matrice de contraintes C sur un sous-ensemble d'attributs $A \in \mathcal{A}$.

Définition 21. *Soit $A \subset \mathcal{A}$, C^A est la matrice de contraintes réduite au sous ensemble d'attributs A.*

La matrice C^A peut également être obtenue à partir de D^A de la même façon que C est obtenue à partir de D.

Soit $(o, o') \in \Theta$, l'ensemble $A_{(o,o')} \subseteq \mathcal{A}$ des attributs tel que $A_{(o,o')} = \{a \in \mathcal{A} | C[(o, o'), a] = 1\}$ représente donc l'ensemble des attributs différenciant les observations o et o'. S'il existe $(k, k') \in \Theta$ tel que $A_{(k,k')} \subseteq A_{(o,o')}$, c'est à dire tous les attributs différenciant k et k' différencient également o et o', nous dirons que le vecteur $C[(o, o'), .]$ est redondant.
Plus formellement nous avons :

Définition 22. *Un vecteur $C[(o, o'), .]$ est redondant s'il existe un couple $(k, k') \in \Theta$ tel que $(C[(o, o'), .] \wedge C[(k, k'), .]) = C[(k, k'), .]$.*

En utilisant cette notion de redondance, C peut être réduite :

Définition 23. *Soit $\theta \subset \Theta$, la matrice de contraintes projetée sur θ, notée C_θ, est la matrice composée des contraintes $C[(o, o'), .]$ $\forall (o, o') \in \theta$.*

Définition 24. *La matrice de contraintes réduite $C_\succ$ est la matrice de contraintes projetée sur l'ensemble $\Theta' \subseteq \Theta$ où Θ' est l'ensemble de toutes les paires $(o, o') \in \Theta$ tel que $C[(o, o'), .]$ n'est pas redondant.*

Bien évidemment, $C_\succ$ est de taille $|\Theta'| \times |\mathcal{A}|$.

Exemple 18. *Reprenons la matrice de contraintes C présentée dans l'exemple 11 :*

Attributs / *Couples*	*Rythme (r)*	*Costume (c)*	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(1,2)	*1*	*0*	*0*	*1*	*1*
(1,3)	*0*	*0*	*1*	*1*	*1*
(1,4)	*0*	*1*	*1*	*1*	*0*
(1,5)	*1*	*1*	*0*	*0*	*0*
(1,6)	*1*	*0*	*1*	*0*	*1*
(2,4)	*1*	*1*	*1*	*0*	*1*
(2,5)	*0*	*1*	*0*	*1*	*1*
(2,6)	*0*	*0*	*1*	*1*	*0*
(3,4)	*0*	*1*	*0*	*0*	*1*
(3,5)	*1*	*1*	*1*	*1*	*1*
(3,6)	*1*	*0*	*0*	*1*	*0*
(4,5)	*1*	*0*	*1*	*1*	*0*
(4,6)	*1*	*1*	*0*	*1*	*1*

On a $C[(3,5),.] = [1,1,1,1,1]$ *et* $C[(1,2),.] = [1,0,0,1,1]$. *On a donc* $C[(3,5),.] \wedge C[(1,2),.] = [1,0,0,1,1] = C[(1,2),.]$. $C[(3,5),.]$ *est donc une contrainte redondante. De la même façon, les couples* $(1,2)$, $(1,3)$, $(1,4)$, $(2,4)$, $(2,5)$, $(4,5)$ *et* $(4,6)$ *forment des contraintes redondantes.*

On a donc $C_\succ$ *la matrice de contraintes réduite suivante :*

Attributs / *Couples*	*Rythme (r)*	*Costume (c)*	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(1,5)	*1*	*1*	*0*	*0*	*0*
(1,6)	*1*	*0*	*1*	*0*	*1*
(2,6)	*0*	*0*	*1*	*1*	*0*
(3,4)	*0*	*1*	*0*	*0*	*1*
(3,6)	*1*	*0*	*0*	*1*	*0*

Remarque 5. *On a* $(C^A)_\succ \neq (C_\succ)^A$. *Par convention, nous noterons* $C^A_\succ$ *la matrice* $(C^A)_\succ$, *c'est à dire la matrice projetée sur laquelle on retire les vecteurs redondants.*

Définition 25. *Soit un ensemble d'attributs* $A \in \mathcal{A}$ *et une contrainte* $C[(o,o'),.]$, *nous dirons que* A *satisfait la contrainte* $C[(o,o'),.]$ *si* $\exists a \in A$ *tel que* $C[(o,o'),a] = 1$, *i.e.* $C^A[(o,o'),.] \neq \mathbb{0}$ *(où* $\mathbb{0}$ *est le vecteur contenant uniquement des valeurs égales à* 0*).*

Exemple 19. *Reprenons la matrice de contraintes réduite $C_{\succ}$ de l'exemple 18 :*

Attributs / *Couples*	*Rythme (r)*	*Costume (c)*	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(1,5)	*1*	*1*	*0*	*0*	*0*
(1,6)	*1*	*0*	*1*	*0*	*1*
(2,6)	*0*	*0*	*1*	*1*	*0*
(3,4)	*0*	*1*	*0*	*0*	*1*
(3,6)	*1*	*0*	*0*	*1*	*0*

Soit $A = \{r, c\}$, A satisfait les contraintes induites par les couples $(1,5)$, $(1,6)$, $(3,4)$ et $(3,6)$ car $C[(1,5),r] = C[(1,6),r] = C[(3,6),r] = 1$ et $C[(1,5),c] = C[(3,4),c] = 1$.

Seule la contrainte $C[(2,6),.]$ n'est pas satisfaite par A car $C^A[(2,6),.] = [0,0]$.

Remarque 6. *Si un ensemble d'attributs $A \in \mathcal{A}$ satisfait toutes les contraintes de $C_{\succ}$, alors il satisfera également toutes les contraintes redondantes.*

Résoudre une instance du PCM consiste à déterminer un sous-ensemble $S \in \mathcal{A}$ tel que tous les couples d'observations issus de groupes différents (i.e. $(o, o') \in \Theta$) diffèrent au moins sur un attribut $s \in S$. Autrement dit, S est une solution s'il satisfait l'ensemble des contraintes $C[(o, o'), .]\ \forall (o, o') \in \Theta$. On a donc :

Propriété 5. *Un ensemble d'attributs $S \subseteq \mathcal{A}$ est une solution du PCM si et seulement si $\nexists (o, o') \in \Theta$ tel que $C^S[(o, o'), .] = \mathbb{0}$.*

Nous pouvons travailler sur $C_{\succ}$ plutôt que sur C pour éviter les contraintes redondantes. Notons que nous pouvons rapidement vérifier l'existence d'au moins une solution :

Propriété 6. *Une instance $(\Omega, \mathcal{A}, D, G)$ est réalisable si et seulement si la matrice de contraintes obtenue grâce à D et G est telle que $\nexists (o, o') \in \Theta$ tel que $C[(o, o'), .] = \mathbb{0}$.*

Si $\exists (o, o') \in \Theta$ tel que $C[(o, o'), .] = \mathbb{0}$, alors nous avons $G(o) \neq G(o')$ et $D[o, .] = D[o', .]$. Or nous avons vu dans la sous-section 2.3.2 qu'il s'agit d'une condition nécessaire et suffisante à la non-existence d'une solution.

3.2 Résolution du *NonDom-PCM*

Nous nous intéressons ici au calcul de solutions non-dominées, et en particulier, à la génération de l'ensemble de ces solutions non-dominées. Dans un premier temps, nous présenterons des méthodes gloutonnes pour générer une solution du PCM. Dans un second temps, nous présenterons des méthodes de résolution dites "contrainte par contrainte", c'est à dire où l'ensemble de solutions est construit en satisfaisant un sous-ensemble de contraintes de plus en plus important. Enfin, dans un troisième temps, nous présenterons des méthodes de résolution dites "solution par solution" où les solutions de l'instance entière seront générées une à une.

3.2.1 Approches gloutonnes

Nous venons de voir avec la propriété 5 qu'un ensemble d'attributs est une solution du PCM s'il satisfait toutes les contraintes. De cette propriété, il est facile d'imaginer un algorithme déterminant une solution (algorithme 13).

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Résultat : S une solution
$S = \{\emptyset\}$
Créer la matrice de contraintes C à partir de D et G
pour tous les $(o, o') \in \Theta$ **faire**
  si $C^S[(o, o'), .] = \mathbb{0}$ **alors**
    Choisir a tel que $C[(o, o'), a] = 1$
    $S = S \cup \{a\}$
retourner(S)

Algorithme 13 : Calcul d'une solution du PCM

Cet algorithme va tout simplement ajouter un attribut pour satisfaire chaque contrainte non encore satisfaite par un ensemble courant d'attributs S.

Notons toutefois que la solution obtenue par un tel algorithme est d'une taille fortement dépendante du choix de $a \in \mathcal{A}$ à chaque étape. L'algorithme présenté dans Chvatal [1979] (introduit dans la sous-section 2.3.4.3) est un algorithme glouton basé sur le même principe, mais choisissant à chaque étape l'attribut $a \in \mathcal{A}$ permettant de satisfaire le plus de contraintes non satisfaites (voir l'algorithme 14).

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Résultat : S une solution
$S = \{\emptyset\}$
Créer la matrice de contraintes C à partir de D et G
pour tous les $(o, o') \in \Theta$ **faire**
  si $C^S[(o, o'), .] = \mathbb{0}$ **alors**
    $\theta = \{(o, o') \in \Theta | C^S[(o, o'), a] = \mathbb{0}\}$
    $a_{best} = max_{a \in \mathcal{A} \setminus S}(\sum_{(o,o') \in \theta} C[(o, o'), a])$
    $S = S \cup \{a_{best}\}$
retourner(S)

Algorithme 14 : Algorithme glouton de Chvatal

3.2.2 Approches contrainte par contrainte

Les approches contrainte par contrainte désignent ici des méthodes de génération de solutions satisfaisant un sous-ensemble de contraintes de plus en plus important jusqu'à la satisfaction de toutes les contraintes. Nous verrons dans cette sous-section comment générer l'ensemble

des solutions non-dominées mais également comment simplifier l'instance et ne générer qu'un sous-ensemble de ces solutions afin d'obtenir un gain en temps de calcul.

3.2.2.1 Calcul des solutions non-dominées

Si les algorithmes gloutons présentés précédement vont rapidement donner une solution du PCM, rien ne garantit la non-dominance ni la minimalité des solutions construites.

Rappel : une solution $S \subseteq \mathcal{A}$ est non dominée s'il n'existe pas de solution $S' \subseteq \mathcal{A}$ telle que $S' \subset S$.

En partant de cette approche, nous pouvons imaginer avoir besoin de déterminer l'ensemble des solutions non-dominées pour garantir la non-dominance d'une solution : le calcul de toutes les solutions non-dominées serait tout autant efficace que le calcul d'une seule solution non-dominée.

Nous proposons ainsi un algorithme permettant de déterminer toutes les solutions non dominées. Soit $\Theta = \{\theta_1, \theta_2, \cdots, \theta_{|\Theta|}\}$, l'idée est de construire progressivement à l'étape j l'ensemble des solutions non dominées du sous-ensemble de contraintes $\Theta_j = \{\theta_1, \theta_2, \cdots, \theta_j\}$ ($j < |\Theta|$), puis d'incrémenter chacune des solutions pour satisfaire la contrainte θ_{j+1} et ainsi obtenir l'ensemble des solutions non-dominées du sous-ensemble de contraintes Θ_{j+1} :

- Étape 1 : Pour chaque solution non dominée $S_j \subseteq \mathcal{A}$ satisfaisant $\Theta_j \subset \Theta$, et pour chaque attribut $a \in \mathcal{A}$ tel que $C[\theta_{j+1}, a] = 1$, l'ensemble des combinaisons $S_j \cup \{a\}$ sera une solution satisfaisant Θ_{j+1}.
- Étape 2 : Comparer chacune de ces nouvelles solutions entre elles pour déterminer celles qui sont non-dominées.

Cette approche correspond à l'algorithme de Berge, vu dans la sous-section 2.4.2. Toutefois nous pouvons proposer une amélioration. En effet, comparer à chaque étape les solutions intermédiaires pour ne conserver que les solutions non-dominées est très coûteux. L'algorithme que nous proposons permet d'éviter cette étape grâce un pré-traitement qui ne créera que des solutions non-dominées.

Idée principale : séparer les solutions satisfaisant $\Theta_j \subset \Theta$ en deux ensembles : celles qui satisfont θ_{j+1} et celles qui ne satisfont pas cette contrainte. Puis, utiliser la propriété suivante afin de ne générer que des solutions non-dominées de Θ_{j+1}.

Propriété 7. *Soit $S_1 \subseteq \mathcal{A}$ et $S_2 \subseteq \mathcal{A}$ deux solutions non-dominées distinctes satisfaisant $\Theta_j \subset \Theta$ et leurs extensions respectives $S'_1 = S_1 \cup \{a\}$ et $S'_2 = S_2 \cup \{a'\}$ ($a, a' \in \mathcal{A}$) satisfaisant Θ_{j+1}, avec S'_1 non-dominée. On a : $S'_1 \subset S'_2 \Rightarrow S'_1 = S_1$, $S'_2 \neq S_2$ et $S_1 \backslash \{a'\} \subset S_2$.*

Autrement dit, parmis les solutions non-dominées satisfaisant Θ_j, celles satisfaisant θ_{j+1} ne peuvent pas être dominées et sont les seules pouvant dominer les solutions satisfaisant Θ_{j+1} à condition qu'elles ne possèdent seulement qu'un attribut de moins.

Preuve 1. *Soit $S_1 \subseteq \mathcal{A}$ et $S_2 \subseteq \mathcal{A}$ deux solutions non-dominées distinctes de $\Theta_j \subset \Theta$, on a donc $S_1 \nsubseteq S_2 \Leftrightarrow \exists a \in \mathcal{A}$ tel que $a \in S_1$ et $a \notin S_2$.*

- *$S_1 \cup \{a'\} \subset S_2 \cup \{a''\}$ implique que $a'' = a$, sinon $a \in S_1 \cup \{a'\}$ et $a \notin S_2 \cup \{a''\} \Rightarrow S_1 \cup \{a'\} \nsubseteq S_2 \cup \{a''\}$. On a donc $a \notin S_2$, d'où $S_2 \cup \{a\} \neq S_2$.*
- *$S'_1 = S_1 \cup \{a'\}$ est non-dominée. On a $S'_2 = S_2 \cup \{a\}$ donc $C[\theta_{j+1}, a] = 1$. Comme $a \in S_1$, S_1 est une solution non-dominée satisfaisant Θ_{j+1}. Si $a' \notin S_1$, on a $S_1 \subset S'_1 \Rightarrow S'_1$ est dominée. Donc $a' \in S_1 \Rightarrow S'_1 = S_1$.*
- *$S_1 \subset S_2 \cup \{a\} \Rightarrow S_1 \cap (S_2 \cup \{a\}) = S_1$. Comme $a \notin S_2$, on a donc $S_1 \cap S_2 = S_1 \backslash \{a\} \cap S_2 = S_1 \backslash \{a\} \Rightarrow S_1 \backslash \{a\} \subset S_2$.*

L'algorithme `NDS` (algorithme 15) utilise ce principe, en séparant les solutions de Θ_j satisfaisant θ_{j+1} dans un ensemble ND_{j+1} des solutions à incrémenter dans l'ensemble ES_{j+1}.

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Résultat : Sol l'ensemble de toutes les solutions non-dominées
$Sol = \{\emptyset\}$
Créer la matrice de contraintes $C_\succ$ à partir de D et G
pour $i = 1$ *à* $|\Theta|$ **faire**
  //Construire un sous-ensemble de solutions ND_i
  $ND_i = \emptyset$
  pour tous les $j \in \mathcal{A}$ *tel que* $C[\theta_i, j] = 1$ **faire**
    pour tous les $S \in Sol \backslash ND_i$ **faire**
      si $j \in S$ **alors**
        $ND_i = ND_i \cup \{S\}$
  //Construire un sous-ensemble de solutions ES_i
  $ES_i = \emptyset$
  pour tous les $j \in \mathcal{A}$ *tel que* $C[\theta_i, j] = 1$ **faire**
    pour tous les $S \in Sol \backslash ND_i$ **faire**
      si $\nexists S' \in ND_i$ *tel que* $S' \subset S \cup \{j\}$ **alors**
        $ES_i = ES_i \cup \{S \cup \{j\}\}$
  $Sol = ND_i \cup ES_i$
retourner(Sol)

Algorithme 15 : Calcul de l'ensemble des solutions non-dominées (`NDS`)

Si l'algorithme retourne toutes les solutions non-dominées, il reste relativement coûteux en temps. Il est toutefois possible de générer moins de solutions afin de gagner un peu de temps

grâce à deux méthodes : réduire l'instance et ne déterminer que les solutions de taille inférieure à une borne.

3.2.2.2 Réduction de l'instance

Considérons une instance $(\Omega, \mathcal{A}, D, G)$ avec $a_1, a_2 \in \mathcal{A}$ tel que $D[., a_1] = D[., a_2]$, alors les attributs a_1 et a_2 apportent la même information. Il est donc possible de réduire l'instance à $(\Omega, \mathcal{A}\backslash\{a_2\}, D^{\mathcal{A}\backslash\{a_2\}}, G)$ ou $(\Omega, \mathcal{A}\backslash\{a_1\}, D^{\mathcal{A}\backslash\{a_1\}}, G)$ (i.e. supprimer l'un des attributs).

Les solutions de l'instance $(\Omega, \mathcal{A}\backslash\{a_2\}, D^{\mathcal{A}\backslash\{a_2\}}, G)$ seront un sous-ensemble des solutions de l'instance $(\Omega, \mathcal{A}, D, G)$ et toutes les solutions contenant a_1 peuvent être transformées en remplaçant a_1 par a_2 pour obtenir les solutions manquantes de l'instance $(\Omega, \mathcal{A}, D, G)$ (Chhel et al. [2013]).

Introduisons maintenant le concept de dominance entre attributs. Soit $a, b \in \mathcal{A}$, on dit que a domine b, dénoté $a \succ b$, si et seulement si $\forall S \subset \mathcal{A}\backslash\{b\}$, si $S \cup \{b\}$ est une solution non-dominée, alors $S \cup \{a\}$ l'est aussi. a apporte donc plus d'information que b dès lors qu'il permet de distinguer au moins les même couples d'observations de Θ que b. Reformulé en considérant la matrice de contraintes, nous avons :

Proposition 1. *Soit* $(a, b) \in \mathcal{A}^2$, $a \succ b \Leftrightarrow (C[., a] \vee C[., b]) = C[., a]$.

Bien évidement, soit $\mathcal{E} \subset \mathcal{A}$ l'ensemble des attributs dominés et $D' = D^{\mathcal{A}\backslash\mathcal{E}}$, l'instance $(\Omega, \mathcal{A}\backslash\mathcal{E}, D', G)$ possède au moins une solution minimale du PCM identique à l'instance $(\Omega, \mathcal{A}, D, G)$.

3.2.2.3 Borne sur la taille des solutions

Avec un tel algorithme, il est possible de ne construire que les solutions de taille inférieure à une borne donnée B. Pour cela, il suffit, à l'étape i, de n'ajouter les solutions $S \cup \{j\}$ que si $|S| < B$. Nous avons donc l'algorithme `NDS-B` (algorithme 16) qui nous donne l'ensemble des solutions de taille inférieure ou égale à la borne B.

3.2.3 Approches solution par solution

Les approches solution par solution désignent ici des méthodes de générations de solutions de plus en plus variées jusqu'à la détermination de toutes les solutions. Nous verrons dans cette sous-section comment vérifier rapidemment qu'une solution est non-dominée, nous présenterons un algorithme glouton générant une solution non-dominée du PCM et enfin une méthode de génération de l'ensemble des solutions non-dominées.

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Une borne B
Résultat : Sol un ensemble de toutes les solutions non-dominées de taille inférieure à B
$Sol = \{\emptyset\}$
Créer la matrice de contraintes $C_{\succ}$ à partir de D et G
pour $i = 1$ *à* $|\Theta|$ **faire**
  //Construire un sous-ensemble de solutions ND_i
  $ND_i = \emptyset$
  pour tous les $j \in \mathcal{A}$ *tel que* $C[\theta_i, j] = 1$ **faire**
    pour tous les $S \in Sol \backslash ND_i$ **faire**
      si $j \in S$ **alors**
        $ND_i = ND_i \cup \{S\}$
  //Construire un sous-ensemble de solutions ES_i
  $ES_i = \emptyset$
  pour tous les $j \in \mathcal{A}$ *tel que* $C[\theta_i, j] = 1$ **faire**
    pour tous les $S \in Sol \backslash ND_i$ *tel que* $|S| < B$ **faire**
      si $\nexists S' \in ND_i$ *tel que* $S' \subset S \cup \{j\}$ **alors**
        $ES_i = ES_i \cup \{S \cup \{j\}\}$
  $Sol = ND_i \cup ES_i$
retourner(Sol)

Algorithme 16 : Calcul des solutions non-dominées bornées (`NDS-B`)

3.2.3.1 Vérification d'une solution non-dominée

Pour l'algorithme `NDS`, nous avons considéré qu'il fallait comparer les solutions entre elles afin de déterminer celles qui étaient non-dominées. Notons toutefois qu'une solution non-dominée est un ensemble d'attributs $S \subseteq \mathcal{A}$ tel que $\nexists S' \subset S$ qui soit une solution, i.e. $\forall S' \subset S$, $\exists (o, o') \in \Theta$ avec $C^{S'}[(o, o'), .] = \mathbb{0}$ (voir propriété 5).

Nous pouvons donc en déduire la propriété suivante :

Propriété 8. *Une solution S est non-dominée si et seulement si $\forall s \in S$, $\exists (o, o') \in \Theta$ tel que $C^{S \backslash \{s\}}[(o, o'), .] = \mathbb{0}$.*

Nous pouvons ainsi présenter une propriété clé pour la suite :

Propriété 9. *Soit une instance $(\Omega, \mathcal{A}, D, G)$ et la matrice de contraintes C associée, un ensemble d'attribut $A \subseteq \mathcal{A}$ respecte l'une des propositions suivantes :*

1. *A n'est pas une solution $\Leftrightarrow C^A_{\succ} = \mathbb{0}$,*
2. *A est une solution non dominée $\Leftrightarrow$ Il existe une permutation particulière φ sur A telle que $C^{\varphi(A)}_{\succ} = Id$ (où Id est la matrice identité),*
3. *Autrement, A est une solution dominée.*

Preuve 2.

1. *A n'est pas une solution $\Leftrightarrow \exists (o, o') \in \Theta$ tel que $C^A[(o, o'), .] = \mathbb{0}$. Or, pour tout vecteur X de taille $|A|$, on a $\mathbb{0} \wedge X = \mathbb{0}$ (i.e. tous les vecteurs X sont redondants), ce qui implique que $C^A_{\succ} = \mathbb{0}$.*

2. *À partir de la propriété 8, on a : A est une solution non-dominée $\Leftrightarrow \forall s \in A, \exists (o, o') \in \Theta$ tel que $C^{A\setminus\{s\}}[(o, o'), .] = \mathbb{0} \Leftrightarrow C^A[(o, o'), .] = [0, \cdots, 0, 1, 0 \cdots, 0]$ (où le 1 est l'élément du vecteur correspondant à l'attribut s) car A est une solution (et donc $C^A[(o, o'), .] \neq \mathbb{0}$). Ainsi, $\forall s \in A$, une contrainte $C^A[\theta, .]$ tel que $C^A[\theta, s] = 1$ ($\theta \neq (o, o')$) sera redondante par rapport à $C^A[(o, o'), .]$. La matrice $C^A_{\succ}$ sera donc une matrice ne contenant qu'un unique 1 par ligne et par colonne $\Leftrightarrow$ Il existe une permutation φ sur A telle que $C^{\varphi(A)}_{\succ} = Id$.*

3. *Si un ensemble est une solution qui n'est pas non-dominée, alors c'est une solution dominée.*

Remarque 7. *La proposition 2 de la propriété 9 est une généralisation de la propriété 4 vue pour les ensembles intersectants.*

Exemple 20. *Reprenons la matrice de contraintes présentée dans l'exemple 11 :*

Variables / Couples	*Rythme (r)*	*Costume (c)*	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(1,2)	*1*	*0*	*0*	*1*	*1*
(1,3)	*0*	*0*	*1*	*1*	*1*
(1,4)	*0*	*1*	*1*	*1*	*0*
(1,5)	*1*	*1*	*0*	*0*	*0*
(1,6)	*1*	*0*	*1*	*0*	*1*
(2,4)	*1*	*1*	*1*	*0*	*1*
(2,5)	*0*	*1*	*0*	*1*	*1*
(2,6)	*0*	*0*	*1*	*1*	*0*
(3,4)	*0*	*1*	*0*	*0*	*1*
(3,5)	*1*	*1*	*1*	*1*	*1*
(3,6)	*1*	*0*	*0*	*1*	*0*
(4,5)	*1*	*0*	*1*	*1*	*0*
(4,6)	*1*	*1*	*0*	*1*	*1*

Nous avions vu que le sous-ensemble d'attributs $A = \{c, l, m\}$ est une solution. Nous avons ainsi $C^A_{\succ}$:

$C^A_\succ =$

Couples \ Variables	Costume (c)	Lunettes (l)	Montre (m)
(1,5)	1	0	0
(1,6)	0	0	1
(2,6)	0	1	0

Soit la permutation φ de A telle que $\varphi(A) = \{c, m, l\}$, on a :

$C^{\varphi(A)}_\succ =$

Couples \ Variables	Costume (c)	Montre (m)	Lunettes (l)
(1,5)	1	0	0
(1,6)	0	1	0
(2,6)	0	0	1

$= Id$

On a bien $C^{\varphi(A)}_\succ = Id$, A est une solution non-dominée.

Nous pouvons donc déterminer si une solution est non-dominée sans avoir à les générer toutes. À partir d'une solution donnée, l'algorithme `VS` (algorithme 17) déterminera s'il s'agit où non d'une solution non-dominée.

Données : Un ensemble $S \subseteq \mathcal{A}$
La matrice de contraintes C
Résultat : 0 si S n'est pas une solution, 1 si S est non-dominée et 2 si S est dominée
//C' est un ensemble de contraintes
$C' = \emptyset$
pour tous les $\theta \in \Theta$ **faire**
  si $C^S[\theta, .] = \mathbb{0}$ **alors**
    retourner(0)
  si $\sum_{s \in S} C^S[\theta, s] = 1$ **alors**
    si $C^S[\theta, .] \notin C'$ **alors**
      $C' = C' \cup \{C^S[\theta, .]\}$
si $|C'| = |S|$ **alors**
  retourner(1)
sinon
  retourner(2)

Algorithme 17 : Vérification d'une solution (`VS`)

Remarque 8. *Si une solution S est dominée, $\forall S' \subset S$ une solution non dominée, $\forall a \in \mathcal{A}$ avec $C^S_\succ[., a] = \mathbb{0}$, on a $a \notin S'$. Autrement dit, nous pouvons travailler sur l'ensemble $S'' = S \backslash \{a | C[., a] = \mathbb{0}\}$ au lieu de travailler sur S et vérifier si S'' est non-dominée.*

3.2.3.2 Algorithme glouton

D'après la propriété 9 présentée dans la section précédente, afin de construire une solution non dominée, il suffit de générer une matrice identité (à une fonction de permutation φ sur A près) à partir de la matrice C.

L'algorithme glouton GA−NDS (algorithme 18) va construire de manière récursive cette matrice en réduisant l'ensemble $A \subset \mathcal{A}$ à partir d'une contrainte θ de façon à ce que cette contrainte respecte $\sum_{s \in A} C[\theta, s] = 1$.

Données : L'ensemble d'attributs $\mathcal{A}$
La matrice de contraintes réduite $C = C_{\succ}$
L'ensemble des contraintes non redondantes Θ
Résultat : Une solution S non-dominée
Fonction *GA-NDS*(C, A, Θ)
  $A = A \backslash \{a \in A | C[., a] = \mathbb{0}\}$
  si $C^A_{\succ} = Id$ *(à une fonction de permutation φ près)* **alors**
    Retourner(A)
  sinon
    $a_{best} = max_{a \in A}(\sum_{\theta \in \Theta} C[\theta, a])$
    $\Theta' = \{\theta \in \Theta | C[\theta, a_{best}] = 0\}$
    Faire
      $\theta_{best} = min_{\theta \in \Theta \backslash \Theta'}(\sum_{a \in A} C[\theta, a])$
      $A' = \{a \in A | C[\theta_{best}, a] = 0\}$
      $\Theta'' = \Theta' \backslash \{\theta \in \Theta' | C^{A'}[\theta, .]$ redondante$\}$
      si $C^{A'}_{\Theta''} = \mathbb{0}$ **alors**
        $\Theta' = \Theta' \cup \{\theta_{best}\}$
    Tant que $C^{A'}_{\Theta''} = \mathbb{0}$;
    Retourner($\{a_{best}\} \cup$ GA-NDS$(C^{A'}_{\Theta''}, A', \Theta'')$)

Algorithme 18 : Fonction gloutonne générant une solution non-dominée (GA−NDS)

L'algorithme donnera nécessairement une solution non-dominée S puisque par construction, $C^S_{\succ}$ sera une matrice identité (à une fonction de permutation φ près).

Optimisation à chaque étape :

- Limiter la récursivité : lors du choix de l'attribut a_{best}, choisir l'attribut présent le plus souvent dans les contraintes non-satisfaites afin de réduire au mieux le nombre de contraintes non satisfaites ;
- Limiter rapidement le choix des attributs : lors du choix de la contrainte θ_{best}, choisir la contrainte constituée du plus grand nombre d'attributs afin de supprimer de notre recherche le plus d'attributs possible.

Toutefois, il s'agit d'un algorithme glouton et si la solution S est non-dominée, rien ne garantit qu'elle soit minimale.

Afin de mieux illustrer cet algorithme, l'exemple ci-dessous le déroule à partir de la matrice de contraintes issue de l'exemple 11 :

Exemple 21. *Reprenons la matrice de contraintes C de l'exemple 11. La matrice $C_{\succ}$ donne :*

Couples \ Variables	*Rythme (r)*	*Costume (c)*	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(1,5)	*1*	*1*	*0*	*0*	*0*
(1,6)	*1*	*0*	*1*	*0*	*1*
(2,6)	*0*	*0*	*1*	*1*	*0*
(3,4)	*0*	*1*	*0*	*0*	*1*
(3,6)	*1*	*0*	*0*	*1*	*0*

L'attribut satisfaisant le plus de contraintes est ainsi l'attribut r. On a donc $\Theta' = \{(2,6),(3,4)\}$ représentant les seules contraintes non satisfaites par $\{r\}$. Concernant θ_{best}, nous avons le choix entre les trois couples $(1,5),(1,6),(3,6)$. Prenons par exemple $\theta_{best} = (1,5)$, on a $A' = \{ch,l,m\}$, $\Theta'' = \Theta'$ et

$C^{A'}_{\Theta''} =$

Couples \ Variables	*Cheveux (ch)*	*Lunettes (l)*	*Montre (m)*
(2,6)	*1*	*1*	*0*
(3,4)	*0*	*0*	*1*

Soit S' une solution non-dominée de la nouvelle instance décrite par $C^{A'}_{\Theta''}$, une solution du PCM pour l'instance principale est $S = \{r\} \cup S'$

Nous obtenons S' par récursivité (en lançant l'algorithme avec $C^{A'}_{\Theta''}$ en entrée). Il est ici facile de voir qu'une des solutions non-dominées possibles est $S' = \{l,m\}$. La solution retournée par l'algorithme pour l'instance générale sera $S = \{r,l,m\}$ et on a bien :

$C^{S}_{\succ} =$

Couples \ Variables	*Rythme (r)*	*Lunettes (l)*	*Montre (m)*
(1,5)	*1*	*0*	*0*
(2,6)	*0*	*1*	*0*
(3,4)	*0*	*0*	*1*

Et donc $C^{S}_{\succ} = Id$, à une fonction de permutation φ sur S près. S est donc une solution non-dominée.

3.2.3.3 Calcul de l'ensemble des solutions non-dominées

Comme il est possible de détecter si une solution est non-dominée, nous pouvons générer un ensemble de solutions sans comparer toutes les solutions. Il est donc possible d'améliorer l'algorithme de Berge (voir la sous-section 2.4.2) de façon drastique avec cette méthode. Toutefois, s'il est possible de ne plus générer de solutions dominées, il faut toutefois éviter de générer des solutions identiques.

Nous introduisons donc la notion d'attribut nié. L'idée est simple, soit une solution S en cours (i.e. une solution d'un sous-problème représenté par $C_{\Theta'},\ \Theta' \subset \Theta$) et un attribut $a \in \mathcal{A}$:

- Étape 1 : Déterminer toutes les solutions non-dominées contenant $S \cup \{a\}$.
- Étape 2 : Ne plus considérer l'attribut a lors du calcul des autres solutions contenant S. Pour cela, nous associons à S un ensemble d'attributs niés $\mathcal{N}_S$ représentant les attributs ne devant jamais être étudiés avec la solution S.

Ainsi, pour un ensemble $A \in \mathcal{A}$ et une contrainte $C[\theta, .]$ non satisfaite, nous choisissons l'un des $a \in \mathcal{A}$ tel que $C[\theta, a] = 1$ pour que $A \cup \{a\}$ satisfasse la contrainte. Pour obtenir d'autres solutions, nous choisissons de ne plus considérer a par la suite afin de ne jamais obtenir de solution dominée ou identique. Nous ajoutons donc a à l'ensemble des attributs niés associées à l'ensemble A.

Remarque 9. *L'ensemble des attributs niés $\mathcal{N}_S$ d'une solution S en cours à un rôle similaire à l'ensemble Υ présenté dans la sous-section 2.4.4. En l'occurrence, $\Upsilon = \mathcal{A} \backslash \mathcal{N}_S$.*

Par la suite, nous représenterons l'ensemble des attributs niés $\mathcal{N}_S$ d'une solution S par un vecteur booléen N_S de taille $\mathcal{A}$ où $N_S[a] = 1$ si $a \in \mathcal{N}_S$, 0 sinon, $\forall a \in \mathcal{A}$.

En utilisant cette méthode, la fonction récursive `Fc-RNDS` (algorithme 19) génère toutes les solutions non-dominées contenant un sous-ensemble S d'attributs et satisfaisant un sous-ensemble de contraintes représentées par la matrice C_{min}.

L'algorithme `RNDS` (algorithme 20) initialise les paramètres avant d'appeler la fonction `Fc-RNDS` qui générera l'ensemble des solutions non-dominées.

L'algorithme cherche donc à satisfaire toutes les contraintes, en incrémentant des solutions non-dominées en cours, tout comme l'algorithme de Berge. À chaque incrémentation, 2 actions seront exécutées à partir de la solutions en cours S' :

- Vérifier si S' est bien non dominée en vérifiant que la matrice C' des contraintes satisfaites et non-redondantes est une matrice identité à une fonction de permutation φ sur $\mathcal{A}$ près ;
- Associer à S' la matrice C'_{min} des contraintes non satisfaites.

Données : Un sous-ensemble d'attributs S
Les attributs niés représentées par le vecteur N_S
La matrice de contraintes C
Une matrice d'un sous-ensemble de contraintes C_{min}
Une contrainte θ_{best}
Un ensemble de solutions Sol
Résultat : Un ensemble de solutions Sol
Fonction *Fc-RNDS*$(S, N_S, C, C_{min}, \theta_{best}, Sol)$
//L'ensemble P représente les attributs possibles à ajouter pour satisfaire $C[\theta_{best}, .]$
$P = \{a|(C_{min}[\theta_{best}, .] \wedge (\neg N_S))[a] = 1\}$
$N_{S'} = N_S$
tant que $P \neq \emptyset$ **faire**
//On travaille sur les attributs de P dans un ordre précis
$a_{best} = max_{a \in P}(\sum_{i \in \{1,\cdots,|C_{min}|\}} C_{min}[\theta_i, a])$
$S' = S \cup \{a_{best}\}$
//On construit maintenant la matrice C'_{min} des contraintes non satisfaites par S' ainsi que la matrice C' vérifiant si S' n'est pas dominée.
$\Theta' = \emptyset$
$\Theta'' = \emptyset$
pour tous les $\theta \in \Theta$ **faire**
si $\sum_{a \in S'} C[\theta, a] = 0$ **alors**
$\Theta' = \Theta' \cup \{\theta\}$
si $\sum_{a \in S'} C[\theta, a] = 1$ **alors**
$\Theta'' = \Theta'' \cup \{\theta\}$
$C'_{min} = C_{\Theta'}$
$\theta_{best} = min_{\theta \in \Theta'}(\sum_{a \in \mathcal{A}|N_{S'}[a]=0} C_{min}[\theta, a])$
$C' = (C^{S'}_{\Theta''})_{\succ}$
$t =$ le nombre de contraintes dans C'
si $t = |S'|$ **alors**
//La solution en cours est non-dominée
si $|C'_{min}| = 0$ **alors**
//S' satisfait toutes les contraintes
$Sol = Sol \cup \{S'\}$
sinon
$Sol =$ Fc-RNDS$(S', N_{S'}, C, C'_{min}, \theta_{best}, Sol)$
//On a trouvée toutes les solutions contenant a_{best}, on l'élimine pour la suite
$P = P \backslash \{a_{best}\}$
$N_{S'}[a_{best}] = 1$
retourner(Sol)

Algorithme 19 : Fonction récursive générant un ensemble de solutions non-dominées (Fc-RNDS)

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Résultat : L'ensemble Sol des solutions non-dominées
Créer la matrice de contraintes C à partir de D et G
$Sol = \emptyset$
$S = \emptyset$
$N_S = \mathbb{O}$
//Nous cherchons la contrainte qui limitera le nombre d'incrémentations de la solution en cours
$\theta_{best} = min_{\theta \in \Theta}(\sum_{a \in \mathcal{A}} C[\theta, a])$
//C_{min} représente la matrice des contraintes non satisfaites par la solution en cours
$C_{min} = C_{\succ}$
//Lancement de l'algorithme récursif
Sol=Fc-RNDS$(S, N_S, C, C_{min}, \theta_{best}, Sol)$
retourner(Sol)

Algorithme 20 : Génération de l'ensemble des solutions non-dominées (RNDS)

Nous optimisons la fonction `Fc-RNDS` sur deux aspects :

- Choix de $\theta \in \Theta'$: Parmi ces contraintes non satisfaites, la prochaine qui sera étudiée sera celle limitant la taille de P, c'est à dire l'ensemble des attributs à considérer pour satisfaire cette contrainte. Ainsi, en limitant la taille de P, nous limitons le nombre de répétitions de la boucle principale.

- Choix de $a \in P$: Pour une contrainte donnée à étudier, nous devons incrémenter la solution en cours pour en faire $|P|$ nouvelles solutions en ajoutant à chaque fois un élément de P différent. À chaque fois que nous incrémentons la solution S en cours avec un attribut a, nous trouvons l'ensemble des solutions contenant $S \cup \{a\}$. Pour l'étude des autres solutions $S \cup \{p\},\ p \in P \backslash \{a\}$, nous ajoutons $\{a\}$ à $\mathcal{N}_S$, ce qui limitera le champs de recherche pour la suite. En fait, plus une solution S est associée à un grand nombre d'attributs niés, plus le calcul de l'ensemble des solutions contenant S et ne contenant pas $\mathcal{N}_S$ sera rapide. L'idée est donc de choisir d'incrémenter S avec l'attribut satisfaisant le plus de contraintes en priorité, pour limiter le nombre de récursivité pour les incrémentations S' ayant le moins d'attributs niés associées.

Nos expérimentations montrent un gain du temps d'exécution deux à trois fois supérieur à un choix de $a \in P$ et $\theta \in \Theta'$ aléatoire.

Remarque 10. *Cet algorithme est très proche de l'algorithme `MMCS` présenté dans la section 2.4.4 en considérant chaque contrainte $C_{min}[\theta, .]$ comme un élément de leur ensemble $uncov$. La différence principale étant que l'algorithme `MMCS` conserve des ensembles de contraintes associés à chaque solution afin de vérifier la non-dominance d'une solution en cours tandis que l'algorithme `RNDS` le fait à partir de la matrice de contraintes C. De plus, l'algorithme `MMCS`*

choisit la contrainte à étudier durant une étape ainsi que l'ordre de traitement des attributs aléatoirement tandis que l'algorithme `RNDS` *les choisit grâce à une étude sur la matrice* C.

Remarque 11. *Il est possible de ne déterminer que les solutions non-dominées de taille inférieure ou égale à une borne* B *donnée avec l'algorithme* `RNDS`*. En effet, pour cela, l'étape où l'algorithme* `RNDS` *est appelé avec les paramètres* $(S', N_{S'}, C, C'_{min}, \theta_{best})$ *ne doit être faite que si* $|S'| < B$.

3.3 Résolution du *MinAll-PCM*

Nous nous intéressons ici à la génération de l'ensemble des solutions de taille minimale. Nous verrons, tout comme dans la section précédente, deux approches différentes : l'approche contrainte par contrainte et l'approche solution par solution.

3.3.1 Approche contrainte par contrainte

Dans cette sous-section nous allons présenter une méthode de génération des solutions minimales satisfaisant les contraintes une à une, puis nous présenterons un cas particulier d'utilisation de cette méthode.

3.3.1.1 Calcul des solutions minimales

Dans cette section, nous nous intéressons à la résolution du *MinAll-PCM*, c'est à dire au calcul des solutions du PCM de plus petite taille, c'est à dire des solutions S telles qu'il n'existe pas de solution S' du PCM telle que $|S'| < |S|$:

Proposition 2. *Une solution* $S \subseteq \mathcal{A}$ *est minimale si et seulement si* $\nexists S' \subset \mathcal{A}$ *avec* $|S'| < |S|$ *telle que* $\forall \theta \in \Theta$, $C^{S'}[\theta, .] \neq \mathbb{0}$

Notons qu'une solution S_1 ne peut être dominée que par une solution de plus petite taille S_2 puisque $S_2 \succ S_1 \Leftrightarrow S_2 \subset S_1$ (où $S_2 \succ S_1$ signifie que S_2 domine S_1). Si S_1 est de taille minimale, alors il n'existe pas de solution de plus petite taille pouvant la dominée. On a donc la propriété suivante :

Propriété 10. *Une solution minimale est toujours non-dominée.*

L'ensemble des solutions minimales est donc un sous-ensemble de l'ensemble des solutions non-dominées. Une première approche serait donc de générer l'ensemble des solutions non-dominées et de les comparer afin de ne conserver que celles de plus petites tailles. Toutefois, il est possible de ne générer que des solutions minimales durant un processus de génération des solutions non-dominées. Le principe est simple, si nous connaissons la taille T_{min} des solutions

minimales, alors il nous suffit de générer l'ensemble des solutions non-dominées inférieures ou égales à une borne $B = T_{min}$. Les algorithmes pouvant générer l'ensemble des solutions inférieures à une borne peuvent donc tous servir à générer l'ensemble des solutions minimales. De plus, une solution minimale étant toujours non-dominée, il n'est pas nécessaire de vérifier la dominance des solutions, seulement d'éviter la construction de solutions identiques grâce aux attributs niés par exemple.

Remarque 12. *Il peut être intéressant de vérifier la dominance à chaque étape pour limiter le nombre de solutions générées qui ne seront pas minimales dès lors que la vérification de la dominance est moins coûteuse en temps que la génération de solutions dominées.*

La problématique est donc de déterminer la taille T_{min} des solutions minimales. Soit nbg le nombre de groupes de l'instance, nous savons que $T_{min} \in [\lceil log_2(nbg) \rceil, |\mathcal{A}|]$ car il faut au moins un attribut pour distinguer un groupe contre tous les autres. Nous proposons donc une méthode "par le bas" pour déterminer cette borne : si nous trouvons une solution avec la borne $B = \lceil log_2(nbg) \rceil$, alors cette solution est nécessairement minimale. Sinon, il n'existe pas de solution de taille B et s'il existe une solution de taille $B + 1$, alors elle sera minimale. En relançant l'algorithme jusqu'à obtenir au moins une solution, nous déterminons l'ensemble des solutions minimales.

L'algorithme `Min-NDS` (algorithme 21) déterminera toutes les solutions minimales en utilisant n'importe quel algorithme déterminant l'ensemble des solutions (non-dominées ou non) inférieures à une borne B, que nous appellerons par la fonction $AlgoSol(C, B)$, avec les entrées C pour la matrice de contraintes et B pour la borne.

Données : Une instance $(\Omega, \mathcal{A}, D, G)$,
Nombre de groupes nbg
Résultat : L'ensemble Sol des solutions non-dominées
//Borne minimale
$B = \lceil log_2(nbg) \rceil$
Générer la matrice de contraintes C à partir de D et G
$Sol = \emptyset$
Faire
  //Calcul des solutions inférieures à B
  $Sol = AlgoSol(C, B)$
  //Incrémentation de la borne si besoin
  si $Sol = \emptyset$ **alors**
    $B = B + 1$
Tant que $Sol = \emptyset$;
retourner(Sol)

Algorithme 21 : Algorithme de génération de l'ensemble des solutions minimales (`Min-NDS`)

S'il n'existe pas de solution inférieure à une borne, les solutions en cours sont éliminées

rapidement car supérieures à la borne. L'algorithme travaille donc sur moins de solutions et est donc moins coûteux en temps d'exécution.

3.3.1.2 Extention de l'algorithme `NDS-B`

Si l'algorithme `MIN-NDS` fonctionne avec n'importe quel algorithme déterminant l'ensemble des solutions non-dominées bornées du PCM, alors nous pouvons nous servir de l'algorithme `NDS-B` (algorithme 16).

Comme une solution minimale est toujours non-dominée, il est donc inutile de vérifier la non-dominance des solutions en cours puisque, par construction, il n'en existe pas de plus petite. Il est donc possible de supprimer cette étape de vérification. Il faut toutefois éviter de générer des solutions identiques, c'est pourquoi il est nécessaire dans ce cas d'utiliser les attributs niés introduites dans la section 3.2.3.3.

En utilisant ce principe, l'algorithme `MWAD` (algorithme 22) déterminera l'ensemble des solutions minimales.

Notons que la vérification utilisé dans l'algorithme `NDS-B` était la tâche la plus coûteuse en terme de temps d'exécution. Ne plus l'utiliser permet de considérablement augmenter les performances de l'algorithme. De plus, alors que l'algorithme `NDS-B` n'est pas parallélisable (car les solutions en cours doivent être comparées entre elles), celui-ci l'est grâce à l'utilisation des attributs niés.

3.3.2 Approche solution par solution

Tout comme nous pouvons déterminer la taille minimale T_{min} avec une approche "par le bas", il est possible d'avoir une approche "par le haut", c'est à dire déterminer une borne supérieure B et dès que l'on trouve une solution de $T < B$, diminuer cette borne $B = T$. Une telle approche s'adapte donc parfaitement à un algorithme qui détermine les solutions par un "parcours en profondeur", c'est-à-dire un algorithme qui va incrémenter une solution en cours jusqu'à ce qu'elle satisfasse toutes les contraintes avant de passer à la solution en cours suivante (il s'agit donc d'une approche de type *branch and bound*). Par exemple, l'algorithme `NDS` n'est pas adapté à cette approche car il détermine toutes les solutions en cours satisfaisant les j premières contraintes avant de passer à la contrainte $j + 1$. Il ne déterminera donc les solutions de l'instance qu'à la fin et ne pourra donc pas faire diminuer la borne en cours de recherche. En revanche, l'algorithme `RNDS` (algorithme 20) est quant à lui intéressant dès lors qu'il détermine une solution avant de passer à la suivante mais, en plus, cherche à éviter la récursivité lors du calcul des premières solutions, et par conséquent tente de déterminer les solutions de plus petite taille en priorité.

Concernant la borne initiale, il est nécessaire qu'elle soit supérieure ou égale à la taille des

Données : Une instance $(\Omega, \mathcal{A}, D, G)$,
Nombre de groupes nbg
Résultat : L'ensemble Sol des solutions non-dominées
$b = \lceil log_2(nbg) \rceil$
$\mathscr{N}_\emptyset = \emptyset$
Faire
  $Sol = \{\emptyset\}$
  pour $i = 1$ *à* $|\Theta|$ **faire**
    //Construire un sous-ensemble de solutions ND_i
    $ND_i = \emptyset$
    pour tous les $j \in \mathcal{A}$ *tel que* $C[\theta_i, j] = 1$ **faire**
      pour tous les $S \in Sol \backslash ND_i$ **faire**
        si $j \in S$ **alors**
          $ND_i = ND_i \cup \{S\}$
    //Construire un sous-ensemble de solutions ES_i
    $ES_i = \emptyset$
    pour tous les $S \in Sol \backslash ND_i$ **faire**
      si $|S| < b$ **alors**
        //Construction progressive de l'ensemble d'attributs niés $\mathscr{N}$
        $\mathscr{N} = \mathscr{N}_S$
        pour tous les $j \in \mathcal{A}$ *tel que* $C[\theta_i, j] = 1$ *et* $j \notin \mathscr{N}_S$ **faire**
          $ES_i = ES_i \cup \{S \cup \{j\}\}$
          $\mathscr{N}_{S \cup \{j\}} = \mathscr{N}$
          //Incrémentation de $\mathscr{N}$
          $\mathscr{N} = \mathscr{N} \cup \{j\}$
    $Sol = ND_i \cup ES_i$
    si $Sol = \emptyset$ **alors**
      $b = b + 1$
      //On sort de la boucle "pour"
      $i = |\Theta|$
Tant que $Sol = \emptyset$;
retourner(Sol)

Algorithme 22 : Algorithme de génération de l'ensemble des solutions minimales (`MWAD`)

solutions minimales T_{min}. Comme $T_{min} \leqslant |\mathcal{A}|$, nous pouvons choisir $B = |\mathcal{A}|$. Toutefois, en utilisant un algorithme glouton, nous pouvons obtenir rapidement une solution S nous donnant donc une borne $B = |S|$. L'algorithme `GA-NDS` (algorithme 18) nous donne non seulement une solution non-dominée (ce qui limite donc la taille de la solution), mais tente en outre de la choisir de façon à limiter sa taille. Toutefois, si l'algorithme utilisé pour déterminer l'ensemble des solutions offre une meilleure borne dès les premières solutions qu'elle détermine (comme pourrait le faire l'algorithme `RNDS`), l'utilisation d'un algorithme glouton est rapidement superflue.

La fonction `Fc-Min-RNDS` (algorithme 23) décrit les modifications apportées à la fonction `Fc-RNDS` afin de ne déterminer que les solutions minimales.

Données : Un sous-ensemble d'attributs S
Les attributs niés représentées par le vecteur N_S
La matrice de contraintes C
Une matrice d'un sous-ensemble de contraintes C_{min}
Une contrainte θ_{best}
Une borne b
Un ensemble de solutions Sol
Résultat : Un ensemble de solutions Sol

Fonction *Fc-Min-RNDS*$(S, N_S, C, C_{min}, \theta_{best}, b, Sol)$
- $P = \{a \in \mathcal{A} | (C_{min}[\theta_{best}, .] \wedge (\neg N_S))[a] = 1\}$
- $N_{S'} = N_S$
- **tant que** $P \neq \emptyset$ **faire**
  - $a_{best} = max_{a \in P}(\sum_{i \in \{1, \cdots, |C_{min}|\}} C_{min}[\theta_i, a])$
  - $S' = S \cup \{a_{best}\}$
  - $\Theta' = \emptyset$
  - $\Theta'' = \emptyset$
  - **pour tous les** $\theta \in \Theta$ **faire**
    - **si** $\sum_{a \in S'} C[\theta, a] = 0$ **alors**
      - $\Theta' = \Theta' \cup \{\theta\}$
    - **si** $\sum_{a \in S'} C[\theta, a] = 1$ **alors**
      - $\Theta'' = \Theta'' \cup \{\theta\}$
  - $C'_{min} = C_{\Theta'}$
  - $\theta_{best} = min_{\theta \in \Theta'}(\sum_{a \in \mathcal{A} | N_{S'}[a]=0} C_{min}[\theta, a])$
  - $C' = (C^{S'}_{\Theta''})_{\succ}$
  - $t =$ le nombre de contrainte dans C'
  - **si** $t = |S'|$ **alors**
    - **si** $|C'_{min}| = 0$ **alors**
      - **si** $|S'| < b$ **alors**
        - $b = |S'|$
        - $Sol = \emptyset$
      - $Sol = Sol \cup \{S'\}$
    - **sinon**
      - **si** $|S'| < b$ **alors**
        - $Sol =$ Fc-Min-RNDS$(S', N_{S'}, C, C'_{min}, \theta_{best}, b, Sol)$
  - $P = P \backslash \{a_{best}\}$
  - $N_{S'}[a_{best}] = 1$
- retourner(Sol)

Algorithme 23 : Fonction récursive générant un ensemble de solutions minimales (`Fc-Min-RNDS`)

Ainsi, l'algorithme `Min-RNDS` (algorithme 24) initialisera les bons paramètres pour que la fonction Fc-Min-RNDS puisse calculer l'ensemble des solutions minimales.

Données : Une instance $(\Omega, \mathcal{A}, D, G)$
Résultat : L'ensemble Sol des solutions non-dominées
// Initialisation de la borne
$b = \mathcal{A}$
Créer la matrice de contraintes C à partir de D et G
$Sol = \emptyset$
$S = \emptyset$
$N_S = \mathbb{O}$
$\theta_{best} = min_{\theta \in \Theta}(\sum_{a \in \mathcal{A}} C[\theta, a])$
$C_{min} = C_{\succ}$
//Lancement de l'algorithme récursif
$Sol =$ Fc-Min-RNDS$(S, N_S, C, C_{min}, \theta_{best}, b, Sol)$
retourner(Sol)

Algorithme 24 : Algorithme de génération de l'ensemble des solutions minimales (`Min-RNDS`)

3.4 Génération de prime patterns

Dans cette section, nous proposons d'étendre l'approche de résolution du PCM afin de générer des patterns pour l'analyse logique de données (LAD) (voir section 2.2).

Nous commencerons par rappeler quelques concepts et définitions de LAD avant de présenter une méthode de génération de l'ensemble des prime patterns dans un premier temps et une méthode de filtrage pour ne concerver que les strong prime patterns dans un second temps.

3.4.1 Quelques rappels sur l'analyse logique de données

Rappelons ici les principaux concepts de LAD vus dans la section 2.2 :

- **Définition 3 :**
  Une fonction booléenne partiellement définie (pdBf) sur $\mathbb{B}^n$ est un couple (P,N) tel que P,$N \subseteq \mathbb{B}^n$ et $P \cap N = \emptyset$.

- **Définition 4 :**
  Étant donnés deux sous- ensembles $\sigma^+, \sigma^- \subseteq \{1, 2, ..., n\}$ représentant les indices des attributs et tel que $\sigma^+ \cap \sigma^- = \emptyset$, un terme t_{σ^+,σ^-} est une fonction booléenne dont l'ensemble positif $T(t_{\sigma^+,\sigma^-})$ est de la forme :

$$T(t_{\sigma^+,\sigma^-}) = \{x \in \mathbb{B}^n | x_i = 1 \; \forall i \in \sigma^+ \text{ et } x_j = 0 \; \forall j \in \sigma^-\}$$

où x_i représente un littéral associé à l'attribut i.
Un terme est représenté par une *conjonction élémentaire*, i.e., une expression booléenne de la forme :

$$t_{\sigma^+,\sigma^-}(x) = (\bigwedge_{i\in\sigma^+} x_i) \wedge (\bigwedge_{j\in\sigma^-} \bar{x}_j)$$

- **Définition 5 :**
  Un pattern d'une pdBf (P,N) est un terme t_{σ^+,σ^-} tel que $|P \cap T(t_{\sigma^+,\sigma^-})| > 0$ et $|N \cap T(t_{\sigma^+,\sigma^-})| = 0$.

Un pattern est donc un terme satisfait par au moins un vecteur positif et aucun vecteur négatif. Il est toutefois intéressant de connaître les vecteurs positifs satisfaisant ce pattern :

- **Définition 6 :**
  La *couverture* d'un pattern p, noté $Cov(p)$ est l'ensemble $Cov(p) = P \cap T(p)$.

Les patterns étant nombreux et redondants, nous ne cherchons qu'à déterminer les "meilleurs" en fonction d'une ou plusieurs préférences. Nous nous intéresserons dans cette section à deux préférences :

- **Préférence de simplicité** (définition 7) :
  La préférence de simplicité σ, notée $\succcurlyeq_\sigma$, est une relation binaire sur un ensemble de patterns $\mathcal{P}$ tel que pour un couple $(p_1, p_2) \in \mathcal{P}^2$, on a $p_1 \succcurlyeq_\sigma p_2$ si et seulement si $Lit(p_1) \subseteq Lit(p_2)$ (où $Lit(p)$ désigne l'ensemble des littéraux d'un pattern p).

- **Préférence évidentielle** (définition 9) :
  La préférence évidentielle $\mathcal{E}$, notée $\succcurlyeq_\mathcal{E}$, est une relation binaire sur un ensemble de patterns $\mathcal{P}$ tel que pour un couple $(p_1, p_2) \in \mathcal{P}^2$, on a $p_1 \succcurlyeq_\mathcal{E} p_2$ si et seulement si $Cov(p_2) \subseteq Cov(p_1)$.

Nous dirons donc qu'un pattern p est :

- **prime** s'il n'existe pas de pattern p' tel que $p' \succ_\sigma p$,
- **strong** s'il n'existe pas de pattern p' tel que $p' \succ_\mathcal{E} p$,
- **strong prime** si p est à la fois strong et prime.

3.4.2 Génération de prime patterns

Le PCM étant une extension de LAD, nous proposons ici une méthode de génération de l'ensemble des prime patterns basée sur les algorithmes de calcul de l'ensemble des solutions non-dominées du PCM.

L'analyse logique de données étant définie sur deux groupes (observations positives et négatives), nous avons besoin de considérer seulement des instances du PCM à deux groupes (ou un groupe contre tous les autres). Une pdBf étant définie par un couple (P, N), on peut lui associer une instance du PCM $(\Omega, \mathcal{A}, D, G)$ où :

- $\Omega = P \cup N$
- $G : \Omega \mapsto \{P, N\}$, $P = \{o \in \Omega | G(o) = P\}$ et $N = \{o \in \Omega | G(o) = N\}$.

Un des objectifs de l'analyse logique de données est de déterminer un pattern couvrant un nombre maximal d'observations de P, tel qu'aucune observation de N ne contienne ce pattern.

Du point de vue du PCM, la notion de solution est un peu différente. Soit une solution S d'une instance du PCM définie comme ci-dessus avec seulement deux groupes, S peut être confondu avec l'ensemble des littéraux d'un pattern seulement si les observation de P sont identiques sur S. En particulier, si $|P| = 1$, l'ensemble des solutions du PCM coïncide en terme d'attributs avec l'ensemble des patterns maximaux, i.e., l'ensemble des patterns qui couvrent l'unique observation dans P. De plus, les solutions non-dominées correspondent aux prime patterns, car dans les deux cas, aucun attribut ne peut être retiré.

Soit une solution S non dominée du PCM qui considère une unique observation $o \in P$, la fonction `Transforme_en_pattern` (algorithme 25) est une fonction simple qui transforme S en prime pattern p.

Données : Une solution non-dominée S du PCM,
Le vecteur $D[o, .]$ représentant la seule observation o de P
Résultat : Un prime pattern p
$pos = \emptyset$
$neg = \emptyset$
pour tous les $a \in S$ **faire**
  si $D[o, a] = 1$ **alors**
    $pos = pos \cup \{a\}$
  sinon
    $neg = neg \cup \{a\}$
$p = (\bigwedge_{i \in pos} x_i) \wedge (\bigwedge_{j \in neg} \neg x_j)$
retourner(p)

Algorithme 25 : Fonction `Transforme_en_pattern`

Ainsi, pour chaque observation, il est possible de générer l'ensemble des prime patterns couvrant cette observation en déterminant les solutions non-dominées de l'instance où le groupe P n'est constitué que de cette observation. Si nous générons tous les primes patterns de toutes les observations, nous obtenons l'ensemble des prime patterns ainsi que l'ensemble $Cov(p)$ de chaque observation couverte par chaque prime pattern p.

Soit un algorithme $Algo\_Sol\_Non\_Dom(C)$ générant l'ensemble des solutions non-dominées d'une instance à partir de la matrice de contraintes C, l'algorithme PPC (algorithme 26), pour **P**rime **P**attern **C**omputation, génère par ce principe l'ensemble des prime patterns de l'instance.

Données : Une instance $(\Omega, \mathcal{A}, D, G)$ avec deux groupes P et N
Résultat : L'ensemble Pat des prime patterns,
L'ensemble Cov de la couverture de chaque prime pattern
$Pat = \emptyset$
$Cov = \emptyset$
pour tous les $o \in P$ **faire**
  Générer la matrice C_o des contraintes $(o, o'),\ \forall o' \in N$
  $Sol = Algo\_Sol\_Non\_Dom(C_o)$
  pour tous les $S \in Sol$ **faire**
    $p = Transforme\_en\_pattern(S, D[o, .])$
    si $p \notin Pat$ **alors**
      $Pat = Pat \cup \{p\}$
      //Nous créons un nouvel élément V_p de Cov qui sera l'ensemble des observations couvertes par p
      $V_p = \{o\}$
      $Cov = Cov \cup \{V_p\}$
    sinon
      //V_p est déjà dans Cov, on l'incrémente
      $V_p = V_p \cup \{o\}$
retourner(Pat et Cov)

Algorithme 26 : Algorithme de génération des prime patterns (PPC)

Remarque 13. *Il n'est pas nécessaire de générer l'ensemble Cov des couvertures pour générer l'ensemble des prime patterns. Dans ce cas, chaque étape travaillant sur l'ensemble Cov peut être retirée de l'algorithme.*

Exemple 22. *Soit les données suivantes :*

Observations	*Groupes*	*Attributs*			
		a	*b*	*c*	*d*
1	*P*	*1*	*1*	*1*	*1*
2		*1*	*1*	*0*	*0*
3		*0*	*0*	*0*	*0*
4	*N*	*1*	*0*	*1*	*1*
5		*1*	*1*	*0*	*1*
6		*0*	*1*	*1*	*1*

En considérant l'observation 1 comme unique observation du groupe P, nous obtenons la matrice de contraintes C_1 suivante :

Contraintes	*Attributs*			
	a	*b*	*c*	*d*
(1,4)	*0*	*1*	*0*	*0*
(1,5)	*0*	*0*	*1*	*0*
(1,6)	*1*	*0*	*0*	*0*

Nous remarquons très vite que l'unique solution non-dominée est $S_1 = \{a, b, c\}$*, ce qui nous donne le pattern* $p_1 = a \wedge b \wedge c$ *qui est bien un prime pattern. Remarquons que s'il existait un pattern* p *couvrant au moins une autre observation et tel que* $Lit(p) \subset Lit(p_1)$*, alors* $Var(p)$ *serait une solution qui dominerait* S_1*. On a donc* p_1 *qui couvre l'observation 1,* $Cov(p_1) = \{1\}$*.*

Considérons maintenant l'observation 2 comme unique observation du groupe P*, on obtient la matrice* C_2 *suivante :*

Contraintes	*Attributs*			
	a	*b*	*c*	*d*
(2,4)	*0*	*1*	*1*	*1*
(2,5)	*0*	*0*	*0*	*1*
(2,6)	*1*	*0*	*1*	*1*

Ici, $S_2 = \{d\}$ *est l'unique solution non-dominée et donc le pattern* $p_2 = \bar{d}$ *est un prime pattern qui couvre l'observation 2 :* $Cov(p_2) = \{2\}$*.*

Enfin, considérons l'observation 3 comme unique observation de P*, et nous obtenons* C_3 *:*

Contraintes	*Attributs*			
	a	*b*	*c*	*d*
(3,4)	*1*	*0*	*1*	*1*
(3,5)	*1*	*1*	*0*	*1*
(3,6)	*0*	*1*	*1*	*1*

Les solutions non-dominées sont $S_3 = \{a, b\}$*,* $S_4 = \{a, c\}$*,* $S_5 = \{b, c\}$ *et* $S_6 = \{d\}$ *ce qui nous donne les patterns* $p_3 = \bar{a} \wedge \bar{b}$*,* $p_4 = \bar{a} \wedge \bar{c}$*,* $p_5 = \bar{b} \wedge \bar{c}$ *et* $p_6 = \bar{d}$*. Or* $p_6 = p_2$ *on a donc* $Cov(p_2) = \{2, 3\}$*. Autrement,* $Cov(p_3) = Cov(p_4) = Cov(p_5) = \{3\}$*.*

Au final, l'ensemble des prime patterns est $Pat = \{p_1, p_2, p_3, p_4, p_5\}$ *et l'ensemble des couvertures est* $Cov = \{Cov(p_1), Cov(p_2), Cov(p_3), Cov(p_4), Cov(p_5)\}$*.*

Notons que nous pouvons également générer des patterns de taille inférieure à une borne B dès lors que l'on utilise un algorithme de génération de solutions du PCM qui en soit capable. Nous pouvons également utiliser un algorithme de génération de solutions minimales si nous souhaitons obtenir les patterns de plus petite taille par observation. Notons également que l'algorithme est parallélisable même si l'algorithme de génération de solutions du PCM ne l'est

pas car nous pouvons travailler sur chaque $o \in P$ de façon indépendante. Dès lors, nous pouvons calculer en parallèle l'ensemble des patterns couvrant chaque observation pour finalement comparer les couvertures une fois la génération des patterns finie.

3.4.3 Génération des strong prime patterns

En utilisant l'ensemble des prime patterns et leur couverture, il est possible de déterminer ceux qui sont strong en comparant les couvertures deux à deux. Nous obtenons ainsi des prime patterns qui sont strong, donc des strong prime patterns (d'après la propriété 2 de la sous-section 2.2.2). L'algorithme `SPPC` (algorithme 27) détermine ainsi les strong prime patterns.

Données : L'ensemble Pat des prime patterns,
L'ensemble Cov de la couverture de chaque prime pattern
Résultat : l'ensemble SPP des strong prime patterns
$SPP = \emptyset$
pour tous les $p \in Pat$ **faire**
  si $\nexists p' \in Pat$ *tel que* $Cov(p) \subset Cov(p')$ **alors**
    $SPP = SPP \cup \{p\}$
retourner(SPP)

Algorithme 27 : Sélection des strong prime patterns (`SPPC`)

3.5 Minimisation de la formulation du PCM

Lorsque nous générons l'ensemble des solutions non-dominées du PCM, nous obtenons régulièrement beaucoup de solutions. Dès lors, il est compliqué de déterminer les solutions les plus appropriées en fonction du critère de l'utilisateur.

Nous avons vu un premier critère qui consiste à minimiser la taille des solutions. Cette approche peut être utile pour concevoir des tests biologiques pour l'identification de groupes par exemple (voir section 2.1). En effet, ces tests consistent à évaluer pour une observation donnée (une bactérie dans notre exemple) un sous-ensemble d'attributs (de gènes) afin de déterminer de quel groupe (pathovar) provient cette observation. Minimiser la taille des solutions permet ainsi de minimiser le nombre d'attributs à étudier pour évaluer l'observation.

Un autre critère peut être la minimisation de la formule logique associée à la solution sous sa forme normale disjonctive (DNF) par soucis de simplicité, de lisibilité et de facilité d'interprétation.

Définition 26. *La taille d'une formule logique DNF est le nombre de conjonctions de littéraux connectés par le connecteur logique* $\vee$ *(i.e. le nombre de connecteurs +1 de ces conjonctions).*

La minimisation de la formulation logique d'un groupe P revient à trouver la solution du PCM pour une instance consituée d'un groupe (P) contre les autres (N) telle qu'il n'existe pas de solution dont la formule associée au groupe P soit de taille inférieure selon la définition 26.

Remarque 14. *Dans cette section nous étudierons les solutions minimisant la formule d'un seul groupe pour une instance du PCM. Nous présenterons donc les méthodes de calcul de solutions d'un groupe contre tous les autres.*
En prenant comme solution liée à l'instance globale l'union de toutes les solutions de chaque groupe, nous aurons une formule générale, mais pas nécessairement de plus petite taille (en terme d'attributs).

Exemple 23. *Considérons l'exemple suivant :*

Observations	*Groupes*	*Attributs*							
		a	*b*	*c*	*d*	*e*	*f*	*g*	*h*
1	*P*	*0*	*1*	*0*	*1*	*0*	*1*	*1*	*0*
2		*1*	*1*	*0*	*1*	*1*	*0*	*0*	*1*
3		*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*
4	*N*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*1*
5		*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
6		*1*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
7		*0*	*0*	*1*	*0*	*1*	*0*	*1*	*0*

$S_1 = \{f, g\}$ est une solution du PCM. La formule logique associée à la solution S_1 pour le groupe P est la DNF $(f \wedge g) \vee (\neg f \wedge \neg g)$. Cette formule booléenne est de taille 2 car elle est constituée de deux conjonctions de littéraux connectées par le connecteur logique $\vee$ (disjonction) : $(f \wedge g)$ ou $(\bar{f} \wedge \bar{g})$.

La taille d'une formule d'un groupe associée à une solution S correspond donc au nombre d'observations de ce groupe différentes sur S, ou autrement dit, au nombre de patterns p_i nécessaire pour couvrir tout le groupe tel que $\forall i,\ Var(p_i) \subseteq S$.

L'objectif d'un tel critère est d'obtenir des observations les plus similaires possible sur la solution. La description d'un groupe parait plus robuste si l'ensemble des observations respectent une unique description plutôt qu'une multitude de possibilités de description.

L'approche par patterns peut être particulièrement utile pour déterminer les solutions qui optimisent ce critère car plus petit est le nombre de prime patterns nécessaire pour couvrir le groupe, plus petite est la taille de la formule booléenne associée à la solution qui coïncide avec ces patterns (en terme d'attributs). Le nombre de patterns qui couvrent le groupe correspond à la taille de la formule logique.

Remarque 15. *Il est possible de ne travailler que sur les strong prime patterns car par définition, la couverture de chaque pattern "non strong" sera strictement incluse dans la couverture d'un strong pattern. Ainsi, parmi les ensembles de patterns de plus petite taille couvrant P, au moins l'un d'eux sera composé de strong patterns uniquement.*

Le problème de minimisation de la formule peut être décrit comme un problème classique de couverture par ensembles dès lors que l'on cherche le plus petit ensemble de patterns qui couvrent chaque observation du groupe P :
Soit Pat l'ensemble des (strong) prime patterns, le vecteur booléen $X = [x_1, \cdot, x_{|Pat|}]$ représentera la présence(1)/absence(0) de chaque pattern dans notre solution.

Nous voulons ainsi minimiser

$$\sum_{i=1}^{|Pat|} x_i$$

Pour chaque observation o, nous introduisons $m_{o,i} = 1$ si le i-ième pattern p_i de Pat couvre o, 0 sinon. Ainsi, nous voulons que chaque observation $o \in P$ soit couverte par au moins un pattern $p_i \in Pat$ de la solution, i.e. tel que $x_i = 1$ (où p_i est le i-ième pattern de Pat). On a donc :

$$\forall o \in P,\ \sum_{i=1}^{|Pat|} m_{o,i}\ x_i \geqslant 1$$

Nous pouvons regrouper toutes les valeurs $m_{o,i}$ dans une matrice M de taille $|P| \times |Pat|$ où :

- Chaque ligne de M correspond à une observation du groupe P,
- Chaque colonne correspond à un pattern,
- $M[o, i] = m_{o,i}$,
- Une ligne $M[o, .]$ représente le vecteur booléen de couverture/non couverture des patterns pour l'observation o,
- Une colonne $M[., i]$ représente le vecteur booléen de couverture/non couverture du i-ième pattern pour chaque observation.

Nous obtenons ainsi le programme linéaire suivant :

$$\begin{aligned}
&min : \sum_{i=1}^{|Pat|} x_i \\
&s.t. : \\
&M\ .\ X^t \geqslant \mathbb{1}^t \\
&X \in \{0,1\}^{|Pat|}, X = [x_1, ..., x_{|Pat|}]
\end{aligned}$$

où $\mathbb{1}$ représente un vecteur composé de 1 et V^t est la transposée du vecteur V.

La matrice M est alors facile à générer dès lors qu'on utilise l'ensemble des couvertures Cov des patterns généré par exemple par l'algorithme `PPC` (algorithme 26) puisque $M[o,i] = 1$ si $o \in Cov(p_i)$, 0 sinon.

Les algorithmes présentés dans la sous-section 3.3 permettent de résoudre le programme linéaire et d'obtenir ainsi des solutions représentant un ensemble de patterns à connecter par le connecteur logique $\vee$ pour obtenir la formulation minimale du groupe.

Soit $\mathcal{P} = \{p_1, \cdots, p_{|\mathcal{P}|}\}$ l'ensemble des patterns d'une solution du programme linéaire, l'ensemble $S = \bigcup_{i=1}^{|\mathcal{P}|} Var(p_i)$ sera donc une solution minimisant la formule DNF du groupe P.

Remarque 16. *Soit une instance $(\Omega, \mathcal{A}, D, G)$ avec deux groupes P et N, une solution S minimisant la formule du groupe P n'est pas nécessairement une solution non-dominée de l'instance, même si la formule est constituée de prime patterns.*

Exemple 24. *Considérons l'instance représentée par le tableau suivant :*

Observations	*Attributs* / *Groupes*	*a*	*b*	*c*	*d*	*e*
1	*P*	*0*	*0*	*0*	*1*	*1*
2		*0*	*0*	*1*	*1*	*1*
3		*1*	*1*	*0*	*0*	*1*
4		*1*	*1*	*0*	*1*	*0*
5	*N*	*1*	*1*	*1*	*0*	*1*
6		*1*	*1*	*1*	*1*	*0*
7		*0*	*0*	*0*	*0*	*0*

La formule $(a \wedge \bar{c}) \vee (d \wedge e)$ est une formule de taille minimale décrivant le groupe P, mais $S = Var(a \wedge \bar{c}) \cup Var(d \wedge e) = \{a, c, d, e\}$ est une solution dominée par la solution $S' = \{c, d, e\}$. Remarquons que S' ne permet pas de créer une autre formule minimale du groupe P.

Remarque 17. *L'algorithme `VS` (algorithme 17) permet de manière très efficace de déterminer pour une liste de solutions minimisant la formule d'un groupe lesquelles sont des solutions non-dominées.*

3.6 Synthèse

Nous avons présenté dans ce chapitre de nombreux algorithmes permettant de résoudre plusieurs problèmes. Nous proposons ici un tableau récapitulatif listant ces algorithmes et leurs utilisations.

Algorithme	Numéro Page	*Nom* Description
NDS	15 p. 89	*Non Dominated Solutions* Calcule l'ensemble des solutions non-dominées
NDS-B	16 p. 91	*Non Dominated Solutions with Bound* Calcule les solutions non dominées de taille inférieure à une borne
VS	17 p. 93	*Vérifie Solution* Vérifie si un ensemble d'attributs est une solution non-dominée
GA-NDS	18 p.94	*Greedy Algorithm : Non Dominated Solution* Algorithme glouton : Calcule une seule solution non-dominée
RNDS	20 p.98	*Rapid Non Dominated Solution* Calcule l'ensemble des solutions non-dominées
MWAD	22 p.102	*Merging With Attributes Denied* Calcul l'ensemble des solutions minimales Détermine les patterns générant la formule logique de taille minimale
Min-RNDS	24 p.104	*Minimal-Rapid Non Dominated Solution* Calcule l'ensemble des solutions minimales Détermine les patterns générant la formule logique de taille minimale
PPC	26 p.107	*Prime Patterns Computation* Calcule l'ensemble des prime patterns et leur couverture
SPPC	27 p.109	*Strong Prime Patterns Computation* Sélectionne les strong prime patterns

4

Expérimentations

Dans ce chapitre nous allons, dans un premier temps, présenter les instances sur lesquelles nous travaillerons ainsi que des outils d'analyse de ces instances. Dans un second temps, nous allons étudier les performances des algorithmes présentés dans le chapitre 3 :

- Les algorithmes résolvant le *NonDom-PCM*, c'est à dire calculant des solutions non-dominées du PCM.
- Les algorithmes résolvant le *MinAll-PCM*, c'est à dire calculant l'ensemble des solutions minimales du PCM.
- Les algorithmes générant l'ensemble des prime patterns pour LAD.

Ensuite, nous comparerons dans la section 4.3 et la section 4.4 les observations au sein des groupes et en fonction de leurs attributs. La Figure 4.1 schématise notre approche :
Soit une instance $(\Omega, \mathcal{A}, D, G)$, cette instance présente des solutions minimales S_{min} (en vert sur la Figure 4.1) pour lesquelles de nombreux patterns sont nécessaires pour couvrir les groupes et présente également des solutions $S_{formule}$ minimisant la taille de la formule DNF pour chaque groupe (en rouge sur la Figure 4.1) constituées de nombreux attributs.

À partir de cette instance, nous étudierons plusieurs aspects :

- En utilisant des méthodes de classification, nous construirons de nouvelles fonctions de groupes (en orange sur la Figure 4.1) que nous comparerons à la fonction originale (Comp1 sur la figure),
- Nous comparerons les solutions minimales S_{min} et celles minimisant la taille de la formule $S_{formule}$ en termes de nombre d'attributs et nombre de patterns nécessaires pour couvrir un groupe (Comp2 sur la figure),
- En méthodes de sélections de variables, nous évaluerons statistiquement l'attribut/l'ensemble d'attributs le plus corrélé à la fonction de groupes (en bleu sur la figure) et nous comparerons cet ensemble d'attributs avec les solutions du PCM (Comp3 sur la figure).

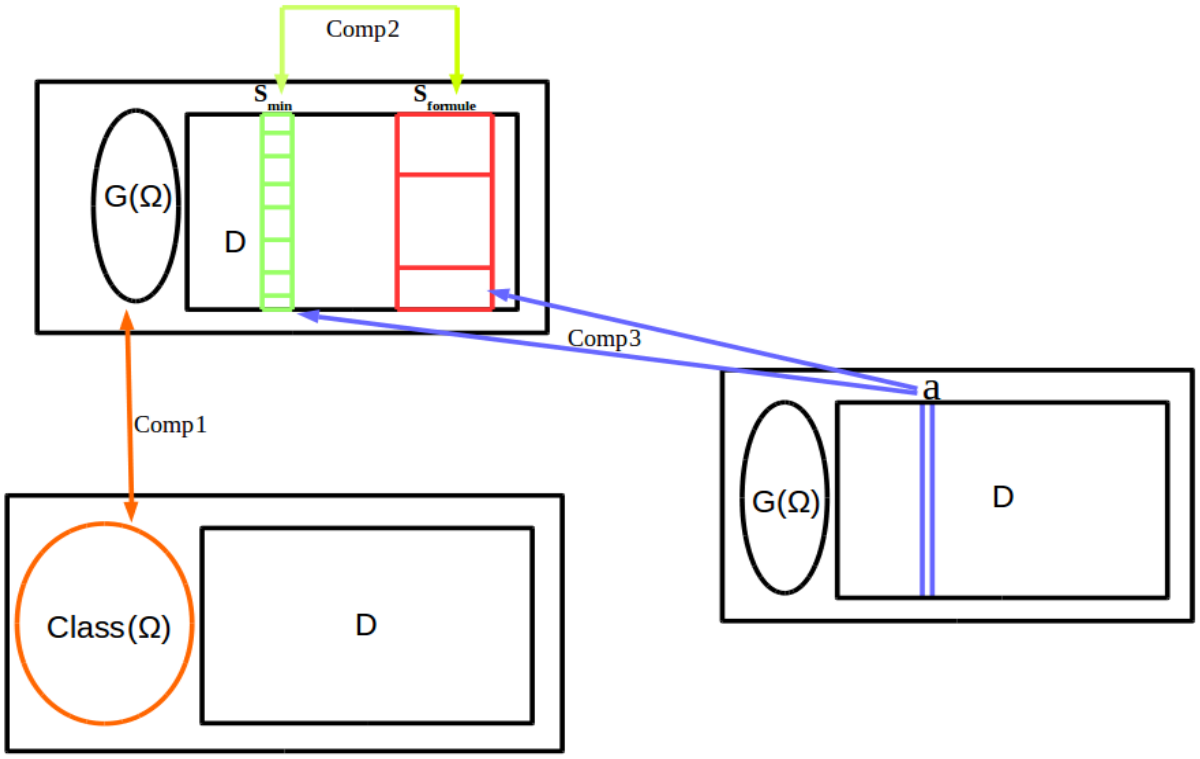


FIGURE 4.1 – Présentation naïve des expérimentations

Enfin, nous étudierons la répartition des solutions du PCM dans la section 4.5 sur plusieurs points :

- en fonction de leur taille,
- la répartition des patterns issus de l'approche LAD et leurs capacités à couvrir les groupes pour résoudre les instances du PCM via l'approche par patterns.

4.1 Description des instances et outils d'analyse

Dans cette section nous commencerons par introduire des mesures pour analyser la diversité des données et pouvoir les comparer.

4.1.1 Mesures de similarité

Les mesures que nous présenterons dans cette section se séparent en deux groupes :

- Similarité des données d'une instance : mesures permettant d'évaluer la ressemblance entre les observations d'un même groupe et la différence entre les observation de groupes différents.
- Comparaison de différentes fonctions de groupes pour un même jeu de données : mesure pour comparer deux classement différent d'un même ensemble d'observations.

4.1.1.1 Mesures de similarité des observations

Dans cette section nous définissons les différentes mesures afin d'étudier les propriétés des observations et leurs distributions dans le groupe. Considérons une instance $(\Omega, \mathcal{A}, D, G)$ et sa matrice de contraintes C.

La définition suivante présente la notion de similarité entre deux observations.

Définition 27. *Soit $(o, o') \in \Omega^2$, $A \subseteq \mathcal{A}$, la similarité $sim(D^A, (o, o'))$ entre o et o' est la moyenne des valeurs du vecteur $D^A_{[o,.]} \odot D^A_{[o',.]}$:*

$$sim(D^A, (o, o')) = \frac{1}{|A|} \times \sum_{a \in A} (D^A[o, a] \odot D^A[o', a])$$

En utilisant la définition précédente, nous pouvons maintenant définir la notion de similarité intragroupe d'un groupe.

Définition 28. *La similarité intragroupe d'un groupe est défini comme la moyenne des similarités entre chaque couple d'observations du groupe.*

Pour $A \subseteq \mathcal{A}$ et un groupe $G_g \subset \Omega$ t.q. $G_g = \{o | G(o) = g\}$,

$$sim_g(D^A) = \frac{2}{(|G_g|)(|G_g| - 1)} \times \sum_{o, o' \in G_g} sim(D^A, (o, o'))$$

La similarité intragroupe d'une instance est ainsi définie par la moyenne des similarités de chaque couple d'observations d'un même groupe.

Définition 29. *Pour $A \subseteq \mathcal{A}$,*

$$sim_{intra}(D^A) = \frac{1}{|\Omega^2 \backslash \Theta|} \times \sum_{(o,o') \in \Omega^2 \backslash \Theta} sim(D^A, (o, o'))$$

La similarité globale d'une instance est définie comme la moyenne des similarités pour chaque couple d'observations.

Définition 30.

$$sim_{over}(D^A) = \frac{1}{|\Omega^2|} \times \sum_{(o,o') \in \Omega^2} sim(D^A, (o, o'))$$

On veut maintenant évaluer la différence entre les observations de groupes différents.

Définition 31. *Soit $(o, o') \in \Omega^2$, $A \in \mathcal{A}$, la difference entre o et o', notée $diff(D^a, (o, o'))$, est la moyenne des valeurs du vecteur $D^A[o, .] \oplus D^A[o', .]$:*

$$diff(D^A, (o, o')) = \frac{1}{|A|} \times \sum_{a \in A} (D^A[o, a] \oplus D^A[o', a])$$

Définition 32. *La différence intergroupe d'une instance est définie comme la moyenne des différences des couple d'observations de groupes différents.*

Pour $A \in \mathcal{A}$,

$$diff_{inter}(D^A) = \frac{1}{|\Theta|} \times \sum_{(o,o') \in \Theta} diff(D^A, (o, o'))$$

Nous avons la propriété suivante.

Proposition 3. *Soit $A \in \mathcal{A}$,*

$$diff_{inter}(D^A) = \frac{1}{|\Theta|} \times \sum_{(i,j) \in \Theta} \sum_{c \in A} C^A_{[(i,j),c]}$$

Calculer la différence intergroupe revient à déterminer la moyenne du nombre de différences entre les observation de l'ensemble Θ.

4.1.1.2 Mesure de similarité des groupes

Dans cette section, nous nous intéressons à la comparaison entre les différentes manières de grouper les observations, i.e. à la comparaison de deux instances $(\Omega, \mathcal{A}, D, G)$ et $(\Omega, \mathcal{A}, D, G')$. Une fonction de groupes G sera proche d'une autre fonction G' si elles ont tendance à regrouper les observations de manière similaire indépendamment des noms des groupes (par exemple, les numéros associés). Dans nos expériences, nous comparons différentes fonctions de groupes et nous voulons définir une notion claire de similarité entre ces fonctions en ce qui concerne les groupes résultants.

Malheureusement, les indicateurs tels que l'indice Jaccard ne sont pas suffisants pour évaluer cette similarité car ils tiennent compte de la valeur des groupes. Par conséquent, il est difficile d'utiliser de tels index s'il y a plus de deux groupes.

Considérons deux instances $(\Omega, \mathcal{A}, D, G)$ et $(\Omega, \mathcal{A}, D, G')$. $\Delta_{G,G'}$ est la matrice de contingence des fonctions de groupes G et G' , i.e., $\Delta_{G,G'}[i,j] = k$ si et seulement si il y a k observations appartenant au groupe i selon G et au groupe j selon G'. Notons que la matrice de contingence est sensible à la valeur du groupe (c'est-à-dire l'indice du groupe). Afin d'éviter cet inconvénient, nous devons considérer les permutations de cette matrice de contingence.

Soit Φ_n l'ensemble de toutes les fonctions de permutation sur $1 \dots n$ (n étant le nombre de groupes). Soit $\varphi \in \Phi$, $\Delta^{\varphi}_{G,G'}$ est la matrice de contingence dont les lignes ont été réarrangées selon φ.

On défini notre indice de similarité entre G et G' ainsi :

$$\sigma_{G,G'} = \frac{max_{\varphi \in \Phi}\ (diag(\Delta^{\varphi}_{G,G'}))}{|\Omega|}$$

où $diag(M)$ est la fonction qui retourne la somme des valeurs de la diagonale de la matrice carré M.

Notons que $argmax_{\varphi \in \Phi}\, diag(\Delta^{\varphi}_{G,G'})$ retourne la meilleures permutation du nom des groupes entre G et G'. Notons que, pour une matrice de contingence donnée de taille $|n| \times |n|$, il y a $n!$ permutations possibles. Ainsi, si nous avons de nombreux groupes, calculer σ peut nécessiter beaucoup de temps.

Exemple 25. *Soit G, G', G'' trois fonctions de groupes. On représente ici ces fonctions par les vecteurs suivants, $G = [1,1,1,1,2,2,2,2,3]$, $G' = [2,2,2,2,3,3,3,3,1]$ et $G'' = [1,1,2,2,1,2,3,3,3]$. $G([o])$ (resp. $G'[o]$ et $G''[o]$) représente l'indice du groupe assigné à l'observation o avec la fonction G (resp. G' et G''). Dans notre exemple, $|\Omega| = 9$.*

On a $\Delta_{G,G'} = \begin{pmatrix} 0 & 4 & 0 \\ 0 & 0 & 4 \\ 1 & 0 & 0 \end{pmatrix}$

Il existe $\varphi \in \Phi$ tel que $\Delta^{\varphi}_{G,G'} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix}$

$diag(\Delta^{\varphi}_{G,G'}) = 4 + 4 + 1 = 9$ *et donc* $\sigma_{G,G'} = \frac{9}{9} = 1$.

On a $\Delta_{G,G''} = \begin{pmatrix} 2 & 2 & 0 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$ *et* $\sigma_{G,G''} = \frac{4}{9}$.

Par conséquent, nous observons que G et G' sont similaires (par construction, malgré les valeurs des groupes assignés) alors que G et G'' sont plus différents.

Notons que, pour une instance $(\Omega, \mathcal{A}, D, G)$, avec $G : \Omega \rightarrow \{1, \cdots, nb_G\}$ (où nb_G est le nombre de groupes), et une autre fonction de groupes $G' : \Omega \rightarrow \{1, \cdots, nb_G\}$, la similarité entre G et G' est bornée :

$$\sigma_{G,G'} \in [\frac{\lceil |\Omega|/nb_G \rceil}{|\Omega|}, 1]$$

4.1.2 Instances

Nous utiliserons dans ce chapitre de nombreuses instances issues pour la plupart de problèmes biologiques. Les caractéristiques de ces instances sont décrites dans la table 4.1 où :

- *Instances* est le nom des instances,
- *Observations* est le nombre d'observations $|\Omega|$,
- *Attributs* est le nombre d'attibuts $|\mathcal{A}|$,
- *Groupes* est le nombre de groupes de l'instance,

Instances	Observations	Attributs	Groupes
rch8	132	37	21
ra_rep1	112	155	7
ra_rep2	112	155	7
ra_phy	113	155	4
ra_phv	109	155	8
ra100_phy	113	99	4
ra100_phv	109	99	8
xanthostgen	173	107	33
ralstophy2s	112	34	4
Random	20	35	2
vote_r	435	16	2
cr60	289	14	2
os1	289	14	2
rel1	259	14	2
Leukemia	35	8389	2
HM(G_1)	34	17	2
HM(G_2)	34	17	2

TABLE 4.1 – Caractéristiques des instances

Les instances sont constituées comme suit :

- Les instances *rch8, ra_rep1, ra_rep2, ra_phy, ra_phv, ra100_phy, ra100_phv, xanthostgen* et *ralstophy2s* [1] sont tirées de problèmes de caractérisation provenant en partie de

[1] http ://www.info.univ-angers.fr/~gh/Idas/Ccd/ce_f.php

l'API BioMérieux[2] basée sur des propriétés biochimiques de l'espèce Ralstonia et sur des expressions de gènes de virulences de l'espèce Xanthomonas. Ces propriétés biochimiques sont un ensemble de fonctions métaboliques de la bactérie comme la présence d'une activité enzymatique particulière. Une description plus détaillée de ces instances est formulée dans l'annexe p.166.

- *Random* est une instance aléatoire constituée de deux groupes (positif et négatif) construite avec seulement une observation dans le groupe positif, et composée de valeurs binaires déterminées aléatoirement et indépendamment des autres. Cette instance est utilisée comme un test basique et permet, par son unique observation positive, d'obtenir des résultats dépendant uniquement du nombre d'attributs et non du nombre de combinaisons d'observations.

- L'instance *vote_r*[3] est une instance composée de données binaires. Cet ensemble de données comprend des votes pour chacun des membres de la Chambre des représentants des États-Unis sur les 16 votes clés identifiés par le CQA. Les attributs sont des informations sur des sujets politiques divers, comme par exemple le parti politique du votant (démocrate ou républicain) ou sa position en ce qui concerne l'immigration (favorable ou non). Cette instance a des données manquantes que l'on a complétées par des valeurs binaires aléatoires. Une description plus détaillée de cette instance est formulée dans l'annexe p.163.

- Les instances *cr60*, *os1* et *rel1* sont des jeux de données correspondant à des patients souffrant de leucémie. Les attributs sont multiples (caractères phénotypiques, génomiques, caryotype, ...). Ici, l'objectif est de trouver des gènes qui pourraient aider à améliorer le pronostic et à sélectionner les traitements les plus appropriés selon les profils des patients. Une description plus détaillée de ces instances est formulée dans l'annexe p.164.

- L'instance *Leukemia* est également un jeu de données basé sur des patients souffrant de leucémie. Les observations sont des patients souffrant de leucémies et les attributs sont des gènes mutés qui sont soupçonnés de jouer un rôle dans la leucémie.. L'objectif de la caractérisation est d'identifier une combinaison de gènes afin de prédire le risque de rechute d'un patient. Le génome humain est plus grand qu'une bactérie, nous avons alors un nombre d'attributs plus important dans cette instance. Une description plus détaillée de cette instance est formulée dans l'annexe p.165.

Instances "faites main"

Afin de considérer des cas particuliers avec des propriétés extrêmes spécifiques, nous construi-

[2]http ://www.biomerieux-usa.com/clinical/api
[3]http ://tunedit.org/repo/UCI/vote.arff

sons deux instances de plus, les instances HM(G_1) et HM(G_2).
Considérons $|\Omega| = 34$ et $|\mathcal{A}| = 17$ ($\mathcal{A} = \{a, b, c, d, ..., p, q\}$) et la matrice de données suivante :

$$\begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 \\ 1 & \cdots & 1 & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

On considère deux fonctions de groupes $G_1, G_2 : \Omega \mapsto \{1, 0\}$ tel que :

- $G_1(o_i) = 1$ si l'indice de l'observation $i < 18$, $G_1(o_i) = 2$ if $i \geq 18$.
- $G_2(o_i) = 1$ si l'indice de l'observation i est pair, $G_2(o_i) = 2$ si i est impair.

Nous obtenons les valeurs suivantes :

Fonction de groupes	Taille de la solution minimale	sim_{over}	similarité intragroupe	différence intergroupe
G_1	3	0.485	0.882	0.889
G_2	8	0.485	0.478	0.508

Notons que les patterns $p_1 = a \wedge b$ et $p_2 = c \wedge d$ couvrent ensemble le groupe 1 obtenu avec la fonction de groupes G_1.
Les patterns $p_3 = b \wedge d \wedge f \wedge h \wedge j \wedge l \wedge n \wedge p$ et $p_4 = \bar{a} \wedge \bar{c} \wedge \bar{e} \wedge \bar{g} \wedge \bar{i} \wedge \bar{k} \wedge \bar{m} \wedge \bar{o} \wedge \bar{q}$ couvrent ensemble le groupe 1 obtenu avec la fonction de groupes G_2.

Ici, nous pouvons voir que pour la même matrice de données, un groupe d'observations donnant une forte similarité intragroupe et une forte différence intergroupes nous donnera des solutions plus petites.

4.2 Performance des algorithmes

Dans cette section, nous souhaitons comparer les temps d'exécution de nos algorithmes entre eux mais également par rapport à d'autres algorithmes de la littérature présentés dans le chapitre 2. Nous comparerons dans un premier temps les algorithmes permettant de calculer l'ensemble (ou un sous-ensemble) des solutions non-dominées, dans un second temps ceux permettant de

calculer uniquement celles de taille minimale et enfin, dans un troisième temps, nous comparerons les algorithmes permettant de générer l'ensemble des prime patterns pour l'analyse logique de données.

4.2.1 Algorithmes résolvant le *NonDom-PCM*

La performance des algorithmes présentés dans la section 3.2 va maintenant être évaluée sur un ensemble d'instances.

Instances étudiées :

- *rch8* || *ra_rep1* || *ra_rep2* || *ra_phy* || *ra_phv* || *ra100_phy* || *ra100_phv* || *xanthostgen* || *ralstophy2s* : instances présentées dans la Table 4.1.

Algorithmes étudiés :

- `GA-NDS` || `NDS-B` || `NDS` || `RNDS` : algorithmes présentés dans le chapitre 3,
- `MMCS` : algorithme présenté dans Murakami and Uno [2014].

Remarque 18. *Le lien du code C de l'algorithme `MMCS` est fourni dans l'article*[4].

Observations :

- *Taille solution `GA-NDS` :* la taille de la solution retournée par l'algorithme `GA-NDS` (algorithme 18). Pour chaque instance, la taille de la solution trouvée est identique à celle de l'algorithme glouton de Chvatal (algorithme 14), mais la solution trouvée est nécessairement non-dominée.
- *`NDS-B` avec `GA-NDS` :* le temps de calcul et le nombre de solutions trouvées par l'algorithme `NDS-B` (algorithme 16) en utilisant la valeur de la colonne précédente comme borne.
- *`NDS` :* le temps de calcul de l'algorithme NDS (algorithme 15).
- *`RNDS` :* le temps de calcul de l'algorithme RNDS (algorithme 20).
- *`MMCS` :* le temps de calcul de l'algorithme MMCS.

Conditions expérimentales :

- Les expériences ont été lancées sur un ordinateur avec Intel Core i7-4910MQ CPU (8×2.90 GHz), 31.3 GB RAM..
- Le délai d'exécution a été limité à 6H et "-" signifie un temps d'exécution supérieur à cette limite. Pour les cas où les algorithmes ont dépassé la limite, les fichiers retournés représentant l'ensemble des solutions trouvées jusque là étaient trop important pour être ouvert par un ordinateur de 31.3 GB RAM.

[4]http ://research.nii.ac.jp/ uno/dualization.html

La Table 4.2 fournit les résultats obtenus sur les instances. Les 3 premières colonnes sont le nom de l'instance, le nombre de contraintes générées $|\Theta|$ et le nombre de solutions du PCM. Les 5 colonnes suivantes présentent les résultats obtenus par nos algorithmes comme décrit dans le tableau des observations. Les temps de calculs sont exprimés en secondes.

Instance	Nb contr.	Nb solutions	Taille solution GA−NDS	NDS−B avec GA−NDS		NDS	RNDS	MMCS
				Time	# Sol.			
rch8	8229	980	10	0.48	222	0.95	0.12	**0.02**
ra_rep1	4587	3402921	14	113.00	37382	-	245.51	**12.256**
ra_rep2	4609	8481769	13	132.04	42067	-	877.29	**36.829**
ra_phy	3479	-	7	3719.52	4576	-	-	-
ra_phv	4146	-	7	3760.59	1853	-	-	-
ra100_phy	3479	2236420	8	**10.23**	860	-	1064.52	16.36
ra100_phv	4146	2029133	10	76.18	3674	-	742.90	**14.88**
xanthostgen	14293	-	14	-	164537	-	-	-
ralstophy2s	3374	267	9	0.21	162	0.14	0.05	**0.02**

TABLE 4.2 – Résultats des algorithmes résolvant le *NonDom-PCM*

Dans un premier temps, nous constatons que le nombre de solutions non-dominées est rapidement très important. La borne obtenue permet de fortement limiter le nombre de solutions générées. Nous verrons en effet dans la section 4.5.1 la répartitions des solutions en fonction de leur taille sur quelques instances et nous constaterons que les solutions déterminées par ces algorithmes font en effet partie des plus petites.
Concernant les algorithmes, on constate que :

- Les algorithmes MMCS et RNDS sont bien plus efficaces que NDS.
- MMCS est plus rapide que RNDS. Les algorithmes étant très proches, il est probable que cette différence de temps puisse s'expliquer par les techniques de programmation ou le choix des structures de données.
- L'algorithme NDS−B donne des performances intéressantes pouvant dans certains cas rivaliser avec RNDS et même MMCS, mais ne propose qu'un sous-ensemble de solutions. Toutefois, au vue du grand nombre de solutions, il peut être intéressant dans certains cas de ne considérer que les plus petites.

Notons que les instances *ra_phy* et *ra_phv* sont particulièrement difficile à résoudre par rapport à la taille des jeux de données. Dans le cas de la résolution du PCM sur ces instances avec l'algorithme NDS−B et la borne obtenue par GA−NDS, nous constatons que malgré le fait qu'il y ait peu de solutions obtenues, les temps de calcul sont beaucoup plus importants que pour

les autres instances. Une explication probable est qu'il existe beaucoup de solutions du PCM sur ces instances, mais que très peu sont de petite taille. Ainsi, lors des premières itérations, le grand nombre de solutions impliquerait que les algorithmes soient très coûteux en temps de calcul, mais nous ne conservons au final que très peu de solutions de taille inférieure à la borne.

4.2.2 Algorithmes résolvant le *MinAll-PCM*

Dans cette sous-section, nous évaluons les performances des algorithmes résolvant le Min-PCM.

Instances étudiées :

- *rch8* ‖ *ra_rep1* ‖ *ra_rep2* ‖ *ra_phy* ‖ *ra_phv* ‖ *ra100_phy* ‖ *ra100_phv* ‖ *xanthostgen* ‖ *ralstophy2s* : instances présentées dans la Table 4.1.

Algorithmes étudiés :

- `NDS-B` ‖ `MWAD` ‖ `Min-RNDS` : algorithmes présentés dans le chapitre 3,
- `CPLEX`[5](`CPLEX-CP` ‖ `CPLEX-LP`) : il s'agit un solveur de programmation linéaire. Nous l'utilisons pour résoudre le programme linéaire associé à la résolution du min-PCM (voir section 2.3.4.2). Fondamentalement, les contraintes de la matrice de contraintes C sont codées en inéquations pseudo-booléennes linéaires. Par conséquent, `CPLEX` requière la génération de toutes les contraintes. Pour de grandes instances, le solveur `CPLEX` peut nécessiter une quantité excessive de mémoire. Notons que si `CPLEX` peut renvoyer des solutions minimales, il est impossible de gérer la non-dominance. `CPLEX` possède beaucoup de paramètres, donc, pour les configurer, nous utilisons l'outil de réglage proposé par `CPLEX`. Nous comparons nos méthodes avec deux solveurs possibles inclus dans `CPLEX` : un solveur de programmation par contraintes `CPLEX-CP` et un solveur de programmation linéaire `CPLEX-LP` (partie entière). Notons que le solveur de programmation par contraintes n'est capable de renvoyer qu'une seule solution, tandis que `CPLEX-LP` renvoie toutes les solutions.

Observations :

- *NDS_{Kmin} :* le temps d'exécution de l'algorithme 16 avec la borne $B = Kmin$ valant la taille des solutions minimales.
- *`MWAD` :* le temps d'exécution de l'algorithme 22.
- *`Min-RNDS` :* le temps d'exécution de l'algorithme 24.
- *`CPLEX-CP` :* Le temps d'exécution du solveur `CPLEX` en programmation par contraintes.
- *`CPLEX-LP` :* Le temps d'exécution du solveur `CPLEX` en programmation linéaire.

[5]http ://www.ibm.com/software/integration/optimization/cplex-optimizer/

Remarque 19. *Étant donné que* `CPLEX` *requiert la matrice de contraintes complète, nous prenons en compte le temps correspondant à sa génération.*

Conditions expérimentales :

- Les expériences ont été lancées sur un ordinateur avec Intel Core i7-4910MQ CPU (8×2.90 GHz), 31.3 GB RAM.
- Les algorithmes de `CPLEX` sont parallélisés.

La Table 4.3 fournit les résultats obtenus sur les instances. Les quatre premières colonnes sont le nom de l'instance, le nombre de contraintes générées $|\Theta|$, la taille $Kmin$ de la solution minimale et le nombre de solutions minimales. Les 3 prochaines colonnes donnent les résultats obtenus par nos algorithmes. Enfin, les deux dernières colonnes correspondent aux résultats `CPLEX` (voir description dans le tableau des observations pour plus de détails). Les temps de calculs sont exprimés en secondes.

Instance	# Contr.	$Kmin$	# Sol. opt.	$\texttt{NDS}_{Kmin}$	MWAD	Min−RNDS	CPLEX−CP	CPLEX−LP
							1 sol.	toutes sol.
rch8	8229	9	16	0.40	0.11	**0.09**	1.70	1.66
ra_rep1	4587	12	14	0.72	**0.18**	0.19	3.32	2.96
ra_rep2	4609	12	1298	4.86	3.93	**1.30**	6.06	23.58
ra_phy	3479	6	4	98.32	**2.63**	90.51	75.87	15.94
ra_phv	4146	6	6	94.31	**2.37**	379.09	95.37	15.83
ra100_phy	3479	7	63	1.16	**0.29**	1.83	6.78	11.00
ra100_phv	4146	9	119	6.32	**1.23**	3.77	15.54	17.00
xanthostgen	14293	13	2410	1529.00	126.06	122.12	58713.00	**56.59**
ralstophy2s	3374	8	26	0.05	0.04	**0.03**	0.92	0.59

TABLE 4.3 – Résultat des algorithmes résolvant le *MinAll-PCM*

Nous remarquons que :

- Malgré la parallélisation de `CPLEX`, `MWAD` et `Min−RNDS` sont globalement plus rapides. Notons également que `CPLEX−LP` utilise beaucoup plus de mémoire que les autres algorithmes.
- $\texttt{NDS}_{Kmin}$ est le plus lent des algorithmes. Non seulement, il n'est pas capable de déterminer la taille des solutions minimales sans résoudre le Min-PCM (donc il ne peut déterminer $Kmin$ en pratique), mais même s'il l'obtient avec un algorithme glouton, il reste le moins efficace.

- Concernant `Min-RNDS`, il semble rapide sauf pour les instances *ra_phy* et *ra_phv*. Toutefois, pour les quelques instances où il est plus rapide que `MWAD`, la différence de temps d'exécution est faible.

- Au final, `MWAD` semble être globalement le plus efficace, alors que `NDS` (dont `MWAD` est tiré) est beaucoup moins efficace que `RNDS` (dont `Min-RNDS` est tiré) pour résoudre les instances du PCM d'après la section 4.2.1.

4.2.3 Algorithmes générant les prime patterns

Dans cette section, nous comparons les performances de notre algorithme de génération de l'ensemble des prime patterns `PPC` (voir section 3.4) avec l'algorithme `Algo_Boros` (voir section 2.2.3.2) qui génère également l'ensemble des prime patterns.

4.2.3.1 Données des instances

Nous considérons ici plusieurs instances présentées dans la Table 4.1 dont certaines seront modifiées afin de correspondre au contexte LAD.

Instances étudiées :

- *Random* où le groupe P (positif) est limité à seulement une observation pour observer le comportement des algorithmes déterminant les prime patterns pour une observation.

- *rch8*, *ra_rep2* et *ralsto_phy2s* : Initialement, beaucoup de groupes sont proposés dans ces instances. Toutefois, nous n'étudions ici que le premier groupe de souches bactériennes comme groupe positif et l'union des autres groupes sera considérée comme le groupe négatif. Les résultats sont similaires si on considère un autre groupe comme le groupe positif.

- *vote_r* || *cr60* || *os1* || *rel1*.

Les instances sont décrites plus précisément dans la Table 4.4 avec leur nombre d'observations, leur nombre d'observations dans le groupe positif et le nombre maximal d'attributs. Pour chaque instance, nous expérimenterons les algorithmes sur des sous-ensembles d'attributs de tailles différentes pour évaluer les performances en fonction de ce nombre d'attributs.

4.2.3.2 Résultats

Algorithmes étudiés :

- `PPC :` algorithme présenté dans le chapitre 3,

- `Algo_Boros :` algorithme de génération de prime patterns issu de Boros et al. [2000].

Instances	observations	taille du groupe positif	attributs
Random	20	1	35
rch8	132	5	37
vote_r	435	168	16
ra_rep2	112	37	73
ralsto_phy2s	73	27	23
cr60	289	58	14
os1	289	224	14
rel1	259	200	14

TABLE 4.4 – Caractéristiques des instances

Remarque 20.

- `PPC` *(Algorithm 26) utilise le principe présenté dans la section 3.4, codé en C++ avec les structures de données et opérateurs de la librairie* boost [6].
- `Algo_Boros` *a été représenté dans la section 2.2.3.2. Notons que le code source présenté dans [Boros et al., 2000; Chikalov et al., 2013] n'est plus disponible. Il a été réimplémenté en C++ en utilisant les mêmes structures de données et opérateurs de la librairie* boost *que pour* `PPC`.

Observations :

- `PPC(RNDS)` *:* le temps d'exécution de l'algorithme `PPC` en utilisant l'algorithme `RNDS` comme fonction calculant les solutions non-dominées du PCM dont on a besoin pour la génération de patterns (voir section 3.4).
- `PPC(NDS)` *:* le temps d'exécution de l'algorithme `PPC` en utilisant l'algorithme `NDS` comme fonction calculant les solutions non-dominées du PCM dont on a besoin pour la génération de patterns.
- `Algo_Boros` *:* le temps d'exécution de l'algorithme de génération de prime patterns issu de Boros et al. [2000].
- `Algo_Boros` *avec taille max :* le temps d'exécution de `Algo_Boros` utilisé avec une borne égale à la taille du plus grand prime pattern. Bien sûr, lorsque l'on utilise cette borne, nous obtenons les mêmes résultats, mais `Algo_Boros` devient plus rapide car il peut s'arrêter plus tôt. Notons qu'en pratique, la valeur de cette borne n'est pas connue jusqu'à ce que l'ensemble des prime patterns ait été calculé.

[6] http ://www.boost.org/doc/libs/1_36_0/libs/dynamic_bitset/dynamic_bitset.html

Conditions expérimentales :

- Les expériences ont été lancées sur un ordinateur avec Intel Core i7-4910MQ CPU (8×2.90 GHz), 31.3 GB RAM.
- Le délai d'utilisation est limité à 24 heures. "- " correspond à des temps d'exécution supérieurs à cette limite.
- Le nombre d'attributs utilisés pour chaque instance varie et est inscrit à coté du nom de l'instance.

La Table 4.5 fournit les résultats obtenus sur les instances pour calculer l'ensemble des prime patterns :

- La première colonne correspond au nom de l'instance, avec le nombre n des attributs utilisés,
- La deuxième colonne correspond au nombre de prime patterns pour chaque instance,
- Les trois colonnes suivantes sont les résultats obtenus pour les algorithmes `PPC(RNDS)`, `PPC(NDS)` (i.e. l'algorithme `PPC` en utilisant respectivement les algorithmes `RNDS` et `NDS` comme fonctions calculant les solutions non-dominées) et `Algo_Boros` (voir description dans le tableau des observations),
- Les deux dernières colonnes correspondent à la taille maximale des patterns calculés (en termes de nombre d'attributs) et les résultats de `Algo_Boros` *avec taille max*.

Les temps de calculs sont exprimés en secondes.

Nous remarquons que :

- Le temps d'exécution de chaque algorithme augmente bien évidemment dès lors que le nombre d'observations augmente, mais surtout, dès lors que le nombre d'attributs augmente. Toutefois, `PPC` est beaucoup moins sensible à ces augmentations d'observations et d'attributs que ne l'est `Algo_Boros`.
- En étendant le temps de calcul à plus de 24 heures, nous avons remarqué que `Algo_Boros` est capable de générer l'ensemble des prime patterns de l'instance *Random(35)* en 3 jours. De même nous avons pu remarquer qu'une semaine n'est pas suffisant concernant l'instance *rch8(37)*. `PPC(NDS)` est capable de générer l'ensemble des prime patterns de l'instance *ra_rep2(65)* en un peu plus d'une heure tandis que `Algo_Boros` n'est pas capable de résoudre le problème sur la même instance avec seulement 20 attributs en un temps raisonnable.

Instance	nombre de prime patterns	temps PPC(RNDS)	temps PPC(NDS)	temps Algo_Boros	taille max	Algo_Boros avec taille max
Random(11)	61	0.008	**0.003**	0.006	4	0.003
Random(15)	170	0.009	**0.011**	0.029	5	0.021
Random(20)	476	0.021	**0.020**	2.306	6	1.185
Random(25)	1143	**0.026**	0.028	113.469	6	16.343
Random(27)	1529	**0.030**	0.052	338.708	6	38.045
Random(30)	2382	**0.052**	0.110	4154.002	6	170.906
Random(35)	4505	**0.111**	0.283	-	6	1668.373
rch8(15)	1	0.005	**0.004**	5.988	2	0.004
rch8(20)	7	**0.009**	**0.009**	4261.190	4	0.163
rch8(25)	26	0.017	**0.012**	-	4	0.676
rch8(30)	43	0.019	**0.016**	-	4	2.384
rch8(37)	131	0.030	**0.021**	-	6	77836.071
vote_r(10)	169	0.092	**0.060**	0.876	6	0.669
vote_r(13)	1047	0.376	**0.250**	52.711	8	48.427
vote_r(16)	4454	1.092	**0.842**	4138.240	9	3466.591
ra_rep2(10)	11	0.004	**0.002**	0.096	4	0.008
ra_rep2(15)	46	0.006	**0.005**	133.096	4	0.158
ra_rep2(20)	126	0.008	**0.007**	-	6	413.723
ra_rep2(25)	303	0.017	**0.009**	-	7	-
ra_rep2(30)	745	0.027	**0.017**	-	7	-
ra_rep2(35)	2309	0.105	**0.060**	-	9	-
ra_rep2(40)	6461	0.496	**0.403**	-	10	-
ra_rep2(45)	17048	**2.226**	2.315	-	10	-
ra_rep2(50)	43762	**11.004**	14.596	-	10	-
ra_rep2(55)	101026	78.887	**68.378**	-	11	-
ra_rep2(60)	254042	**529.799**	840.734	-	12	-
ra_rep2(65)	720753	**4128.182**	4617.556	-	14	-
ra_rep2(73)	2474630	**50109.837**	60740.333	-	15	-
ralstophy(10)	22	0.006	**0.003**	0.047	5	0.029
ralstophy(15)	132	0.023	**0.008**	35.286	6	6.839
ralstophy(20)	361	0.031	**0.013**	39725.665	6	436.524
ralstophy(23)	1073	0.088	**0.040**	-	8	-
cr60(10)	55	0.027	**0.018**	0.923	6	0.585
cr60(14)	196	0.033	**0.026**	350.849	8	219.009
os1(10)	107	0.040	**0.018**	0.958	6	0.571
os1(14)	286	0.061	**0.035**	379.423	7	102.318
rel1(10)	135	0.032	**0.024**	0.976	6	0.604
rel1(14)	388	0.071	**0.033**	376.540	7	107.410

TABLE 4.5 – Performance des algorithmes générant les prime patterns

- Dans `Algo_Boros`, le nombre d'itérations est lié au nombre d'attributs. On observe dans la dernière colonne qu'en utilisant la bonne borne (borne égale à la taille du plus grand pattern), les prime patterns peuvent être générés plus rapidement. Toutefois, le temps d'exécution reste toujours beaucoup plus important qu'avec les algorithmes `PPC`. De plus, pour appliquer cette approche, il faudrait connaitre la taille du plus grand pattern.

- Si `RNDS` est plus efficace que `NDS` pour déterminer l'ensemble des solutions non dominées (voir section 4.2.1), il est pourtant plus efficace d'utiliser `PPC(NDS)` pour générer les prime patterns des instances avec un faible nombre d'attributs. Toutefois, dès que le nombre d'attributs devient important, `PPC(RNDS)` devient plus efficace.

4.3 Étude des groupes

Le but de nos expériences est d'étudier avec précision les propriétés des solutions trouvées pour le PCM selon les propriétés topologiques des instances et en particulier les caractéristiques des groupes définis dans les instances. Étant donné qu'il s'agit de caractériser des groupes de données, nous considérons les méthodes classiques qui peuvent être utilisées pour regrouper les données en classes en utilisant des distances. Notre objectif est d'étudier les propriétés des groupes en ce qui concerne leur similarité ainsi que les propriétés des solutions que nous calculons en fonction de différentes fonctions de groupes sur des observations similaires.

4.3.1 Protocole expérimental

La Figure 4.2 présente notre protocole expérimental afin d'examiner avec précision la notion de solution dans le contexte de PCM. En particulier, nous cherchons à étudier si les attributs identifiés dans les solutions ainsi que la taille des solutions peuvent être liés aux propriétés structurelles de l'instance.

Pour chaque instance $(\Omega, \mathcal{A}, D, G)$ nous considérons plusieurs fonctions de groupes :

- La fonction de groupes d'origine G correspond à des groupes définis par des experts dans des cas réels (G en haut à gauche et en haut à droite dans la figure 4.2) ;

- Les fonctions de groupes obtenues par des méthodes de classification sur les données initiales ($kmeans_D$ en bas à gauche dans la Figure 4.2) et sur les données réduites aux attributs des solutions ($kmeans_S$ en bas à droite dans la Figure 4.2). Les variantes suivantes sont prises en considération :

  – des fonctions de groupes correspondant aux groupes générés par la méthode K-means (voir la sous-section 2.5.1.2),

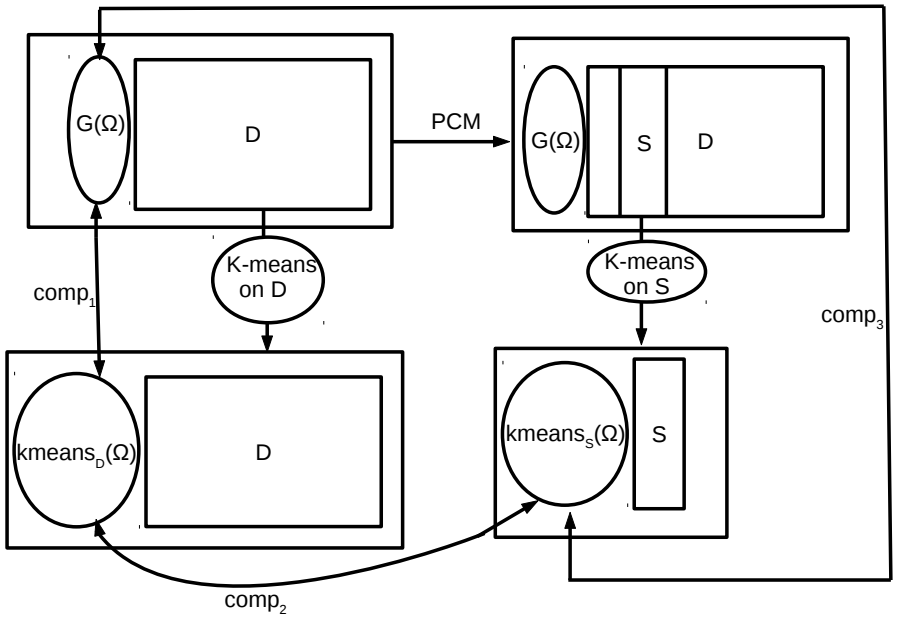


FIGURE 4.2 – Les comparaisons à étudier

- des fonctions de groupes correspondant aux groupes générés par la méthode K-means avec des centroides fixés, appelées ici *center k-means*,
- des fonctions de groupes correspondant aux groupes générés par la méthode K-médoïdes (voir la sous-section 2.5.1.2),
- des fonctions de groupes générées aléatoirement.

Notons que toutes ces fonctions considèrent le même nombre de groupes, correspondant au nombre de groupes de la fonction de groupes d'origine des instances initiales.

Le but de cette première analyse, correspondant à $comp_1$ dans la Figure 4.2, est d'évaluer l'impact de la fonction de groupes sur la taille des solutions ainsi que sur la similarité des groupes.

Pour chaque instance avec les fonctions de groupes mentionnées ci-dessus, nous calculons des solutions (flèche PCM dans la Figure 4.2). À partir de ces solutions, nous considérons encore les mêmes groupes mais restreints à l'ensemble des attributs qui apparaissent dans les solutions.

Les comparaisons $comp_2$ et $comp_3$ correspondent donc à l'analyse de l'impact des attributs des solutions sur les groupes issus des différentes fonctions de groupes par rapport à la fonction de groupes d'origine ($comp_3$) et aux fonctions de groupes résultant d'autres méthodes de classification ($comp_2$).

4.3.2 Classement des solutions

En utilisant les algorithmes décrits dans la section 3.2, de nombreuses solutions non dominées peuvent être calculées. Par conséquent, choisir la solution la plus appropriée est un problème difficile car nous n'avons pas de critères objectifs. Néanmoins, comme le but de la résolution du PCM est de caractériser des groupes d'observations, nous pouvons analyser l'influence des attributs sélectionnés d'une solution sur la topologie du groupe. En utilisant les mesures définies dans la section 4.1.1, pour d'une instance $(\Omega, \mathcal{A}, D, G)$, une solution S pourrait être préférée à une solution S' si :

- les observations appartenant au même groupe projetées sur S sont plus similaires que sur S' (i.e., $sim_{intra}(D^S) > sim_{intra}(D^{S'})$, ou
- les observations de différents groupes projetées sur S sont plus différentes que sur S' (i.e. $diff_{inter}(D^S) > diff_{inter}(D^{S'})$.

Le premier cas correspond à des solutions qui ont tendance à se concentrer sur des attributs qui ont des valeurs similaires pour les observations d'un même groupe. Notons que la similarité intragroupe a un impact sur les patterns qui peuvent être calculés à partir du point de vue LAD. Par conséquent, en ce qui concerne ce critère, pour une instance $(\Omega, \mathcal{A}, D, G)$ et un ensemble de solutions $\mathcal{S}ol$, nous désignons $\bar{S} \in \mathcal{S}ol$ les solutions telles que $\forall S' \in \mathcal{S}ol$, $sim_{intra}(D^{\bar{S}}) \geq sim_{intra}(D^{S'})$ et $\underline{S} \in \mathcal{S}ol$ les solutions telles que $\forall S' \in \mathcal{S}ol, sim_{intra}(D^{\underline{S}}) \leq sim_{intra}(D^{S'})$.

4.3.3 Comparaison des groupes

Dans cette section, nous nous intéressons à la comparaison des différentes manières de grouper les observations, i.e. à la comparaison de deux instances $(\Omega, \mathcal{A}, D, G)$ et $(\Omega, \mathcal{A}, D, G')$. L'objectif est d'étudier différentes fonctions de groupes obtenues avec des méthodes de classification et de les comparer pour déterminer s'il y a corrélation entre la structure de l'instance et la fonction de groupes originale.

4.3.3.1 Description des instances étudiées

Instances étudiées :

- *xanthostgen* ‖ *ra_rep1* ‖ *ra_rep2* ‖ *ra_phy* ‖ *ra_phv* ‖ *ra100_phy* ‖ *ra100_phv* ‖ *ralstophy2s* ‖ *Leukemia* : instances réelles présentées dans la Table 4.1,
- *HM(G_1)* ‖ *HM(G_2)* : instances faites main présentées également dans la Table 4.1.

Les instances théoriques $HM(G_1)$ et $HM(G_2)$ permettront de visualiser les résultats que nous pouvons obtenir avec un "bon" jeu de données et avec un "mauvais" jeu de données dans le sens

où un bon jeu de donnée possède des observations de même groupe très proches en terme d'attributs et un mauvais jeu de donnée possède des observations de groupes différents très proches. Rappelons que ces deux instances possèdent le même jeu de données et seule la répartition des observations dans les groupes change. Faire une classification sur une de ces instances est donc équivalent à faire la même classification sur l'autre.

4.3.3.2 Calcul des solutions minimales

La table 4.6 fournit la taille des solutions minimales pour chaque instance avec différentes fonctions de groupes.

Nous pouvons constater que les groupes aléatoires sont plus difficiles à caractériser et nécessitent donc plus d'attributs, car, comme prévu, aucune régularité ne peut être exploitée dans les données. Les autres résultats montrent que les groupes initiaux et les groupes obtenus selon les méthodes de classification ne sont pas si différents. Par conséquent, une analyse plus poussée devrait être effectuée sur ces groupes.

4.3.3.3 Comparaison des fonctions de groupes

Afin de comparer les différentes fonctions de groupes présentées dans la section 4.3.1, nous utilisons d'abord la fonction objectif de la méthode k-means.

Rappel de la sous-section 2.5.1.2 :
Étant donné un ensemble Ω d'observations, la méthode K-means vise à créer k clusters $S_1, \cdots, S_k$ tels que $\bigcup_{1 \leq i \leq k} S_i = \Omega$, optimisant la fonction objectif suivante :

$$min \sum_{i=1}^{k} \sum_{o \in S_i} ||o - \mu_i||^2$$

où μ_i est le centroide (moyenne des valeurs des points) de S_i.

Cette valeur nous fournit des informations sur la structure des groupes en termes de concentration d'observations. Étant donnée une instance, pour chaque groupe, nous calculons la valeur de la fonction objective de la méthode k-means. Ces valeurs sont présentées dans le tableau 4.7 ainsi que leurs valeurs normalisées. En effet, plus la taille de la matrice de données est élevée (en nombre d'observations et d'attributs), plus la valeur de la fonction objectif est élevée. Par conséquent, nous normalisons ces valeurs en fonction du nombre d'observations et du nombre d'attributs.

La table 4.8 fournit la similarité entre les fonctions de groupes avec la fonction de groupes d'origine en fonction de l'indice de similarité défini dans la section 4.3.3 ($comp_1$ dans Fig 4.2). Notons que pour l'instance *xanthostgen*, le nombre de groupes est trop élevé pour calculer l'indice. Notons bien évidement que la similarité est de 1 pour les fonctions de groupes d'origine.

Instances	Fonction de groupes	Taille de la solution minimale
xanthostgen	Original	13
xanthostgen	K-medoid	11
xanthostgen	K-means	12
xanthostgen	center k-means	12
xanthostgen	Random	33
ra_rep1	Original	12
ra_rep1	K-medoid	9
ra_rep1	K-means	7
ra_rep1	center k-means	7
ra_rep1	Random	16
ra_rep2	Original	12
ra_rep2	K-medoid	9
ra_rep2	K-means	8
ra_rep2	center k-means	8
ra_rep2	Random	16
ra_phy	Original	6
ra_phy	K-medoid	8
ra_phy	K-means	6
ra_phy	center k-means	6
ra_phy	Random	15
ra_phv	Original	6
ra_phv	K-medoid	9
ra_phv	K-means	8
ra_phv	center k-means	7
ra_phv	Random	16
ra100_phy	Original	7
ra100_phy	K-medoid	8
ra100_phy	K-means	7
ra100_phy	center k-means	7
ra100_phy	Random	16
ra100_phv	Original	9
ra100_phv	K-medoid	10
ra100_phv	K-means	9
ra100_phv	center k-means	10
ra100_phv	Random	16
ralstophy2s	Original	8
ralstophy2s	K-medoid	7
ralstophy2s	K-means	5
ralstophy2s	center k-means	3
ralstophy2s	Random	13
Leukemia	Original	2
Leukemia	K-médoid	4
Leukemia	K-means	3
Leukemia	center k-means	3
Leukemia	Random	2

TABLE 4.6 – Taille des solutions minimales

Comme prévu, les fonctions de groupes aléatoires sont très différentes des fonctions de groupes d'origines. Nous pouvons observer que les fonctions de groupes résultant des méthodes

Instances	Fonction de groupes	Taille solution minimale	Valeur initiale fonction objective k-means	Valeur normalisée fonction objective k-means
xanthostgen	Original	13	186.500	0.019
xanthostgen	K-medoid	11	136.768	0.014
xanthostgen	K-means	12	155.385	0.016
xanthostgen	center k-means	12	154.393	0.016
xanthostgen	Random	33	990.117	0.103
ra_rep1	Original	12	755.022	0.092
ra_rep1	K-medoid	9	583.971	0.071
ra_rep1	K-means	7	570.379	0.070
ra_rep1	center k-means	7	570.379	0.070
ra_rep1	Random	16	982.077	0.120
ra_rep2	Original	12	754.501	0.092
ra_rep2	K-medoid	9	583.971	0.071
ra_rep2	K-means	8	590.362	0.072
ra_rep2	center k-means	8	575.717	0.070
ra_rep2	Random	16	981.837	0.120
ra_phy	Original	6	795.025	0.097
ra_phy	K-medoid	8	694.224	0.085
ra_phy	K-means	6	665.324	0.081
ra_phy	center k-means	6	665.324	0.081
ra_phy	Random	15	1003.030	0.123
ra_phv	Original	6	542.583	0.072
ra_phv	K-medoid	9	487.656	0.065
ra_phv	K-means	8	500.239	0.066
ra_phv	center k-means	7	480.067	0.064
ra_phv	Random	16	892.476	0.118
ra100_phy	Original	7	516.776	0.097
ra100_phy	K-medoid	8	416.205	0.078
ra100_phy	K-means	7	405.415	0.076
ra100_phy	cent kmeans	7	418.636	0.078
ra100_phy	Random	16	594.883	0.111
ra100_phv	Original	9	373.076	0.074
ra100_phv	K-medoid	10	300.570	0.060
ra100_phv	K-means	9	304.891	0.060
ra100_phv	cent kmeans	10	295.815	0.059
ra100_phv	Random	16	546.836	0.108
ralstophy2s	Original	8	205.019	0.122
ralstophy2s	K-medoid	7	169.080	0.101
ralstophy2s	K-means	5	162.639	0.097
ralstophy2s	cent kmeans	3	162.482	0.097
ralstophy2s	Random	13	247.306	0.147
Leukemia	Original	2	8798.400	0.144
Leukemia	K-médoid	4	8756.820	0.143
Leukemia	K-means	3	8734.740	0.143
Leukemia	cent kmeans	3	8717.610	0.143
Leukemia	Random	2	8813.620	0.144

TABLE 4.7 – Valeur des fonctions objectives k-means.

Instances	Fonction de groupes G	Taille solution minimale	similarité $\sigma_{G,O}$ avec la fonction de groupes originale
ra_rep1	Original	12	1
ra_rep1	K-medoid	9	0.437
ra_rep1	K-means	7	0.536
ra_rep1	center k-means	7	0.536
ra_rep1	Random	16	0.232
ra_rep2	Original	12	1
ra_rep2	K-medoid	9	0.429
ra_rep2	K-means	8	0.420
ra_rep2	center k-means	8	0.5
ra_rep2	Random	16	0.214
ra_phy	Original	6	1
ra_phy	K-medoid	8	0.589
ra_phy	K-means	6	0.562
ra_phy	center k-means	6	0.562
ra_phy	Random	15	0.321
ra_phv	Original	6	1
ra_phv	K-medoid	9	0.667
ra_phv	K-means	8	0.722
ra_phv	center k-means	7	0.796
ra_phv	Random	16	0.259
ra100_phy	Original	7	1
ra100_phy	K-medoid	8	0.457
ra100_phy	K-means	7	0.514
ra100_phy	center kmeans	7	0.619
ra100_phy	Random	16	0.505
ra100_phv	Original	9	1
ra100_phv	K-medoid	10	0.584
ra100_phv	K-means	9	0.505
ra100_phv	center kmeans	10	0.594
ra100_phv	Random	16	0.396
ralstophy2s	Original	8	1
ralstophy2s	K-medoid	7	0.575
ralstophy2s	K-means	5	0.697
ralstophy2s	center kmeans	3	0.644
ralstophy2s	Random	13	0.370
Leukemia	Original	2	1
Leukemia	K-medoid	4	0.514
Leukemia	K-means	3	0.543
Leukemia	center kmeans	3	0.571
Leukemia	Random	2	0.771

TABLE 4.8 – Similarité $\sigma_{G,O}$ entre les différentes méthodes de groupement et le groupement original (O).

de classification sont en effet plutôt différentes des fonctions de groupes d'origine, bien que nous ayons déjà observé que ces fonctions de groupes génèrent des groupes dont les cohérences sont plutôt similaires (selon le critère k-mean).

Afin d'étudier d'avantage la caractéristique des groupes induits par les fonctions de groupes, nous évaluons les cas avec plus de critères topologiques présentés dans la section 4.1.1.

4.3.3.4 Plus de caractéristiques sur les instances

La Table 4.9 résume les caractéristiques des instances avec les différentes fonctions de groupes définies précédemment. Pour les exemples xanthostgen, ra100_phy, ra100_phv, ralstophy2s et Leukemia, nous présentons pour chaque fonction de groupes la taille minimale de la solution, la similarité globale des observations de l'instance, la similarité intragroupe et la différence intergroupes (voir les définitions dans la section 4.1.1).

Notons que la similarité globale est calculée sur l'ensemble des observations et est évidemment identique pour toutes les variantes de l'instance (c'est-à-dire avec différentes fonctions de groupes).

Nous observons que :

- La similarité globale est assez élevée, ce qui s'explique par le fait que les observations ne sont pas générées de manière aléatoire et partagent des valeurs similaires sur un nombre suffisant d'attributs. Cette observation est plutôt raisonnable en particulier lorsque les attributs sont des gènes, dont les présences/absences sont en effet similaires pour les différents individus.

- La similarité intragroupe des fonctions de groupes aléatoires est proche de la similarité globale, ce qui est totalement cohérent, car la fonction de groupes aléatoire rassemble les observations initiales uniformément, en gardant ainsi leur similarité initiale.

- La similarité intragroupe est élevée pour les fonctions de groupes initiales ainsi que pour les fonctions de groupes basées sur les k-means, ce qui corrobore l'analyse précédente. En fait, ces fonctions de groupes ont des propriétés topologiques similaires en termes de similarités de groupe et de longueur de solution. Néanmoins, les groupes résultants sont différents.

- En ce qui concerne la différence intergroupe, les scores sont plutôt bas. Ce fait peut s'expliquer par la forte similarité globale des observations. Il est en effet difficile de générer des groupes d'observations très différents.

- L'instance *Leukemia* est légèrement différente. Pour chaque fonction de groupes, même la fonction de groupes aléatoire, la similarité intragroupe et la différence entre les groupes

Instances	Fonction de groupes	Taille sol min.	sim_{over}	sim_{intra}	$diff_{inter}$
$HM(G_1)$	G_1	3	0.485	0.882	0.889
$HM(G_2)$	G_2	8	0.485	0.478	0.508
xanthost	Original	13	0.719	0.942	0.289
xanthost	K-med	11	0.719	0.960	0.289
xanthost	K-means	12	0.719	0.954	0.289
xanthost	center K-means	12	0.719	0.954	0.288
xanthost	Random	33	0.719	0.708	0.281
ra_rep1	Original	12	0.747	0.793	0.270
ra_rep1	K-med	9	0.747	0.855	0.273
ra_rep1	K-means	7	0.747	0.870	0.280
ra_rep1	center K-means	7	0.747	0.870	0.280
ra_rep1	Random	16	0.747	0.742	0.253
ra_rep2	Original	12	0.747	0.792	0.269
ra_rep2	K-med	9	0.747	0.855	0.273
ra_rep2	K-means	8	0.747	0.857	0.277
ra_rep2	center K-means	8	0.747	0.865	0.277
ra_rep2	Random	16	0.747	0.741	0.252
ra_phy	Original	6	0.747	0.808	0.303
ra_phy	K-med	8	0.747	0.830	0.285
ra_phy	K-means	6	0.747	0.837	0.284
ra_phy	center K-means	6	0.747	0.837	0.284
ra_phy	Random	15	0.747	0.747	0.253
ra_phv	Original	6	0.748	0.847	0.293
ra_phv	K-med	9	0.748	0.867	0.273
ra_phv	K-means	8	0.748	0.871	0.281
ra_phv	center K-means	7	0.748	0.878	0.280
ra_phv	Random	16	0.748	0.742	0.251
ra100phy	Original	7	0.757	0.808	0.303
ra100phy	K-med	8	0.757	0.839	0.270
ra100phy	K-means	7	0.757	0.843	0.271
ra100phy	center K-means	7	0.757	0.838	0.278
ra100phy	Random	16	0.757	0.774	0.255
ra100phv	Original	9	0.757	0.833	0.271
ra100phv	K-med	10	0.757	0.872	0.263
ra100phv	K-means	9	0.757	0.871	0.262
ra100phv	center K-means	10	0.757	0.878	0.264
ra100phv	Random	16	0.757	0.764	0.246
ralsto	Original	8	0.676	0.854	0.253
ralsto	K-med	7	0.676	0.790	0.363
ralsto	K-means	5	0.676	0.797	0.370
ralsto	center K-means	3	0.676	0.795	0.366
ralsto	Random	13	0.676	0.678	0.326
Leuk	Original	2	0.694	0.695	0.309
Leuk	K-med	4	0.694	0.696	0.308
Leuk	K-means	3	0.694	0.697	0.309
Leuk	center K-means	3	0.694	0.697	0.309
Leuk	Random	2	0.694	0.693	0.304

TABLE 4.9 – Similarités (comme definie dans la section 4.1.1) pour les différentes fonctions de groupes

sont beaucoup plus proches. On peut remarquer que la taille minimale des solutions est très faible, surtout en ce qui concerne l'ensemble des attributs initiaux très important.

On peut remarquer que la taille minimale des solutions est très faible, surtout par rapport à l'ensemble des attributs initiaux.

Nous avons observé que les fonctions de groupes initiales possèdent des propriétés topologiques proches des groupes générés par les algorithmes K-means. Néanmoins, les groupes

résultants sont très différents. Par conséquent, nous pouvons conclure que le groupe initial ne peut pas être obtenu en utilisant des algorithmes de type K-means avec une distance de Hamming classique. Nous pourrions essayer de modifier cette distance, mais nous n'avons préalablement aucune information disponible qui peut être utilisée pour sélectionner les attributs les plus appropriés.

Une autre question est donc : notre approche de réduction des attributs nous permet-elle de caractériser les groupes qui pourraient être obtenus par des algorithmes de classification si les attributs appropriés avaient été identifiés *a priori* ?

4.3.3.5 Analyses des solutions

Dans cette partie, nous nous concentrons sur les solutions calculées pour les différentes instances, en utilisant différentes fonctions de groupes. Selon notre protocole expérimental présenté dans la figure 4.2, nous sommes intéressés quant à analyser si les attributs des solutions peuvent être exploités par des algorithmes k-means en termes de similarité.

Pour ce faire, nous appliquons les différents algorithmes de clustering sur les observations restreintes aux attributs qui apparaissent en solution (évidemment, les valeurs de distance changent). Ensuite, nous comparons la similarité des groupes résultants obtenus par regroupement avec les fonctions de groupes d'origine ($comp_3$ et $comp_2$).

Mais, avant d'examiner ces comparaisons, nous devons analyser les solutions. Soit une instance $(\Omega, \mathcal{A}, D, G)$, la résolution du problème de caractérisation multiple minimale conduit à de nombreuses solutions optimales. Pour chaque solution, la similarité intragroupe peut être différente.

La Table 4.10 montre la similarité intragroupe, la similarité globale et la différence intergroupes pour la solution $\bar{S}$ qui maximise la similarité intragroupe et pour la solution $\underline{\mathrm{S}}$ qui minimise cette similarité intragroupe. La dernière colonne correspond à la solution S' qui fournit la plus grande similarité d'un groupe, c'est-à-dire $S' = argmax_{S \in Sol}(max_g(sim_g(D^{S'})))$ (Voir Définition 28).

Bien évidement, on a $sim_{intra}(D^{\bar{S}}) \geq sim_{intra}(D^{\underline{\mathrm{S}}})$. Pour chaque instance à l'exception de *ra_rep1*, on observe que $diff_{inter}(D^{\underline{\mathrm{S}}}) \geq diff_{inter}(D^{\bar{S}})$. De plus, la valeur de similarité globale est plus élevée pour $D^{\bar{S}}$ que pour $D^{\underline{\mathrm{S}}}$.

Notons que lorsque la similarité d'un groupe est élevée, cela signifie que nous pouvons nous attendre à ce que le groupe puisse être couvert par moins de patterns.

Les algorithmes de type K-means utilisés ici sont basés sur la distance euclidienne entre les observations. Mais nous avons vu que les fonctions de groupes initiales ne peuvent pas être expliquées par une similarité de distance.

Nous voulons maintenant étudier si, une fois projetés sur les attributs sélectionnés dans les solutions obtenues pour le MCP, les groupes d'observations résultants présentent des caractéris-

Instance	Taille sol minimale	$sim_{intra}(D^{\bar{S}})$ max	$sim_{over}(D^{\bar{S}})$	$dif\,f_{inter}(D^{\bar{S}})$	$sim_{intra}(D^{\underline{S}})$ min	$sim_{over}(D^{\underline{S}})$	$dif\,f_{inter}(D^{\underline{S}})$	$sim_g(D^{S'})$ max
$HM(G_1)$	3	0.882	0.485	0.889	0.882	0.485	0.889	0.882
$HM(G_2)$	8	0.485	0.485	0.516	0.485	0.485	0.516	0.485
xanthostgen	13	0.958	0.664	0.346	0.908	0.646	0.363	1
ra_rep1	12	0.671	0.603	0.422	0.636	0.595	0.420	0.75
ra_rep2	12	0.691	0.637	0.382	0.605	0.561	0.454	0.917
ra_phy	12	0.869	0.725	0.392	0.843	0.594	0.607	0.906
ra_phv	12	0.883	0.630	0.474	0.815	0.565	0.538	0.921
ra100_phy	7	0.841	0.710	0.388	0.701	0.576	0.516	0.875
ra100_phv	9	0.782	0.620	0.439	0.670	0.549	0.494	1
ralstophy2s	8	0.736	0.638	0.422	0.650	0.562	0.493	0.770
Leukemia	2	0.992	0.889	0.5	0.537	0.503	0.629	1

TABLE 4.10 – Similarité des solutions

tiques différentes en ce qui concerne les approches de regroupement. En d'autres termes, nous voulons vérifier si les groupes obtenus par regroupement sur l'ensemble réduit d'attributs sont similaires aux groupes initiaux.

Revenons à la figure 4.2, $comp_2$ correspond à la comparaison entre les groupes obtenus en regroupant les observations initiales et les groupes obtenus en regroupant les observations réduites aux attributs des solutions optimales. $comp_3$ correspond à la comparaison entre les groupes obtenus par classification sur les observations réduites aux attributs des solutions optimales et des groupes initiaux des instances.

La Table 4.11 présente la similarité σ entre chacune de ces fonctions de groupes. Nous pouvons voir que plus petite est la taille minimale de la solution, plus la similarité entre la fonction de groupes d'origine et les fonctions de groupes basées sur les k-means calculées sur les solutions est élevée. La similarité entre les fonctions de groupes basées sur les k-means et la fonction de groupes d'origine est proche de la similarité entre les fonctions de groupes basées sur les k-means initiales et les fonctions de groupes k-means sur les solutions.

4.4 Étude des attributs

Dans cette section, nous souhaitons comparer les résultats issus des méthodes de caractérisation avec les résultats issus des méthodes de sélection de variables. Nous comparerons, sur une liste d'instances, dans un premier temps le nombre d'attributs obtenus par les solutions minimisant

Instances	Fonction de groupes	Similarité σ avec l'originale	Similarité σ avec k-means sur la sol	Similarité σ avec "center k-means" sur la sol
$HM(G_1)$	G_1	1	1	1
$HM(G_1)$	K-means	1	1	1
$HM(G_1)$	center k-means	1	1	1
ra_rep1	Original	1	0.339	0.455
ra_rep1	K-means	0.536	0.464	0.464
ra_rep1	center k-means	0.536	0.464	0.464
ra_rep2	Original	1	0.375	0.411
ra_rep2	K-means	0.420	0.491	0.455
ra_rep2	center k-means	0.5	0.562	0.509
ra_phy	Original	1	0.652	0.875
ra_phy	K-means	0.562	0.491	0.634
ra_phy	center k-means	0.562	0.491	0.634
ra_phv	Original	1	0.787	0.889
ra_phv	K-means	0.722	0.694	0.713
ra_phv	center k-means	0.796	0.778	0.843
ra100_phy	Original	1	0.743	0.771
ra100_phy	K-means	0.514	0.581	0.619
ra100_phy	center k-means	0.619	0.667	0.762
ra100_phv	Original	1	0.545	0.663
ra100_phv	K-means	0.505	0.475	0.564
ra100_phv	center k-means	0.594	0.465	0.594
ralstophy2s	Original	1	0.534	0.644
ralstophy2s	K-means	0.699	0.644	0.740
ralstophy2s	center k-means	0.644	0.699	0.808
Leukemia	Original	1	0.943	1
Leukemia	K-means	0.543	0.543	0.543
Leukemia	center k-means	0.571	0.514	0.571

TABLE 4.11 – Comparaison des fonctions de groupes

le nombre d'attributs et celles minimisant le nombre de patterns et dans un second temps le nombre de patterns caractérisant le groupe en utilisant les attributs des solutions obtenues par ces deux approches. Ensuite, nous comparerons les ensembles d'attributs des solutions du PCM avec ceux retournés par la méthode de sélection de variable `CFS` (présentée dans la sous-section 2.5.2.2). Enfin, nous vérifierons si un prétraitement obtenu par des méthodes de sélection de variables permet de mettre en évidence des attributs appartenant aux solutions du PCM.

4.4.1 Instances

Afin d'évaluer et de comparer les différents aspects d'optimisation des solutions en fonction des attributs et des patterns avec les méthodes de sélection de variables (voir section 2.5.2), nous considérons maintenant les instances suivantes décrite dans la Table 4.1.

Instances étudiées :

- *rch8*, *ra_rep1*, ra_rep2, *ralsto_phy2s*, *ra100_phv*, *ra100_phy*, *ra_phv* et *ra_phy* : Initialement, beaucoup de groupes sont proposés dans ces instances. Toutefois, nous n'étudions ici que le premier groupe de souches bactériennes comme groupe positif et l'union des autres groupes sera considérée comme le groupe négatif. Les résultats sont similaires si on considère un autre groupe comme le groupe positif.

- *vote_r*

Des caractéristiques plus détaillées des instances sont décrites dans la Table 4.12 : nombre d'observations, taille du groupe positif et nombre d'attributs.

Instances	Observations	Taille groupe positive	# Attributs
rch8	132	5	37
vote_r	435	168	16
ra_rep1	112	38	155
ra_rep2	112	37	73
ralsto	73	27	23
ra100_phv	101	21	50
ra100_phy	105	31	51
ra_phv	108	22	70
ra_phy	112	31	73

TABLE 4.12 – Caractéristiques des instances

Rappelons les principales caractéristiques des trois méthodes que nous envisagerons dans ces expériences, notre objectif étant de comparer ces méthodes en fonction des attributs sélectionnés.

Méthodes utilisées :

- `MinS` : calcule un ensemble minimal d'attributs pour identifier le groupe positif P, ce qui correspond à une solution minimale du PCM puisque les instances n'ont que deux groupes (voir section 2.3 et 3.3),

- `Pattern` : calcule une couverture du groupe positif P en utilisant des patterns (voir section 2.2 et 3.5),

- `CFS` : calcule un ensemble d'attributs pertinents pour le groupe positif P basé sur la corrélation des attributs (voir les méthodes de filtrage de la sous-section 2.5.2.2).

Notons que chaque méthode nous permet d'identifier un sous-ensemble d'attributs pouvant être utilisé pour expliquer ou comprendre les caractéristiques du groupe positif P. L'objectif de ces expérimentations est donc d'examiner et comparer ces sous-ensembles d'attributs.

4.4.2 Comparaison du nombre d'attributs composant les solutions

Dans la Table 4.13, nous comparons pour chaque méthode (`MinS` ou `Pattern`) le nombre total d'attributs utilisés. En ce qui concerne `MinS`, cela correspond au nombre d'attributs dans une solution minimale. En ce qui concerne `Pattern`, nous considérons le nombre minimal d'attributs qui doivent être utilisés pour couvrir totalement l'ensemble P par un minimum de patterns. Notons que, dans les deux cas, plusieurs solutions peuvent exister pour la même instance.

Instance	# Attributs `MinS`	# Attributs `Pattern`
rch8	3	3
vote_r	10	14
ra_rep1	12	22
ra_rep2	11	19
ralsto	5	5
ra100_phv	2	2
ra100_phy	3	4
ra_phv	2	2
ra_phy	3	4

TABLE 4.13 – Nombre d'attributs utilisés par `MinS` et `Pattern`

Remarquons que le nombre d'attributs est plutôt similaire pour les deux méthodes lors de l'étude de petites instances. Néanmoins, nous observons une différence sur les instances *ra_rep1* et *ra_rep2*. Mais, notons que pour *ra_rep2*, cette différence représente seulement 11% du nombre total d'attributs de l'instance, et seulement 6,5% pour *ra_rep1*.

4.4.3 Comparaison du nombre de patterns pour couvrir le groupe positif

Dans la Table 4.14, nous nous concentrons sur les patterns. La valeur #Patterns `MinS` correspond au nombre minimal de patterns nécessaires pour couvrir les groupes positifs, mais en utilisant uniquement l'ensemble minimal d'attributs calculé par `MinS` (dans ce cas, nous recalculons les patterns pour l'ensemble des attributs sélectionnés avec `MinS`, voir la section 3.5). La valeur #Patterns `Pattern` est le nombre minimal de patterns nécessaire pour couvrir le groupe positif, mais cette couverture est basée sur tous les patterns possibles.

Dans la Table 4.14, nous observons que les attributs calculés par `MinS` ne conviennent pas pour trouver un bon ensemble de patterns couvrants (en fonction de ces attributs, le nombre de patterns pour obtenir une couverture de l'ensemble positif augmente). Cela signifie que l'explication fournie par `MinS` diffère de celle fourni par `Pattern` en termes de patterns.

Instance	#Patterns `MinS`	#Patterns `Pattern`
rch8	1	1
vote_r	71	10
ra_rep1	27	7
ra_rep2	25	6
ralsto	7	3
ra100_phv	1	1
ra100_phy	5	2
ra_phv	1	1
ra_phy	5	2

TABLE 4.14 – Nombre de patterns pour `MinS` et `Pattern`

4.4.4 Comparaison de *MinS* avec *CFS*

Dans la Table 4.15, nous présentons les observations suivantes :

Observations :

- Le nombre de solutions minimales calculées par `MinS`,
- Le nombre d'attributs de ces solutions minimales,
- Le nombre d'attributs de la meilleure solution calculée par `CFS`,
- Le nombre maximal d'attributs communs entre le sous-ensemble calculé par `CFS` et des solutions minimales de `MinS`. Étant donné que plusieurs solutions peuvent maximiser le nombre d'attributs communs, nous indiquons également entre parenthèses combien de solutions correspondent à ce critère.

Rappelons que la méthode `CFS` calcule un sous-ensemble d'attributs pertinents pour évaluer le groupe (dans le sens où l'ensemble d'attributs sera corrélé au groupe). Afin d'évaluer la pertinence de ce sous-ensemble d'attributs, nous proposons de vérifier qu'ils peuvent être utiles pour classer les données. Par conséquent, nous utilisons un algorithme de classification et vérifions que les classes résultantes correspondent aux groupes positifs et négatifs initiaux. Nous utilisons ici un algorithme k-means (Hartigan and Wong [1979]; MacQueen et al. [1967]) car c'est une technique de classification simple et bien connue (bien sûr, le nombre de classes est fixé à 2).

La précision Acc du classement est évaluée en fonction de la similarité entre le groupe prédit c et le groupe réel original r. Soit n observations, nous considérons deux vecteurs de dimensions $n : C$ et R, tels que c_i est le groupe prédictif de l'observation i et r_i correspondent à son groupe original réel. Nous définissons la précision comme suit :

$$Acc(C,R) = |2 \times (\frac{1}{n}\sum_{i=1}^{n}(r_i - c_i)^2) - 0.5|$$

Notons que $Acc(C,R) \in [0,1]$. Les valeurs proche de 1 correspondent ainsi à une précision élevée.

Pour chaque instance, la précision du groupement en fonction des attributs sélectionnés par `CFS` est présentée entre parenthèses dans la troisième colonne.

Puisque k-means est une méthode statistique, nous avons répété le processus de classification et évalué la valeur moyenne de précision sur 20 exécutions indépendantes.

Instance	# solution `MinS`	# attributs dans solution minimale	# attributs `CFS` (précision)	# attributs commun (# solution)
rch8	1	3	6 (0.439)	0 (1)
vote_r	1	10	4 (0.706)	3 (1)
ra_rep1	134	12	6 (0.393)	3 (1)
ra_rep2	106	11	11(0.536)	3 (26)
ralsto	5	5	3 (0.288)	2 (1)
ra100phv	1	2	4 (0.465)	0 (1)
ra100phy	1	3	1 (0.562)	1 (1)
ra_phv	1	2	3 (0.519)	0 (1)
ra_phy	1	3	1 (0.589)	1 (1)

TABLE 4.15 – Similarité entre attributs obtenus par `MinS` et par `CFS`

L'utilisation d'une technique de sélection de variables telle que `CFS` pour expliquer nos données par des techniques de classification (c'est-à-dire des méthodes basées sur la distance) n'est pas vraiment pertinente ici. Notons que les attributs sélectionnés par `CFS` sont différents des attributs calculés pour `MinS`. En outre, `MinS` réduit le nombre d'attributs utilisés pour la caractérisation. Notons que nous avons également effectué une classification en utilisant les attributs calculés dans les solutions `MinS`, ce qui conduit également à une mauvaise précision de classification (mais `MinS` n'est pas une méthode de sélection de variables car elle recherche des combinaisons d'attributs).

Notons que de meilleurs résultats sont obtenus pour l'instance *vote_r*, où 3 des 4 attributs sélectionnés par `CFS` sont également présents dans la solution minimale unique de `MinS`. En outre, la précision est plus élevée (0.7). Néanmoins, `CFS` sélectionne seulement 4 attributs, tandis que 10 sont nécessaires pour caractériser l'instance via l'approche réalisée par `MinS`. Par conséquent, cette technique de sélection de variables ne peut pas être utilisée comme prétraitement pour améliorer les résultats du PCM.

4.4.5 Analyse des attributs pour `MinS` et `Pattern`

Dans Table 4.16 et Table 4.17, nous évaluons la pertinence des attributs au moyen de scores. Nous voulons évaluer la capacité d'un attribut à discriminer les observations selon les groupes initiaux. Étant donné un attribut a, le score, dénoté $score(a)$, est calculé en comptant le nombre d'observations où la valeur de l'attribut a diffère dans un même groupe.

$$score(a) = |2 \times (\frac{1}{n} \sum_{j=1}^{n} (g_j - a_j)^2) - 0.5|$$

où :

- a_j est la valeur de l'attribut a pour l'observation j,
- $g_j \in \{0, 1\}$ est le groupe affecté à l'observation j.

Notons qu'un attribut qui discriminerait complètement les groupes (c'est-à-dire, dont les valeurs seront toujours identiques ou toujours différentes pour toutes les observations des deux groupes) aura une valeur de score de 1. Étant donné un ensemble d'attributs (par exemple, représentant une solution MinS ou Pattern), ses scores sont la moyenne des scores de ses attributs.
Dans la Table 4.16, nous nous concentrons sur `MinS` :

- La première colonne est le meilleur score possible obtenu par une solution parmi l'ensemble des solutions minimales.
- Dans la seconde colonne, afin d'évaluer ce score, nous calculons le ratio entre ce score et le score maximal qui peut être obtenu lors de la sélection du même nombre d'attributs mais en utilisant les meilleurs attributs (ceux qui maximisent cette notion de score). Un ratio de 1 signifie que la solution minimale est construite uniquement à l'aide d'attributs avec les scores les plus élevés.
- La troisième colonne indique le meilleur score possible parmi les attributs qui n'apparaissent dans aucune solution (c'est-à-dire qui ne sont pas sélectionnés dans l'approche `MinS`).
- La quatrième colonne indique le meilleur score possible parmi l'ensemble des attributs.

À l'exception de l'instance *vote_r*, nous notons que la valeur du ratio est faible, ce qui montre que les scores des solutions `MinS` sont plutôt faibles. Ces solutions sont donc composées d'attributs à faible score. En outre, le meilleur score des attributs n'apparaissant dans aucune solution est proche du meilleur score possible. Cela signifie que les attributs avec un score élevé ne sont que rarement présents dans les solutions. Cela suggère que les solutions `MinS` ne sont pas construites avec les attributs les plus corrélés aux groupes. Encore une fois,

Instance	meilleur score sol	ratio	meilleur score att. hors sol	score max possible
rch8	0.323	0.35	0.924	0.924
vote_r	0.439	0.7	0.683	0.894
ra_rep1	0.217	0.593	0.321	0.518
ra_rep2	0.239	0.636	0.357	0.5
ralsto	0.43	0.602	0.836	0.89
ra100phv	0.624	0.663	0.96	0.96
ra100phy	0.549	0.653	0.829	0.905
ra_phv	0.648	0.686	0.963	0.963
ra_phy	0.577	0.634	0.929	0.929

TABLE 4.16 – Scores des attributs pour `MinS`

ce système de score ne permet pas le prétraitement des données et offre des informations différentes des solutions du PCM.

Dans la Table 4.17, nous étudions les solutions obtenues par `Pattern`. Étant donné que le nombre d'attributs est variable, nous indiquons entre parenthèses dans la première colonne le nombre d'attributs dans la solution possédant le meilleur score.

Instance	meilleur score sol (#att)	ratio	meilleur score att hors sol	score max possible
rch8	0.515(4)	0.557	0.924	0.924
vote_r	0.468(15)	0.930	0.531	0.894
ra_rep1	0.211(26)	0.616	0.339	0.518
ra_rep2	0.158(19)	0.447	0.5	0.5
ralsto	0.333(6)	0.514	0.836	0.89
ra100phv	0.624(2)	0.663	0.96	0.96
ra100phy	0.467(4)	0.58	0.829	0.905
ra_phv	0.648(2)	0.686	0.963	0.963
ra_phy	0.527(4)	0.581	0.893	0.929

TABLE 4.17 – Scores des attributs pour `Patterns`

Comme pour `MinS`, nous observons que les scores des variables ne sont pas corrélés aux exigences pour la construction d'une couverture au moyen de patterns.

4.5 Étude de la répartition des solutions du PCM et des patterns

L'objectif de cette section est de présenter une visualisation de la répartition des solutions du PCM et de la capacité des primes patterns à couvrir un groupe en fonction de leur taille. Nous

allons donc présenter dans un premier temps la répartition des solutions du problème de caractérisation multiple obtenues sur quelques instances en fonction de leur taille et dans un second temps nous nous concentrerons sur le pouvoir de couverture des primes patterns en fonction de leur taille.

4.5.1 Répartition des solutions du problème de caractérisation multiple

Dans cette section, nous souhaitons observer la distribution des solutions du PCM en fonction de leur taille sur quelques instances. La Figure 4.3 présente cette répartition des solutions du PCM sur 4 instances *ra100_phv*, *rch8*, *ralsto_phy2s*, *ra100_phy* (présentées dans la section 4.2.1).

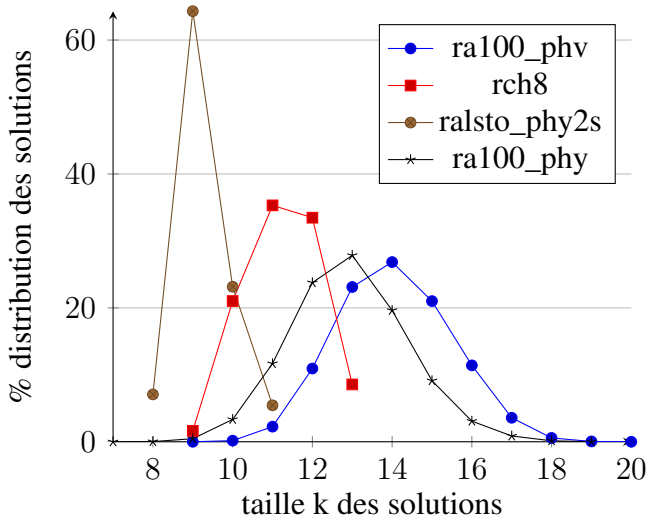


FIGURE 4.3 – Distributions des solutions non-dominées en fonction de leur taille k.

Nous visualisons que les distributions des solutions de chaque instance sont similaires. Plus la taille augmente, plus le nombre de solutions augmente jusqu'à un certain niveau où il diminue aussi rapidement qu'il a augmenté. Ces résultats sont cohérents avec ce que nous pouvions espérer. En effet, plus nous utilisons d'attributs, plus nous avons de chance d'obtenir une solution puisque chaque attribut en plus est un moyen en plus de différencier les observations de groupes différents. Toutefois, plus nous ajoutons d'attributs, plus nous augmentons le risque que notre solution soit dominée.

La Figure 4.4 présente la distribution cumulée des solutions précédentes en fonction de leur taille, c'est à dire qu'elle illustre le nombre de solutions de taille inférieure ou égale à une valeur donnée. Par exemple, pour l'instance *rch8*, 60% des solutions sont constituées d'au plus 11 attributs.

Nous constatons que le nombre de solutions non-dominées augmente très rapidement lorsque les solutions sont d'une taille proche de celle des solutions minimales. Ainsi, lorsque l'on

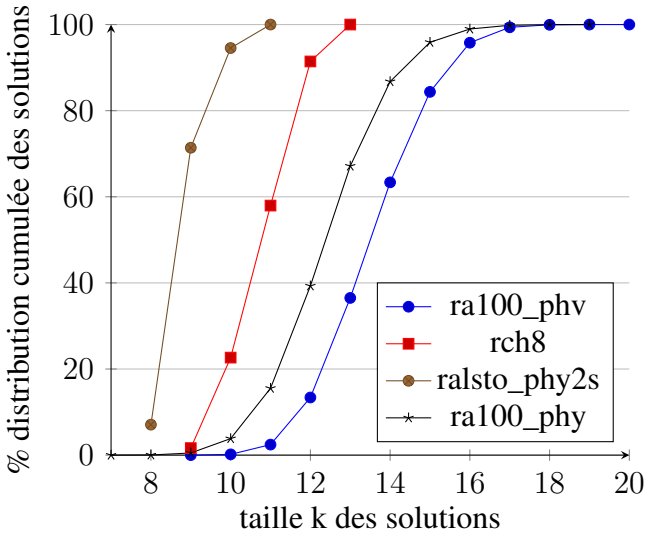


FIGURE 4.4 – Distribution cumulée des solutions non-dominées en fonction de leur taille k.

cherche les solutions non-dominées de taille inférieure à une taille donnée, il n'est pas nécessaire que cette taille soit très éloignée de la taille des solutions minimales pour trouver une grande partie de l'ensemble des solutions non-dominées. Toutefois, cela implique qu'en termes de temps d'exécution, une augmentation marginale de la taille de la borne augmente grandement le temps d'exécution de l'algorithme si cette borne est proche de la taille des solutions minimales.

Notons qu'au vu de ces résultats, la solution non-dominée donnée par un des algorithmes glouton présentés dans la section 3.2 dont on retrouve les résultats dans la Table 4.2 est de taille satisfaisante par rapport à l'ensemble des solutions non-dominées. Par exemple, pour l'instance *rch8*, ces algorithmes gloutons retournent chacun une solution de taille 10, or seulement 20% des solutions non-dominées de l'instance sont de taille inférieur ou égale à 10.

La Figure 4.5 présente la distribution de l'ensemble des solutions (dominées ou non) de l'instance *ralsto_phy2s* ainsi que l'ensemble des solutions non-dominées en fonction de leur taille. L'objectif de cette représentation est de présenter le rapport entre le nombre de solutions totales (dominées ou non) et le nombre de solutions non-dominées. Notons que le nombre de solutions est représenté sur une ordonnée logarithmique pour une question de visualisation.

Nous constatons que le nombre de solutions non-dominées est très faible par rapport au nombre de solutions globales dès lors que leur taille est supérieure à la taille des solutions minimales. Si ne calculer que les solutions non-dominées a un intérêt certain dans l'apprentissage (information non redondante), cela permet également d'écarter de la recherche un nombre considérable de solutions qui pourtant sont très facile à reconstruire puisqu'il s'agit d'ajouter n'importe quels attributs à n'importe quelle solution non-dominée.

Notons que l'instance *ralsto_phy2s* est une "petite" instance. Nous ne présentons ici que

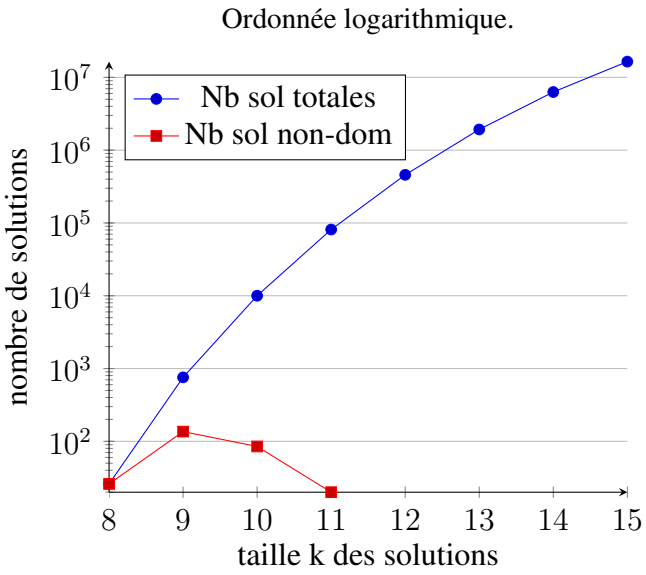


FIGURE 4.5 – Distribution des solutions en fonction de leur taille pour l'instance *ralsto_phy2s*.

celle-ci car il est impossible de visualiser l'ensemble des solutions non-dominées par rapport à l'ensemble global des solutions sur les autres instances "plus grandes", même en utilisant une ordonnée logarithmique.

4.5.2 Étude de la caractérisation par pattern

Dans cette section, nous étudions plus précisément les prime patterns qui ont été calculés dans la section 4.2.3. Dans ce but, nous considérons uniquement les cas réels avec leur ensemble complet d'attributs (l'instance aléatoire n'est pas considérée).

Pour chaque instance, le nombre total de prime patterns est présenté dans le tableau 4.5 ainsi que la taille des plus grands prime patterns (encore en termes de nombre d'attributs). Rappelons que, pour chaque instance, l'ensemble des prime patterns contient des patterns de différentes tailles jusqu'à cette taille maximale.

Parmi ces prime patterns, nous sommes intéressés par ceux qui couvrent le nombre maximal d'observations dans le groupe P. Ces patterns permettent à l'utilisateur d'identifier des caractéristiques intéressantes partagées par autant d'observations que possible. Ces patterns peuvent aider cet utilisateur à formuler une explication ou un diagnostic possible en fonction des valeurs des attributs (notion de préférence évidentielle, voir définition 9). Pour chaque taille de patterns qui appartiennent à l'ensemble des prime patterns calculés, la figure 4.5.2 présente le pourcentage maximal d'observations satisfaites par un pattern de cette taille. Par exemple, considérons l'instance *rch8*, il existe au moins un pattern de taille 3 (et un pattern de taille 4) qui couvre toutes les observations du groupe P.

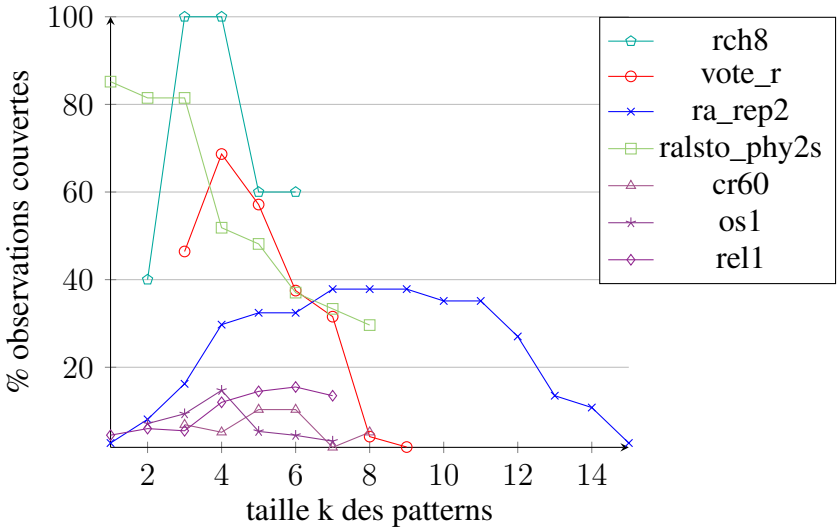


FIGURE 4.6 – Couverture maximale des patterns en fonction de leur taille k.

Nous remarquons que :

- Les plus petits patterns ont une couverture plus petite (sauf pour l'instance *ralsto_phy2s*) car ils impliquent moins d'attributs. Un petit ensemble de valeurs d'attributs est en effet plus susceptible d'apparaître dans les observations de P, mais il est également plus susceptible d'apparaître dans des observations de N, ce qui n'est pas compatible avec la définition d'un pattern. Une remarque similaire est valable pour des patterns plus importants : un pattern plus large est plus susceptible de discriminer des observations de N mais peut ne pas être présent dans la plupart des observations de P. Par conséquent, pour plusieurs cas, un compromis doit être atteint pour trouver des tailles de patterns qui couvrent un nombre significatif d'observations du groupe P.

- *ralsto_phy2s* est un cas particulier car un attribut est suffisant pour caractériser plus de 80% d'observations du groupe P. Cet attribut peut être trouvé par une analyse statistique très simple. Néanmoins, cet attribut unique n'est pas suffisant pour caractériser l'ensemble du groupe P. Par conséquent, la notion de pattern est entièrement pertinente pour aider l'utilisateur à mieux comprendre le groupe P.

- Concernant les instances *cr60*, *os1* et *rel1*, le pourcentage d'observations couvertes est assez faible. Ceci est dû au fait que des observations similaires sont présentes dans les groupes P et N. Néanmoins, il est vraiment difficile pour le praticien de décider celles qui ne sont pas pertinentes. Bien qu'il n'y ait pas de solution du PCM, certains patterns qui couvrent des observations non dupliquées peuvent encore être trouvés. Ces patterns permettent au praticien d'affiner son analyse sur les données afin de mieux définir les groupes de patients ou de considérer plus d'attributs.

La Figure 4.7 montre la répartition des prime patterns en fonction de leurs tailles, c'est-à-dire combien de prime patterns de chaque taille ont été calculés pour chaque instance. Ce chiffre met en évidence que les patterns de taille moyenne sont plus susceptibles de couvrir plus d'observations. Il existe en effet un plus grand nombre de patterns de taille moyenne. Encore une fois, considérer peu d'attributs n'est pas suffisant pour exclure les observations du groupe N.

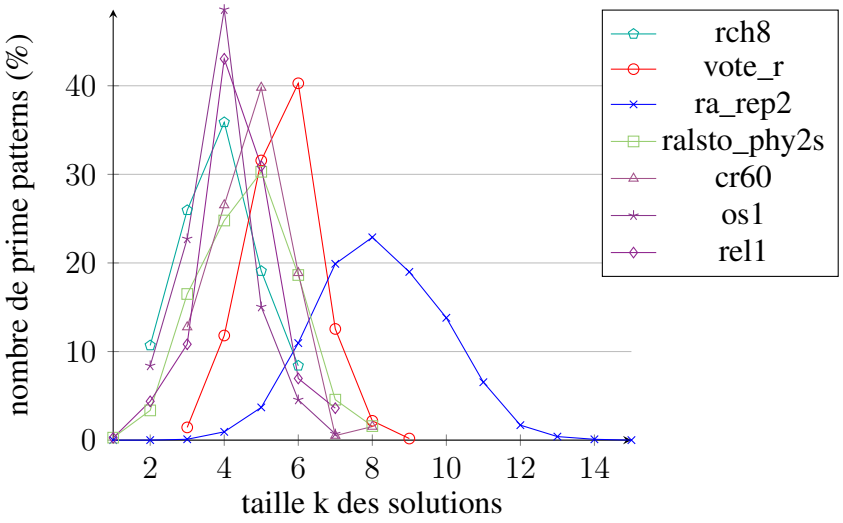


FIGURE 4.7 – Distribution des prime patterns pour chaque taille k

Enfin, nous montrons comment les prime patterns peuvent être utilisés pour caractériser toutes les observations du groupe P. Pour chaque instance, le nombre minimal de patterns nécessaire pour couvrir toutes les observations du groupe P est enregistré dans la Table 4.18.

Instances	taille groupe positive	#patterns	#attributs	couverture
rch8	5	1	3	100%
vote_r	168	10	14	100%
ra_rep2	37	6	19	100%
ralsto_phy2s	27	3	5	100%
cr60	58	14	14	51.724%
os1	224	26	14	75.446%
rel1	200	25	14	86%

TABLE 4.18 – Nombre de patterns nécessaire pour couvrir le groupe positif

Comme déjà mentionné, l'instance *rch8* ne nécessite qu'un seul pattern. Bien sûr, plusieurs combinaisons de patterns peuvent être pertinentes et peuvent donc être proposées à l'utilisateur. Notons que, comme mentionné ci-dessus, *cr60*, *os1* et *rel1* ne peuvent pas être totalement couvertes par des patterns car certaines observations similaires sont présentes dans les deux ensembles P et N. Néanmoins, la couverture calculée correspondant à un ensemble de patterns peut être utilisée pour mieux identifier une explication partielle de ces deux groupes.

5

Conclusion et perspectives

Les contributions de cette thèse touchent deux approches logiques pour l'analyse de données : la caractérisation multiple de données et l'analyse logique de données. De nouveaux algorithmes, plus efficaces, ont été présentés pour chacune d'elles. Les expérimentations réalisées sur des données biologiques réelles apportent des résultats intéressants, et nous offre de nouvelles opportunités de travaux futurs.

Contribution à la résolution du problème de caractérisation multiple

Nous avons présenté les liens entre le calcul de solutions non-dominées du PCM et le calcul d'ensembles intersectants. Pour calculer l'ensemble de ces solutions non-dominées, nous avons défini de nouveaux algorithmes, mais également utilisé des algorithmes déjà existants, calculant des ensembles intersectants. De plus, nous avons développé des méthodes permettant de calculer uniquement les solutions de plus petite taille (*MinAll-PCM*).

Les travaux sur l'analyse logique de données ont permis de calculer des formules descriptives de groupes, comme une disjonction de patterns, en minimisant le nombre de ces patterns. Toutefois, déterminer quelle formule minimale (en terme de nombre de patterns) doit être choisie pour chaque groupe afin de minimiser le nombre total d'attributs pour calculer une solution du PCM reste un problème difficile.

Enfin, nos expérimentations nous ont permis de remarquer que la caractérisation de données nous apporte une information différente de l'analyse de données et qu'il n'est pas possible d'établir de prétraitements statistiques afin de diminuer la taille du problème.

Notons également que la reformulation du problème de caractérisation multiple à partir de la

matrice de contraintes permet d'ouvrir l'analyse à des jeux de données non-booléens. En effet, dès lors que la matrice de contraintes pointe des différences entre les observations, il suffit de concevoir un critère de "différentiation" des données pour transformer une matrice de données composée de nombres réels en une matrice de contraintes composée de valeurs binaires.

Contribution à l'analyse logique de données

Nous avons présenté les liens existants entre l'analyse logique de données et la caractérisation multiple. Cela nous a permis de définir un nouvel algorithme générant l'ensemble des prime patterns d'un groupe d'une instance plus efficacement que les algorithmes existants. De plus, pour chaque pattern, l'algorithme détermine l'ensemble des observations qu'il couvre, ce qui permet d'obtenir une information importante supplémentaire.

Notons que les prime patterns ont un rôle important dans l'analyse descriptive des données puisqu'il s'agit d'une information soulignant un diagnostic exact et non-redondant.

Perspectives de recherche

En continuité des travaux présentés dans cette thèse, plusieurs aspects restent à développer :

- Déterminer une caractérisation multiple minimale minimisant la caractérisation des groupes d'une instance : lors du calcul de solutions minimisant la formule associée à un groupe (i.e. minimisant le nombre de patterns pour couvrir le groupe), nous travaillons avec un groupe face à tous les autres. Pour chaque groupe, plusieurs solutions sont disponibles. Choisir une des solutions de chaque groupe telle que l'union des solutions forme la solution du PCM avec le moins d'attributs reste un problème difficile.

- Recherche de solutions du PCM en optimisation multi-objectif : nous avons développé deux axes d'optimisation des solutions du problème de caractérisation multiple, en fonction du nombre d'attributs et en fonction du nombre de patterns couvrant les groupes. Il peut toutefois être intéressant de développer des études multi-objectifs pour mettre en avant des résultats axés sur ces deux objectifs simultanément.

- Comparaison entre la caractérisation et les itemsets : les règles d'associations suggèrent une relation forte et causale entre deux ensembles d'attributs. Ce genre de relations pourrait permettre de mieux comprendre la constitution des solutions (et améliorer le temps de calcul de ces solutions) mais également de proposer une nouvelle relation d'ordre entre les solutions.

- Étendre le problème de la caractérisation de données à une matrice de contraintes non booléennes : en pratique les données ne sont pas nécessairement binaires. Nous pouvons aujourd'hui travailler sur de telles données à condition que la différence entre les données

soit une relation binaire (soit les données sont similaires, soit elles sont différentes).
Par exemple, nous pouvons décider qu'une valeur $D[o, a]$ est différente d'une valeur $D[o', a]$ si et seulement si $|D[o, a] - D[o', a]| > \alpha_a$ (et dans ce cas $C[(o, o'), a] = 1$).
Avec un tel procédé, la "binarisation" des données n'est plus obligatoire, mais le résultat de la caractérisation dépendra fortement de α_a. En étendant le problème à une matrice de contraintes non binaire (par exemple où $C[(o, o'), a] = |D[o, a] - D[o', a]|$), nous pourrions nous intéresser aux solutions maximisant la somme des différences entre observations de groupes différents.

- Mise en place d'une plateforme pour les biologistes : pour une utilisation facile et intuitive des algorithmes, le développement d'une interface graphique parait être une contribution essentielle (Plateforme Galaxie, INRA, ...).

Annexes

Algorithmes développés au cours de la thèse

La Figure 1 présente les principaux algorithmes que nous avons développés. Ces algorithmes sont répartis en plusieurs catégories :

- Génération des fichiers d'entrée au bon format.
- Réduction de la matrice : Ces algorithmes sont optionnels. Ils permettent de faire un prétraitement des données en éliminant les attributs identiques ou dominés.
- Résolution du PCM : Ce sont les algorithmes que nous avons présentés dans cette thèse et qui permettent de générer l'ensemble ou un sous-ensemble des solutions non-dominées et minimales. Notons que nous avons également développé un algorithme permettant de ne générer qu'un sous-ensemble de solutions S_i disjointes (i.e. $\bigcap_i S_i = \emptyset$). Les résultats sont très dépendants des premières solutions générées mais cela permet d'obtenir des solutions radicalement différentes.
- Tri des solutions : Nous avons vu qu'il peut exister de nombreuses solutions. Ces algorithmes permettent de les trier selon différents critères afin de mettre en évidence celles présentant des propriétés intéressantes. Le critère fréquence de caractère nous pousse à rechercher la solution composée des attributs apparaissant le plus dans l'ensemble des solutions. Les critères similarité et différence nous demandent de chercher les solutions sur lesquelles la projection des données offrent la similarité intra-groupe et la différence inter-groupe les plus importantes. Enfin, le critère de regroupement des attributs corrélés n'est applicable que pour les instances ayant des attributs pour lesquels nous pouvons mesurer une similarité. Nous fixons un seuil de similarité α tel que si deux attributs se ressemble à α près, nous considérons qu'ils sont identiques. En regroupant de cette manière les attributs, des solutions disparaissent. Plus α est élevé, moins il nous reste de solutions. La meilleure solution selon ce critère sera celle qui restera avec le plus fort taux α. Ce critère nous permet de construire des solutions moins vulnérables à une erreur de séquençage par exemple.

- Maximisation de similarité : Cet algorithme déterminera les solutions maximisant la similarité intra-groupe des données projetées sur ces solutions.
- Amélioration par suppression d'observations : En supprimant des observations, il est possible de réduire la taille des solutions minimales. Cet algorithme va donc faire de l'optimisation multi-objectif en minimisant à la fois la taille des solutions minimales et le nombre d'observations supprimées. Le résultat est donc un front de Pareto de solutions minimales, i.e. une liste de solutions telles qu'il n'existe pas de solution à la fois de plus petite taille et pour laquelle on a supprimé moins d'observation.
- Approche par pattern : Il s'agit ici de l'algorithme de génération des prime patterns présenté dans cette thèse. Cette approche permet de couvrir les groupes en utilisant le moins de patterns possibles.

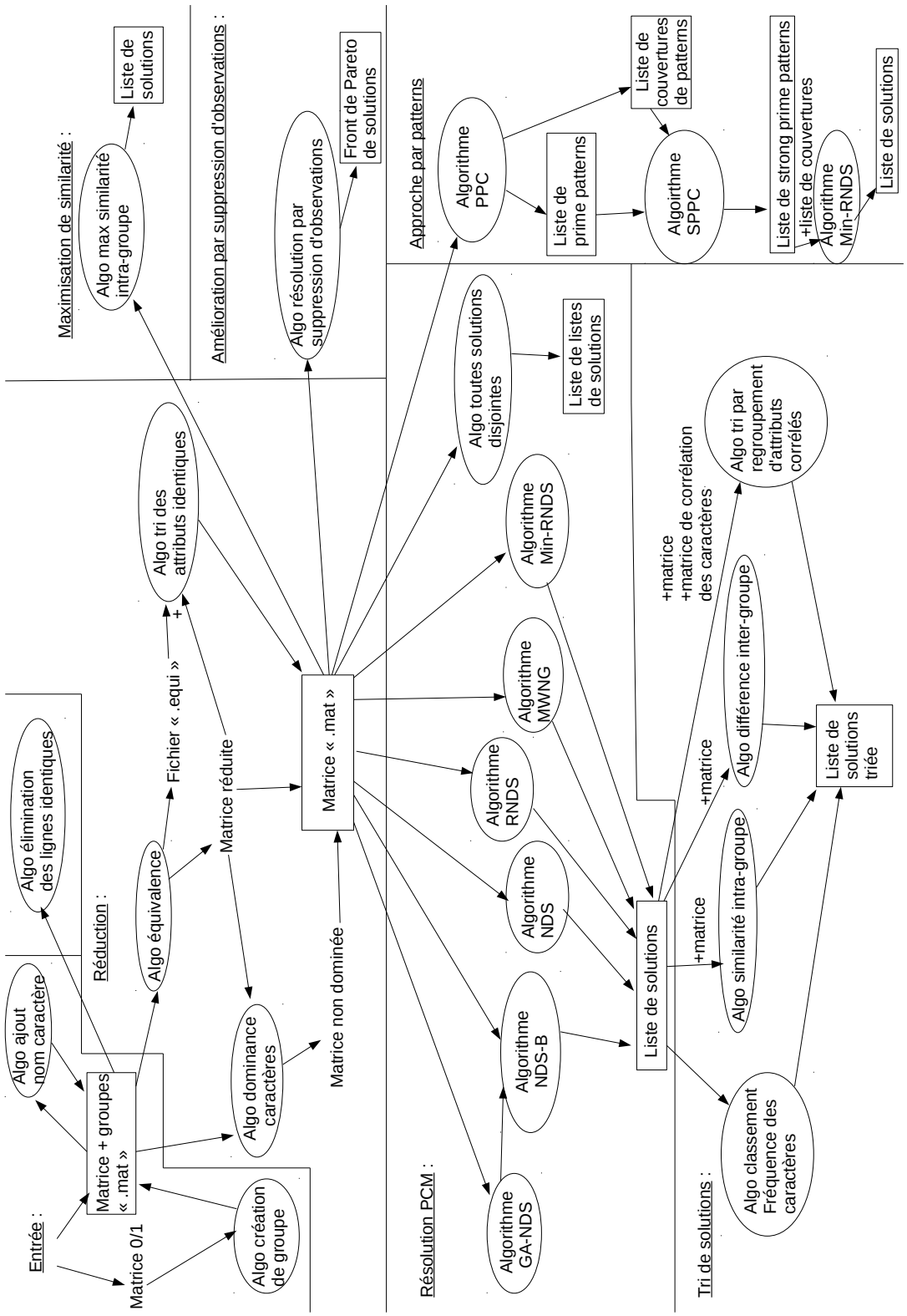


FIGURE 1 – Liste des algorithmes

Publications au cours de la thèse

Publications acceptées :

- Conférence internationale :
  Chambon, A., Boureau, T., Lardeux, F., Saubion, F., & Le Saux, M. (2015, November) : Characterization of Multiple Groups of Data. In *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on (pp. 1021-1028). IEEE.*

- Revue :
  Chambon, A., Boureau, T., Lardeux, F., & Saubion, F. : Logical Characterization of Groups Data : a Comparative Study. To appear in *Applied Intelligence* (2017).

Articles soumis :

- Chambon, A., Boureau, T., Lardeux, F., & Saubion, F. : Computing Sets of Patterns for Binary Data. Soumis à *Annals of Mathematics and Artificial Intelligence*

- Chambon, A., Boureau, T., Lardeux, F., & Saubion, F. : Explaining Groups of Data with Binary Attributes. Soumis à *Symposium On Applied Computing (SAC), 2018*.

Précision sur les instances réelles étudiées

Précision sur l'instance *vote_r*

Cet ensemble de données comprend des votes pour chacun des membres de la Chambre des représentants des États-Unis sur les 16 votes clés identifiés par le CQA. Le CQA énumère neuf types de votes différents :

- voté pour, jumelé et annoncé pour (ces trois simplifiés par oui),
- voté contre, apparié et annoncé contre (ces trois simplifiés par non),
- voté blanc, voté blanc pour éviter un conflit d'intérêts, et n'a pas voté (ces trois simplifiés par une valeur manquante et remplacés par une valeur binaire aléatoire dans notre jeu de données).

Groupe et attributs :

- **Groupe :** démocrate ou républicain,
- Bébés handicapés : oui ou non,
- Partage des coûts du projet eau : oui ou non,
- Adoption de la résolution sur le budget : oui ou non
- Gel des frais médicaux : oui ou non
- Aide au salvador : oui ou non
- Groupes religieux dans les écoles : oui ou non
- Interdiction des tests anti-satellite : oui ou non
- Aide aux contras nicaraguayens : oui ou non
- Missile Mx : oui ou non
- Immigration : oui ou non
- Réduction des usines de combustibles : oui ou non
- Dépenses d'éducation : oui ou non
- Droit de poursuivre en justice avec le programme Superfund : oui ou non
- Crime : oui ou non
- Exportations hors taxes : oui ou non
- Loi sur l'administration des exportations d'Afrique du Sud : oui ou non

Précision sur les instances *cr60, os1* et *rel1* (maladie : Leucémie aiguë myéloïque (LAM))

Ces ensembles de données comprennent des renseignements sur plusieurs patients atteints de leucémie. Les patients sont les mêmes dans les 3 jeux de données (l'instance *rel1* ne contient toutefois qu'un sous ensemble de ces patients), seuls les groupes changent. Les attributs sont multiples (génotype, phénotype, caryotype, ...) mais tous binaires. Ces descripteurs ont été choisis car on les suspecte de jouer un rôle dans la leucémie. Les descripteurs génomiques correspondent à des gènes mutés dont le lien est connu avec la maladie.

Groupes et attributs :

- **Groupe cr60 :** rémission complète à 60 jours ou non,
- **Groupe os1 :** survie à 1 an ou non,
- **Groupe rel1 :** rémission à 1 an ou non,
- Genre : F ou M,
- Caryotype : normal ou translocation chromosomique,
- CD33strat : marqueur des cellules souches exprimé ou non exprimé,
- AlloSCT : transplantation de cellule ou non (greffe),
- DNMT3A, IDH1, IDH2, FLT3itd, NPM1, CEBPAdm, WT1, ASXL1 : gènes exprimés ou non exprimé,

Précision sur l'instance *Leukemia*

La problématique de cette instance est la même que pour les instances *cr60, os1* et *rel1* . Les observations sont des patients souffrant de leucémie et les attributs sont des gènes mutés qui sont soupçonnés de jouer un rôle dans la maladie. L'objectif de la caractérisation est d'identifier une combinaison de gènes afin de prédire ou d'expliquer le risque de rechute d'un patient.
Notons que trois gènes, FLT3 ITD, NPM1 et CEBPA, sont utilisés par la classification ELN (European Leukemia Net) pour faire un pronostic (la présence de NPM1 muté ou CEBPA sans FLT3 ITD donne un bon pronostic, la présence seule du gène muté FLT3 ITD donne un mauvais pronostic).

Groupe et attributs :

- **Groupe :** Caryotype : normal (nk) ou translocation 15-17,
- FLT3 ITD : expression de la mutation ou non,
- NPM1 : expression de la mutation ou non,
- 8577 gènes répartis sur les 23 paires de chromosomes : expression du gène ou non,

Précision sur les instances *rch8* et *xanthostgen*

Ces instances correspondent à des souches bactériennes pour des bactéries phytopathogènes du genre Ralstonia et Xanthomonas. Les attributs représentent l'expression ou non de gènes de virulence. Il peut s'agir de gènes de ménage, de gènes de résistance ou d'effecteurs de type III. Les groupes sont alors des pathovars, c'est à dire des groupes déterminés en fonction de la pathologie observée et de l'hôte affecté (voir section 2.1).

Groupes :

- ***rch8*** : alfalfae ; allii ; aurantifolii ; axonopodis ; begoniae ; citri ; citrumelo ; dieffenbachiae ; glycines ; malvacearum ; anacardii ; mangiferaeindicae ; manihotis ; fuscans ; NF1 ; NF2 ; NF3 ; ricini ; vasculorum ; vesicatoria ; vignicola.

- ***xanthostgen*** : X.a.pv.alfalfae ; X.a.pv.allii ; X.a.pv.anacardii ; X.a.pv.aurantifolii ; X.a.pv.a ; X.a.pv.begoniae ; X.a.pv.citri ; X.a.pv.citri.306 ; X.a.pv.citri.A*/Aw ; X.a.pv.citrumelo ; X.a.pv.dieffenbachiae ; X.a.pv.glycines ; X.a.pv.malvacearum ; X.a.pv.mangiferaeindicae ; X.a.pv.manihotis ; X.a.pv.musacearum ; X.a.pv.phaseoli.var.fuscans ; X.a.pv.phaseoli.var.non.fuscans.NF1 ; X.a.pv.phaseoli.var.non.fuscans.NF2 ; X.a.pv.phaseoli.var.non.fuscans.NF3 ; X.a.pv.ricini ; X.a.pv.spondiae ; X.a.pv.vasculorum ; X.a.pv.vesicatoria ; X.a.pv.vesicatoria.85-10 ; X.a.pv.vignicola ; X.c ; X.c.pv.armoriaciae ; X.c.pv.c ; X.c.pv.incanae ; X.c.pv.raphani ; X.o.pv.o ; X.o.pv.oryzicola.

Bibliographie

C. C. Aggarwal and C. K. Reddy. *Data clustering : algorithms and applications*. CRC Press, 2013. (cité en page 58).

R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993. (cité en page 72).

R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994. (cité en pages 74 et 77).

G. Alexe and P. L. Hammer. Spanned patterns for the logical analysis of data. *Discrete Applied Mathematics*, 154(7) :1039–1049, 2006. (cité en pages 28, 29 et 31).

Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. *Computational Logic—CL 2000*, pages 972–986, 2000. (cité en pages 73 et 78).

R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks*, 5(4) :537–550, 1994. (cité en page 64).

R. J. Bayardo Jr. Efficiently mining long patterns from databases. *ACM Sigmod Record*, 27(2) : 85–93, 1998. (cité en page 73).

T. Bayes, R. Price, and J. Canton. *An essay towards solving a problem in the doctrine of chances*. C. Davis, Printer to the Royal Society of London, 1763. (cité en page 60).

C. Berge. *Hypergraphs : combinatorics of finite sets*, volume 45. Elsevier, 1984. (cité en page 48).

T. O. Bonates, P. L. Hammer, and A. Kogan. Maximum patterns in datasets. *Discrete Applied Mathematics*, 156(6) :846–861, 2008. (cité en page 24).

C. Borgelt. Frequent item set mining. *Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery*, 2(6) :437–456, 2012. (cité en page 75).

E. Boros, P. L. Hammer, T. Ibaraki, and A. Kogan. Logical analysis of numerical data. *Mathematical Programming*, 79(1-3) :163–190, 1997. (cité en page 20).

E. Boros, P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, 12(2) : 292–306, 2000. (cité en pages 25, 28, 29, 30, 44, 127 et 128).

E. Boros, Y. Crama, P. L. Hammer, T. Ibaraki, A. Kogan, and K. Makino. Logical analysis of data : classification with justification. *Annals OR*, 188(1) :33–61, 2011. doi : 10.1007/s10479-011-0916-1. (cité en page 20).

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992. (cité en page 69).

A. Boudane, S. Jabbour, L. Sais, and Y. Salhi. A sat-based approach for mining association rules. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2472–2478. AAAI Press, 2016. (cité en page 78).

A. Boudane, J. Said, L. Sais, and Y. Salhi. Une approche basée sur sat pour l'énumération des regles d'association non redondantes. *Communication-Journées Francophones de Programmation par Contraintes, JFPC*, 2017. (cité en page 79).

T. Boureau, M. Kerkoud, F. Chhel, G. Hunault, A. Darrasse, C. Brin, K. Durand, A. Hajri, S. Poussier, C. Manceau, et al. A multiplex-pcr assay for identification of the quarantine plant pathogen xanthomonas axonopodis pv. phaseoli. *Journal of microbiological methods*, 92(1) :42–50, 2013. (cité en page 17).

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984. (cité en page 69).

F. Chhel, F. Lardeux, A. Goëffon, and F. Saubion. Minimum multiple characterization of biological data using partially defined boolean formulas. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1399–1405. ACM, 2012. (cité en pages 13, 36, 40, 42 et 43).

F. Chhel, F. Lardeux, F. Saubion, and B. Zanuttini. Application du problème de caractérisation multiple à la conception de tests de diagnostic pour la biologie végétale. *Revue des Sciences et Technologies de l'Information-Série RIA : Revue d'Intelligence Artificielle*, pages 649–668, 2013. (cité en pages 15, 44, 46 et 90).

I. Chikalov, V. Lozin, I. Lozina, M. Moshkov, H. S. Nguyen, A. Skowron, and B. Zielosko. Logical analysis of data : theory, methodology and applications. In *Three Approaches to Data Analysis*, pages 147–192. Springer, 2013. (cité en pages 20 et 128).

H. Chouaib. *Sélection de caractéristiques : méthodes et applications*. PhD thesis, Université Paris Descartes, 2011. (cité en page 67).

V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3) :233–235, 1979. (cité en pages 46 et 87).

S. Dasgupta. *The hardness of k-means clustering*. Department of Computer Science and Engineering, University of California, San Diego, 2008. (cité en page 58).

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977. (cité en pages 56 et 59).

L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3) : 297–302, 1945. (cité en page 56).

G. Dong and J. Li. Mining border descriptions of emerging patterns from dataset pairs. *Knowledge and Information Systems*, 8(2) :178–202, 2005. (cité en page 49).

C. Dupuis, M. Gamache, and J.-F. Pagé. Logical analysis of data for estimating passenger show rates at air canada. *Journal of Air Transport Management*, 18(1) :78–81, 2012. (cité en page 35).

E. Engvall and P. Perlmann. Enzyme-linked immunosorbent assay (elisa) quantitative assay of immunoglobulin g. *Immunochemistry*, 8(9) :871–874, 1971. (cité en page 15).

F. Ferri, V. Kadirkamanathan, and J. Kittler. Feature subset search using genetic algorithms. In *in : IEE/IEEE Workshop on Natural Algorithms in Signal Processing, IEE*. Citeseer, 1993. (cité en page 67).

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2) :179–188, 1936. (cité en page 61).

A. Gainer-Dewar and P. Vera-Licona. The minimal hitting set generation problem : algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1) :63–100, 2017. (cité en pages 49 et 50).

B. Ghattas and A. Ben Ishak. Sélection de variables pour la classification binaire en grande dimension : comparaisons et application aux données de biopuces. *Journal de la société française de statistique*, 149(3) :43–66, 2008. (cité en pages 69 et 70).

D. Goldberg. *Genetic algotithms in Search, optimization anf machine learning*. Addison-wesley, 1989. (cité en page 67).

G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI*, volume 90, 2003. (cité en page 75).

G. Grahne and J. Zhu. Reducing the main memory consumptions of fpmax* and fpclose. In *Proc. Workshop Frequent Item Set Mining Implementations (FIMI 2004, Brighton, UK), Aachen, Germany*, 2004. (cité en page 75).

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar) :1157–1182, 2003. (cité en page 64).

A. Hajri, C. Brin, G. Hunault, F. Lardeux, C. Lemaire, C. Manceau, T. Boureau, and S. Poussier. A «repertoire for repertoire» hypothesis : Repertoires of type three effectors are candidate determinants of host specificity in xanthomonas. *PLoS One*, 4(8) :e6632, 2009. (cité en page 17).

M. A. Hall and L. A. Smith. Feature subset selection : a correlation based filter approach. 1997. (cité en page 65).

P. L. Hammer. Partially defined boolean functions and cause-effect relationships. In *International conference on multi-attribute decision making via or-based expert systems*, 1986. (cité en page 18).

P. L. Hammer and T. O. Bonates. Logical analysis of data—an overview : from combinatorial optimization to medical applications. *Annals of Operations Research*, 148(1) :203–225, 2006. (cité en page 20).

P. L. Hammer, A. Kogan, B. Simeone, and S. Szedmák. Pareto-optimal patterns in logical analysis of data. *Discrete Applied Mathematics*, 144(1) :79–102, 2004. (cité en pages 20, 24, 28 et 33).

P. L. Hammer, A. Kogan, and M. A. Lejeune. A logical analysis of banks' financial strength ratings. *Expert Systems with Applications*, 39(9) :7808–7821, 2012. (cité en page 35).

J. A. Hartigan and M. A. Wong. Algorithm as 136 : A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1) :100–108, 1979. (cité en pages 58 et 145).

J. W. Holley and J. P. Guilford. A note on the g index of agreement. *Educational and psychological measurement*, 24(4) :749–753, 1964. (cité en page 56).

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. (cité en pages 44, 46 et 47).

L. Kaufman and P. J. Rousseeuw. Partitioning around medoids (program pam). *Finding groups in data : an introduction to cluster analysis*, pages 68–125, 1990. (cité en page 59).

D. J. Kavvadias and E. C. Stavropoulos. An efficient algorithm for the transversal hypergraph generation. *J. Graph Algorithms Appl.*, 9(2) :239–264, 2005. (cité en page 49).

D. Knuth. The art of programming. *ITNow*, 53(4), 2011. (cité en page 50).

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97 (1-2) :273–324, 1997. (cité en pages 63 et 66).

W. A. Kosters and W. Pijls. Apriori, a depth first implementation. In *FIMI*, 2003. (cité en page 75).

M. Kudo and J. Sklansky. Comparison of algorithms that select features for pattern classifiers. *Pattern recognition*, 33(1) :25–41, 2000. (cité en pages 67 et 68).

N. Kwak and C.-H. Choi. Input feature selection for classification problems. *IEEE Transactions on Neural Networks*, 13(1) :143–159, 2002. (cité en page 64).

C. Lazar, J. Taminau, S. Meganck, D. Steenhoff, A. Coletta, C. Molter, V. de Schaetzen, R. Duque, H. Bersini, and A. Nowe. A survey on filter techniques for feature selection in gene expression microarray analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4) :1106–1119, 2012. (cité en page 64).

T. Li. A general model for clustering binary data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 188–197. ACM, 2005. (cité en page 70).

D.-I. Lin and Z. M. Kedem. Pincer-search : A new algorithm for discovering the maximum frequent set. In *International conference on Extending database technology*, pages 103–119. Springer, 1998. (cité en page 73).

L. Lin and Y. Jiang. The computation of hitting sets : Review and new algorithms. *Information Processing Letters*, 86(4) :177–184, 2003. (cité en page 49).

H. Liu and R. Setiono. Feature selection and classification-a probabilistic wrapper approach. In *Proceedings of 9th International Conference on Industrial and Engineering Applications of AI and ES*, pages 419–424, 1997. (cité en page 66).

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967. (cité en pages 58 et 145).

T. Marill and D. Green. On the effectiveness of receptors in recognition systems. *IEEE transactions on Information Theory*, 9(1) :11–17, 1963. (cité en page 66).

M.-A. Mortada, T. Carroll, S. Yacout, and A. Lakis. Rogue components : their effect and control using logical analysis of data. *Journal of Intelligent Manufacturing*, 23(2) :289–302, 2012. (cité en page 34).

K. Murakami and T. Uno. Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics*, 170 :83–94, 2014. (cité en pages 50 et 123).

N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *International Conference on Database Theory*, pages 398–416. Springer, 1999. (cité en page 73).

K. Person. On lines and planes of closest fit to system of points in space. philiosophical magazine, 2, 559-572, 1901. (cité en page 61).

P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern recognition letters*, 15(11) :1119–1125, 1994. (cité en page 66).

J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1) :81–106, 1986. (cité en page 69).

J. R. Quinlan. Learning logical definitions from relations. *Machine learning*, 5(3) :239–266, 1990. (cité en page 42).

J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description lenght principle. *Information and computation*, 80(3) :227–248, 1989. (cité en page 69).

C. R. Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2) :159–203, 1948. (cité en page 56).

A. Reddy, H. Wang, H. Yu, T. O. Bonates, V. Gulabani, J. Azok, G. Hoehn, P. L. Hammer, A. E. Baird, and K. C. Li. Logical analysis of data (lad) model for the early diagnosis of acute ischemic stroke. *BMC medical informatics and decision making*, 8(1) :30, 2008. (cité en page 34).

R. Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1) :57–95, 1987. (cité en page 49).

H. S. Ryoo and I.-Y. Jang. Milp approach to pattern generation in logical analysis of data. *Discrete Applied Mathematics*, 157(4) :749–761, 2009. (cité en pages 25, 26 et 27).

P. H. Sneath. Some thoughts on bacterial classification. *Microbiology*, 17(1) :184–200, 1957. (cité en page 56).

R. R. Sokal. A statistical method for evaluating systematic relationships. *Univ Kans Sci Bull*, 38 :1409–1438, 1958. (cité en page 56).

T. Toda. Hypergraph transversal computation with binary decision diagrams. In *International Symposium on Experimental Algorithms*, pages 91–102. Springer, 2013. (cité en page 50).

T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm : An efficient algorithm for enumerating frequent closed item sets. In *FIMI*, volume 90, 2003. (cité en page 75).

V. Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013. (cité en page 69).

A. W. Whitney. A direct method of nonparametric measurement selection. *IEEE Transactions on Computers*, 100(9) :1100–1103, 1971. (cité en page 66).

I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining : Practical machine learning tools and techniques*. Morgan Kaufmann, 2016. (cité en page 65).

F. Wotawa. A variant of reiter's hitting-set algorithm. *Information Processing Letters*, 79(1) : 45–51, 2001. (cité en page 49).

UNIVERSITE BRETAGNE LOIRE

université angers

Thèse de Doctorat

Arthur CHAMBON

Caractérisation logique de données

Application aux données biologiques

Logical Characterization of Data

Application to Biological Data

Résumé

L'analyse de groupes de données binaires est aujourd'hui un défi au vu des quantités de données collectées. Elle peut être réalisée par des approches logiques. Ces approches identifient des sous-ensembles d'attributs booléens pertinents pour caractériser les observations d'un groupe et peuvent aider l'utilisateur à mieux comprendre les propriétés de ce groupe.
Cette thèse présente une approche pour caractériser des groupes de données binaires en identifiant un sous-ensemble minimal d'attributs permettant de distinguer les données de différents groupes.
Nous avons défini avec précision le problème de la caractérisation multiple et proposé de nouveaux algorithmes qui peuvent être utilisés pour résoudre ses différentes variantes. Notre approche de caractérisation de données peut être étendue à la recherche de patterns (motifs) dans le cadre de l'analyse logique de données. Un pattern peut être considéré comme une explication partielle des observations positives pouvant être utilisées par les praticiens, par exemple à des fins de diagnostic. De nombreux patterns existent et plusieurs critères de préférence peuvent être ajoutés pour se concentrer sur des ensembles plus restreints (prime patterns, strong patterns,...). Nous proposons donc une comparaison entre ces deux méthodologies ainsi que des algorithmes pour générer des patterns. Un autre objectif est d'étudier les propriétés des solutions calculées en fonction des propriétés topologiques des instances. Des expériences sont menées sur de véritables ensembles de données biologiques.

Mots clés

Caractérisation logique de données, Données biologiques, Fonctions booléennes.

Abstract

Analysis of groups of binary data is now a challenge given the amount of collected data. It can be achieved by logical based approaches. These approaches identify subsets of relevant Boolean attributes to characterize the observations of a group and may help the user to better understand the properties of this group.
This thesis presents an approach for characterizing groups of binary data by identifying a minimal subset of attributes that allows to distinguish data from different groups.
We have precisely defined the multiple characterization problem and proposed new algorithms that can be used to solve its different variants. Our data characterization approach can be extended to search for patterns in the framework of logical analysis of data. A pattern can be considered as a partial explanation of the positive observations that can be used by practitioners, for instance for diagnosis purposes. Many patterns may exist and several preference criteria can be added in order to focus on more restricted sets of patterns (prime patterns, strong patterns, ...). We propose a comparison between these two methodologies as well as algorithms for generating patterns. The purpose is also to precisely study the properties of the solutions that are computed with regards to the topological properties of the instances. Experiments are thus conducted on real biological data.

Key Words

Logical Characterization of Data, Biological Data, Boolean Functions.