Modèles de classification en classes empiétantes, cas des modèles arborés

Célia Châtel

Aix*Marseille université
Initiative d'excellence

École Doctorale en Mathématiques et Informatique de Marseille

Modèles de classification en classes empiétantes, cas des modèles arborés

THÈSE

présentée et soutenue publiquement le 7 décembre 2018

pour l'obtention du

Doctorat d'Aix Marseille Université

(mention informatique)

par

Célia Châtel

Composition du jury

Rapporteurs :	Pascale Kuntz-Cosperec, Professeur	LINA, Université de Nantes
	Engelbert Mephu-Nguifo, Professeur	LIMOS, Université de Clermont-Ferrand
Examinateurs :	Karell Bertet, Maître de conférences, HdR	L3I, Université de La Rochelle
	Amedeo Napoli, Directeur de recherche	LORIA, CNRS
	Yann Vaxès, Professeur	LIS, Aix Marseille Université
Directeurs :	François Brucker, Professeur	LIS, École Centrale Marseille
	Pascal Préa, Maître de conférences, HdR	LIS, École Centrale Marseille

Laboratoire d'Informatique et Systèmes - UMR 7020

Mis en page avec la classe thesul.

Remerciements

Je voudrais tout d'abord adresser mes remerciements à Pascale Kuntz et Engelbert Mephu Nguifo pour avoir accepté de rapporter cette thèse. Je suis très honorée de l'intérêt qu'ils ont porté à mon travail et tiens à les remercier pour le temps qu'ils ont passé à le lire et pour leurs remarques.

Je tiens également à remercier Karell Bertet, Amedeo Napoli et Yann Vaxès d'avoir accepté d'examiner mon travail.

Merci également à François Brucker et Pascal Préa, mes directeurs de thèse qui m'ont donné envie de poursuivre mes études en informatique quand j'étais étudiante à l'École Centrale de Marseille, qui m'ont ensuite encouragée à réaliser un stage en recherche puis permis de suivre un master d'informatique fondamentale en parallèle de ma troisième année à Centrale. Merci de m'avoir ensuite accueillie en thèse. Je n'en serais pas là sans vous.

Merci à toute l'équipe ACRO pour ses séminaires qui m'ont fait découvrir de nombreux sujets de recherche.

Un grand merci à tout ceux qui m'ont donné envie de me lancer dans la recherche, que je citerai sans ordre particulier : Liva, Sandrine, Pierre, Valentin, Cécile, et tout ceux que j'oublie.

Je tiens à remercier également très sincèrement Sylvie, Nadine, Martine, Kai et Manuel qui nous apportent un soutien administratif, logistique et technique sans faille. Nous avons une grande chance de vous avoir au sein du laboratoire.

Je ne remercierai jamais assez mes parents pour leur soutien permanent et pour avoir cru en moi depuis toujours.

Merci enfin à Laurine pour m'avoir supportée (dans tous les sens du terme) durant ces années parfois intenses. Merci à Delphine pour sa présence malgré la distance. Merci également à Btiste dont la présence soyeuse a égayé de nombreux moments de mon parcours. Merci à Alexandre et Chloé pour leur duo parfois étonnant mais toujours amusant, pour leur tentative de relecture, et pour leur abandon. Enfin, merci à Pichachu et Zerator, pour la distraction continuelle qu'ils ont pu m'apporter.

Sommaire

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction

Nous traitons dans cette thèse de la classification de données du point de vue de l'analyse de données mais aussi de celui de l'apprentissage automatique supervisé. Dans les deux cas, nous disposons d'une description d'un ensemble de données (des individus, des objets) observées dans une certaine base (des attributs). Les données peuvent alors être de deux types :

— des données représentées par une matrice binaire individus/attributs qui indique pour chaque individu s'il possède ou non une certaine caractéristique. Par exemple un ensemble d'animaux et le fait de posséder ou non une queue, d'être vivipare, d'avoir des plumes, etc... ;

— des données vectorielles représentant chaque individu par un vecteur à la manière de la représentation d'un point dans le plan par ses coordonnées.

L'analyse de données vise à représenter des données existantes dans un modèle connu et contraint afin de pouvoir en extraire de l'information et pouvoir interpréter le résultat obtenu [40]. Il s'agit de construire un modèle à la fois intelligible, c'est-à-dire qui peut être compris, mais également interprétable, qui apporte de l'information non triviale sur les données. L'analyse en composantes principales [38] est l'exemple le plus représentatif de l'analyse de données. Il s'agit de projeter les données sur un petit nombre de dimensions et de donner un sens à chaque axe ainsi créé, en perdant le moins d'information possible, mais aussi en mesurant la quantité d'information perdue. Le modèle est lisible : il est constitué de droites et sépare donc simplement les individus entre un côté de la droite et l'autre. Il est intelligible : il est possible de donner un sens aux axes mis en évidence et retrouver les informations évidentes sur les données. Enfin, il est interprétable et permet de trouver des informations supplémentaires sur les données. La classification peut être considérée comme une déclinaison ensembliste de l'analyse en composantes principales : on cherche à généraliser la projection sur une droite.

Dans le contexte de la classification, il s'agira de réussir à mettre ensemble les choses qui se ressemblent et séparer celles qui diffèrent en permettant ainsi d'extraire de l'information sur les données [31]. Le défi est donc de réussir à la fois à regrouper et à séparer ; de créer des groupes d'individus qui se ressemblent au sein d'un groupe tout en étant différents des autres groupes. La notion de ressemblance et de différence représente d'ailleurs un défi en soi. Dans ce cadre, deux visions distinctes se dessinent.

— D'un côté, une approche locale, mise en œuvre par les treillis de concepts, centrée

sur la représentation des données, permet de représenter exhaustivement des données et d'étudier les liens entre les objets et les caractéristiques qui les décrivent. Ces modèles bénéficient de la possibilité d'effectuer des comparaisons locales aisément mais les résultats obtenus sont de grande taille et ne permettent que difficilement une représentation globale tout en étant laborieux à calculer entièrement.

— De l'autre côté, une approche plus globale, dont l'idée principale est de projeter les données sur un modèle contraint et bien connu afin de pouvoir les étudier et de donner un sens aux classes créées. Cette projection revient à approximer et donc perdre une partie de l'information. Néanmoins, elle permet une étude globale et une représentation aisée en limitant le nombre de classes créées par le modèle et en contraignant les relations entre ces classes. Dans ce cas, le choix d'un modèle adapté aux données est primordial.

Ces deux visions complémentaires sont donc utilisées en pratique pour résoudre des problèmes différents en tirant profit de leurs forces respectives.

L'apprentissage automatique, pour sa part, dispose de données dont la structure est définie : chaque exemple est assigné à une classe déterminée à l'avance (par exemple, un ensemble de mails dont le destinataire a renseigné s'ils étaient des spams ou non). L'objectif est alors d'être capable de donner, pour un nouvel exemple, une prédiction de la classe à laquelle il appartient (savoir si un nouvel email est un spam par exemple). Il s'agit donc de retrouver un ordre donné, a priori quelconque, sans chercher à en expliquer la structure. Certains modèles peuvent néanmoins présenter un traitement intéressant des données. C'est le cas par exemple des arbres de décision [14], basés sur la dichotomie : le modèle se pose successivement des questions dont la réponse peut être soit oui soit non.

Ces deux approches de la classification peuvent sembler s'opposer. En effet, les méthodes de l'apprentissage automatique les plus performantes en prédiction sont souvent de véritables boîtes noires. Les méthodes de l'apprentissage profond en sont l'exemple le plus parlant. Il est impossible pour un être humain ou, selon les modèles utilisés, pour une personne non experte, d'expliquer le résultat du modèle ou encore d'en extraire une information sur les données utilisées. Cette question de la compréhension théorique des modèles d'apprentissage profond est d'ailleurs devenue en elle-même un sujet de recherche ([39] pour une bibliographie fournie sur le sujet). Comme se le demandait Ali Rahimi, lors de son exposé à NIPS en 2017 [1] : "L'apprentissage automatique est-il devenu de l'alchimie ?". De l'autre côté, les modèles de classification peuvent parfois paraître inapplicables en pratique si l'utilisateur ne sait pas ce qu'il cherche dans la structure. Les treillis de concepts par exemple, utilisés en analyse de concepts formels (FCA) [25] sont le plus souvent de taille exponentielle et sont donc adaptés à une utilisation locale centrée autour d'un point de départ connu. Ils sont également utiles en recherche d'information [18]. L'objectif n'étant pas le même, dans le cadre de leur utilisation, ces inconvénients n'en sont pas réellement. En effet, les méthodes de l'apprentissage profond visent à fournir une réponse sur la classe de sortie d'un exemple sans en expliquer la raison tandis que les

treillis de concepts permettent de représenter les relations locales individus/attributs de manière exhaustive.

Les liens entre les arbres de décision et les treillis de concepts ont été étudiés ([37] pour une revue des principaux travaux sur ces liens) et des modèles mêlant les deux comme les treillis dichotomiques ([9], [27]) ont déjà été proposés. Nous cherchons ici à approfondir ces liens entre l'apprentissage automatique et l'analyse de données et à créer des modèles s'inspirant des deux mondes pour être à la fois interprétables et utilisables en prédiction.

Les deux modèles les plus utilisés en analyse de données sont les partitions et les hiérarchies. Le partitionnement produit un ensemble de classes deux à deux disjointes tandis que les hiérarchies permettent l'inclusion d'une classe dans une autre. Dans les deux cas, l'empiétance n'est pas permise. C'est-à-dire que l'intersection de deux classes ne peut être que vide ou l'une des deux classes elle-même. Dans les modèles utilisés en apprentissage automatique pour la prédiction, on conserve cette idée de hiérarchie à travers les arbres de décision par exemple. Les classes créées par ces modèles forment une hiérarchie. Les hiérarchies peuvent d'ailleurs toutes être représentées sous la forme d'arbre.

Cette non empiétance des classes permet de produire des modèles simples, facilement interprétables et basés sur de solides théories mathématiques. La simplicité d'utilisation des arbres de décision en est un exemple. Néanmoins, ce choix de modèle non empiétant peut conduire à occulter de l'information. Comment représenter par exemple, le lien entre une plante hybride et les variétés de plantes croisées pour l'obtenir ? Comment prendre en compte la possibilité qu'un texte appartienne à plusieurs genres en analyse textuelle ? La perte de telles informations peut de plus créer de la confusion dans l'interprétation du résultat obtenu.

Cette hypothèse de non empiétance peut être affaiblie par l'utilisation des hiérarchies faibles [6], initialement développées pour la biologie. Dans une hiérarchie faible, l'intersection de trois classes est égale à l'intersection de deux d'entre elles. Une contrainte un peu plus forte sur la structure des classes mais adaptée au traitement de données issues de la phylogénie par exemple, est la contrainte de système totalement équilibré. Ces systèmes de classes sont des systèmes arborés généralisant les arbres phylogénétiques. Ils ne contiennent pas de cycles et sont des hiérarchies faibles. Ces modèles peuvent être vus sous différents angles : celui des treillis (dans lequel ils sont les treillis démontables ou sans couronnes [29], [15]), celui des hypergraphes (dans lequel ils sont les hypergraphes totalement équilibrés [23], [32]), celui des matrices (où ils sont les matrices Γ-free [4]) ou encore dans le monde des dissimilarités, qui proposent une représentation différente des données, pouvant être liée aux descriptions classiques de présence-absence [30]. Chacune de ses structures apportera ses avantages et ses inconvénients, permettant d'utiliser chacune d'entre elles dans un but précis. Nous nous intéresserons particulièrement dans ce travail à toutes ces structures totalement équilibrées et montrerons l'intérêt qu'elles peuvent avoir en classification et en apprentissage.

Nous approfondissons dans cette thèse les liens entre l'apprentissage automatique et les systèmes de classes totalement équilibrés en montrant l'intérêt que peut revêtir l'usage de systèmes totalement équilibrés afin de permettre à la fois l'interprétation et la prédiction. Nous développons pour cela certaines propriétés théoriques des modèles totalement équilibrés. Les contributions principales de ce travail sont les suivantes :

- un modèle d'apprentissage automatique efficace en prédiction et permettant la classification de données sans utiliser les classes de sorties des données lors de la construction du modèle ;
- un modèle de classification utilisant des idées d'apprentissage automatique ;
- une preuve de l'équivalence entre les systèmes totalement équilibrés et les systèmes binarisables ;
- une caractérisation des hypergraphes binaires par une séquence d'arbres mixtes ;
- une approximation des systèmes totalement équilibrés par des systèmes binaires ;
- une approximation d'hypergraphes quelconques par un hypergraphe binaire ;
- l'introduction des dissimilarités totalement équilibrées et l'approximation d'une dissimilarité quelconque par une dissimilarité totalement équilibrée.

Ce travail se compose de six chapitres présentant les résultats principaux obtenus.

Le Chapitre 1 introduit les objets utilisés durant tout le texte et donne quelques propriétés et résultats connus et utiles au développement de ce travail. Il présente l'équivalence entre les objets utilisés dans le cadre de la classification et les formalismes propres à chaque type d'objets : les hypergraphes, les matrices, les systèmes de classes, les treillis et les dissimilarités. Il introduit également un certain nombre de notions issues de l'apprentissage automatique et présente en détails les arbres de décision, inspirations majeures des méthodes d'apprentissage proposées dans cette thèse. Enfin, il présente les modèles totalement équilibrés, objets principaux de ce travail, dans chacun des formalismes décrits plus tôt.

Le Chapitre 2 présente une première approche de l'idée de classification supervisée en utilisant des méthodes d'analyse de données c'est-à-dire sans utilisation des classes de sorties dans la construction du modèle : les arbres de décision K-Means. Ce modèle, basé sur l'utilisation du partitionnement K-Means permet d'obtenir une hiérarchie de classes utilisable comme un arbre de décision pour la prédiction de la classe d'un nouvel exemple. Il ne présente pas encore la possibilité de prendre en compte de l'empiétance entre les classes mais permet de mettre en évidence l'intérêt de la description des données et l'information qu'elle contient. Ce chapitre présente également les résultats obtenus en apprentissage, comparés à ceux obtenus par des arbres de décision classiques sur divers jeux de données réelles. Nous étendons également les modèles d'arbres de décision K-Means aux forêts d'arbres de décision K-Means en nous inspirant des forêts de l'apprentissage automatique.

Le Chapitre 3 propose un modèle totalement équilibré de classification en classes empiétantes par une construction simple basée sur l'utilisation d'arbres couvrants minimum

et inspirée des arbres de décision. Ce modèle permet d'obtenir un ensemble de classes empiétantes représentant les données et utilisables en classification. Il permet de mettre en évidence la possibilité de réaliser de la classification en apprentissage automatique avec de l'empiétance parmi les classes utilisées et propose une approximation par un modèle totalement équilibré de données vectorielles.

En apprentissage automatique, les arbres de décision sont le plus souvent binaires par souci de simplicité. Le Chapitre 4, centré sur la vision du problème par les treillis, étend la notion de binarité des arbres aux treillis et montre que les treillis binarisables (*i.e.* qui peuvent être plongés dans un treillis binaire) sont exactement les treillis démontables. Ce chapitre propose un algorithme permettant d'approximer n'importe quel treillis démontable non binaire par un treillis binaire par l'ajout du moins d'éléments possible. Il présente également une application des résultats dans le cadre de l'analyse de concepts formels.

Le Chapitre 5, rédigé du point de vue des hypergraphes, étend la notion d'arbre binaire à celle d'hypergraphe binaire de manière identique au travail réalisé dans le Chapitre 4 pour les treillis et propose une caractérisation des hypergraphes binaires par une séquence d'arbres mixtes similaire à celle proposée par Lehel [32] pour les hypergraphes totalement équilibrés. Cette caractérisation par une construction simple d'arbres permet de prouver de manière alternative le résultat d'équivalence entre les systèmes totalement équilibrés et les systèmes binaires déjà présenté dans le Chapitre 4. De plus, cette construction permet d'approximer un hypergraphe totalement équilibré par un hypergraphe binaire (d'une manière plus générale que l'approximation d'un treillis démontable par un treillis binaire présenté dans le chapitre précédent) mais également d'approximer un hypergraphe quelconque par un hypergraphe binaire (et donc totalement équilibré).

Enfin, le Chapitre 6 se place dans le cadre des dissimilarités pour définir les dissimilarités totalement équilibrées (équivalentes aux hypergraphes totalement équilibrés). Nous donnons alors un algorithme de calcul des classes associées à une dissimilarité donnée, un algorithme permettant de reconnaître en temps polynomial ces dissimilarités particulières et un algorithme permettant d'approximer n'importe quelle dissimilarité par une dissimilarité totalement équilibrée.

Chapitre 1

Préliminaires

Ce chapitre expose les différentes notions nécessaires à la compréhension du reste du manuscrit. Nous présentons en détails les différentes structures discrètes qui sont utilisées dans le cadre de la classification de données ainsi que leur intérêt dans ce domaine. Nous parlerons ainsi d'*objets* ou d'*individus* pour désigner les éléments observés et d'*attributs* pour faire référence aux caractéristiques utilisées pour les décrire. Nos modèles visent à réunir les individus en un certain nombre de *classes i.e.* de groupes à la fois homogènes et distincts les uns des autres. Nous donnons également quelques notions d'apprentissage automatique. Enfin, nous abordons l'analyse de données et précisons les différences avec l'apprentissage automatique. Nous présentons également en détails les modèles totalement équilibrés et leur intérêt en analyse de données.

1.1 Structures discrètes

Cette section présente les structures discrètes utilisées pour la classification dans la suite du texte. Nous présentons successivement ces structures :

- les *hypergraphes* (Section 1.1.1), qui permettent la généralisation du lien entre deux individus, qui peut être représenté par les arêtes d'un graphe, à des ensembles de n individus ;
- les *matrice binaires* (Section 1.1.2), naturellement liées aux hypergraphes mais permettant, en plus, d'ordonner les classes et les objets à travers l'ordre des lignes et des colonnes, et ainsi d'étudier certains ordres particuliers (comme les ordres doublement lexicaux que nous détaillerons) ;
- les *systèmes de classes* (Section 1.1.3), restriction au cas des hypergraphes clos par intersection, cas d'intérêt en classification pour l'information que contient l'intersection de deux classes ;
- les *treillis* (Section 1.1.4), qui sont très utilisés pour l'exhaustivité de leur description des relations locales entre les objets et les attributs ;
- les *dissimilarités* (Section 1.1.5), qui utilisent une représentation des données par

une description locale en donnant une mesure de l'intensité du lien entre deux individus.

Tous ces objets sont équivalents ou fortement liés et permettent de représenter des données par la description des individus ou la ressemblance entre ces individus. Chacun d'entre eux est utilisé dans un cadre particulier selon le type d'analyse à faire sur les données. Nous ne considérerons tout au long de ce travail que des structures finies. Toutes les références à des hypergraphes, matrices, treillis ou dissimilarités correspondent donc à des objets finis.

1.1.1 Graphes et hypergraphes

Nous présentons dans cette partie quelques définitions générales associées aux hypergraphes qui seront utilisées durant tout le manuscrit. Nous commençons par définir les graphes, structure bien connue dont les hypergraphes sont une généralisation.

Définition 1.1.1. Un *graphe* $\mathcal{G}$ est un couple (V, E) avec V un ensemble et E un ensemble de couples d'éléments de V.

V est l'ensemble de *sommets* et E l'ensemble d'*arêtes*.

Cette structure de données très classique et utilisée dans de nombreuses applications permet de représenter les relations (les arêtes) entre deux individus (les sommets). Il est naturel de vouloir représenter des liens entre plus de deux objets, surtout dans le cadre de la classification où les classes doivent pouvoir être de tailles quelconques.

Les graphes peuvent être *orientés* ou non, ce qui signifie que leurs arêtes ont une direction (on parlera alors d'*arcs*) ou non. On appellera *graphe mixte* un graphe $\mathcal{G} = (V, E, \overrightarrow{E})$ possédant un ensemble d'arêtes non orientées E et un ensemble d'arcs $\overrightarrow{E}$.

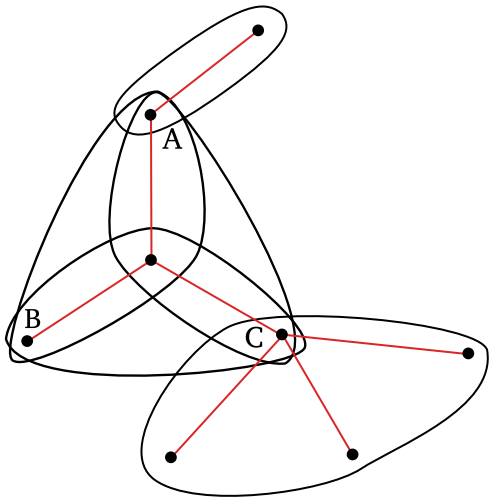
Dans le cadre des graphes non orientés, un *cycle* est une suite d'arêtes consécutives (ayant un sommet en commun) dont les deux extrémités sont identiques. De plus, un graphe est dit *connexe* si pour toute paire de sommets u, v, il existe un chemin d'arêtes liant u à v.

Définition 1.1.2. Un *arbre* est un graphe non orienté, acyclique et connexe.

Nous nous intéresserons dans ce travail à des structures arborés généralisant cette notion d'arbre.

Définition 1.1.3. Un *hypergraphe* $\mathcal{H}$ est un couple (V, E) où V est un ensemble de sommets et $E \subseteq 2^V$.

Cet objet généralise les graphes non orientés en permettant des "arêtes" entre plus de deux sommets. Comme dans les graphes, on appellera les éléments de V des *sommets*. Les éléments de E seront quant à eux appelés des *hyperarêtes*. Ces structures permettent de représenter un modèle de classification de manière simple : les individus ou objets sont des sommets tandis que les classes sont des hyperarêtes.



Arbre support représenté en rouge et hyperarêtes de l'hypergraphe en noir.

FIGURE 1.1 – Exemple d'hyperarbre et d'un arbre support

Le *sous-hypergraphe induit* de $\mathcal{H} = (V, E)$ sur un ensemble $A \subseteq V$ est l'hypergraphe $\mathcal{H}' = (A, \{e \cap A \neq \emptyset \mid e \in E\})$ noté $\mathcal{H}_{|A}$. On notera de même $\mathcal{G}_{|V'}$ le sous-graphe induit de $\mathcal{G}$ sur V' *i.e.* le graphe $(V', \{(u, v) \in E \mid u, v \in V'\}$. Nous utiliserons également cette notation pour désigner la restriction d'un ensemble à un de ses sous-ensembles.

On appelle *graphe support* d'un hypergraphe $\mathcal{H} = (V, E)$ un graphe $G = (V, E')$ tel que toute hyperarête de $\mathcal{H}$ est une partie connexe de G.

Définition 1.1.4. Un hypergraphe $\mathcal{H} = (V, E)$ est un *hyperarbre* ssi il existe un arbre $T = (V, E')$ tel que T est un graphe support de $\mathcal{H}$.

On appelle T un *arbre support*.

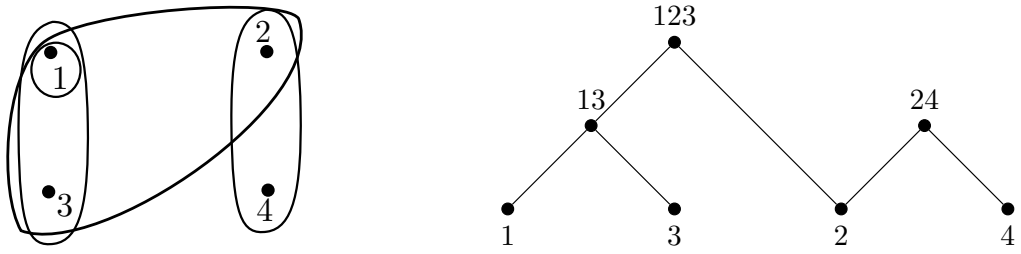
Par exemple, la Figure 1.1 représente un hyperarbre et un arbre support possible.

On appelle *relation de couverture* et on note $\prec$ la relation sur E définie par $u \prec v$ ssi :

— $u \subsetneq v$,

— $\nexists x \in E, u \subsetneq x \subsetneq v$.

Cette relation peut être représentée par un *diagramme de Hasse* : un diagramme tel que toutes les hyperarêtes sont des sommets du diagramme et il existe une arête ascendante d'un sommet a à un sommet b ssi $a \prec b$. Il existe donc un chemin d'arêtes ascendantes entre deux éléments a et b ssi $a \subseteq b$. Notre usage des hypergraphes étant la classification, nous ajouterons toujours les sommets de l'hypergraphe au diagramme de Hasse comme s'ils étaient des hyperarêtes singletons. Cela permet la représentation claire des individus. La Figure 1.2b représente le diagramme de Hasse associé à l'hypergraphe de la Figure 1.2a.



(a) Hypergraphe

(b) Diagramme de Hasse

FIGURE 1.2 – Exemple d'hypergraphe et de son diagramme de Hasse

1.1.2 Matrices

La description de données peut également se faire grâce à des matrices binaires. Cette représentation des données observées est la plus naturelle possible : chaque individu est représenté par une ligne et chaque attribut par une colonne. De plus, en considérant les lignes comme les sommets d'un hypergraphe et les colonnes comme ses hyperarêtes, l'équivalence entre les matrices et les hypergraphes est évidente. Les colonnes d'une matrice peuvent alors être vues directement comme les classes du modèle.

Plus formellement, une matrice binaire $n \times m$ $\mathcal{M}$ est équivalente à l'hypergraphe $\mathcal{H}(\mathcal{M}) = (\{1, \ldots, n\}, \{C_1, \ldots, C_m\})$ avec pour tout $1 \leq j \leq m, C_j = \{1 \leq i \leq n \mid \mathcal{M}_{i,j} = 1\}$. Réciproquement, étant donné un hypergraphe $\mathcal{H} = (V, E)$ avec $V = \{v_1, \ldots, v_n\}$ et $E = \{E_1, \ldots, E_n\}$, $\mathcal{H}$ est équivalent à la matrice $n \times m$ $\mathcal{M}(\mathcal{H})$ avec pour tout i, j, $\mathcal{M}(\mathcal{H})_{i,j} = 1$ si et seulement si $v_i \in E_j$.

Définition 1.1.5. Une matrice $\mathcal{M}$ est dite Γ-*free* si pour tout $i < i', j < j'$, $\mathcal{M}_{i,j} = 1, \mathcal{M}_{i,j'} = 1, \mathcal{M}_{i',j} = 1$ implique $\mathcal{M}_{i',j'} = 1$.

Les matrices permettent de plus d'associer un ordre aux lignes et/ou aux colonnes. On appellera *ordre* Γ-*free* d'une matrice $\mathcal{M}$ un ordre des lignes et des colonnes de $\mathcal{M}$ tel que la matrice réordonnée est Γ-free.

Un autre type d'ordre particulier pouvant être défini pour les matrices binaires est l'ordre *doublement lexical*.

Définition 1.1.6. Soit $\mathcal{M}$ une matrice binaire. Un *ordre doublement lexical* de $\mathcal{M}$ est un ordre des lignes et des colonnes de $\mathcal{M}$ tel que :

- les lignes, considérées comme des entiers binaires lus de droite à gauche apparaissent en ordre croissant ;
- les colonnes, considérées comme des entiers binaires lus de bas en haut apparaissent en ordre croissant.

Toute matrice binaire admet un ordre doublement lexical [36]. La Table 1.1a représente la matrice binaire correspondant à l'hypergraphe de la Figure 1.2a. Par souci de lisibilité,

	c_1	c_2	c_3	c_4
l_1	×		×	×
l_2		×	×	
l_3	×		×	
l_4		×		

(a) Exemple de matrice binaire

	c_2	c_4	c_1	c_3
l_4	×			
l_2	×			×
l_3			×	×
l_1		×	×	×

(b) Ordre doublement lexical de la matrice binaire

TABLE 1.1 – Exemple d'ordre doublement lexical d'une matrice binaire

les 1 sont remplacés par des croix et les 0 par un espace vide. L'ordre de cette matrice n'est pas Γ-free. En effet, $\mathcal{M}_{1,1} = \mathcal{M}_{1,4} = \mathcal{M}_{3,1} = 1$ et $\mathcal{M}_{3,4} = 0$ par exemple, ce qui forme un Γ. La Table 1.1b représente un ordre doublement lexical de cette matrice. Cet ordre est pour sa part Γ-free.

Étant donné une matrice binaire $n \times m$ $\mathcal{M}$, un algorithme de Spinrad [41] permet de calculer un ordre doublement lexical de $\mathcal{M}$ en $\mathcal{O}(nm)$.

1.1.3 Systèmes de classes

Dans le cadre de la classification, les modèles utilisés peuvent être clos par intersection : l'existence de deux classes non disjointes et incomparables entraîne l'existence d'une troisième classe générée par leur intersection. Ceci permet au modèle de contenir plus d'informations, les intersections de classes faisant sens pour l'interprétation.

Définition 1.1.7. Un ensemble S est un *système de classes* d'un ensemble fini V si :

(i) $S \subseteq 2^V$,

(ii) $\forall A, B \in S, A \cap B \in S$,

(iii) S possède un maximum et un minimum *i.e.* il existe $\bot \in S$ et $\top \in S$ tels que pour tout $A \in S, \bot \subseteq A$ et $A \subseteq \top$.

V est l'ensemble des objets du système et chaque élément de S est une *classe*.

La condition (*ii*) signifie que l'ensemble est clos par intersection. Pour tout ensemble S de parties de V, on notera $\overline{S}$ la *clôture par intersection* de S *i.e.* le plus petit ensemble tel que $S \subseteq \overline{S}$ et pour tout $A, B \in \overline{S}, A \cap B \in \overline{S}$. Cet ensemble existe car nous travaillons sur des ensembles finis.

Deux classes A et B de S sont dites *compatibles* si $A \subseteq B$ ou $B \subseteq A$. Si A et B ne sont pas compatibles, on note $A \parallel B$. Une *chaîne* de S est un ensemble de classes de S deux à deux compatibles. Inversement, une *antichaîne* de S est un ensemble de classes de S deux à deux incompatibles.

La clôture par intersection des hyperarêtes d'un hypergraphe $\mathcal{H} = (V, E)$ ainsi que l'ajout si besoin de $\emptyset$ et V à E donnent un système de classes. Inversement, tout système

de classes peut être représenté par un hypergraphe. De même, la clôture par intersection des colonnes d'une matrice binaire ainsi que l'ajout si besoin d'une colonne pleine de 1 et d'une colonne pleine de 0 créent un système de classes. Par exemple, l'hypergraphe de la Figure 1.2a (et donc la matrice de la Table 1.1a) donnent le système de classe $\{\emptyset, \{1\}, \{2\}, \{1,3\}, \{2,4\}, \{1,2,3\}, \{1,2,3,4\}\}$: on ajoute un minimum et un maximum ainsi que $\{2\}$ qui est l'intersection de $\{2,4\}$ et $\{1,2,3\}$.

1.1.4 Treillis

Les systèmes de classes sont également équivalents aux treillis finis [7]. Ces objets sont très utilisés en classification et notamment pour l'analyse de concepts formels. Ils permettent une représentation exhaustive des données tout en fournissant un accès rapide aux liens entre objets et attributs. Les treillis sont utiles pour l'analyse locale des données grâce à la conservation de toute l'information.

Nous définissons dans cette partie les treillis et les notions utiles associées.

Définitions générales

Définition 1.1.8. Un *poset* (ensemble partiellement ordonné) est une paire $(A, \leq)$ telle que A est un ensemble non vide et la relation $\leq$ est une relation binaire :

1. réflexive : pour tout $a \in A, a \leq a$,
2. antisymétrique : pour tout $a, b \in A$, $a \leq b$ et $b \leq a$ équivaut à $a = b$,
3. transitive : pour tout $a, b, c \in A$, si $a \leq b$ et $b \leq c$ alors $a \leq c$.

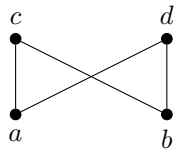
Dans $(A, \leq)$ un poset, on dit que $\leq$ est une *relation d'ordre partiel*. Pour tout $a, b \in A$, si $a \leq b$ ou $b \leq a$, on dit que a et b sont *comparables*. Sinon, on dit qu'ils sont *incomparables* et on note $a \parallel b$.

Dans un poset $(A, \leq)$, on note $\prec$ la relation de *couverture* : pour tout $u, v \in A, u \prec v$ ssi $u < v$ et il n'existe aucun $x \in A$ tel que $u \leq x \leq v$. On dit alors que u *couvre* v (ou que v *est couvert par* u).

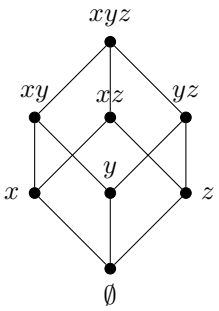
On peut représenter un poset par son *diagramme de Hasse*. Dans un diagramme de Hasse, chaque élément est représenté par un point et une arête ascendante existe entre l'élément a et l'élément b ssi $a \prec b$. Un chemin d'arêtes ascendantes existe donc entre deux éléments a et b ssi $a \leq b$. Nous retrouvons ici la définition du diagramme de Hasse pour un hypergraphe $\mathcal{H} = (V, E)$ qui correspond au diagramme de Hasse du poset $(E, \subseteq)$.

La Figure 1.3a représente le diagramme de Hasse associé au poset $(\{a, b, c, d\}, \leq)$ avec pour seules relations de comparaison les relations $a \leq c, a \leq d, b \leq c, b \leq d$. La Figure 1.3b représente le diagramme de Hasse du poset $(\{\emptyset, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}, \subseteq)$.

Soit $H \subseteq A$ et $a \in A$. a est une *borne supérieure* de H si pour tout $h \in H$, $h \leq a$. Une borne supérieure a de H est une *plus petite borne supérieure* de H ou *supremum* (noté



(a) Exemple de poset qui n'est pas un treillis



(b) Exemple de treillis

FIGURE 1.3 – Exemples de posets

$\sup H$) si pour toute borne supérieure b de H, $a \leq b$. On peut définir symétriquement les notions de *borne inférieure* ($b \in A$ tel que pour tout $h \in H, b \leq h$) et plus grande borne inférieure ou *infimum* notée $\inf H$ ($b \in A$ tel que pour toute borne inférieure a de H, $a \leq b$).

Définition 1.1.9. Un poset $(L, \leq)$ est un *treillis* si pour toute paire d'éléments $a, b \in L$, $\inf\{a, b\} \in L$ et $\sup\{a, b\} \in L$.

Par exemple, le poset représenté sur la Figure 1.3a n'est pas un treillis. En effet, toutes les paires d'éléments incomparables ne possèdent ni inf ni sup : c et d n'ont pas de borne supérieure et n'ont pas de plus grande borne inférieure (a et b sont deux bornes inférieures incomparables de c et d). Symétriquement, a et b n'ont pas de borne inférieure et n'ont pas de plus petite borne supérieure. Au contraire, le poset de la Figure 1.3b est un treillis.

Dans le formalisme des treillis, on note $a \wedge b$ l'infimum de a et b et $a \vee b$ le supremum de a et b.

Un élément a est dit *inf-irréductible* ssi $a = b \wedge c$ implique $a = b$ ou $a = c$. Ceci signifie que l'élément n'est couvert que par un élément. Symétriquement, un élément a est dit *sup-irréductible* ssi $a = b \vee c$ implique que $a = b$ ou $a = c$ *i.e.* l'élément ne couvre qu'un élément. Un élément qui est à la fois sup-irréductible et inf-irréductible est appelé élément *doublement irréductible*.

Si $\mathcal{L} = (L, \leq)$ est un treillis, on appelle *dual* de $\mathcal{L}$ le treillis $(L, \geq)$ défini par la relation d'ordre inverse que nous noterons $\mathcal{L}^{-1}$. Le diagramme de Hasse de $\mathcal{L}^{-1}$ est le diagramme retourné de $\mathcal{L}$.

Équivalence avec les systèmes de classes

Tout treillis fini est équivalent à un système de classes. Ce résultat provient de [7] ou encore de [25] dans le cadre de l'analyse de concepts formels. À partir d'un treillis fini $\mathcal{L} = (L, \leq)$, on peut construire un système de classes S de la manière suivante :

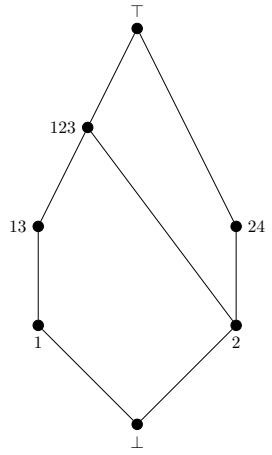


FIGURE 1.4 – Treillis associé à l'hypergraphe de la Figure 1.2a

— $\{x \in L \mid x \text{ sup-irréductible}\}$ est l'ensemble des objets de S,
— $S = \overline{H}$, avec $H = \{\{y \in L \mid y \text{ sup-irréductible}, y \leq x\} \mid x \in L, x \text{ inf-irréductible}\}$.

Cela revient à considérer les éléments sup-irréductibles comme des objets et les éléments inf-irréductibles comme des classes puis à clôturer l'ensemble des classes par intersection.

Les treillis sont ainsi équivalents aux hypergraphes clos par intersection et aux matrices binaires closes par intersection. La Figure 1.4 montre le diagramme de Hasse du treillis associé à l'hypergraphe représenté sur la Figure 1.2a.

1.1.5 Dissimilarités

Si décrire des objets par un ensemble d'attributs qu'ils possèdent ou non peut sembler la manière la plus intuitive de faire, des données peuvent également être décrites par une dissimilarité, c'est-à-dire une mesure de différence entre les objets. Nous donnons dans cette partie les définitions relatives aux dissimilarités nécessaires à ce travail et détaillons certains liens avec des objets déjà évoqués. Les dissimilarités en classification permettent une description locale des données utilisées pour la construction d'un modèle global. Elles représentent l'intensité du lien qui existe entre deux individus. Les dissimilarités permettent donc à partir de données décrites localement d'étudier l'organisation globale des objets en classes.

Définition 1.1.10. Une *dissimilarité* sur un ensemble V est une application d de $V \times V$ dans $\mathbb{R}_+$ telle que :
— $\forall x \in V, d(x, x) = 0$,
— $\forall x, y \in V, d(x, y) = d(y, x)$.

Nous utiliserons uniquement des dissimilarités *propres* dans cette thèse *i.e.* des dissimilarités telles que pour tout x, y, $d(x, y) = 0$ ssi $x = y$.

x	0			
y	3	0		
z	1	1	0	
t	2	2	2	0
	x	y	z	t

TABLE 1.2 – Exemple de dissimilarité

FIGURE 1.5 – Graphes seuils associés à la dissimilarité donnée dans la Table 1.2

On peut associer à une dissimilarité d un ensemble de sous-ensembles de V dit ensemble de classes. Pour cela, on utilise les *graphes seuil* de d. Le *graphe seuil* G_σ de d est le graphe $G_\sigma = (V_G, E_G)$ avec :

— $V_G = V$,

— $\forall x, y \in V, (x, y) \in E_G \iff d(x, y) \leq \sigma$.

On définit pour tout sous-ensemble S de V le diamètre de S : $diam(S) = \max_{x,y \in S} d(x, y)$.

Définition 1.1.11. On appelle *classes* d'une dissimilarité d l'ensemble des cliques maximales de tous les graphes seuils de d.

Définition 1.1.12. Étant donné une dissimilarité $d : V \times V \to \mathbb{R}_+$, pour tout $x \in V$ et pour tout $\sigma \geq 0$, on appelle *boule de rayon* σ *et de centre* x l'ensemble $B(x, \sigma) = \{y \in V \mid d(x, y) \leq \sigma\}$.

On peut donc associer à une dissimilarité d deux hypergraphes :

— l'hypergraphe des classes : $\mathcal{C}(d) = (V, E)$ avec E l'ensemble des classes de d,

— l'hypergraphe des boules : $\mathcal{B}(d) = (V, E)$ avec E l'ensemble des boules de d.

rayon \ centre	x	y	z	t
0	x	y	z	t
1	xz	yz	xyz	t
2	xzt	yzt	xyzt	xyzt
3	xyzt	xyzt	xyzt	xyzt

TABLE 1.3 – Boules de la dissimilarité décrite par la Table 1.2

La Table 1.2 représente une dissimilarité d sur x, y, z, t. Les boules de cette dissimilarité sont données dans la Table 1.3. L'hypergraphe des boules de d est donc $(\{x, y, z, t\}, \{\{x\}, \{y\}, \{z\}, \{t\}, \{x, z\}, \{y, z\}, \{x, y, z\}, \{x, z, t\}, \{y, z, t\}, \{x, y, z, t\}\})$.

La Figure 1.5 représente les graphes seuil associés à d. L'hypergraphe des classes de d est donc $(\{x, y, z, t\}, \{\{x\}, \{y\}, \{z\}, \{t\}, \{x, z\}, \{y, z\}, \{x, z, t\}, \{y, z, t\}, \{x, y, z, t\}\})$.

1.2 Apprentissage automatique

1.2.1 Introduction

Dans cette partie nous introduisons les notions provenant de l'apprentissage automatique qui seront utilisées dans les chapitres suivants. L'apprentissage automatique vise, à partir de techniques statistiques, à permettre à un ordinateur "d'apprendre" c'est-à-dire de s'améliorer sur une certaine tâche en utilisant une base de données afin de produire des prédictions sur de nouvelles données.

L'apprentissage automatique peut être divisé en deux grandes catégories :

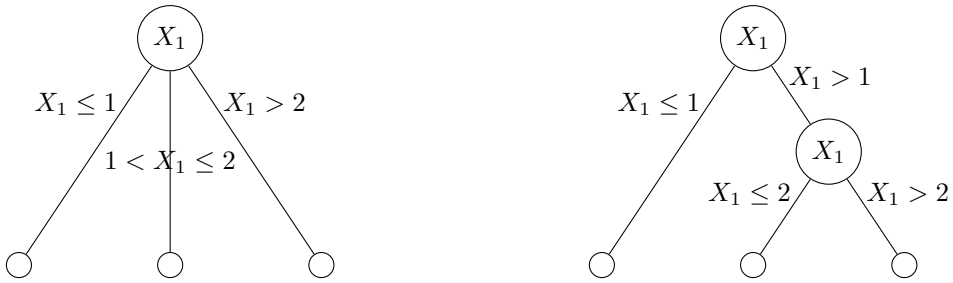
— l'*apprentissage supervisé* dans lequel l'algorithme dispose de données étiquetées (en classification par exemple, chaque donnée est associée à sa classe) et apprend une fonction de l'espace de départ dans l'espace des étiquettes,

— l'*apprentissage non supervisé* dans lequel l'algorithme dispose uniquement de données et doit extraire la structure de ces données.

Dans le cadre de l'apprentissage supervisé, l'ordre est donné, contrairement à l'analyse de données qui cherche à trouver un ordre dans les données. Chaque exemple est assigné à une classe. Le modèle cherche alors à retrouver pour chaque exemple la classe qui lui est assignée et à généraliser à de nouveaux individus.

Nous présenterons dans cette partie des algorithmes et méthodes de l'apprentissage supervisé. L'objectif des modèles d'apprentissage supervisé est d'être généralisable et utilisable pour la prédiction. Les modèles n'ont pas vocation à être interprétables ou à apporter de l'information sur l'ensemble de données fourni.

Nous nous intéresserons plus précisément dans cette thèse à des tâches de classification : à partir d'un ensemble de données connues et réparties dans deux classes ou plus,



(a) Nœud non binaire (b) Arbre binaire équivalent

FIGURE 1.6 – Binarisation d'un nœud non binaire d'un arbre de décision

l'algorithme d'apprentissage produit une fonction permettant d'assigner une classe à de nouvelles données.

L'évaluation des performances de tels modèles est souvent réalisée à partir d'une base connue en utilisant une partie des données pour l'apprentissage et la partie restante pour mesurer les performances du modèle sur des données qui n'ont jamais été utilisées par l'algorithme.

Nous présenterons dans la Section 1.2.2 les arbres de décision, une méthode classique d'apprentissage supervisé ainsi que son extension en forêts d'arbres de décision (Section 1.2.3). Enfin, nous détaillerons dans la Section 1.2.4 la méthode utilisée dans ce travail pour évaluer les performances des modèles produits.

1.2.2 Arbres de décision

Les arbres de décision, introduits par Breiman [14], sont une méthode d'apprentissage supervisé très utilisée, car ils sont efficaces pour la prédiction dans de nombreuses situations et sont faciles à comprendre. Un arbre de décision est un modèle construit à partir de données d'apprentissage dans lequel chaque nœud situé à l'intérieur de l'arbre représente une division des données en plusieurs sous-ensembles distincts. Ces sous-ensembles correspondent à une division selon la valeur d'un attribut donné en chaque nœud. Chaque nœud peut être divisé en autant de sous-ensembles que voulu, mais en pratique, les divisions binaires sont les plus utilisées dans un souci d'efficacité et de simplicité. De plus, chaque division en k sous-ensembles peut être transformée en divisions successives en deux sous-ensembles. Un exemple de transformation d'un nœud à trois enfants en un nœud binaire est représenté sur la Figure 1.6. Chaque feuille de l'arbre est quant à elle associée à une classe de sortie. Un nœud de l'arbre est dit *pur* si tous ses exemples sont de la même classe. Les feuilles d'un arbre de décision sont pures.

Pour prédire la classe d'un point, on parcourt une branche de l'arbre en passant d'un nœud à l'autre selon la valeur de notre donnée à prédire jusqu'à arriver dans une feuille qui donne la classe prédite. La Figure 1.7 montre un exemple d'arbre de décision pour des

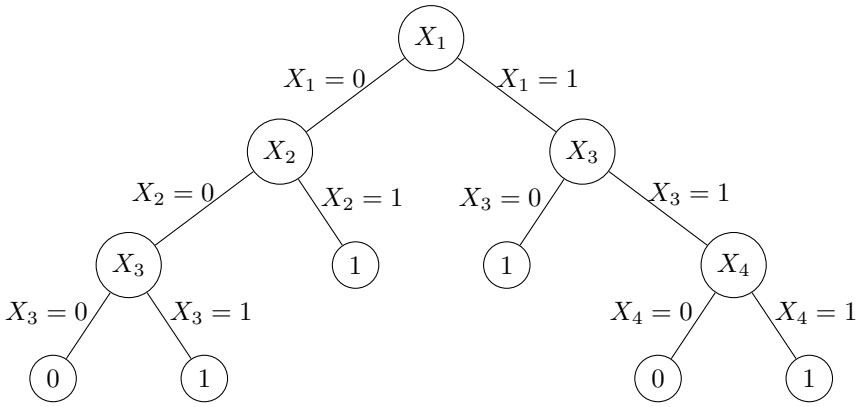


FIGURE 1.7 – Exemple d'arbre de décision

données représentées sur une base de 4 attributs binaires A_1, A_2, A_3, A_4. Sur cet exemple, la nouvelle donnée $x = (1, 0, 1, 0)$ serait classée dans la classe 0. Ces arbres peuvent également être utilisés sur des attributs continus en réalisant des divisions du type $A_i \leq \alpha$, $A_i > \alpha$.

Algorithme

L'algorithme de création d'un arbre de décision à partir de données représentées sur la base $A_1, \ldots, A_d$ est présenté dans l'Algorithme 1. Il consiste à diviser récursivement chaque nœud de manière à maximiser une certaine quantité appelée le *gain en impureté*.

Critères d'impureté

Les deux critères les plus utilisés dans la construction d'arbres de décision pour diviser les nœuds sont l'*impureté de Gini* et le gain en *entropie*.

Supposons que les données sont réparties dans J classes de sortie appelées $1, 2, \ldots, J$. L'*impureté de Gini* d'un sous-ensemble de points P est définie par :

$$I_G(P) = \sum_{i=1}^{J} f_i \sum_{k \neq i} f_k$$

avec f_i la fréquence d'apparition de la classe i dans P. Cela revient à calculer la probabilité de se tromper en attribuant une classe de sortie à un exemple au hasard en suivant la distribution des classes dans l'ensemble. La valeur de ce critère d'impureté est donc de 0 quand l'ensemble est composé d'éléments d'une unique classe. Ce critère peut être utilisé pour diviser les nœuds d'un arbre de décision en minimisant la moyenne des impuretés de Gini des fils pondérées par la taille des ensembles.

L'*entropie* est définie pour un ensemble P par :

$$H(P) = -\sum_{i=1}^{J} f_i \log_2 f_i$$

Algorithme 1 : Construction d'un arbre de décision

```
1  Créer un nœud racine contenant tous les exemples
2  Créer une file F contenant la racine
3  tant que la file F est non vide faire
4      Dépiler un nœud de F
5      si tous les exemples que le nœud contient ne sont pas de la même classe alors
6          pour chaque attribut A_i dans A_1, ..., A_d faire
7              pour chaque division possible s sur cet attribut faire
8                  Calculer le gain en impureté obtenu pour la division s sur l'attribut
                   A_i
9              fin
10         fin
11         Choisir A_best l'attribut fournissant le meilleur gain et s_best la division
           associée
12         Créer un nœud enfant du nœud courant pour chaque partie de s_best
13         Empiler les nœuds enfants dans F
14     fin
15 fin
```

avec f_i la fréquence d'apparition de chaque classe dans P. L'entropie peut être utilisée pour diviser un nœud en deux en maximisant le gain en entropie c'est-à-dire en maximisant la différence entre l'entropie du nœud et la somme pondérée des entropies des fils créés par la séparation.

Les deux critères visent à créer des nœuds enfants les plus purs possibles comparés à leurs parents.

Complexité

Lors de la construction d'un arbre de décision, la division de chaque nœud se fait en calculant pour chaque division possible et pour chaque attribut une valeur d'impureté. Le nombre de divisions possibles pour un nœud contenant n exemples pour chaque attribut est de n. En effet, supposons qu'un exemple x prend la valeur a pour l'attribut A et un exemple y la valeur b pour ce même attribut. Supposons de plus que aucun exemple ne prend une valeur entre a et b pour l'attribut A. Dans ce cas, si A est continu, l'infinité de divisions possibles entre a et b ($A \leq \frac{a+b}{2}, A \leq \frac{a+b}{42}$ par exemple) donnent toutes la même partition et donc la même valeur d'impureté. La méthode optimale consiste à ordonner les points par valeur croissante selon chaque attribut, ce qui devient alors l'opération dominante. L'algorithme permet donc de construire un arbre de décision en $\mathcal{O}(nd \log n)$ avec n le nombre d'exemples et d le nombre d'attributs.

Pour l'utilisation d'un arbre de décision déjà construit, la propagation d'un exemple x

dans un nœud est une opération constante (simple comparaison de la valeur d'un attribut de x avec une valeur fixée). La complexité est donc bornée par la profondeur de l'arbre et est donc en $\mathcal{O}(\log n)$.

Conclusion

S'agissant de la construction des arbres de décision, nous pouvons remarquer que :
— les deux critères présentés sont basés sur l'optimisation d'un paramètre lié à la classe de sortie des données observées,
— la construction se fait par un processus *top-down* c'est-à-dire en partant de l'ensemble de données entier et en le divisant petit à petit.

1.2.3 Forêts d'arbres de décision

Sur des données réelles, la différence de gain (en impureté de Gini ou en entropie) entre des divisions selon des attributs différents peut être très faible. Or, le choix des attributs influence grandement le modèle et donc ses résultats. Les méthodes d'apprentissage par construction d'un arbre de décision sont donc instables et présentent une grande variance. Pour remédier à cela, on construit souvent un ensemble d'arbres avec une perturbation aléatoire. Pour prédire la classe d'un nouvel exemple, on fait voter les arbres : on utilise chaque arbre individuellement pour prédire une classe puis on lui attribue la classe majoritaire. Parmi les méthodes les plus utilisées, on trouve le *bagging* [11], les *random subspaces* [28], les *random patches* [34], les *random forests* [13] ou encore les *extra-trees* [26].

Le *bagging* consiste à construire t arbres chacun sur un sous-ensemble (avec remise) des exemples, le *random subspace* consiste quant à lui à construire t arbres sur un sous-ensemble (avec remise) des attributs tandis que les *random patches* font les deux à la fois.

En pratique les *random forests* sont les plus utilisées. La méthode consiste à construire t arbres sur un sous-ensemble d'attributs de taille p avec un paramètre $0 < K \leq 1$ en modifiant légèrement la méthode de division de chaque nœud :

1. Tirer au hasard $\lceil K \times p \rceil$ attributs
2. Pour chaque attribut X_i choisi et chaque division possible sur cet attribut, calculer le gain obtenu (selon un critère d'impureté préalablement choisi) en divisant selon l'attribut.
3. Choisir X_{best} l'attribut fournissant le meilleur gain et la division associée.
4. Créer un nœud divisant les exemples selon la division choisie sur X_{best}.

La variable K représente donc la fraction d'attributs de l'ensemble des attributs qui sera utilisée pour la division du nœud. On peut noter que si ce paramètre est fixé à 1, l'arbre est construit de manière déterministe et on retrouve l'algorithme de construction d'un

arbre de décision classique. Si K est très petit par contre, l'arbre est construit de manière totalement aléatoire. Enfin, la méthode des *extra-trees* étend les random forests classiques en choisissant au hasard une seule division par attribut lors de l'étape 2. Ces deux dernières méthodes étant les plus efficaces en pratique, nous les utiliserons pour évaluer les résultats de nos modèles.

1.2.4 Évaluation des modèles de classification

Validation croisée

La *validation croisée* est un ensemble de méthodes d'évaluation des modèles de classification. Afin de déterminer si un modèle de classification est efficace ou non, on évalue sa capacité de généralisation à des données inconnues (sur lesquelles il n'a pas été entraîné). Pour cela, on sépare l'ensemble de données $\mathcal{L} = \{(x_i, y_i)\}_{1 \leq i \leq n}$ avec pour tout $i, x_i \in X$ l'espace des données et $y_i \in Y$ l'ensemble des classes de sortie, en deux ensembles complémentaires :

— un ensemble de données, appelé *ensemble d'apprentissage*, sur lequel le modèle va être construit,

— un ensemble de données, appelé *ensemble de test*, dont le classifieur devra prédire la classe de sortie.

Soit $f : X \to Y$ un classifieur. On appelle *score* de f sur $\mathcal{S} \subseteq \mathcal{L}$:

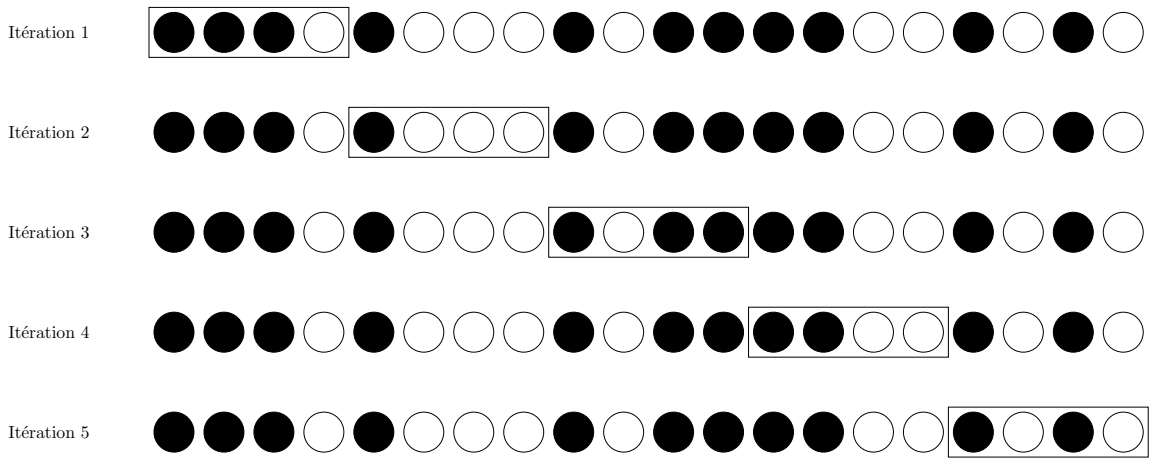
$$s_f(\mathcal{S}) = \frac{|\{(x_i, y_i) \in \mathcal{S} \mid f(x_i) = y_i\}|}{|\mathcal{S}|}$$

qui représente la fraction d'exemples bien classés par le classifieur. Un score de 1 par exemple indique que le classifieur n'a commis aucune erreur sur l'ensemble considéré. On appelle *score d'apprentissage* le score obtenu par le classifieur sur l'ensemble d'apprentissage et *score de test* le score obtenu sur l'ensemble de test.

Afin de réduire la variance de la méthode d'évaluation, la plupart des méthodes de validation croisée répètent le processus un certain nombre de fois en changeant le partitionnement ensemble d'apprentissage/ensemble de test.

K-Fold

Nous nous intéressons ici plus spécifiquement à la méthode des *k-fold* que nous utiliserons pour évaluer nos modèles de classification. Cette méthode consiste à séparer en k échantillons disjoints de tailles égales l'ensemble de données puis à utiliser un des échantillons comme ensemble de test et les $k - 1$ restants comme ensemble d'apprentissage et ce, pour les k séparations possibles. On considère ensuite la moyenne des scores obtenus sur les k ensembles de tests. Par exemple, une 3-fold sur un ensemble $\mathcal{L} = \{(x_i, y_i)\}_{1 \leq i \leq n}$ de données consiste à partitionner au hasard $\mathcal{L}$ en trois ensembles $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ de tailles égales puis à :



Les exemples sont représentés par des disques de couleurs différentes selon leur classe de sortie. Les rectangles indiquent les exemples utilisés pour le test à chaque itération. Le reste sera utilisé pour l'apprentissage.

FIGURE 1.8 – Schéma de la validation croisée k-fold pour $k = 5$.

— entraîner un classifieur sur $\mathcal{L}_1 \cup \mathcal{L}_2$ et le tester sur $\mathcal{L}_3$,
— entraîner un classifieur sur $\mathcal{L}_2 \cup \mathcal{L}_3$ et le tester sur $\mathcal{L}_1$,
— entraîner un classifieur sur $\mathcal{L}_1 \cup \mathcal{L}_3$ et le tester sur $\mathcal{L}_2$,
— calculer la moyenne des trois scores de tests.

La Figure 1.8 illustre le processus pour $k = 5$. Les tests effectués sur les modèles de classification testés présenteront la moyenne de 10 répétitions de validation croisée 10-fold.

Jeux de données

Les modèles de classification proposés seront testés sur des jeux de données classiques de l'apprentissage automatique. Ceux-ci présentent des tailles et dimensions variées récapitulées dans la Table 1.4. Toutes ces données peuvent être trouvées sur le site de l'UCI Machine Learning Repository [2].

Les domaines dont sont issues ces données sont divers :

— *breast-cancer* représente les caractéristiques de noyaux cellulaires (rayon, texture, périmètre, aire, compacité, concavité...) présents dans des prélèvements faits sur des tumeurs présentes dans des seins. La classe de sortie indique si la tumeur est bénigne ou maligne.
— *digits* est une base d'images de chiffres écrits à la main.
— *iris* [22] représente trois types d'iris différents selon la longueur et largeur de leurs sépales et pétales.
— *ringnorm* est un jeu de données artificiel créé par Breiman [12] sur lequel les arbres de décision sont peu performants.

	n	d	c
breast-cancer	683	9	2
digits	1797	64	10
iris	150	4	3
ringnorm	7400	20	2
satellite	6435	36	7
spambase	6301	57	2
vehicle	846	18	4
waveform	5000	21	3
zoo	101	16	7

n représente le nombre d'exemples, d la dimension de l'espace dans lequel les objets sont représentés et c le nombre de classes

TABLE 1.4 – Description des jeux de données utilisés pour tester les modèles de classification

— *satellite* contient des images multispectrales de la surface de la Terre étiquetées par la présence de champs de coton, de végétation...
— *spambase* représente des e-mails sur une base particulière utilisant la fréquence d'apparition de certains mots et lettres. Le but de classification est de déterminer si l'e-mail est un spam ou non.
— *vehicle* est composé de silhouettes de véhicules présentés sous différents angles, représentées sur une base spécifique. Il s'agit ici de déterminer le type de véhicule (Opel, Saab, bus, van).
— *waveform* est une base de données artificielle [14] générée en combinant deux parmi trois objets de base et en ajoutant du bruit aux attributs.
— *zoo* représente des animaux dans une base de 16 attributs tels que le nombre de pattes, la présence de plumes, de fourrure, etc.

1.3 Analyse de données et modèles totalement équilibrés

1.3.1 Introduction

Nous nous attardons à présent sur l'analyse de données, différente de l'apprentissage supervisé dans ses objectifs. Là où l'apprentissage automatique supervisé cherche à utiliser la globalité des données afin de prédire l'appartenance à une classe de nouvelles données, l'analyse de données cherche à expliquer les données, à en extraire l'information. Pour cela, l'un des principaux courants de l'analyse de données consiste à projeter les données sur un modèle afin de les expliquer.

Le modèle le plus simple d'analyse de données est la partition. Réaliser une *partition* d'un ensemble S consiste à séparer S en n sous-ensembles disjoints de S, $S_1, \ldots, S_n$ tels que $S_1 \cup S_2 \cup \cdots \cup S_n = S$. Ce modèle est un modèle de discrimination : il sépare des ensembles de données différents.

La Figure 1.9a représente un exemple de partition de cinq éléments. Les Figures 1.9b et 1.9c ne sont par contre pas des partitions. En effet, les ensembles formés ont des intersections non vides.

D'autres systèmes de classes particuliers peuvent être étudiés dont l'un des plus classiques : les *hiérarchies*.

Définition 1.3.1. On appelle *hiérarchie* un système de classes S tel que :

$$\forall A, B \in S, A \cap B \in \{A, B, \emptyset\}.$$

Les partitions sont des cas particuliers de hiérarchies. La Figure 1.9a représente donc une hiérarchie. La Figure 1.9b représente une autre hiérarchie qui n'est cette fois pas une partition.

Cette définition implique que, pour une hiérarchie, deux classes sont soit d'intersection vide, soit compatibles. Il n'existe donc pas d'empiétance des classes au sein de ces systèmes. Ils permettent de créer un modèle représentant la spécialisation des classes. Plus on descend dans la hiérarchie et plus les classes se spécialisent. Le binôme linnéen [43] (aussi connu sous le nom de binomial) en taxonomie est un exemple de hiérarchie. Il permet de classer les espèces selon leur genre puis d'y adjoindre un épithète spécifique permettant de désigner l'espèce au sein du genre voire d'ajouter un sous-genre, une section, sous-section ou encore une série. Par exemple, le nom binomial du chat domestique est *Felis silvestris catus* signifiant qu'il appartient au genre *felis* (les félins), à l'espère *felis silvestris* (chat sauvage) et plus spécifiquement à la sous-espèce des chats domestiques. Les arbres de décision vus dans la section précédente créent également une hiérarchie.

Une condition plus faible permet de définir des systèmes de classes particuliers autorisant l'empiétance, par exemple les *hiérarchies faibles*.

Définition 1.3.2. Une *hiérarchie faible* S est un système de classes tel que :

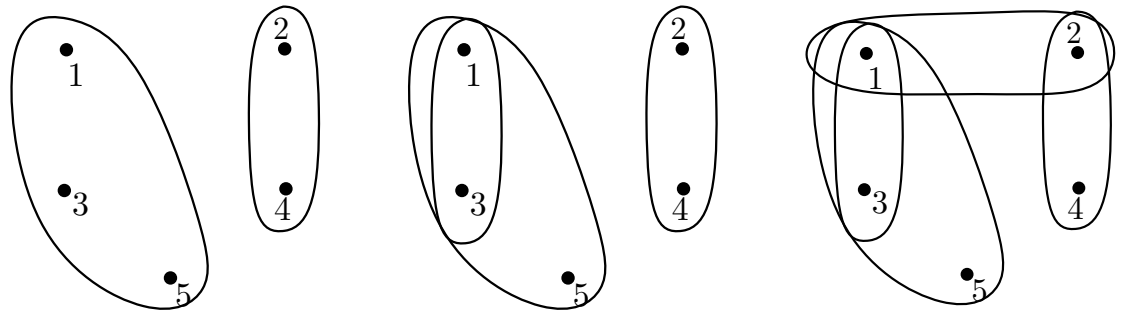
$$\forall A, B, C \in S, A \cap B \cap C \in \{A \cap B, A \cap C, B \cap C\}.$$

Dans de tels systèmes, l'intersection de trois classes est toujours l'intersection de deux de ces classes. Ces ensembles ont été introduits par Bandelt et Dress [6]. La Figure 1.9c représente un exemple de hiérarchie faible qui n'est ni une partition ni une hiérarchie. Les Figures 1.9a et 1.9b sont aussi des hiérarchies faibles.

1.3.2 K-Means

Problème

Les *K-Means* [33] (et leur version plus générale des nuées dynamiques [19]) sont une méthode classique de partitionnement de données que nous utiliserons dans le Chapitre



(a) Partition (b) Hiérarchie (c) Hiérarchie faible

FIGURE 1.9 – Exemples des modèles de classification présentés

2. Étant donné n points, on cherche à les répartir en k groupes appelés *classes* de façon à minimiser la somme des carrés de la distance de chaque point au centre de la classe à laquelle il appartient. Chaque classe est représentée par son centre et chaque point de l'ensemble de données est associé à la classe dont il est le plus proche du centre. Ce partitionnement revient à effectuer un partitionnement de Voronoi sur les centres des classes.

Plus formellement, à partir d'un ensemble de données X, on cherche à déterminer $S = \{S_1, S_2, \ldots, S_k\}$ tel que pour tout i, j, $S_i \cap S_j = \emptyset$, $S_i \subseteq X$ et $\bigcup_{i \leq n} S_i = X$, solution au problème d'optimisation :

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

avec μ_i le centre de S_i.

Trouver la solution optimale au problème de partitionnement K-Means dans un espace euclidien est NP-difficile même pour $k = 2$. Dans ce contexte, des heuristiques ont été développées afin de converger rapidement vers un optimum local qui n'est pas forcément l'optimum global.

Algorithme

L'Algorithme 2 détaille l'algorithme classique de Lloyd-Forgy ([33], [24]) pour la résolution de ce problème qui sera celui utilisé dans les travaux présentés dans ce manuscrit. À chaque étape jusqu'à convergence, on crée chaque classe en associant tous les points à la classe dont le centre est le plus proche (ligne 3) puis on met à jour les centres des classes en fonction des points qui y sont associés (ligne 4). Si un point est à égale distance de deux centres de classe, on l'associe à un seul au hasard. Cet algorithme peut converger vers un minimum local arbitrairement éloigné de l'optimum global.

Les centres de classes initiaux peuvent être choisis au hasard, mais il existe aussi des heuristiques permettant d'accélérer la convergence et d'améliorer la qualité de la solution

Algorithme 2 : Algorithme de Lloyd (K-Means)

Données : un ensemble de points X, k points initiaux $\mu_1^{(1)}, \mu_2^{(1)}, \ldots, \mu_k^{(1)}$
Résultat : un partitionnement S de X

```
t = 1
tant que des μ_i changent faire
    S_i^(t) = {x_p ∈ X | ∀j, 1 ≤ j ≤ k, ‖x_p − μ_i^(t)‖² ≤ ‖x_p − μ_j^(t)‖²}
    μ_i^(t+1) = (1/|S_i^(t)|) Σ_{x_j ∈ S_i^(t)} x_j
    t = t + 1
fin
retourner S = {S_i^(t−1) | 1 ≤ i ≤ k}
```

renvoyée comparée à la solution optimale. On peut citer par exemple la méthode des *K-Means++* qui sera celle utilisée chaque fois qu'on utilisera un K-Means, qui consiste à choisir un centre au hasard parmi les points de l'ensemble puis à répéter les deux étapes suivantes jusqu'à avoir k centres :

1. pour chaque point $x \in X$, calculer $D(x)$ la distance de x au centre de classe le plus proche ;
2. choisir un point au hasard comme nouveau centre en suivant une distribution de probabilité proportionnelle à $D(x)^2$.

Cet algorithme garantit une approximation de ratio $\mathcal{O}(\log k)$ [5].

Complexité

La complexité de cet algorithme est $\mathcal{O}(ndki)$ avec k le nombre de classes, d la dimension de l'espace, n le nombre de points et i le nombre d'itérations de l'algorithme. En pratique, quand les données présentent une structure favorable au partitionnement, le nombre d'itérations nécessaires avant convergence de la méthode est petit. De plus, nous utiliserons les K-Means avec $k = 2$ dans le Chapitre 2 et nous considérerons donc que la complexité de l'algorithme est $\mathcal{O}(nd)$.

1.3.3 Modèles totalement équilibrés

Les modèles présentés auparavant tels que les partitions ou les hiérarchies permettent de nombreuses applications en analyse de données mais peuvent parfois être trop restrictifs en ne permettant pas l'empiétance des classes. Selon les domaines d'application, divers modèles peuvent être utilisés, permettant de correspondre au mieux aux données étudiées. En phylogénie par exemple, les modèles d'arbres sont très présents, respectant l'idée que l'évolution des espèces suivrait une structure arborée. Les espèces sont ainsi représentées par les nœuds d'un arbre dont les autres nœuds sont des ancêtres communs permettant

d'expliquer l'évolution. Nous étudions particulièrement dans cette thèse les modèles dits *totalement équilibrés* qui sont une généralisation des arbres : ils sont sans cycles, peuvent être caractérisés par une séquence d'arbres [32] ou comme étant les hypergraphes qui admettent un arbre support récursivement.

Définition 1.3.3. Soit S un système de classes. On appelle *cycle* de S une séquence $(X_0, X'_0, X_1, \ldots, X_{k-1}, X'_{k-1}), k \geq 3$ avec pour tout $0 \leq i \leq k-1$:
— $X_i \in S, X'_i \in S$,
— $X_i \subsetneq X'_{i-1 \pmod k}, X_i \subsetneq X'_i$,
— $\forall j \neq i-1 \pmod k, j \neq i, X_i \parallel X'_j$.

Intuitivement, chaque X_i est inclus dans exactement deux X'_j de la séquence, c'est donc l'ensemble $\{X'_j\}_{0 \leq j \leq n}$ qui forme le cycle. On utilise le modulo dans la définition pour avoir $X'_{i-1 \pmod k} = X'_{k-1}$ si $i = 0$.

Définition 1.3.4. Soit S un système de classes. S est dit *totalement équilibré* si S ne possède aucun cycle.

Notons qu'un système de classes totalement équilibré est une hiérarchie faible. En effet les hiérarchies faibles sont des systèmes de classes sans 3-cycles.

Ces systèmes sont équivalents aux graphes fortement cordés ([15] pour l'équivalence entre les treillis sans couronne et les graphes fortement cordés, [21] pour l'équivalence entre les matrices totalement équilibrées et les graphes fortement cordés).

Hypergraphes totalement équilibrés

Comme remarqué précédemment, les hypergraphes clos par intersection sont équivalents aux systèmes de classes et nous pouvons donc nous intéresser aux hypergraphes *totalement équilibrés* de manière équivalente. Le lien entre les systèmes totalement équilibrés et les arbres transparaît particulièrement dans le formalisme des hypergraphes : les hypergraphes totalement équilibrés sont les hypergraphes qui ne contiennent pas un certain type de cycles (les cycles spéciaux), tout comme les arbres sont les graphes ne contenant pas de cycles. Nous n'utiliserons dans ce manuscrit aucun autre type de cycles que les cycles spéciaux et les appellerons donc simplement *cycles*.

Définition 1.3.5. Soit $\mathcal{H} = (V, E)$. Un *cycle* de $\mathcal{H}$ est une séquence $(v_0, E_0, v_1, \ldots, v_{k-1}, E_{k-1}), k \geq 3$ avec pour tout $0 \leq i \leq k-1$:
— $v_i \in V, E_i \in E$,
— $v_i \in E_{i-1 \pmod k} \cap E_i$,
— $\forall j \neq i-1 \pmod k, j \neq i, v_i \notin E_j$.

La Figure 1.10 montre un hypergraphe contenant un cycle de taille 4.

Définition 1.3.6. Un hypergraphe $\mathcal{H} = (V, E)$ est *totalement équilibré* si il ne possède pas de cycle.

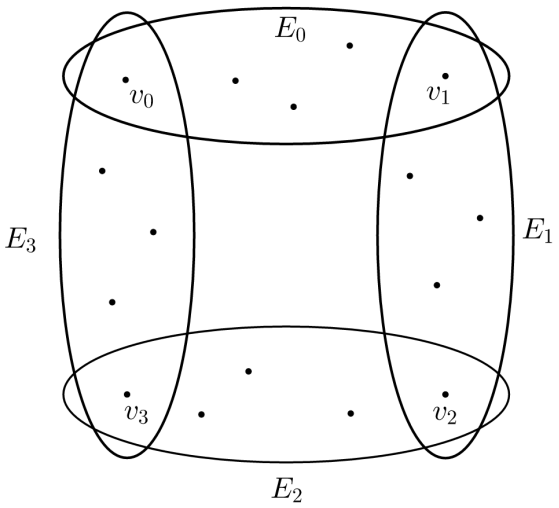


FIGURE 1.10 – 4-cycle

La définition de cycle donnée peut être généralisée à une séquence $(E_0, E'_0, E_1, \ldots, E_{k-1}, E'_{k-1}), k \geq 3$ avec pour tout $0 \leq i \leq k-1$:

— $E_i \in E, E'_i \in E$,
— $E_i \subsetneq E'_{i-1 \pmod n} \cap E'_i$,
— $\forall j \neq i-1, j \neq i, E_i \parallel E'_j$.

En effet, l'existence d'une telle séquence entraîne l'existence d'un cycle tel que défini auparavant. Les deux définitions sont de plus équivalentes dans le cas des hypergraphes clos par intersection (le cycle $(v_0, E_0, v_1, \ldots, v_{k-1}, E_{k-1})$ entraîne l'existence du cycle $(E_0, E_0 \cap E_1, E_1, E_1 \cap E_2, \ldots, E_{k-1}, E_{k-1} \cap E_0)$.

Par extension de la définition de clôture par intersection donnée pour les ensembles, pour tout hypergraphe $\mathcal{H} = (V, E)$, sa clôture par intersection est définie par $\overline{\mathcal{H}} = (V, \overline{E})$. La propriété suivante permet de plus de s'affranchir de l'hypothèse de clôture par intersection :

Propriété 1.3.1. *Soit $\mathcal{H} = (V, E)$ un hypergraphe. $\mathcal{H}$ est totalement équilibré ssi $\overline{\mathcal{H}}$ est totalement équilibré.*

Démonstration. Soit $\mathcal{H}$ un hypergraphe contenant un cycle $(v_0, E_0, v_1, \ldots, v_{k-1}, E_{k-1})$. Pour tout $i \leq k-1$, $E_i \in \overline{E}$ donc $(v_0, E_0, v_1, \ldots, v_{k-1}, E_{k-1})$ est un cycle de $\overline{\mathcal{H}}$. Donc $\overline{\mathcal{H}}$ totalement équilibré implique $\mathcal{H}$ totalement équilibré.

Réciproquement soit $\mathcal{H} = (V, E)$ un hypergraphe totalement équilibré et $\overline{\mathcal{H}} = (V, \overline{E})$ sa clôture par intersection. Supposons que $\overline{\mathcal{H}}$ n'est pas totalement équilibré. Par définition, il existe un cycle $(v_0, E'_0, v_1, \ldots, v_{k-1}, E'_{k-1})$ avec pour tout $i \leq k, E'_i \in \overline{E}$. Pour tout $i \leq k$, nous avons deux cas possibles, soit $E'_i \in E$ soit il existe $E^i_1, E^i_2 \in E, E'_i = E^i_1 \cap E^i_2$. Dans ce cas, pour tout $j \neq i$, si $E^i_1 = E^j_1$ alors $E^i_2 \neq E^j_2$. Il existe donc un cycle dans $\mathcal{H}$, ce qui est une contradiction. □

Les liens des hypergraphes totalement équilibrés avec les arbres sont nombreux. On peut par exemple remarquer qu'un hypergraphe totalement équilibré est un hyperarbre. Lehel fournit également [32] une caractérisation des hypergraphes totalement équilibrés par les hyperarbres tirée des résultats de Duchet [20], Flament [23].

Théorème 1.3.2 (Lehel [32]). *Un hypergraphe $\mathcal{H}$ est totalement équilibré ssi tout sous-hypergraphe induit de $\mathcal{H}$ est un hyperarbre.*

L'hypergraphe représenté sur la Figure 1.1 est un hyperarbre, mais n'est pas totalement équilibré. En effet, l'hypergraphe induit par les trois sommets A, B et C $(\{A, B, C\}, \{\{A, B\}, \{A, C\}, \{B, C\}\})$ n'est pas un hyperarbre. Ces trois sommets et hyperarêtes forment un cycle.

On peut également caractériser ces hypergraphes par l'existence d'un ordre particulier sur les sommets.

Définition 1.3.7. Étant donné un hypergraphe $\mathcal{H} = (V, E)$. Un ordre $v_1 < v_2 < \cdots < v_n$ sur les éléments de V est dit *totalement équilibré* si les ensembles $\{X_{|v_i,\ldots,v_n} \mid v_i \in X, X \in E\}$ sont des chaînes pour tout $1 \leq i \leq n$.

Théorème 1.3.3 (Brucker, Gely [15]). *Un hypergraphe est totalement équilibré ssi il admet un ordre totalement équilibré.*

Caractérisation des hypergraphes totalement équilibrés par une séquence d'arbres

Les liens entre les hypergraphes totalement équilibrés et les arbres sont développés plus avant dans [32] grâce à une caractérisation des hypergraphes totalement équilibrés par une séquence d'arbres.

Théorème 1.3.4 (Lehel [32]). *Soit $\mathcal{H} = (V, E)$ un hypergraphe. $\mathcal{H}$ est totalement équilibré ssi il existe une séquence d'arbres $T_0, \ldots, T_{n-1}$ où pour tout $i, T_i = (V_i, E_i)$ tel que pour tout $1 \leq i \leq n-1$:*

1. *les sommets de T_i sont des ensembles à $i+1$ éléments de V,*
2. $V_0 = V$,
   $V_i = \{x \cup y \mid x, y \in V_{i-1}, (x, y) \in E_{i-1}\}$,
3. $(u, v) \in E_i \implies \exists x, y, z \in V_{i-1}, u = x \cup y, v = y \cup z$,
4. $\forall e \in E, \exists i \leq n, \exists x \in V_i, x = e$.

On peut tirer de ce résultat un algorithme permettant de générer tous les hypergraphes totalement équilibrés. Cet algorithme est détaillé dans l'Algorithme 3 qui crée une séquence d'arbres respectant les contraintes énoncées dans le Théorème 1.3.4. À partir de cette construction, $\mathcal{H} = (V, E)$, avec E n'importe quel sous-ensemble de l'ensemble des sommets de ces arbres est un hypergraphe totalement équilibré.

L'algorithme utilise la notion d'*arbre couvrant*.

Algorithme 3 : Construction d'une séquence d'arbres générant des hypergraphes totalement équilibrés

Données : Ensemble de sommets V

Résultat : Une séquence d'arbres générant des hypergraphes totalement équilibrés dont l'ensemble de sommets est V

1 $n = |V|$
2 $T_0 = (V, E_0)$ arbre aléatoire
3 **pour** i *de* 1 *à* $n-1$ **faire**
4 $\quad V_i = \{x \cup y \mid x, y \in V_{i-1}, (x, y) \in E_{i-1}\}$
5 $\quad G = (V_i, \{(u, v) \mid u \in V_i, v \in V_i, \exists x, y, z \in V_{i-1}, u = x \cup y, v = y \cup z\})$
6 $\quad T_i = (V_i, E_i) =$ `arbre_couvrant`(G)
7 **fin**
8 **retourner** $T_0, T_1, \ldots, T_{n-1}$

Avec `arbre_couvrant`(G) une fonction qui renvoie un arbre couvrant du graphe G.

Définition 1.3.8. Un graphe $T = (V', E')$ est appelé *arbre couvrant* de $G = (V, E)$ si :
- T est un arbre,
- $V' = V$,
- $E' \subseteq E$.

On peut également définir les *arbres couvrants minimaux* (resp. *maximaux*) comme étant des arbres couvrants tels que la somme des poids de leurs arêtes (ou leur nombre si les arêtes ne sont pas pondérées) est minimale (resp. maximale).

Nous utiliserons cette caractérisation dans la Section 3.3 pour montrer le caractère totalement équilibré d'un modèle d'apprentissage. Nous présenterons de plus dans la Section 5.3 une construction similaire à celle-ci permettant de caractériser les hypergraphes binaires (introduits dans la Section 5.2) en utilisant des arbres mixtes (c'est-à-dire utilisant à la fois des arêtes orientées et des arêtes non orientées) et en adaptant l'idée de contraction des arêtes présente dans cette construction.

La Figure 1.11 donne un exemple de séquence d'arbres construite par la méthode de Lehel.

Ce résultat illustre de manière simple un résultat de Anstee [3] : l'hypergraphe totalement équilibré de taille maximum à n sommets possède $m = \binom{n+1}{2}$ hyperarêtes avec $n - k + 1$ hyperarêtes de taille k pour tout k entre 1 et n.

Matrices totalement équilibrées

Définition 1.3.9. Soit $\mathcal{M}$ une matrice binaire. $\mathcal{M}$ est dite *totalement équilibrée* si son hypergraphe associé est totalement équilibré.

Ces matrices sont donc équivalentes aux autres systèmes sans cycles déjà évoqués. Elles ont été caractérisées par l'existence d'ordres Γ-free par Anstee et Farber [4].

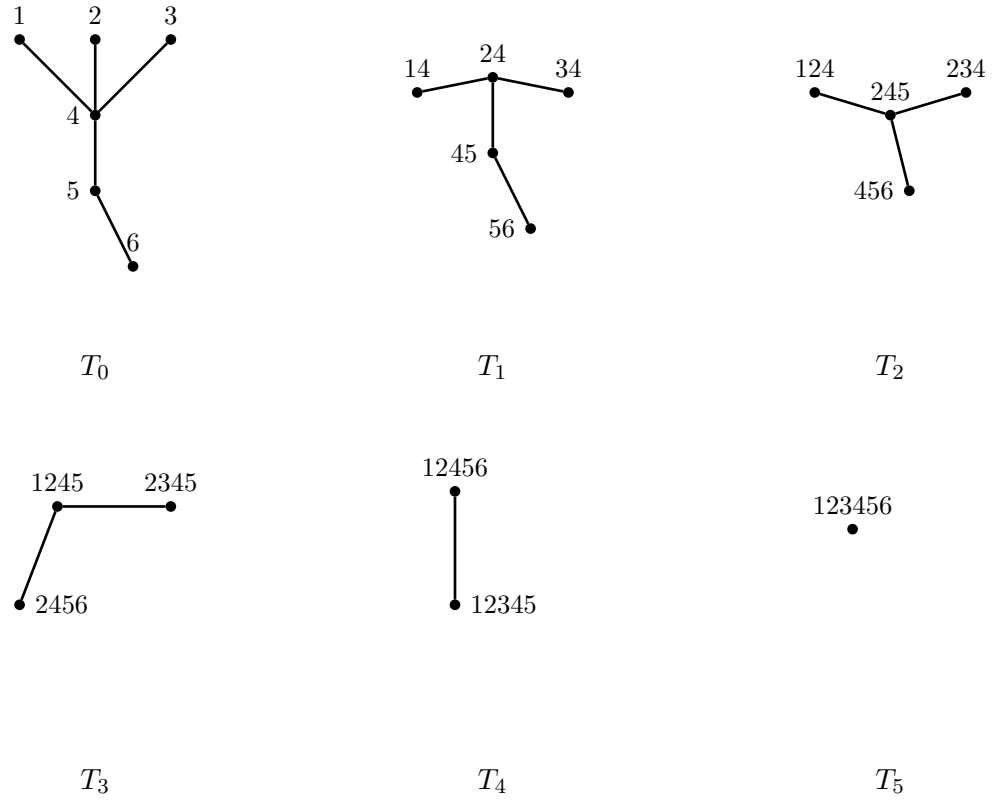


FIGURE 1.11 – Exemple de construction d'une séquence d'arbres de l'Algorithme 3

	c_1	c_2	c_3	c_4
l_1	×			×
l_2	×	×		
l_3		×	×	
l_4			×	×

(a) Exemple de matrice non totalement équilibrée

	c_1	c_2	c_4	c_3
l_2	×	×		
l_1	×		×	
l_3		×		×
l_4			×	×

(b) Ordre doublement lexical de la matrice

TABLE 1.5 – Exemple de matrice non totalement équilibrée

Théorème 1.3.5 (Anstee, Farber [4]). *Soit $\mathcal{M}$ une matrice binaire. Les trois propositions suivantes sont équivalentes :*

(i) $\mathcal{M}$ est totalement équilibrée ;

(ii) $\mathcal{M}$ admet un ordre doublement lexical Γ-free ;

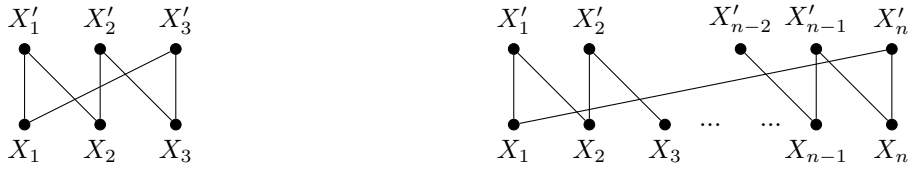
(iii) Tous les ordres doublement lexicaux de $\mathcal{M}$ sont Γ-free.

D'après ce résultat, la Table 1.1b prise en exemple précédemment pour illustrer le cas d'une matrice Γ-free est donc totalement équilibrée. La Table 1.5a représente la matrice associée à un 4-cycle et n'est donc pas totalement équilibrée. En effet, la Table 1.5b représente la matrice réordonnée selon un ordre doublement lexical et elle contient toujours un Γ.

Une fois une matrice quelconque réordonnée, il suffit de vérifier si la matrice est Γ-free pour savoir si elle est totalement équilibrée. Un algorithme de Spinrad permet de le faire en $\mathcal{O}(nm)$, avec n le nombre lignes et m le nombre de colonnes de la matrice. Enfin, si la matrice obtenue n'est pas Γ-free, un algorithme d'approximation consistant à ajouter des 1 à la matrice pour supprimer les Γ peut être appliqué. Ces différents algorithmes sont décrits dans [42]. Les ordres Γ-free d'une matrice totalement équilibrée permettent notamment une représentation claire des classes d'un système [16].

Treillis démontables

L'absence de cycles dans les différents modèles se décline également dans le cas des treillis. Ces treillis sont dits *sans couronnes* [29]. Une *couronne* est un poset $(X_0, X'_0, \ldots, X_{n-1}, X'_{n-1})$ tel que pour tout i, $X_i < X'_{i-1 \mod n}$, $X_i < X'_i$ et toutes les autres paires d'éléments sont incomparables. L'équivalence entre les treillis sans couronne et les systèmes de classes sans cycles est immédiate. Le diagramme de Hasse d'une 3-couronne est représenté sur la Figure 1.12a et celui d'une n-couronne sur la Figure 1.12b. Nous pouvons noter que si $(L, \leq)$ est un treillis sans 3-couronne, pour tout $a, b, c \in L$, $a \wedge b \wedge c \in \{a \wedge b, a \wedge c, b \wedge c\}$, ce qui rejoint la définition des hiérarchies faibles donnée dans la Section 1.3.1. Les treillis sans couronnes sont également connus sous le nom de treillis *démontables*.



(a) 3-couronne (b) n-couronne

FIGURE 1.12 – Diagramme de Hasse de couronnes

Définition 1.3.10. Soit $\mathcal{L} = (L, \leq)$ un treillis. $\mathcal{L}$ est un treillis *démontable* ssi il existe $L_0, \ldots, L_n$ tel que :

— $(L_0, \leq)$ est un treillis vide,

— $L_n = L$,

— $\forall i \in \{1, \ldots, n\}, \exists x \in L_i, x$ doublement irréductible, $L_{i-1} = L_i \backslash \{x\}$

Cette définition caractérise les treillis démontables par l'existence d'un ordre d'élimination des éléments doublement irréductibles.

Les treillis sans couronnes seront principalement utilisés dans le Chapitre 4 où nous montrerons l'équivalence entre les treillis binarisables et les treillis sans couronnes.

Dissimilarités totalement équilibrées

L'équivalent dans le monde des dissimilarités des structures totalement équilibrées présentées auparavant n'existe pour l'instant pas. Néanmoins, les dissimilarités peuvent être liées à des objets totalemnt équilibrés.

Dans la Section 1.1.5, les boules de la dissimilarité décrite par la Table 1.2 étaient données dans la Table 1.3. L'hypergraphe des boules de d était donc $(\{x, y, z, t\}, \{\{x\}, \{y\}, \{z\}, \{t\}, \{x, z\}, \{x, y, z\}, \{x, z, t\}, \{y, z, t\}, \{x, y, z, t\}\})$. Cet hypergraphe possède un cycle : $\{x\}, \{x, z, t\}, \{t\}, \{y, z, t\}, \{y\}, \{x, y, z\}$. La Figure 1.5 représente les graphes seuil associés à d. L'hypergraphe des classes de d est donc $(\{x, y, z, t\}, \{\{x\}, \{y\}, \{z\}, \{t\}, \{x, z\}, \{y, z\}, \{x, z, t\}, \{y, z, t\}, \{x, y, z, t\}\})$. Cet hypergraphe est totalement équilibré et admet pour ordre totalement équilibré $x < y < z < t$.

Il est possible d'associer à tout graphe $G = (V, E)$ une dissimilarité d_G sur V définie par $d_G(x, y) = 1$ si $(x, y) \in E$ et $d_G(x, y) = 2$ sinon. D'après [21], $\mathcal{B}(d_G)$ est totalement équilibré ssi $\mathcal{C}(d_G)$ est totalement équilibré. Ceci n'est plus vrai dans le cas d'une dissimilarité quelconque comme illustré par l'exemple donné dans le paragraphe précédent. L'un des deux sens de l'équivalence reste néanmoins vrai comme nous le montrerons dans la Section 6.2 (Proposition 6.2.1).

Nous définirons dans le Chapitre 6 les *dissimilarités totalement équilibrées*.

Chapitre 2

Arbres de décision K-Means

2.1 Introduction

Ce chapitre présente un modèle de classification supervisée semblable aux arbres de décision, utilisant des idées de classification non supervisée (partitionnement de données). Comme remarqué dans la Section 1.2.2, les arbres de décision classiques utilisent les classes de sortie des exemples pour diviser les nœuds et atteindre des nœuds purs le plus vite possible. Nous cherchons ici à utiliser uniquement la description des données et non les classes qui leur ont été assignées.

Un modèle de classification de données ne peut être interprétable que si la structure des données (ici, les attributs) contient des informations sur la classe de sortie. Le bon fonctionnement de nos modèles sur des données réelles suppose donc que la représentation des données choisie (la base dans laquelle elles sont décrites) et les distances qui en résultent sont cohérentes avec les classes de sortie. La description des données doit les représenter de manière pertinente : deux éléments proches dans la base choisie doivent se ressembler et inversement. Dans le cas contraire, les données seront inexploitables. Nous cherchons dans ce travail à utiliser la structure des données, à travers les distances entre les objets, et non leurs classes de sortie, pour construire un modèle de classification interprétable. Nous utilisons donc des notions d'analyse de données au sein d'une application pour l'apprentissage automatique.

En théorie, nous intégrons de l'analyse de données dans de l'apprentissage automatique et proposons un modèle de classification dont la construction s'effectue en $\mathcal{O}(nd \log n)$ (avec n le nombre d'individus et d la dimension) et la prédiction de la classe d'un nouvel exemple en $\mathcal{O}(d \log n)$.

En pratique, notre modèle est aussi efficace en prédiction de classes que les arbres de décision et construit un modèle dont les classes sont interprétables.

Ce chapitre est structuré comme suit : la Section 2.2 présente le nouveau modèle de classification proposé appelé *arbres de décision K-Means* en présentant l'algorithme (Section 2.2.1) ainsi qu'un comparatif des performances obtenues sur des ensembles de

données classiques avec les modèles d'arbres habituels (Section 2.2.2). La Section 2.3 étend le modèle à des forêts et présente l'algorithme (Section 2.3.1), l'intérêt d'utiliser des forêts plutôt que des arbres (Section 2.3.2) et les performances comparées aux autres modèles de forêts (Section 2.3.3).

2.2 Arbres de décision K-Means

2.2.1 Algorithme des arbres de décision K-Means

Algorithme de base

L'Algorithme 4 présente la méthode utilisée pour construire un arbre de décision K-Means. Le système obtenu est une hiérarchie permettant la prédiction pour les problèmes de classification et rappelle le principe des K-Means hiérarchiques [17] utilisés pour le partitionnement de données. L'algorithme produit un partitionnement hiérarchique dont les clusters les plus petits (au sens de l'inclusion) sont purs (pour les classes de sortie des exemples).

L'algorithme utilisé pour les K-Means est celui de Lloyd présenté dans la section 1.3.2 avec une initialisation des centres de départ par la méthode K-Means++ présentée dans la même section.

Algorithme 4 : Construction d'un arbre de décision K-Means de base

Données : Un ensemble d'exemples étiquetés
Résultat : Un arbre de décision K-Means

```
Créer un nœud racine contenant tous les exemples
Créer une file F contenant la racine
tant que la file F est non vide faire
    Dépiler un nœud de F
    si tous les exemples que le nœud contient ne sont pas de la même classe alors
        Calculer un K-Means avec k = 2
        Créer un nœud enfant du nœud courant pour chaque cluster obtenu
        Empiler les nœuds enfants dans F
    fin
fin
```

L'Algorithme 5 indique comment prédire la classe d'un exemple particulier. La méthode consiste simplement à descendre dans l'arbre en allant à chaque fois dans le nœud dont le centre est le plus proche de l'exemple à prédire.

La Figure 2.1 représente la construction d'un arbre de décision K-Means sur les données iris projetées en deux dimensions par analyse en composantes principales. Chaque figure représente la propagation des exemples dans un niveau de l'arbre de K-Means créé : la

Algorithme 5 : Prédiction de la classe d'un exemple par un arbre de décision K-Means

Données : Un arbre de décision K-Means et un exemple x à prédire
Résultat : La classe de sortie prédite pour x

```
1 nœud = racine de l'arbre
2 tant que nœud n'est pas une feuille faire
3 |   Propager x à l'enfant de nœud dont le centre est le plus proche de x
4 fin
5 retourner classe de nœud
```

première figure représente toutes les données ; la deuxième correspond à la séparation de l'ensemble de données par un K-Means avec $k = 2$; la troisième représente la séparation de chacune des deux zones de la figure précédente en deux zones par application d'un K-Means ; etc... Cette représentation illustre bien que chaque niveau de l'arbre de K-Means forme une partition tandis que la structure entière est une hiérarchie.

Perturbations aléatoires

Afin de pouvoir construire des forêts d'arbres de décision K-Means (Section 2.3), nous appliquons une perturbation aléatoire aux arbres de décision K-Means. Nous nous inspirons donc des arbres de décision classiques pour modifier le modèle d'arbres : à chaque nœud, le K-Means est calculé sur un sous-ensemble aléatoire et de taille fixée des variables. L'Algorithme 6 détaille la méthode qui diffère de l'Algorithme 4 par l'ajout d'un paramètre et la modification de la séparation (ligne 6).

La propagation d'un exemple dans ces arbres perturbés peut être modifiée. Nous pouvons remplacer la ligne 3 de l'Algorithme 5 par la propagation de l'exemple au nœud dont le centre projeté dans le sous-espace utilisé pour le K-Means est le plus proche de x projeté dans le même espace. En pratique, cette propagation donne naturellement de meilleurs résultats que la propagation au centre le plus proche sans tenir compte du sous-espace sur lequel le nœud a été divisé.

Complexité

Cette méthode de construction appelle au maximum n fois l'algorithme des K-Means, mais en pratique beaucoup moins. La complexité dans le pire des cas est donc $\mathcal{O}(n^2 d)$ avec d la dimension de l'espace et n le nombre d'éléments, car les K-Means ont une complexité de $\mathcal{O}(nd)$ (Section 1.3.2). Dans le cas d'un arbre binaire équilibré, la division d'un nœud de taille n revient à effectuer un calcul de K-Means (en $\mathcal{O}(nd)$)) puis à rappeler récursivement la division sur deux nœuds de taille $\frac{n}{2}$. L'algorithme de construction a donc une complexité de $\mathcal{O}(nd \log n)$, égale à celle de la construction d'un arbre de décision classique (Section 1.2.2).

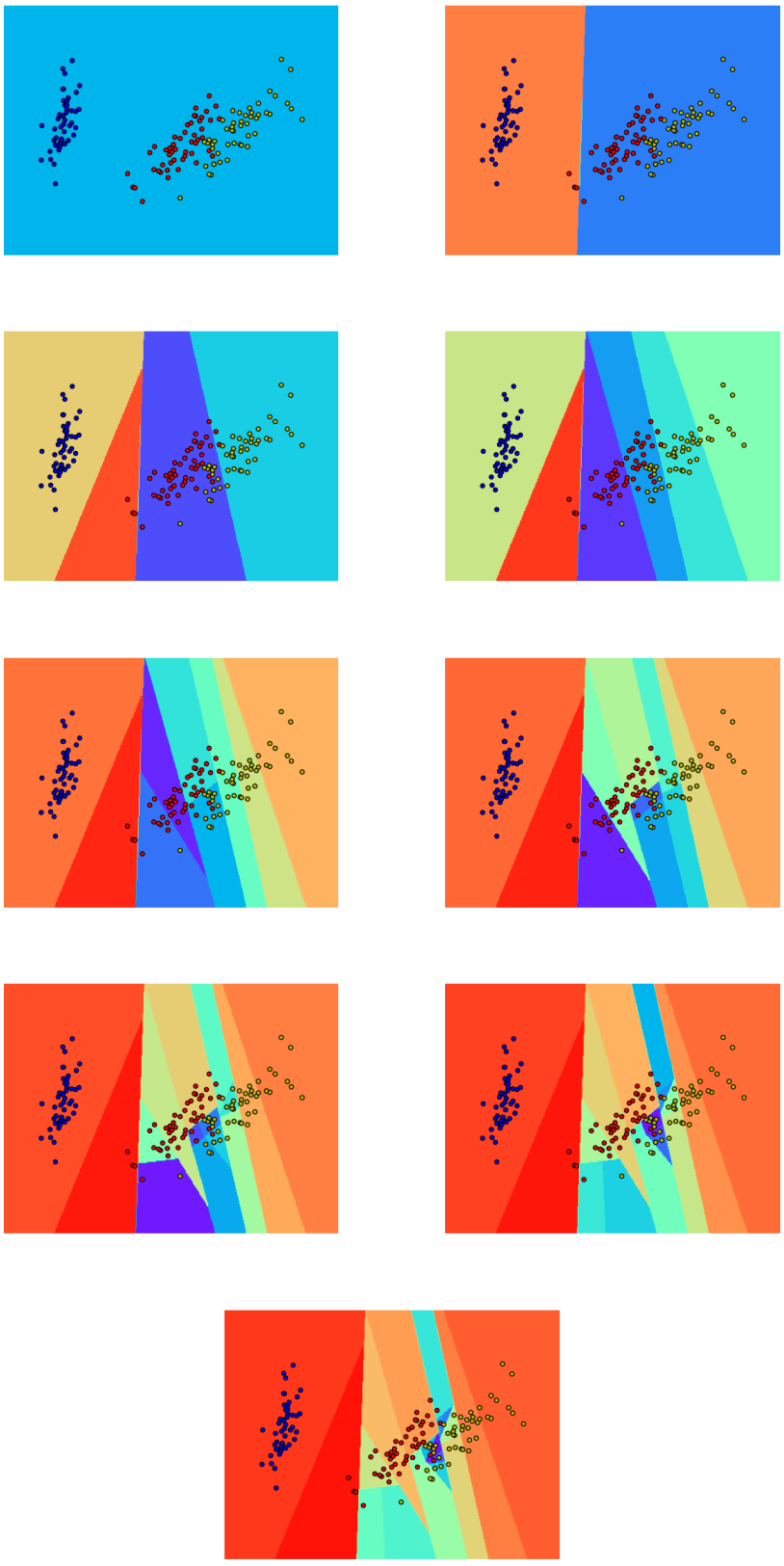


FIGURE 2.1 – Exemple d'application de l'Algorithme 4

Algorithme 6 : Construction d'un arbre de décision K-Means

Données : Un ensemble d'exemples étiquetés, `max_attributs` le nombre d'attributs à utiliser pour diviser les nœuds

Résultat : Un arbre de décision K-Means

```
Créer un nœud racine contenant tous les exemples
Créer une file F contenant la racine
tant que la file F est non vide faire
    Dépiler un nœud de F
    si tous les exemples que le nœud contient ne sont pas de la même classe alors
        Choisir au hasard max_attributs attributs
        Calculer un K-Means avec k = 2 sur le sous-ensemble défini par ces
         attributs
        Créer un nœud enfant du nœud courant pour chaque cluster obtenu
        Empiler les nœuds enfants dans F
    fin
fin
```

Pour la propagation d'un exemple dans un arbre de décision K-Means déjà construit, le choix de descendre dans un enfant ou l'autre d'un nœud se fait en calculant une distance et donc en $\mathcal{O}(d)$. La propagation d'un exemple est donc en $\mathcal{O}(d \log n)$ ce qui est supérieur d'un facteur d aux arbres classiques.

2.2.2 Comparaison des arbres de décision K-Means avec les modèles classiques d'arbres

Comparaison des résultats en apprentissage

Tous les résultats présentés dans cette section sont ceux obtenus par les paramètres donnant les meilleurs résultats :

- critère d'entropie ou de Gini pour les arbres classiques ;
- nombre d'attributs à utiliser lors de la séparation des nœuds pour les arbres classiques et les arbres de K-Means ;
- type de propagation pour les arbres K-Means : sur l'espace entier ou sur le sous-espace utilisé lors de la division.

La Table 2.1 présente les résultats obtenus sur certains ensembles de données classiques par dix répétitions de 10-folds. Les ensembles de données ainsi que la méthode d'évaluation sont décrits dans la Section 1.2.4.

La Table 2.1 compare également les modèles d'arbres à un *K-plus-proches-voisins*. Ce classifieur consiste simplement à assigner à un nouvel exemple la classe majoritaire parmi ses K plus proches voisins. Le résultat présenté est le meilleur obtenu en réglant le

	DT	KMDT	KN
breast-cancer	0.9198	**0.9357**	0.9601
digits	0.8679	**0.9495**	0.987
iris	0.9443	**0.9522**	0.9639
ringnorm	**0.8976**	0.8252	0.6902
satellite	0.8578	**0.8784**	0.9087
spambase	**0.9129**	0.7735	0.7978
vehicle	**0.7166**	0.6067	0.6495
waveform	0.7565	**0.7899**	0.8197
zoo	0.8773	**0.9080**	0.7820

DT : arbre de décision, KMDT : arbre de décision K-Means, KN : K-plus-proches-voisins
Résultat en gras quand le modèle est significativement meilleur que l'autre.

TABLE 2.1 – Performances des arbres de décision K-Means comparées à celles des arbres de décision classiques

paramètre K. Ce classifieur permet d'évaluer l'adéquation des données et de leurs classes de sorties avec une méthode de partitionnement. Dans l'hypothèse où l'information sur la classe de sortie des données est contenue dans leur structure même, il est raisonnable de supposer qu'un point est dans la même classe de sortie que des points qui lui ressemblent (et qui sont donc proches de lui). Les résultats de ce modèle permettent d'évaluer ce lien entre la structure des données et leurs classes de sortie mais il n'est pas utilisable en pratique à cause de sa complexité et il ne s'agit donc pas ici de comparer ses performances à celles des modèles évalués.

Nous pouvons constater sur cette table que les arbres de décision K-Means (KMDT) obtiennent des performances supérieures aux arbres de décision classiques (DT) sur 6 des 9 ensembles de données testés (breast-cancer, digits, iris, satellite, waveform et zoo). Ces ensembles sont de tailles et dimensions variées tout comme ceux sur lesquels les KMDT sont moins efficaces que les DT. Néanmoins, ils font partie des ensembles pour lesquels les K-plus-proches-voisins (KNN) sont les plus efficaces, indiquant une structure des données claire et propice à notre méthode. Nous pouvons de plus remarquer que sur les jeux de données où les arbres de décision K-Means réalisent de meilleures performances que les arbres de décision classiques, le gain est de quelques pour cent tandis que dans le cas contraire, la différence entre les deux modèles est beaucoup plus grande. À nouveau, c'est la structure des données qui est déterminante dans le résultat.

La Table 2.2a indique les tailles moyennes des arbres (en nombre de nœuds) construits pour les différents jeux de données. Les KMDT contiennent systématiquement plus de nœuds que les DT ce qui est inévitable puisque les deux modèles s'arrêtent quand leurs feuilles sont pures mais que les DT optimisent un critère de pureté tandis que les KMDT ne tiennent pas compte des classes de sortie dans la division. La Table 2.2b permet quant

	n	DT	KMDT
breast-cancer	683	55.94	141.76
digits	1797	282.54	474.50
iris	150	17.20	38.32
ringnorm	7400	643.94	3635
satellite	6435	947.62	2826.38
spambase	6301	505.94	3487.56
vehicle	846	269.62	917.56
waveform	5000	995.86	3616.68
zoo	101	22.62	29.62

(a) Nombres de nœuds des arbres de décision K-Means comparés aux arbres classiques

	KMDT/DT	Différence
breast-cancer	2.53	+1.59
digits	1.68	+8.16
iris	2.23	+0.79
ringnorm	5.64	-7.21
satellite	2.98	+2.06
spambase	6.89	-13.94
vehicle	3.40	-10.99
waveform	3.63	+3.34
zoo	1.30	+3.07

(b) Rapport de la taille des arbres et différence de performances

DT : arbre de décision, KMDT : arbre de décision K-Means
KMDT/DT : $\frac{\text{nombre de nœuds du meilleur modèle d'arbre de décision K-Means}}{\text{nombre de nœuds du meilleur modèle d'arbre de décision classique}}$
Difference : gain en score de test

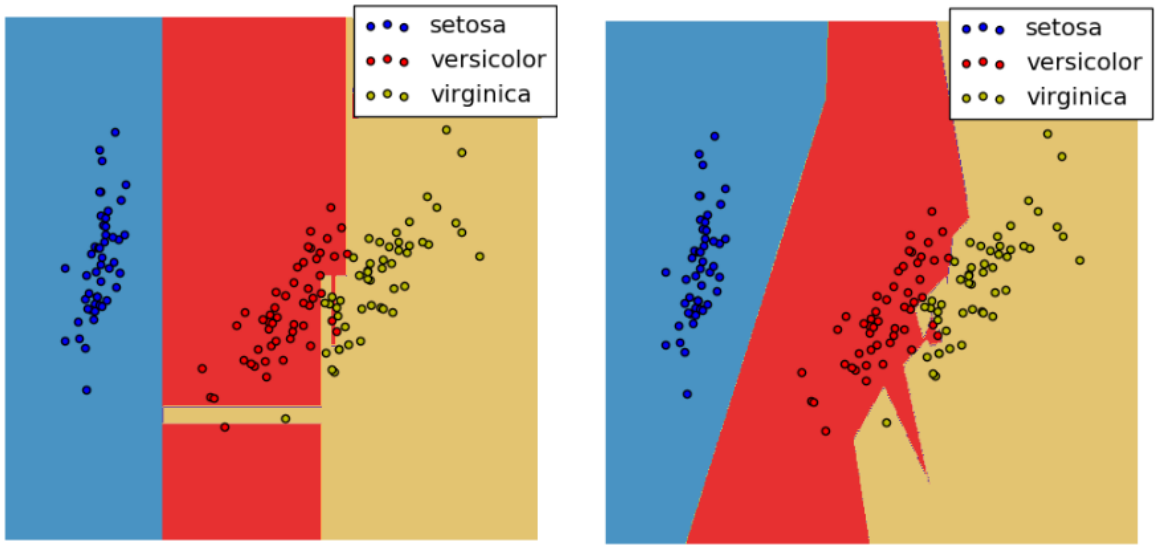
TABLE 2.2 – Comparaison des tailles des arbres de décision K-Means et des arbres de décision classiques

à elle de comparer le rapport entre la taille des KMDT et la taille des DT et la différence en performance sur l'ensemble de test. Nous pouvons remarquer que les jeux de données sur lesquels les KMDT sont beaucoup moins performants que les DT sont aussi ceux où la différence de taille est grande (5-6). La simplicité du modèle construit est donc déterminante dans sa capacité à généraliser. Les bases de données où les KMDT sont beaucoup moins performants que les DT (ringnorm, spambase, vehicle) sont aussi ceux où les arbres construits sont d'une taille de l'ordre de $\frac{n}{2}$ voire n ce qui montre la difficulté du modèle à atteindre des nœuds purs.

Comparaison des surfaces de décision

La Figure 2.2 représente les surfaces de décision d'un arbre de décision classique (à gauche) et d'un arbre de décision K-Means (à droite) obtenues sur une projection en deux dimensions de la base de données *iris* (par analyse en composantes principales) ainsi que les points utilisés pour obtenir ce résultat. La surface de décision correspond aux zones qui sont associées à chaque classe. On représente par exemple en rouge sur la figure la zone dans laquelle se trouvent tous les points qui seront classifiés comme rouges (de la classe versicolor) par le modèle.

La différence majeure entre les deux modèles réside dans la possibilité de créer des séparations non parallèles aux axes entre les classes pour les arbres de décision K-Means. En effet, la surface de décision d'un arbre de décision classique est l'assemblage de plusieurs rectangles (en deux dimensions) dont les côtés sont parallèles aux axes. Dans l'exemple



(a) Arbre de décision classique (b) Arbre de décision K-Means

FIGURE 2.2 – Surfaces de décision des arbres de décision K-Means et des arbres de décision classiques sur un exemple en deux dimensions

donné sur la Figure 2.2, la séparation entre la classe rouge et la classe bleue des arbres de décision K-Means paraît par exemple plus naturelle que celle proposée par les arbres de décision classiques.

Les deux modèles présentent également des séparations peu naturelles comme le rectangle jaune qui coupe en deux la classe rouge pour les arbres de décision ou encore le pic rouge dans la classe jaune pour les arbres K-Means.

2.3 Forêts de décision K-Means

2.3.1 Algorithme de forêts de décision K-Means

Comme expliqué dans la Section 1.2.3, en apprentissage automatique, on utilise souvent des ensembles d'arbres de décision dont la construction est légèrement perturbée de manière aléatoire afin de réduire la variance. Nous appliquons cette même idée aux arbres de décision K-Means avec l'Algorithme 7.

La prédiction de la classe d'un exemple se fait ensuite en le propageant dans chacun des arbres. La classe attribuée à l'exemple est alors la classe majoritairement renvoyée par les arbres.

Algorithme 7 : Construction d'une forêt d'arbres de décision K-Means

Données : un ensemble d'exemples étiquetés
t le nombre d'arbres à construire
`max_attributs` le nombre d'attributs à utiliser

1 **pour** *i de* 1 *à t* **faire**
2 Construire un arbre de décision K-Means avec le paramètre `max_attributs` sur l'ensemble d'exemples
3 **fin**

	KMDT	KMRF
breast-cancer	0.9357	0.9546
digits	0.9495	0.9789
iris	0.9522	0.9584
ringnorm	0.8252	0.9056
satellite	0.8784	0.9099
spambase	0.7735	0.9295
vehicle	0.6067	0.7134
waveform	0.7899	0.8362
zoo	0.9080	0.9263

KMDT : arbre de décision K-Means, KMRF : forêt d'arbre de décision K-Means

TABLE 2.3 – Performances des forêts de décision K-Means comparées à celles des arbres de décision K-Means

	RF	ET	KMRF
breast-cancer	0.9541	0.9537	0.9546
digits	0.9607	0.9682	**0.9789**
iris	0.9450	0.9523	0.9584
ringnorm	0.9540	**0.9638**	0.9056
satellite	0.9097	0.9083	0.9099
spambase	**0.9461**	**0.9470**	0.9295
vehicle	0.7495	**0.7603**	0.7134
waveform	0.8218	0.8239	**0.8362**
zoo	0.8993	0.9037	**0.9263**

RF : random forests, ET : extra trees, KMRF : forêt d'arbre de décision K-Means
Résultat en gras quand le modèle est significativement meilleur qu'un autre.

TABLE 2.4 – Performances des forêts de décision K-Means comparées à celles des forêts classiques

2.3.2 Impact de l'utilisation des forêts sur les résultats en apprentissage

La Table 2.3 compare les résultats obtenus par les arbres de décision K-Means et par les forêts de décision K-Means. Comme dans la section précédente, les résultats obtenus présentés sont les meilleurs pour chaque modèle utilisant 10 arbres.

Pour tous les jeux de données, l'utilisation des forêts d'arbres de décision augmente significativement les performances, ce qui est naturel. Nous pouvons de plus remarquer que sur les ensembles présentant des performances assez faibles pour les arbres de décision K-Means (ringnorm, spambase, vehicle), l'augmentation des performances permise par l'utilisation des forêts est la plus forte avec des augmentations comprises entre +8% et +15% alors que pour les autres ensembles le gain est souvent de +2-3%.

2.3.3 Comparaison des forêts de K-Means avec les modèles classiques de forêts

La Table 2.4 permet de comparer les performances en apprentissage des forêts d'arbres de décision K-Means (KMRF) avec les forêts d'arbres classiques et plus particulièrement les random forests (RF) et les extra-trees (ET) présentés dans la Section 1.2.3. Comme précédemment, les résultats présentés sont les meilleurs obtenus en réglant les différents paramètres, pour dix arbres dans chaque forêt.

Les KMRF sont :

— plus performants que les RF et ET sur 3 ensembles de données sur 9,
— équivalents aux RF et ET sur 3 ensembles de données sur 9,
— moins performants que les RF et ET sur 3 ensembles de données sur 9.

Les ensembles de données sur lesquels les KMRF sont moins performants que les autres modèles de forêts sont les mêmes que ceux sur lesquels les arbres K-Means sont moins performants que les autres modèles d'arbres. Néanmoins l'écart entre les modèles est réduit. En effet il passe de :

— -7.24% à -5.82% pour l'ensemble ringnorm,
— -13.94% à -1.75% pour l'ensemble spambase,
— -10.99% à -4.69% pour l'ensemble vehicle.

Le même phénomène se produit sur les ensembles sur lesquels les arbres de K-Means sont plus performants que les arbres classiques :

— écart de $+8.16\%$ pour les arbres contre $+1.07\%$ pour les forêts sur digits,
— $+3.34\%$ pour les arbres contre $+1.23\%$ pour les forêts sur waveform,
— $+3.07\%$ pour les arbres contre $+2.26\%$ pour les forêts sur zoo.

L'utilisation des forêts par rapport aux arbres tend donc à lisser les différences entre les modèles en matière de performance en apprentissage.

2.4 Conclusion

Dans ce chapitre, nous avons présenté un nouveau modèle de classification pour l'apprentissage automatique. Ce modèle inspiré des arbres de décision n'utilise dans sa construction que les K-Means, méthode d'analyse de données, et ne prend donc pas en compte les classes de sortie des exemples, contrairement aux modèles classiques d'apprentissage automatique.

Les résultats obtenus par le modèle proposé en classification sur des ensembles de données classiques sont bons. En effet, selon l'ensemble de données étudié, les résultats peuvent être meilleurs ou équivalents à ceux des arbres de décision classiques. Ils sont aussi parfois nettement moins bons sur certains ensembles dont la structure ne reflète pas les classes de sortie. Il convient de plus de souligner que l'un des ensembles sur lesquels les arbres de décision K-Means sont moins performants que les arbres de décision classique est un ensemble généré et non pas un ensemble de données réelles. Ceci marque une différence entre la méthode proposée, capable de retrouver la structure des classes de sortie dans les données si la description y correspond, et les méthodes d'apprentissage automatique supervisé, destinées à retrouver une structure prédéterminée et quelconque.

Ces résultats (rappelés intégralement dans la Table 2.5) tendent à confirmer l'intérêt de l'approche structurelle de la classification que nous proposons. En effet, le fait de réussir à classifier les exemples de manière efficace sans utiliser la classe de sortie des données lors de la construction du modèle prouve que la structure des données contient l'information de leur classe de sortie dans le cas de données réelles. Le système de classes produit par l'arbre (en définissant les classes comme les ensembles d'éléments qui se propagent jusqu'à un nœud donné) est une hiérarchie et ne crée donc pas de classes empiétantes.

Nous avons de plus montré que les performances des arbres K-Means peuvent être

	DT	KMDT	RF	ET	KMRF	KN
breast-cancer	0.9198	**0.9357**	0.9541	0.9537	0.9546	0.9601
digits	0.8679	**0.9495**	0.9607	0.9682	**0.9789**	0.9870
iris	0.9443	**0.9522**	0.9450	0.9523	0.9584	0.9639
ringnorm	**0.8976**	0.8252	0.9540	**0.9638**	0.9056	0.6902
satellite	0.8578	**0.8784**	0.9097	0.9083	0.9099	0.9087
spambase	**0.9129**	0.7735	**0.9461**	**0.9470**	0.9295	0.7978
vehicle	**0.7166**	0.6067	0.7495	**0.7603**	0.7134	0.6495
waveform	0.7565	**0.7899**	0.8218	0.8239	**0.8362**	0.8197
zoo	0.8773	**0.9080**	0.8993	0.9037	**0.9263**	0.7820

TABLE 2.5 – Récapitulatif des performances de tous les modèles liés aux arbres de décision classiques et de K-Means testés

améliorées en construisant des forêts d'arbres inspirés des modèles classiques de forêts comme les random forests ou les extra trees. Les performances des forêts d'arbres de décision K-Means ont été meilleures que celles des arbres K-Means sur tous les ensembles testés, montrant l'intérêt des ensembles, comme pour les arbres de décision classiques et particulièrement dans les cas où un arbre de K-Means seul obtenait des performances assez basses. De plus, l'utilisation des forêts permet de réduire les différences de résultats entre les arbres K-Means et les arbres classiques.

Chapitre 3

Systèmes de classes de centres de gravité

3.1 Introduction

Le Chapitre 2 présente un modèle de classification utilisant l'analyse de données pour créer un arbre de décision. Cette approche valide la pertinence de l'utilisation de la description des données comme élément de classification, suffisant à la prédiction de classes de nouveaux objets. Néanmoins, le modèle proposé ne permet pas encore la création de classes empiétantes. Il permet l'utilisation de notions d'analyse de données pure dans le cadre de l'apprentissage automatique. Ce chapitre réalise la démarche inverse et propose un modèle appelé *systèmes de classes de centres de gravité*, basé également sur la structure des données et des notions de distance mais permettant cette fois l'empiétance des classes ; empruntant à l'apprentissage automatique et plus particulièrement aux arbres de décision pour réaliser un modèle de classification totalement équilibré.

Pour cela, nous partons de l'ensemble des données et rassemblons petit à petit les éléments qui se ressemblent en utilisant des arbres couvrants minimum du graphe de distances des éléments considérés. Notre construction est locale : il s'agit de regrouper les éléments proches, en construisant un arbre, localement, autour de chaque élément pour obtenir un arbre global. Nous proposons une construction simple dont l'utilisation est inspirée de celle des arbres de décision.

En théorie, nous proposons une première approche d'un modèle totalement équilibré, autorisant l'empiétance des classes, utilisable en apprentissage automatique. Nous intégrons de l'apprentissage automatique dans des idées issues de l'analyse de données et construisons notre modèle localement. Nous analysons également l'intérêt de réaliser la construction de manière locale plutôt que globale. Le modèle propose de plus, une approximation de données vectorielles par un modèle totalement équilibré.

En pratique, le modèle est utilisable pour la prédiction mais ne fonctionne que sur les jeux de données de petite taille et petite dimension. Il permet de mettre en évidence

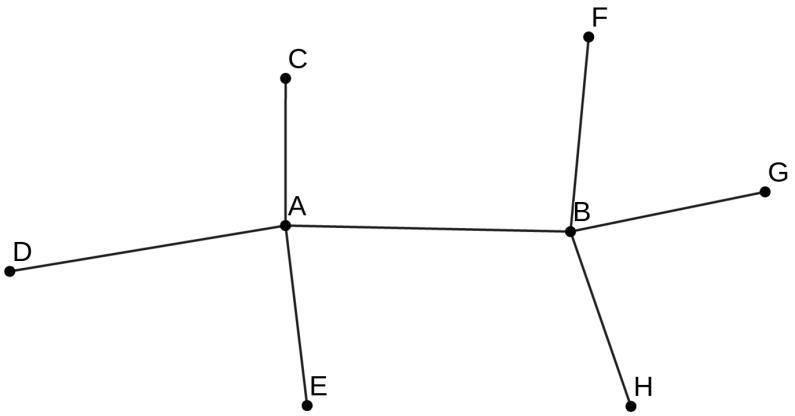


FIGURE 3.1 – Exemple d'arbre couvrant minimum constituant la première étape de l'Algorithme 8

le besoin de construire les modèles en divisant petit à petit les données plutôt qu'en les rassemblant pour les utiliser dans des tâches de prédiction.

Nous proposons un modèle de classification en classes empiétantes (Section 3.2). Nous montrons dans la Section 3.3.1 que le modèle présenté respecte les conditions définies par Lehel ([32] et détaillées dans la Section 1.1.1) et est donc un hypergraphe totalement équilibré. Le modèle est ainsi un système de classes totalement équilibré. Nous proposons une variante plus naturelle du modèle dans la Section 3.3.2 et montrons l'avantage de la méthode locale choisie. Nous présentons enfin quelques résultats d'utilisation de cette méthode dans le cadre de l'apprentissage automatique dans la Section 3.4.

3.2 Algorithme de systèmes de classes de centres de gravité

Algorithme de base

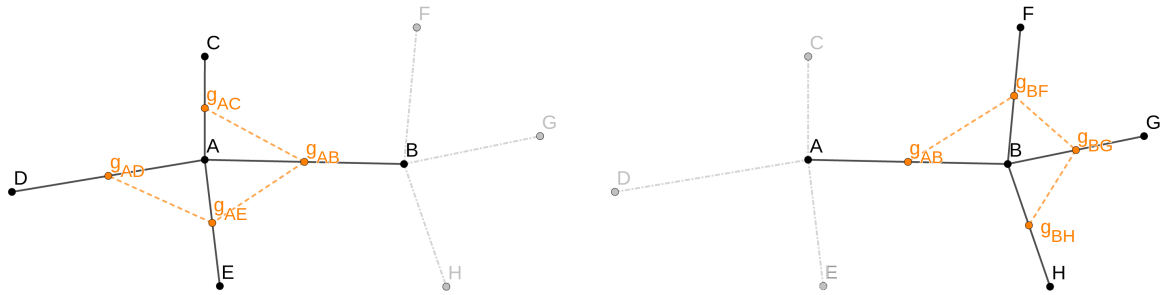
L'Algorithme 8 présente la méthode de création d'un système de classes de centre de gravité. Dans cet algorithme, la notation g_{xy} désigne le centre de gravité des points x et y, et la lettre d représente la distance euclidienne. L'idée générale est de construire le modèle couche par couche, en partant de l'ensemble de tous les objets donnés, puis, à chaque couche, de rassembler des groupes d'exemples proches les uns des autres deux par deux en créant un nouveau groupe représenté par son centre de gravité. Un système de classes de centre de gravité créé à partir de n exemples contient donc n couches telles que les éléments de la couche k contiennent k exemples.

Algorithme 8 : Création d'un système de classes de centres de gravité

Données : $X = (x_1, \ldots, x_n)$ avec pour tout $i, x_i \in \mathbb{R}^m$
Résultat : S un système de classes de centres de gravité

$D_0 = (X, \{(x_i, x_j) \in X \times X \mid i, j \leq n\}, (x_i, x_j) \in X \times X \mapsto d(x_i, x_j))$
$T_0 =$ `arbre_couvrant_minimum`$(D_0) = (V_0, E_0)$
$i = 0$
tant que $|V_i| \neq 1$ **faire**
  $D_{i+1} = (\{g_{xy} \mid (x, y) \in E_i\}, \{(g_{xy}, g_{st}) \mid (x, y) \in E_i, (s, t) \in E_i\}, d)$
  Créer file F
  Choisir un nœud x dans T_i
  Mettre x dans F
  Marquer x
  tant que $F \neq \emptyset$ **faire**
    Dépiler l'élément x de F
    pour *chaque voisin* y *de* x *dans* T_i **faire**
      Empiler y dans F
      Marquer y
    fin
    $T^x_{i+1} =$ `arbre_couvrant_minimum`$(D_{i+1|\{g_{xy}|(x,y)\in E_i\}})$
  fin
  $T_{i+1} = \bigcup_x T^x_{i+1}$
  $i = i + 1$
fin
retourner $\bigcup_i V_i$

Avec `arbre_couvrant_minimum`(G) une fonction retournant un arbre couvrant minimum du graphe G



(a) Arbre couvrant minimum des milieux des arêtes liées à A

(b) Arbre couvrant minimum des milieux des arêtes liées à B

La partie utilisée de l'arbre de base est représentée en noir tandis que la partie ignorée à ce moment de l'algorithme est en gris pointillés. L'arbre couvrant local réalisé est représenté en orange pointillés.

FIGURE 3.2 – Arbres couvrants minimum locaux pour l'arbre de la Figure 3.1

La Figure 3.1 représente un arbre couvrant minimum d'un ensemble de 8 points sur lequel appliquer la construction. Les points C, D, E, F, G et H n'ont qu'un voisin chacun. L'arbre couvrant minimum créé à la ligne 16 pour chacun de ces points est donc constitué d'un point unique (respectivement $g_{AC}, g_{AD}, g_{AE}, g_{BF}, g_{BG}, g_{BH}$).

La Figure 3.2 représente les arbres couvrants minimum locaux construits à la ligne 16 de l'algorithme pour les points A (Figure 3.2a) et B (Figure 3.2b). Les deux arbres couvrants sont ensuite réunis pour faire un seul arbre et terminer une étape de l'algorithme. L'arbre obtenu est représenté en orange sur la Figure 3.3.

La propagation dans la structure pour l'utilisation en apprentissage automatique se fait de manière similaire à celle des arbres de décision K-Means présentée dans l'Algorithme 5 :

— Tant que le nœud n'est pas pur : propager l'exemple à l'enfant du nœud courant dont le centre est le plus proche de lui,
— Retourner la classe du nœud pur atteint.

La seule différence notable est que, dans le cas des arbres de décision K-Means, les nœuds purs sont exactement les feuilles tandis que dans le cas des systèmes de classes de centre de gravité, un nœud pur peut se trouver au milieu de la structure. Cela résulte du sens de construction des deux modèles. En effet, les arbres de décision K-Means, comme les arbres de décision classiques, sont construits de haut en bas (dit "top-down") c'est-à-dire en partant de l'ensemble des données et s'arrêtent donc dès qu'ils créent un nœud pur. Les systèmes de classes de centres de gravité quant à eux sont construits de bas en haut (dit "bottom-up"), en partant de chaque point individuellement et en les rassemblant petit à petit. Ils ne peuvent donc pas tenir compte de la pureté des nœuds lors de la construction.

La construction bottom-up a ainsi l'inconvénient de créer des modèles plus gros que

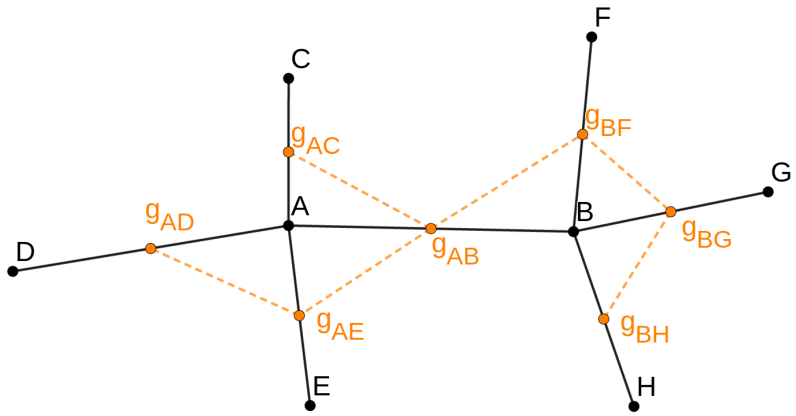


FIGURE 3.3 – Arbre obtenu pour l'arbre de la Figure 3.1 par une étape de l'Algorithme 8

les arbres de décision K-Means et les arbres de décision classiques. Là où les arbres de décision K-Means réalisaient déjà des structures plus grandes que les arbres de décision car ils n'optimisent pas le critère d'arrêt de construction, les systèmes de classes de centres de gravité doivent être construits entièrement et possèdent exactement $\frac{n(n+1)}{2}$ éléments avec n le nombre d'éléments dans l'ensemble de données.

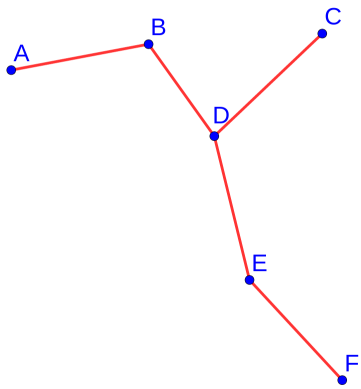
Exemple

La Figure 3.4 représente les différents arbres couvrants minimums construits lors de l'exécution de l'Algorithme 8 sur l'ensemble de points A, B, C, D, E, F représenté sur le premier arbre. La représentation respecte la position des points en tant que milieux de segments entre deux points de l'arbre précédent.

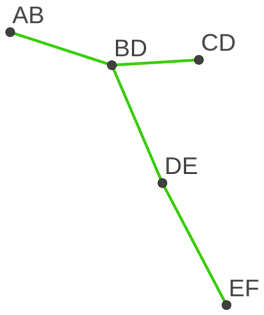
La Figure 3.5 représente le modèle obtenu correspondant à l'exécution détaillée dans la Figure 3.4. La représentation consiste à dessiner les sommets d'un arbre par niveau, en commençant par le niveau le plus bas puis à relier deux sommets de deux niveaux consécutifs si le sommet du niveau supérieur est obtenu à partir du sommet du niveau inférieur. En d'autres termes, si i et j sont des sommets d'un niveau et que g_{ij} est un sommet du niveau supérieur, alors i et j sont reliés à g_{ij} dans la représentation proposée. Ainsi la propagation d'un nouvel exemple dans la structure est visuelle. En partant du sommet le plus haut, l'exemple descend pour rejoindre le sommet adjacent dont il est le plus proche.

Complexité

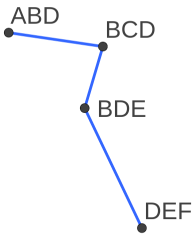
La construction a une complexité totale de $\mathcal{O}(n(nd + n^2))$. En effet, l'algorithme construit exactement n arbres (avec n le nombre d'éléments) et pour passer d'un arbre



T_0



T_1



T_2

T_3

T_4

T_5

FIGURE 3.4 – Exemple d'arbres couvrants construits par l'Algorithme 8

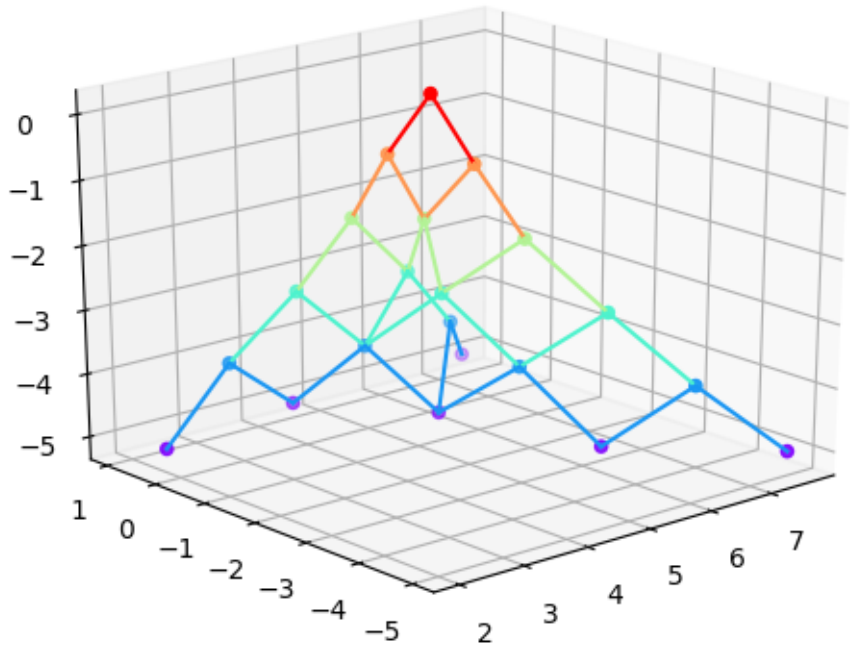


FIGURE 3.5 – Représentation 3D du modèle obtenu correspondant aux arbres de la Figure 3.4

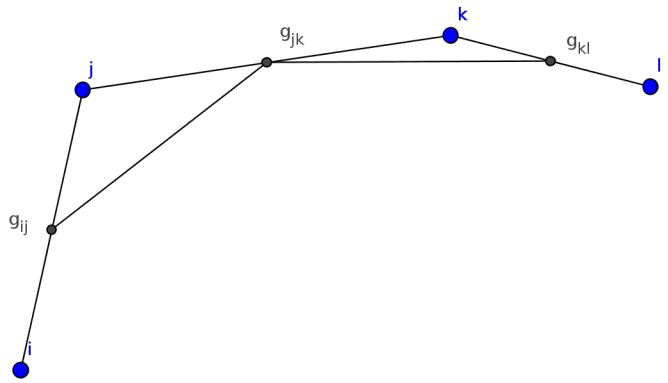


FIGURE 3.6 – Illustration d'une étape de l'Algorithme 8 pour un chemin de taille 4

à l'autre, il calcule un arbre couvrant minimum local pour chaque sommet de l'arbre et calcule les milieux de chaque arête. Le calcul des milieux de chaque arête s'effectue en nd avec d la dimension. Le calcul de l'arbre couvrant à partir des milieux par l'algorithme de Prim a une complexité $\mathcal{O}(n^2)$.

La complexité de la propagation d'un exemple est pour sa part $\mathcal{O}(nd)$. En effet, la structure comporte n étages et chaque propagation d'un nœud à l'autre nécessite de calculer deux distances.

3.3 Propriétés

3.3.1 Un système totalement équilibré

Les systèmes de classes de centres de gravité constituent une première approche des modèles de classification en classes empiétantes que nous souhaitons développer. Nous allons montrer dans cette partie que la structure construite respecte la construction proposée par Lehel [32] pour les hypergraphes totalement équilibrés et détaillée dans la Section 1.1.1. Ces systèmes de classes sont donc totalement équilibrés.

Nous commençons par poser quelques notations. Notons g_{ij} le centre de gravité des points i et j. Nous définissons de plus D_X le *graphe des distances* d'un ensemble de points $X = x_0, \ldots, x_{n-1}$ par $D_X = (X, A_X, w_X)$ avec $A_X = \{(x_i, x_j) \mid x_i, x_j \in X\}$ et $w_X : (x_i, x_j) \in X \times X \mapsto d(x_i, x_j)$ avec d la distance euclidienne.

Théorème 3.3.1. *Soit S obtenu par l'Algorithme 8. S est un système de classes totalement équilibré.*

Démonstration. Par construction, l'Algorithme 8 construit une séquence de graphes T_0, $\ldots, T_m$ (avec $T_i = (V_i, E_i)$ pour tout i) telle que pour tout $i \leq m$, pour tout $x \in V_i$ et pour tout $y \in V_i$ tels que $(x, y) \in E_i$, $g_{xy} \in V_{i+1}$.

De plus T_0 est un arbre et le passage de T_i à T_{i+1} conserve la structure d'arbre. Pour tout i, T_i est donc un arbre.

De plus, toujours par construction, pour tout i, $(g_{xy}, g_{st}) \in E_{i+1}$ seulement si $y = s$ et $(x, y) \in E_i, (s, t) \in E_i$.

Les arbres construits par l'Algorithme 8 respectent donc la construction de Lehel (Théorème 1.3.4). L'algorithme produit ainsi un hypergraphe totalement équilibré. Par construction l'hypergraphe produit est clos par intersection, l'algorithme produit donc un système de classes totalement équilibré. □

3.3.2 Variante globale du modèle

L'algorithme proposé permet de construire un modèle pour une analyse globale des données en réalisant des opérations locales et en ne considérant que des informations locales (un élément et ses voisins). Une variante naturelle de l'algorithme donné serait de réaliser la même construction mais de manière globale plutôt que locale à chaque étape : au lieu de faire localement un arbre minimum couvrant de chaque ensemble de sommets issus d'un même sommet de l'arbre précédent, il serait envisageable de calculer tous les milieux de paires de sommets voisins dans l'arbre précédent et réaliser un arbre couvrant minimum de l'ensemble de ces sommets. L'Algorithme 9 détaille cette construction.

Algorithme 9 : Création d'une structure de centres de gravité par une méthode globale

Données : $X = (x_1, \ldots, x_n)$ avec pour tout $i, x_i \in \mathbb{R}^m$
Résultat : S un système de classes de centres de gravité

$D_0 = (X, \{(x_i, x_j) \in X \times X \mid i, j \leq n\}, (x_i, x_j) \in X \times X \mapsto d(x_i, x_j))$
$T_0 =$ `arbre_couvrant_minimum`$(D_0) = (V_0, E_0)$
$i = 0$
tant que $|V_i| \neq 1$ **faire**
  $D_{i+1} = (\{g_{xy} \mid (x, y) \in E_i\}, \{(g_{xy}, g_{st}) \mid (x, y) \in E_i, (s, t) \in E_i\}, d)$
  $T_{i+1} =$ `arbre_couvrant_minimum`(D_{i+1})
  $i = i + 1$
fin
retourner $\bigcup_i V_i$

Avec `arbre_couvrant_minimum`(G) une fonction retournant un arbre couvrant minimum du graphe G

Sur un chemin de taille 4, les deux algorithmes donnent le même résultat comme le montre la Proposition suivante. Le résultat est illustré par la Figure 3.6 pour le cas en deux dimensions : l'application de l'Algorithme 8 comme de l'Algorithme 9 sur un chemin de taille 4 (i, j, k, l) donne le chemin de taille 3 (g_{ij}, g_{jk}, g_{kl}).

Proposition 3.3.2. *Soient* $i, j, k, l \in \mathbb{R}^m$ *tels que* $T = (\{i, j, k, l\}, \{(i, j), (j, k), (k, l)\})$ *est*

un arbre couvrant minimum de $D_{\{i,j,k,l\}}$. *Alors* $T_g = (\{g_{ij}, g_{jk}, g_{kl}\}, \{(g_{ij}, g_{jk}), (g_{jk}, g_{kl})\})$ *est un arbre couvrant minimum de* $D_{\{g_{ij},g_{jk},g_{kl}\}}$.

Démonstration. Soient $i, j, k, l \in \mathbb{R}^m$ tels que $T = (\{i, j, k, l\}, \{(i, j), (j, k), (k, l)\})$ est un arbre couvrant minimum de $D_{\{i,j,k,l\}}$. Supposons, pour simplifier la preuve, que l'axe x passe par le point i et par le point k. On note x_i la coordonnée x de i.

Dans ce cas, $x_{g_{kl}} \leq f$ avec $f = \frac{x_i+2x_j+x_k}{4}$ (équation de la médiatrice de g_{ij} et g_{jk}).

$$\begin{array}{rl} x_{g_{kl}} \leq f & \iff \frac{x_k+x_l}{2} \leq f \\ & \iff x_l \leq 2f - x_k \end{array}.$$

Or $d(k, l) \leq d(i, l)$ car $T = (\{i, j, k, l\}, \{(i, j), (j, k), (k, l)\})$ est un arbre couvrant minimum de $D_{\{i,j,k,l\}}$ (sinon l'arête (k, l) serait remplacée par (i, l)). Donc $x_l \geq d$ avec $d = \frac{x_i+x_k}{2}$ (équation de la médiatrice de i et k).

Nous avons donc $x_l \leq 2f - x_k$ seulement si $d \leq 2f - x_k$. Ce qui équivaut à

$$\frac{x_i + x_k}{2} \leq \frac{x_i + 2x_j + x_k}{2} - x_k$$

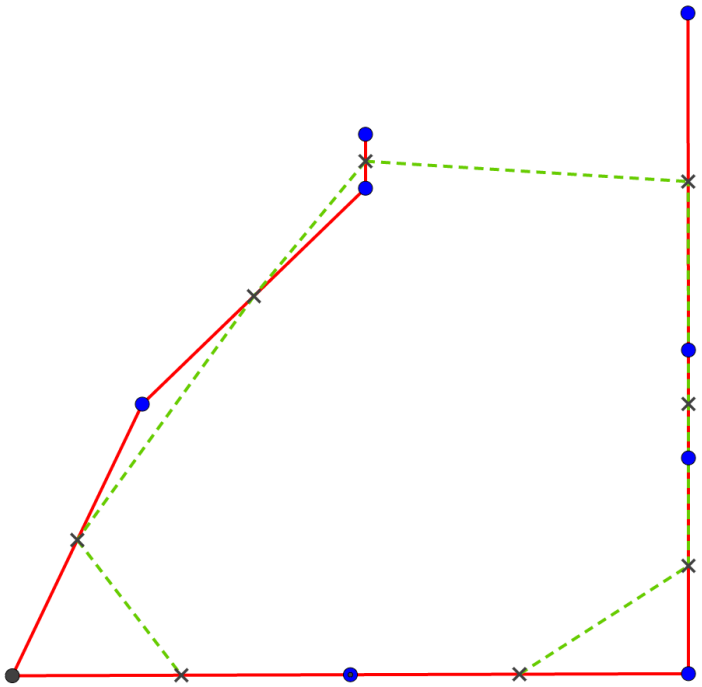
qui équivaut à $x_k \leq x_j$. Ceci est impossible car $T = (\{i, j, k, l\}, \{(i, j), (j, k), (k, l)\})$ est un arbre couvrant minimum de $D_{\{i,j,k,l\}}$ donc $d(i, j) \leq d(i, k)$ (sinon l'arête (i, j) serait remplacée par (i, k)). □

Néanmoins cet algorithme ne respecte pas la caractérisation des hypergraphes totalement équilibrés donnée par Lehel et le modèle issu d'une telle construction ne serait donc pas totalement équilibré. Il est en effet possible de construire un ensemble de points tel que la construction globale et la construction locale sont différentes et tel que la construction globale ne respecte pas les conditions données par Lehel.

La Figure 3.7 représente la construction globale d'un cas dans lequel la construction globale et la construction locale diffèrent. L'arbre couvrant minimum (en traits pleins) de l'ensemble de points considérés (les points ronds) est un simple chemin tout comme l'arbre couvrant minimum (en traits pointillés) des milieux des arêtes (en forme de croix) de l'arbre précédent. Néanmoins ces deux chemins ne respectent pas la construction donnée dans la caractérisation de Lehel. En effet, les extrémités du chemin pointillés ne sont pas le milieu d'arêtes dont l'une des extrémités est une extrémité du chemin en traits pleins.

Ce type de configuration, qui peut paraître trop spécifique pour arriver dans le cadre de données réelles, survient en réalité de plus en plus quand la densité de points augmente, rendant l'utilisation d'un tel modèle en supposant l'équilibre total impossible. La construction du modèle par la méthode locale est donc nécessaire à l'obtention d'un modèle totalement équilibré.

La Table 3.1a présente les résultats expérimentaux obtenus sur la différence entre la construction globale et la construction locale. Pour chaque combinaison dimension/nombre de points, 100 tests ont été effectués et cette table présente le nombre de fois où les deux constructions ont été différentes. La diminution entre la dimension 5 et la dimension 10



Les points ronds représentent les points de départ ; l'arbre en traits pleins, un arbre couvrant minimum de ces points ; les points en forme de croix sont les milieux des arêtes de l'arbre plein et l'arbre en pointillés est un arbre couvrant minimum des points en forme de croix.

FIGURE 3.7 – Contre-exemple de construction par la méthode globale

d \ n	20	100	500	1000
2	0	0	4	65
3	1	5	9	61
5	0	6	21	74
10	0	0	16	78

(a) Nombre de fois où la construction globale diffère de la construction locale

d \ n	20	100	500	1000
2	×	×	1	209.86
3	1	1	1	190.97
5	×	1.16	1	214.9
10	×	×	1	213.6

(b) Nombre moyen de différences entre la construction locale et la construction globale d'un même ensemble de points

n : nombre d'essais, d : dimension

TABLE 3.1 – Différence entre la construction locale et la construction globale sur des ensembles de points tirés au hasard

pour 100 et 500 points peut s'expliquer par le changement de densité de points dûe à l'augmentation de la dimension. On observe clairement qu'à dimension constante, l'augmentation du nombre de points amplifie la fréquence du phénomène jusqu'à se produire pour 1000 points dans la plupart des cas. La construction globale n'est donc, en théorie comme en pratique, pas un modèle totalement équilibré.

La Table 3.1b présente quant à elle le nombre de fois où les deux constructions diffèrent lors de l'exécution de la méthode sur le même ensemble de points. Pour cela, dès que la construction locale et la globale diffèrent, on continue l'exécution à partir de la construction locale. Pour des faibles nombres de points, nous pouvons observer que les deux constructions diffèrent une seule fois dans l'exécution. Il pourrait donc paraître envisageable de réaliser une approximation de la construction à l'étape où les deux modèles diffèrent. Néanmoins, dans le cas de 1000 points, quelle que soit la dimension, le nombre de fois où les deux constructions diffèrent explose et il devient inenvisageable d'utiliser la construction globale. En effet, celle-ci s'éloigne trop des modèles totalement équilibrés.

3.4 Résultats des systèmes de classes de centres de gravité en apprentissage

La Table 3.2 présente les résultats obtenus en apprentissage par le modèle des systèmes de classes de centres de gravité. Ces résultats sont, comme pour le chapitre précédent, des moyennes des résultats de dix validations croisées 10-folds. Les résultats obtenus sur la plupart des ensembles de données sont très mauvais. Les seuls ensembles dont les résultats sont proches de ceux des arbres de décision sont les ensembles de données de petite taille (iris et zoo). Cela peut être dû à la taille du modèle. En effet, comme indiqué précédemment, un système de classes de centres de gravité possède $\frac{n(n+1)}{2}$ éléments et les modèles construits sur des données nombreuses sont ainsi de très grandes tailles et semblent perdre de l'information. La création des modèles de manière top-down semble ainsi être plus efficace dans le cadre de la classification : pour obtenir un modèle efficace et généralisable, il vaut mieux séparer petit à petit ce qui diffère que rassembler petit à petit ce qui se ressemble.

L'obtention de résultats équivalents aux arbres de décision dans le cas de petits jeux de données permet néanmoins de valider que les modèles totalement équilibrés ne sont pas incompatibles avec la classification de données en apprentissage automatique.

3.5 Conclusion

Ce chapitre présente une première approche de la construction de modèles de classification pour l'apprentissage automatique, inspirés d'arbres de décision mais permettant l'empiétance des classes. Le modèle s'appuie sur une construction locale simple permet-

	DT	GDS
breast-cancer	0.9236	0.7831
digits	0.8719	0.1717
iris	0.9415	0.937
satellite	0.8601	0.2279
spambase	0.9172	0.5612
vehicle	0.7255	0.4322
waveform	0.7609	0.4293
zoo	0.8853	0.8697

DT : arbre de décision, GDS : système de centres de gravité

TABLE 3.2 – Performances en classification des systèmes de classes de centres de gravité comparées à celles des arbres de décision

tant de rassembler petit à petit les ensembles d'objets proches (représentés par des milieux de représentants de groupes de points). Cette construction respecte la caractérisation des hypergraphes totalement équilibrés proposée par Lehel [32] et inscrit donc le modèle dans le monde des structures totalement équilibrées.

Une construction plus intuitive viserait à réaliser la même opération que celle proposée en considérant le problème de manière non plus locale, mais globale. Nous avons néanmoins montré que dans ce cas, le modèle sortait clairement du cadre des structures totalement équilibrées à travers des simulations sur des ensembles de points aléatoires.

Nous avons enfin présenté des résultats obtenus pour le modèle en apprentissage automatique sur des jeux de données classiques. Ces résultats sont très mauvais sur la plupart des ensembles et le modèle ne parvient à de bons résultats que sur les ensembles de données les plus petits et de dimension raisonnable. Ces résultats tendent à montrer que l'utilisation de modèles totalement équilibrés autorisant l'empiétance peut permettre d'obtenir de bons résultats de classification (équivalents à ceux des arbres de décision) mais que la construction du modèle de manière bottom-up est contre-productive pour l'application en apprentissage automatique sur des ensembles de données plus conséquents.

Chapitre 4

Binarisation des treillis démontables

4.1 Introduction

Dans le cadre de l'apprentissage automatique, les arbres de décision sont le plus souvent binaires afin de préserver une certaine simplicité du modèle. Dans ce domaine comme dans bien d'autres, les structures binaires sont à la fois efficaces et simples. Nous cherchons donc dans cette section à étendre la notion de binarité des arbres aux treillis. Par treillis binaire, nous entendons treillis tel que chaque élément couvre au maximum deux autres éléments et est couvert par au maximum deux. Nous nous intéresserons aux treillis binarisables *i.e.* qui peuvent être plongés dans un treillis binaire. Cette transformation d'un treillis non binaire en un treillis binaire est similaire à celle d'un nœud non binaire en une succession de nœuds binaires dans un arbre de décision. Par contre, contrairement aux arbres, tous les treillis ne peuvent pas être transformés en treillis binaires. Nous relions dans ce chapitre les systèmes binaires et les systèmes totalement équilibrés et cherchons à expliquer ce qui permet à ces structures d'être binaires. Cette approche vise à renforcer un peu plus les liens entre structures totalement équilibrées et arbres. En effet, l'intérêt de considérer des structures binaires dans le cas des arbres n'est plus à démontrer.

En théorie, nous définissons la notion de treillis binaires et binarisables et montrons que les treillis binarisables sont exactement les treillis démontables. Nous donnons aussi un algorithme efficace qui ajoute le moins d'éléments possible à un treillis démontable pour l'approximer par un treillis binaire.

En pratique, nous proposons une approximation des treillis démontables par un treillis binaire, ce qui permet de simplifier l'interprétation des modèles et détaillons l'application de cet algorithme en analyse de concepts formels. Nous montrons également que les éléments ajoutés au treillis pour l'approximation sont interprétables. De plus, le résultat théorique d'équivalence entre les treillis démontables et binarisables permet de plus de désigner les treillis démontables comme parfaits candidats au développement de modèles de prédiction admettant l'empiétance des classes.

Ce chapitre est organisé comme suit : la Section 4.2 présente les définitions des nou-

veaux types de treillis que nous proposons, la Section 4.3 détaille l'algorithme de binarisation d'un treillis démontable, la Section 4.4 utilise l'algorithme pour démontrer l'équivalence entre les treillis démontables et les treillis binaires. Enfin, la Section 4.5 propose une application en analyse de concepts formels (FCA) en définissant l'analyse de concepts formels et les liens avec notre sujet (Section 4.5.1) puis en détaillant l'application de la méthode sur un exemple (Section 4.5.2).

4.2 Treillis binaires

Nous définissons les treillis binaires en nous inspirant de la définition d'un arbre de décision binaire. Un arbre de décision binaire est tel que chaque nœud interne possède deux enfants. De plus, tous les nœuds sauf la racine possèdent un unique parent. Nous étendons cette définition en prenant en compte le fait qu'un nœud d'un treillis peut être couvert par plus d'un élément. Dans ces structures, l'équivalent des questions binaires de la forme "$X \leq \alpha$?" posées dans les arbres de décision devient une question à propos de deux attributs. Un objet peut donc avoir un des deux attributs considérés ou les deux pour être propagé dans la structure.

Définition 4.2.1. Soit $\mathcal{L} = (L, \leq)$ un treillis fini. $\mathcal{L}$ est *binaire* ssi :

$$\forall v \in L, \begin{cases} |\{u \in L \mid u \prec v\}| \leq 2 \\ |\{w \in L \mid v \prec w\}| \leq 2 \end{cases}$$

Si seule la première condition est vérifée, on dira que $\mathcal{L}$ est *couverture-binaire*.

Dans cette définition, tous les éléments doivent être binaires. En pratique, nous exclurons l'élément $\bot$ de la définition et autoriserons $|\{u \in L \mid \bot \prec u\}| > 2$. En effet, pour l'interprétation en classification, le fait que $\bot$ soit couvert par plus de deux éléments indique simplement l'existence de plus de deux objets. Forcer cet élément à n'être couvert que par deux éléments créerait donc de nombreux nouveaux éléments sans apporter d'information supplémentaire intéressante.

La Figure 4.1 présente un treillis simple non binaire. En effet, l'élément $\top$ et l'élément X couvrent trois éléments.

Nous cherchons ici à binariser un treillis non binaire.

Définition 4.2.2. $\mathcal{L} = (L, \leq)$ est *binarisable* ssi il existe $\mathcal{L}' = (L', \leq)$ tel que $\mathcal{L}'$ est binaire et $\mathcal{L}$ peut être plongé dans $\mathcal{L}'$ (*i.e.* il existe $f : L \to L'$ telle que pour tout $x, y \in L$, $x \leq y$ ssi $f(x) \leq f(y)$).

La Figure 4.2 propose deux treillis binaires dans lesquels le treillis initial de la Figure 4.1 peut être plongé. Dans les deux cas, deux éléments sont ajoutés au treillis, ce qui est le minimum possible.

On dira qu'un treillis $\mathcal{L}'$ est une *binarisation* d'un treillis $\mathcal{L}$ ssi $\mathcal{L}'$ est binaire et $\mathcal{L}$ peut être plongé dans $\mathcal{L}'$. Si $\mathcal{L}'$ respecte seulement la première condition de la Définition 4.2.1, on dira que $\mathcal{L}'$ est une *couverture-binarisation* de $\mathcal{L}$.

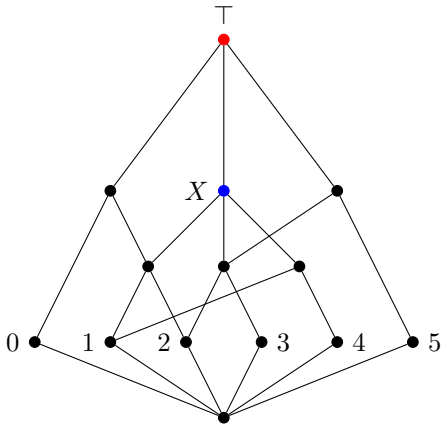


FIGURE 4.1 – Exemple de treillis non binaire

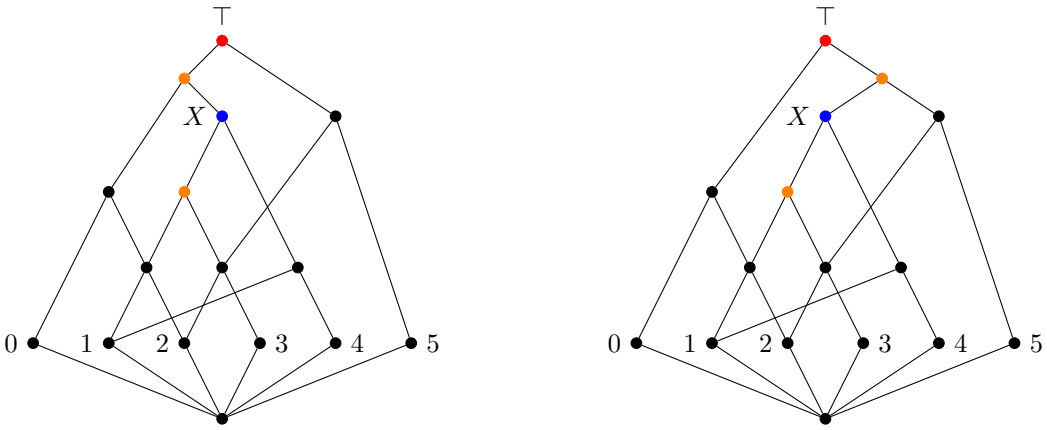


FIGURE 4.2 – Deux binarisations possibles du treillis de la Figure 4.1

Algorithme 10 : Couverture-binarisation d'un système de classes

Données : S un système de classes

Résultat : $\mathcal{B}(S)$ un système de classes tel que chaque élément en couvre au plus deux et tel que S peut être plongé dans $\mathcal{B}(S)$.

```
1  B(S) = S
2  pour Y ∈ S faire
3  |  C = {X ∈ B(S) | X ≺ Y}
4  |  tant que |C| > 2 faire
5  |  |  Choisir X_i, X_j d'intersection maximale parmi C
6  |  |  B(S) = B(S) ∪ {X_i ∪ X_j}
7  |  |  C = C ∪ {X_i ∪ X_j}\{X_i, X_j}
8  |  fin
9  fin
10 retourner B(S)
```

4.3 Algorithme de binarisation des treillis démontables

Nous présentons dans cette partie un algorithme permettant de couverture-binariser un système de classes totalement équilibré. Les systèmes de classes et les treillis étant équivalents, le résultat de cet algorithme donne un système de classes binaires équivalent à un treillis couverture-binaire. Nous étendons toutes les définitions données dans ce chapitre pour les treillis (binaire, binarisable...) aux systèmes de classes.

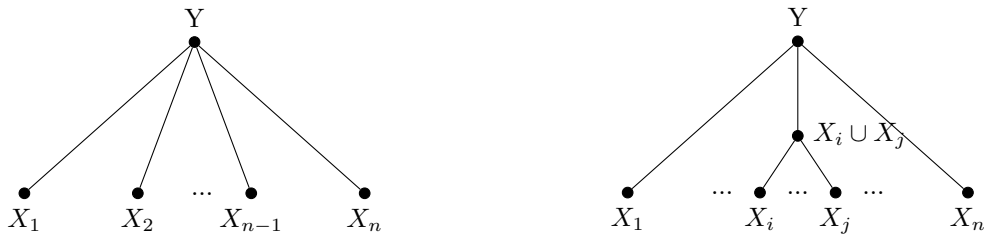
Le processus de binarisation consiste à couverture-binariser chaque élément non binaire en créant l'union de certains des éléments qu'il couvre. Pour conserver la clôture par intersection en ajoutant un élément à la fois, les éléments utilisés pour en créer de nouveaux éléments doivent être choisis avec soin.

Définition 4.3.1. Soit $\{X_1, \ldots, X_n\}$ un ensemble de sous-ensembles incomparables du même ensemble. X_i et X_j sont d'*intersection maximale* parmi $\{X_1, X_2, \ldots X_n\}$ ssi il n'existe aucun $k \neq i, j$ tel que $X_i \cap X_j \subsetneq X_i \cap X_k$ ou $X_i \cap X_j \subsetneq X_j \cap X_k$.

Le processus de couverture-binarisation est décrit formellement dans l'Algorithme 10 et une étape du processus en est illustrée sur la Figure 4.3. L'algorithme proposé permet ici de créer un système de classes tel que chaque classe couvre au maximum deux autres classes. Afin d'obtenir un système binaire, il faut donc appliquer l'algorithme sur le système de classes associé au dual du treillis équivalent au système obtenu.

Le processus ajoute aussi peu d'éléments que possible au système de classes pour le transformer en un système binaire. En effet, si un élément Y couvre k éléments $X_1, \ldots, X_k$, notre construction ajoute exactement $k - 2$ éléments.

Le choix de deux éléments d'intersection maximale permet de prouver la correction de notre algorithme, mais est aussi interprétable d'un point de vue applicatif. En effet,



(a) Configuration initiale

(b) Exemple de première étape

FIGURE 4.3 – Illustration d'une étape du processus de binarisation

le fait de rassembler deux classes dont l'intersection est la plus grosse possible revient à mettre ensemble les classes qui se ressemblent le plus.

Notre algorithme revient à calculer toutes les intersections de classes deux à deux. Les systèmes considérés sont totalement équilibrés et possèdent donc au plus n^2 éléments avec n le nombre d'objets [3]. La complexité de notre algorithme est donc de $\mathcal{O}(n^5)$.

Les lemmes techniques qui suivent permettront de démontrer la Proposition 4.3.5 qui prouve que notre algorithme produit une couverture-binarisation d'un système de classes. Nous pouvons remarquer que tous ces lemmes s'appliquent à des systèmes de classes sans 3-couronnes mais pas forcément sans couronnes de toutes tailles. L'hypothèse de démontabilité ne sera utilisée que pour garantir la possibilité d'itérer le processus.

Lemme 4.3.1. *Soit L un système de classes sans 3-couronne et $\{X_1, \ldots, X_n\} \subset L$ un ensemble d'éléments incomparables de $\mathcal{L}$.*

Soient X_i et X_j d'intersection maximale parmi $\{X_1, \ldots X_n\}$.

Alors,

$$\forall l \in \{1, \ldots, n\}, X_l \cap (X_i \cup X_j) = \begin{cases} X_i \cap X_l \\ ou \\ X_j \cap X_l \end{cases}$$

Démonstration. Soient X_i, X_j d'intersection maximale parmi $\{X_1, \ldots, X_n\}$.

L ne possède pas de 3-couronnes, donc, pour tout $l, X_i \cap X_j \cap X_l \in \{X_i \cap X_j, X_i \cap X_l, X_j \cap X_l\}$.

X_i et X_j sont d'intersection maximale donc pour tout $l, X_i \cap X_j \not\subset X_l \cap X_i$ et $X_i \cap X_j \not\subset X_l \cap X_j$. D'où $X_i \cap X_j \cap X_l \in \{X_i \cap X_l, X_j \cap X_l\}$ *i.e.* $X_j \cap X_l \subseteq X_i \cap X_l$ ou $X_i \cap X_l \subseteq X_j \cap X_l$.

Cependant,

$$\forall l, X_l \cap (X_i \cup X_j) = (X_i \cap X_l) \cup (X_j \cap X_l)$$

D'où

$$\forall l, X_l \cap (X_i \cup X_j) = \begin{cases} X_i \cap X_l \\ ou \\ X_j \cap X_l \end{cases} \iff \begin{cases} X_j \cap X_l \subseteq X_i \cap X_l \\ ou \\ X_i \cap X_l \subseteq X_j \cap X_l \end{cases}$$

□

Lemme 4.3.2. *Soit L un système de classes sans 3-couronnes et $\{X_1, \ldots, X_n\} \subset L$ l'ensemble d'éléments couverts par un élément $Y \in L$. Soient X_i, X_j, deux éléments d'intersection maximale parmi $\{X_1, \ldots X_n\}$:*

$$\forall Z \in L \setminus \{X_1, \ldots, X_n, Y\}, Z \cap (X_i \cup X_j) \in L \cup \{Xi \cup X_j\}.$$

Démonstration. Si $Z \cap X_i \subseteq X_j$ alors $Z \cap (X_i \cup X_j) = Z \cap X_j$. L est clos par intersection donc $Z \cap X_j \in L$. Le même raisonnement s'applique à X_j.

Supposons à présent que $Z \cap X_i \setminus \{X_j\} \neq \emptyset$ et $Z \cap X_j \setminus \{X_i\} \neq \emptyset$. Nous avons $X_i \cap X_j \subseteq Z$, car L ne possède pas de 3-couronne. Donc $X_i \cap X_j \subseteq Z \cap Y$.

Si $Y \subseteq Z$, le résultat est évident. Si $Z \parallel Y$ ou $Z \subsetneq Y$ alors $Z \cap Y \subsetneq Y$ donc il existe un k tel que $Z \cap Y \subseteq X_k$, car Y couvre exactement les éléments $(X_1, \ldots, X_n)$. On a $X_i \cap X_j \subseteq Z \cap Y \subseteq X_k$ et X_i, X_j sont d'intersection maximale donc $k = i$ ou $k = j$. D'où $Z \cap Y \subseteq X_i$ (ou de manière symétrique $Z \cap Y \subseteq X_j$) ce qui conduit à $Z \cap Y \subseteq Z \cap X_i$ donc $Z \cap Y \subseteq Z \cap (X_i \cup X_j)$.

De plus, comme $X_i \cup X_j \subseteq Y$, $Z \cap (X_i \cup X_j) \subseteq Z \cap Y$, par double inclusion $Z \cap (X_i \cup X_j) = Z \cap Y$. Enfin, L est clos par intersection donc $Z \cap Y \in L$ ce qui termine la preuve. □

Lemme 4.3.3. *Soit L un système de classes sans 3-couronnes et $\{X_1, \ldots, X_n\} \in L$ l'ensemble d'éléments couverts par $Y \in L$. Pour tout X_i, X_j d'intersection maximale parmi $\{X_1, \ldots, X_n\}$, les éléments de $\{X_1, \ldots X_n\} \cup \{X_i \cup X_j\} \setminus \{X_i, X_j\}$ sont incomparables.*

Démonstration. Montrons que,

$$\forall k \neq i, j, X_i \cup X_j \parallel X_k.$$

Pour tout $k \neq i, j$, X_i, X_j et X_k sont incomparables donc $X_i \nsubseteq X_k$ et $X_j \nsubseteq X_k$ d'où $X_i \cup X_j \nsubseteq X_k$.

Comme L ne possède pas de 3-couronnes,

$$X_i \cap X_j \cap X_k \in \{X_i \cap X_j, X_i \cap X_k, X_j \cap X_k\}$$

donc X_i et X_j étant d'intersection maximale, $X_i \cap X_j \not\subset X_i \cap X_k$ et $X_i \cap X_j \not\subset X_j \cap X_k$. Donc $X_i \cap X_j \cap X_k \in \{X_i \cap X_k, X_j \cap X_k\}$.

Supposons que $X_i \cap X_j \cap X_k = X_i \cap X_k$. Nous avons alors $X_i \cap X_k \subseteq X_j$. De plus, $(X_1, \ldots, X_n)$ étant incomparables, il existe $x \in X_k$ tel que $x \notin X_j$. Or $X_i \cap X_k \subseteq X_j$, donc $x \notin X_i$. Donc il existe $x \in X_k, x \notin X_i \cup X_j$ donc $X_k \nsubseteq X_i \cup X_j$. □

Proposition 4.3.4. *L'Algorithme 10 appliqué sur un système de classes S retourne un système de classes $\mathcal{B}(S)$ tel que :*

— *pour tout $X \in \mathcal{B}(S)$, X couvre au plus deux éléments,*

— *$S \subseteq \mathcal{B}(S)$,*

— *pour tout $X \in S$, X est couvert par le même nombre d'éléments dans S et dans $\mathcal{B}(S)$.*

Démonstration. D'après les Lemmes 4.3.1 et 4.3.2, la création de nouveaux éléments par la méthode décrite dans la construction préserve la clôture par intersection du système. Le Lemme 4.3.1 montre que l'intersection de n'importe quel élément de la partie du système traitée et du nouvel élément est déjà dans le système tandis que le Lemme 4.3.2 montre la même chose pour les autres éléments. De plus, l'élément minimum et l'élément maximum sont inchangés car l'algorithme n'ajoute que des unions de deux éléments existants. Le résultat de l'algorithme est donc bien un système de classes. Le Lemme 4.3.3 montre que si X_i et X_j sont sélectionnés dans l'ensemble d'éléments incomparables $\{X_1, \ldots, X_n\}$ couverts par un élément Y pour créer un nouvel élément, les seuls changements opérés dans la relation de couverture du système sont :

— $X_i \cup X_j \prec Y$;
— $X_i \prec X_i \cup X_j$;
— $X_j \prec X_i \cup X_j$.

Ceci prouve qu'aucune couronne n'est ajoutée au système de classes. De plus, pour $k = i, j$, X_k n'est plus couvert par Y mais par $X_i \cup X_j$ et les autres éléments sont inchangés. Le nombre d'éléments qui couvrent un élément donné est donc inchangé. Le Lemme 4.3.3 montre également que le processus peut être itéré si le système de classes ne possède toujours pas de 3-couronnes.

L'algorithme termine quand tous les éléments du système initial couvrent au plus deux éléments. Les éléments ajoutés par la construction couvrent toujours exactement deux éléments par construction. Le système obtenu est donc bien tel que chaque élément couvre au maximum deux éléments. □

Proposition 4.3.5. *Soit $\mathcal{L} = (L, \leq)$ un treillis fini. Si $\mathcal{L}$ est démontable alors $\mathcal{L}$ est binarisable.*

Démonstration. Soit A le système de classes associé à $\mathcal{L}$ par la méthode détaillée dans la Section 1.1.4. L'Algorithme 10 peut être appliqué sur ce système de classes. D'après la Proposition 4.3.4, le système de classes obtenu $\mathcal{B}(A)$ est tel que $A \subseteq \mathcal{B}(A)$ et $\mathcal{B}(A)$ est tel que tous ses éléments en couvrent au plus deux. De plus, $\mathcal{L}$ peut être trivialement plongé dans $\mathcal{L}' = (\mathcal{B}(A), \subseteq)$.

Soit B le système de classes associé au dual de $\mathcal{L}'$ par la même méthode que précédemment. D'après la Proposition 4.3.4, $\mathcal{B}(B)$, le résultat de l'Algorithme 10 appliqué à B est binaire. Finalement, $\mathcal{L}'$ peut être plongé dans le treillis binaire $\mathcal{L}'' = (\mathcal{B}(B), \subseteq)$ donc $\mathcal{L}$ peut également être plongé dans $\mathcal{L}''$ par transitivité du plongement. □

4.4 Équivalence treillis démontables/treillis binaires

Les résultats présentés dans la section précédente permettent de prouver par une construction que les treillis démontables sont binarisables. Prouvons à présent l'autre sens du résultat principal de ce chapitre. Nous utiliserons pour cela la propriété suivante.

Propriété 4.4.1. *Soit $\mathcal{L} = (L, \leq)$ un treillis fini. Si $(X_0, X'_0, \ldots, X_{n-1}, X'_{n-1})$ est une couronne de $\mathcal{L}$ alors $(X_0, X_0 \vee X_1, \ldots, X_{n-1}, X_{n-1} \vee X_0)$ est une couronne de $\mathcal{L}$.*

Démonstration. Soient $\mathcal{L}$ un treillis et $(X_0, X'_0, \ldots, X_{n-1}, X'_{n-1})$ une couronne de $\mathcal{L}$. Montrons que

$$\forall j \neq i, i+1(\mod n), X_i \vee X_{i+1} \parallel X_j.$$

Soient i et $j \neq i, i+1(\mod n)$, $X_i \parallel X_j$ et $X_{i+1} \parallel X_j$ par définition d'une couronne donc $X_i \vee X_{i+1} \not\leq X_j$.

De plus, $X_j \parallel X'_i$ par définition d'une couronne et pour tout $i \leq n$, $X_i \leq X'_i$ et $X_{i+1} \leq X'_i$ donc $X_i \vee X_{i+1} \leq X'_i$. D'où $X_j \not\leq X_i \vee X_{i+1}$.

Donc $X_j \parallel X_i \vee X_{i+1}$. □

Une couronne quelconque implique donc l'existence d'une couronne de la forme $(X_0, X_0 \vee X_1, \ldots, X_{n-1}, X_{n-1} \vee X_0)$. Nous considérerons à partir de maintenant uniquement des couronnes de cette forme.

Proposition 4.4.2. *Soit $\mathcal{L} = (L, \leq)$ un treillis fini. Si $(L, \leq)$ est binarisable alors L est démontable.*

Démonstration. Montrons le résultat par récurrence.

Supposons que $(L, \leq)$ est un treillis binaire contenant une 3-couronne. Soit $(X_0, X_0 \vee X_1, X_1, X_1 \vee X_2, X_2, X_2 \vee X_0)$ une 3-couronne et $Y = \sup(X_0, X_1, X_2) = \sup(X_0 \vee X_1, X_1 \vee X_2, X_2 \vee X_0)$. Le treillis $(L, \leq)$ est binaire donc Y couvre au plus deux éléments Y' et Y'' et $\{X_i \leq Y' \cup Y''\} = \{X_0, X_1, X_2\}$. Y est le supremum de $X_0 \vee X_1, X_1 \vee X_2, X_2 \vee X_0$ et $Y' \prec Y, Y'' \prec Y$ donc, par le principe des tiroirs, nous pouvons supposer sans perte de généralité que $X_1 \vee X_2 \leq Y'$ et $X_2 \vee X_0 \leq Y'$. Donc $X_0 < Y'$ et $X_1 < Y'$ donc $X_0 \vee X_1 \leq Y'$ donc $Y' = Y$ ce qui est une contradiction.

Supposons à présent que tout treillis $(L, \leq)$ contenant une couronne de taille inférieure ou égale à $n-1$ (avec $n > 3$) n'est pas binaire. Soit $(L, \leq)$ un treillis binaire contenant une n-couronne. Soit $(X_0, X_0 \vee X_1, \ldots X_{n-1}, X_{n-1} \vee X_0)$ une couronne et $Y = \sup(X_0, \ldots X_{n-1})$. $(L, \leq)$ est binaire donc Y couvre au plus deux éléments Y' et Y''. $Y = \sup(X_0, \ldots, X_{n-1})$ donc

$$\begin{cases} 1 \leq |\{X_i \leq Y' \mid i \leq n-1\}| < n \\ 1 \leq |\{X_i \leq Y'' \mid i \leq n-1\}| < n. \end{cases}$$

Y est binaire donc

$$\{X_i \leq Y' \mid i \leq n-1\} \cup \{X_i \leq Y'' \mid i \leq n-1\} = \{X_i \mid 1 \leq i \leq n-1\}.$$

Nous pouvons donc supposer sans perte de généralité que $|\{X_i \leq Y' \mid i \leq n-1\}| \geq 2$. Supposons de plus que $X_0 \not\leq Y'$. Soit $j = \min\{i \leq n-1 \mid X_i \leq Y'\}$ et $j' = \max\{i \leq n-1 \mid X_i \leq Y'\}$. $(X_{j'}, X_{j'} \vee X_{j'+1(\mod n)}, \ldots, X_0, \ldots, X_j, Y')$ est une couronne de taille inférieure ou égale à $n-1$ et supérieure à 3. En effet, pour tout $i < j$ et pour tout $i > j'$, $X_i \not\leq Y'$. Par hypothèse de récurrence, le treillis n'est pas binaire.

Par définition du plongement d'un treillis dans un autre et d'une couronne, pour tout treillis non démontable $(L, \leq)$, si $(L, \leq)$ peut être plongé dans $(L', \leq)$ par une fonction f, alors $(X_0, X'_0, \ldots X_{n-1}, X'_{n-1})$ est une couronne de $(L, \leq)$ ssi $f(X_0), f(X'_0), \ldots, f(X_{n-1})$, $f(X'_{n-1})$ est une couronne de $(L', \leq)$. D'après le résultat précédent, $(L', \leq)$ n'est pas binaire donc $(L, \leq)$ n'est pas binarisable.

Dans cette partie de la preuve, nous avons utilisé uniquement un sens de la définition de treillis binaire. En effet, nous nous servons de l'argument "Y élément d'un treillis binaire $(L, \leq)$ implique Y couvre au maximum deux éléments", mais pas de l'argument "Y élément d'un treillis binaire $(L, \leq)$ implique Y est couvert au maximum par deux éléments". Si Y est couvert par 3 éléments ou plus, la même démonstration peut être appliqué au dual du treillis dans lequel Y couvre plus de 3 éléments.

□

La combinaison des Propositions 4.3.5 et 4.4.2 donne le résultat principal.

Théorème 4.4.3. *Soit $(L, \leq)$ un treillis fini. $(L, \leq)$ est binarisable ssi $(L, \leq)$ est démontable.*

Nous présenterons une preuve alternative de ce résultat en le considérant du point de vue des hypergraphes dans le Chapitre 5. La preuve de "$\mathcal{H}$ hypergraphe binarisable implique $\mathcal{H}$ totalement équilibré" (Proposition 5.6.1) sera identique à celle proposée ici tandis que l'autre implication utilisera une construction différente et plus générale.

Les treillis sans couronnes étant équivalents aux treillis binaires, ils semblent être une bonne généralisation des arbres de décision aux systèmes autorisant l'empiétance. On peut en effet aisément imaginer l'utilisation d'un treillis binaire comme d'un arbre de décision en passant d'un nœud à l'autre par une question du type "L'exemple considéré possède-t-il l'attribut X_i et/ou l'attribut X_j ?". De plus, la non-unicité de la solution de binarisation pour chaque élément permet d'envisager la création de systèmes de décision interactifs permettant à l'utilisateur d'influer sur le choix de la binarisation.

4.5 Application en analyse de concepts formels

Cette section consiste à appliquer le résultat de binarisation à des treillis dits *de concepts* dans le cadre de l'analyse de concepts formels.

4.5.1 Analyse de concepts formels

Cette sous-section vise à introduire les notions d'analyse de concepts formels nécessaires à l'application de notre algorithme.

Définition 4.5.1. Un *contexte formel* est un triplet $K = (G, M, I)$ avec G un ensemble d'objets, M un ensemble d'attributs et $I \subseteq G \times M$ une relation binaire.

Un contexte formel peut donc être représenté par une matrice. En effet, supposons que $G = (l_1, \ldots, l_n)$ et $M = (c_1, \ldots, c_m)$. Le contexte formel K est équivalent à la matrice binaire $n \times m$ $\mathcal{M}$ avec $\mathcal{M}_{i,j} = 1$ ssi $l_i I c_j$. Cette matrice peut être vue comme une base de données avec les lignes représentant les individus et les colonnes les attributs qu'ils peuvent avoir. La Table 4.1 donne un exemple de matrice associée à un contexte formel. Pour plus de lisibilité de la matrice, les 1 sont remplacés par des croix et les 0 par un vide.

On définit également deux opérateurs :
— pour $A \subseteq G : A^{\uparrow} = \{y \in M \mid \forall x \in A,\ xIy\}$
— pour $B \subseteq M : B^{\downarrow} = \{x \in G \mid \forall y \in B,\ xIy\}$.

Définition 4.5.2. Un *concept formel* associé à un contexte formel $K = (G, M, I)$ est une paire (A, B) avec :
— $A \subseteq G$, $B \subseteq M$
— $A^{\uparrow} = B$
— $B^{\downarrow} = A$.
A est appelé une *extension* et B une *intension*.

La Table 4.2 donne les concepts formels associés à la Table 4.1. On y trouve par exemple le concept $(\{l_2\}, \{c_6, c_7, c_8\})$, car l_2 possède les attributs c_6, c_7 et c_8, est le seul à les posséder tous les trois et n'en possède pas d'autres. De même, la paire $(\{l_2, l_3\}, \{c_6, c_8\})$ est un concept, car c_6 et c_8 sont les deux seuls attributs que l_2 et l_3 ont en commun et l_2 et l_3 sont les seuls à les posséder tous les deux.

Comme évoqué précédemment, un contexte formel peut être représenté par une matrice binaire. À l'inverse, toute matrice binaire de taille $n \times m$ représente un contexte formel $K_{\mathcal{M}} = (\{1, \ldots, n\}, \{1, \ldots, m\}, I_{\mathcal{M}})$ avec $I_{\mathcal{M}}$ tel que $i I_{\mathcal{M}} j$ ssi $\mathcal{M}_{i,j} = 1$. En notant l_i la ligne i de la matrice $\mathcal{M}$ et c_j la colonne j, on a [7] :
— la clôture par intersection de $\{c_1, \ldots, c_m, \{1, \ldots, n\}\}$ est égale à $\{A \subseteq G \mid A^{\uparrow\downarrow} = A\}$,
— la clôture par intersection de $\{l_1, \ldots, l_m, \{1, \ldots, m\}\}$ est égale à $\{B \subseteq M \mid B^{\uparrow\downarrow} = B\}$.
Il y a donc équivalence entre les contextes formels et les matrices binaires closes par intersection.

On appelle *treillis des extensions* le treillis des extensions de tous les concepts formels d'un contexte formel muni de la relation d'inclusion des ensembles. Par exemple, le treillis des extensions du contexte formel de la Table 4.1 est $(\{\{l_1\}, \{l_2\}, \{l_3\}, \{l_5\}, \{l_2, l_4\}, \{l_2, l_3\},$

	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
l_1			×	×	×			
l_2						×	×	×
l_3		×	×	×		×		×
l_4							×	×
l_5	×	×	×					

TABLE 4.1 – Exemple de matrice associée à un contexte formel

$(\emptyset, \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\})$
$(\{\boldsymbol{l_2}\}, \{c_6, c_7, c_8\})$
$(\{\boldsymbol{l_1}\}, \{c_3, c_4, \boldsymbol{c_5}\})$
$(\{\boldsymbol{l_3}\}, \{c_2, c_3, c_4, c_6, c_8\})$
$(\{\boldsymbol{l_5}\}, \{\boldsymbol{c_1}, c_2, c_3\})$
$(\{l_2, \boldsymbol{l_4}\}, \{\boldsymbol{c_7}, c_8\})$
$(\{l_2, l_3\}, \{\boldsymbol{c_6}, c_8\})$
$(\{l_1, l_3\}, \{c_3, \boldsymbol{c_4}\})$
$(\{l_3, l_5\}, \{\boldsymbol{c_2}, c_3\})$
$(\{l_2, l_3, l_4\}, \{\boldsymbol{c_8}\})$
$(\{l_1, l_3, l_5\}, \{\boldsymbol{c_3}\})$
$(\{l_1, l_2, l_3, l_4, l_5\}, \emptyset)$

TABLE 4.2 – Concepts associés à la matrice de contexte de la Table 4.1

$\{l_1, l_3\}, \{l_3, l_5\}, \{l_2, l_3, l_4\}, \{l_1, l_3, l_5\}, \{l_1, l_2, l_3, l_4, l_5\}\}, \subseteq)$. On définit symétriquement le *treillis des intensions* comme étant le treillis des intensions des concepts formels d'un contexte formel muni de la relation d'inclusion. Ces deux treillis sont le dual l'un de l'autre.

Définition 4.5.3. Le *treillis de concepts* $\mathcal{L} = (L, \leq)$ associé au contexte formel K est le treillis défini par L l'ensemble des concepts formels de K et la relation $(A_1, B_1) \leq (A_2, B_2)$ ssi $A_1 \subseteq A_2$ et $B_2 \subseteq B_1$.

Un concept (A, B) représente un objet l_i si A est la plus petite extension au sens de l'inclusion contenant l_i. Symétriquement, un concept (A, B) représente un attribut c_j si B est la plus petite intension au sens de l'inclusion contenant c_j. Les objets et attributs représentés par un concept sont indiqués en gras dans la Table 4.2.

La Figure 4.4 représente le diagramme de Hasse du treillis de concepts associé au contexte formel de la Table 4.1. Chaque nœud représente un concept (listé dans la Table 4.2). Les nœuds sont étiquetés par l'objet et/ou l'attribut qu'ils représentent. Si le concept représente un objet, il est représenté par un demi-cercle noir. S'il représente un attribut, il est représenté par un demi-cercle bleu. Le treillis de concepts d'un contexte formel peut être vu comme la fusion du treillis des intensions retourné et du treillis des extensions.

Les liens entre treillis de concepts et arbres de décision ont déjà été largement étudiés. Les treillis de concepts peuvent par exemple être considérés comme une famille d'arbres de décision superposés et ont été utilisés pour la construction d'arbres de décision [8]. Nous pouvons également citer le développement d'un modèle de classification hybride entre le treillis de concepts et l'arbre de décision : les treillis dichotomiques [9] [27]. De nombreuses méthodes de classification supervisée mais aussi non supervisée utilisant les

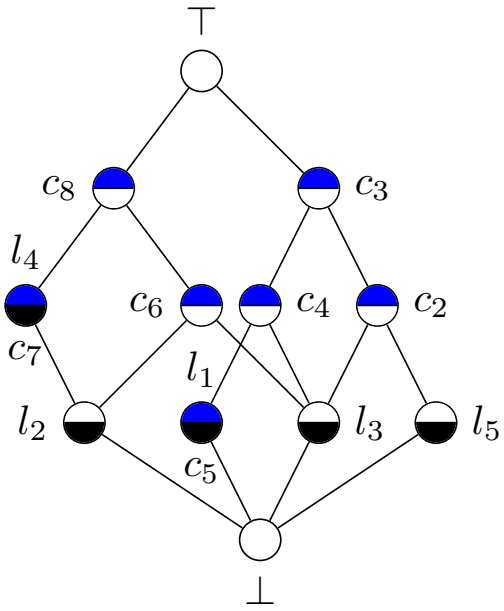


FIGURE 4.4 – Treillis de concepts associé au contexte formel de la Table 4.1

treillis de concepts peuvent être trouvées dans [37].

4.5.2 Binarisation de treillis de concepts

Nous allons maintenant appliquer l'Algorithme 10 de binarisation d'un système de classes sans couronnes à l'analyse de concepts formels. Nous pouvons l'appliquer sur les systèmes de classes associés respectivement au treillis des intensions et au treillis des extensions d'un ensemble de concepts.

Notre algorithme binarise dans un premier temps le système de classes associé au treillis initial puis celui associé à son dual. Il s'agit donc de binariser le treillis des extensions puis le treillis des intensions puisque l'un est le dual de l'autre. Appliquer notre algorithme d'abord sur le treillis des extensions par exemple revient à modifier le treillis des extensions pour que tous les éléments ne couvrent que deux éléments maximum (ce qui ajoute de nouveaux attributs au treillis) puis à modifier le treillis des intensions pour que chaque élément ne soit couvert que par deux éléments maximum (ce qui va ajouter de nouveaux objets au treillis).

La Table 4.3 donne un petit contexte formel décrivant des animaux sur lequel appliquer notre algorithme. Les concepts formels associés à ce contexte sont donnés dans la Table 4.4. La représentation du diagramme de Hasse du treillis de concepts associé à ce contexte formel (Figure 4.5) permet de voir aisément les concepts non binaires. Ici, le concept représentant le canard est couvert par trois concepts et le concept représentant

	écailles	dents	nage	vole	graines	plumes	air
saumon	×		×				
requin		×	×				
barracuda	×	×	×				
crapaud			×				×
crocodile		×	×				×
aigle				×		×	×
autruche					×	×	×
canard			×	×	×	×	×

TABLE 4.3 – Matrice binaire d'un ensemble d'animaux

({∅}, {écailles, dents, nage, vole, graines, plumes, air})
({*crocodile*}, {dents, nage, air})
({*canard*}, {nage, vole, graines, plumes, air})
({*barracuda*}, {écailles, dents, nage})
({*autruche*, canard}, {*graines*, plumes, air})
({*saumon*, barracuda}, {*écailles*, nage})
({*aigle*, canard}, {*vole*, plumes, air})
({*requin*, barracuda, crocodile}, {*dents*, nage})
({*crapaud*, crocodile, canard}, {air, nage})
({aigle, autruche, canard}, {*plumes*, air})
({crapaud, crocodile, aigle, autruche, canard}, {*air*})
({saumon, requin, barracuda, crapaud, crocodile}, {*nage*})

TABLE 4.4 – Concepts formels du contexte formel des animaux

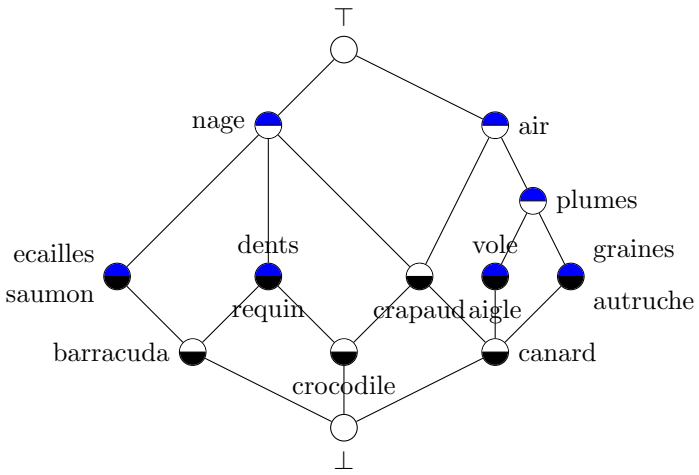


FIGURE 4.5 – Diagramme de Hasse du treillis associé à la Table 4.3

l'attribut *nage* couvre trois concepts. Cette représentation montre également que le treillis est démontable et donc que notre algorithme peut s'y appliquer.

Une binarisation du treillis des extensions (sans appliquer l'algorithme sur son dual) de ce treillis de concepts donne le diagramme de Hasse représenté sur la Figure 4.6a. Sur cette figure, le nouveau nœud latent (un nouvel attribut) est représenté par un rectangle. Le diagramme de Hasse associé à une binarisation des extensions et des intensions est représenté sur la Figure 4.6b et le contexte formel qui lui est associé est indiqué dans la Table 4.5. Le processus ajoute deux nouveaux concepts formels : un associé à un nouvel objet *obj* (qu'on peut interpréter comme l'existence d'un oiseau qui mange des graines, par exemple un canari) et un associé à un nouvel attribut *att* (qui suggère l'existence d'un attribut permettant de distinguer le saumon, le requin, le barracuda et le crocodile du crapaud et du canard) et modifie les autres concepts formels en utilisant ce nouvel attribut et ce nouvel objet.

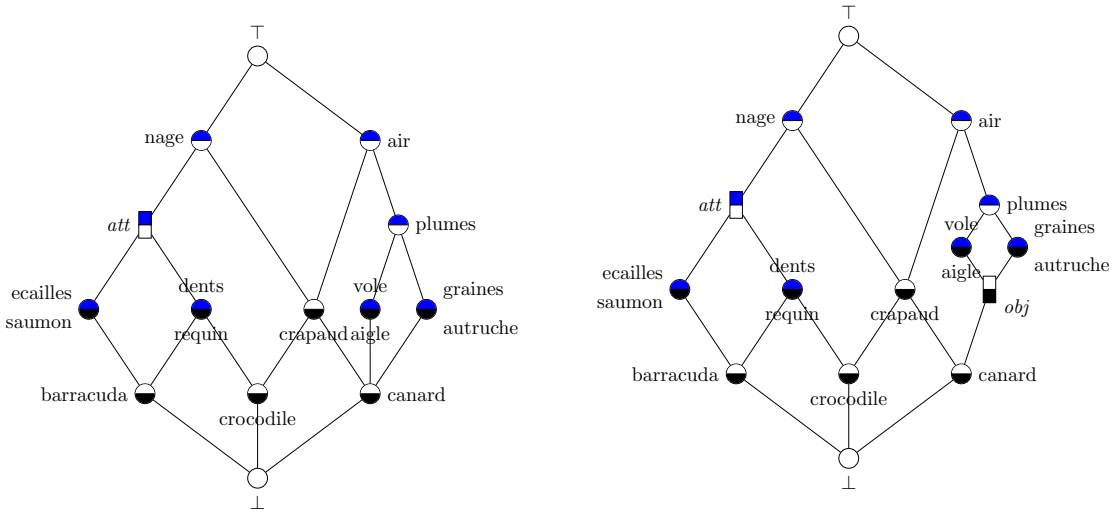
La binarisation ne donne pas une solution unique. En effet, *att* aurait pu être associé à un nouveau concept $\{(requin, crapaud, barracuda, crocodile, canard), (dents, att, nage)\}$ à la place d'être associé à $\{(saumon, requin, barracuda, crocodile), (ecailles, att, nage)\}$, car les deux intersections concernées sont incomparables.

4.6 Conclusion

Ce chapitre présente un algorithme simple (Section 4.3) pour transformer un système de classes sans couronnes non binaire en un système de classes binaire en ajoutant le moins d'éléments possible. Les treillis et systèmes de classes étant équivalents, cet algorithme se généralise aux treillis. La construction permet de prouver l'équivalence entre les treillis binaires et les treillis démontables (Section 4.4). Cette équivalence constitue

	écailles	dents	**att**	nage	vole	graines	plumes	air
saumon	×		×	×				
requin		×	×	×				
barracuda	×	×	×	×				
crapaud				×				×
crocodile		×	×	×				×
aigle					×		×	×
autruche						×	×	×
obj					×	×	×	×
canard				×	×	×	×	×

TABLE 4.5 – Contexte formel binarisé



(a) Binarisation des extensions

(b) Binarisation des extensions et des intensions

FIGURE 4.6 – Diagramme de Hasse de la binarisation du treillis associé à la Table 4.4

un résultat théorique intéressant mais fait également des treillis démontables le parfait candidat pour l'extension d'idées issues de l'apprentissage automatique développées dans les arbres de décision à des systèmes de décision plus généraux et autorisant notamment l'empiétance des classes. De plus, la non-unicité de la solution de binarisation proposée permet d'envisager la création de systèmes de décision interactifs. Enfin, la propriété théorique d'*intersection maximale* présentée et utilisée pour prouver la correction de l'algorithme s'interprète aisément en pratique. En effet, créer de nouvelles classes rassemblant les groupes qui se ressemblent le plus paraît intuitif et cohérent avec les objectifs de la classification.

Nous avons également présenté une application de la binarisation de treillis à l'analyse de concepts formels dans la Section 4.5. Cette méthode permet de produire un modèle simple pour l'interprétation des données et permet de rassembler des individus proches ou de suggérer l'existence d'individus non observés.

Chapitre 5

Caractérisation des hypergraphes binaires

5.1 Introduction

Comme détaillé dans la Section 1.1.1, les hypergraphes totalement équilibrés ont de nombreux liens avec les arbres : ils sont définis comme étant les hypergraphes sans cycles spéciaux [35], ils peuvent être caractérisés par une séquence d'arbres [32] et sont les hyperarbres tels que tout sous hypergraphe induit est un hyperarbre [32].

De plus, comme introduit dans la Section 1.2.2 pour le cas des arbres de décision, en apprentissage automatique, les systèmes de décision binaires sont préférés aux systèmes plus complexes, car ils sont plus faciles à comprendre et à interpréter tout en contenant la même information. Ce chapitre introduit les hypergraphes binaires, strictement équivalents aux treillis binaires définis dans la Section 4.2. Ces hypergraphes sont tels que, en considérant l'ordre partiel d'inclusion de leurs hyperarêtes, chaque hyperarête couvre au plus deux hyperarêtes et est couverte par au plus deux hyperarêtes. Nous introduisons également la notion d'hypergraphe binarisable de manière identique à celle de treillis binarisable également donnée dans la Section 4.2 : un hypergraphe est binarisable s'il peut être plongé dans un hypergraphe binaire. Nous cherchons alors à caractériser ces hypergraphes particuliers par une construction semblable à celle proposée par Lehel [32] pour les hypergraphes totalement équilibrés.

En théorie, nous étendons la notion de binarité aux hypergraphes en définissant les hypergraphes binaires ainsi que les hypergraphes binarisables et nous proposons une caractérisation des hypergraphes binaires clos par intersection par une séquence d'arbres mixtes. Cette caractérisation est similaire à celle proposée par Lehel [32] pour les hypergraphes totalement équilibrés. Notre algorithme de construction, de complexité $\mathcal{O}(n^3)$, permet de prouver qu'un hypergraphe clos totalement équilibré est binarisable et fournit donc une preuve alternative à celle proposée dans la Section 4.3 pour montrer que les treillis sans couronnes sont binarisables. Nous prouvons ainsi l'équivalence entre les

hypergraphes clos totalement équilibrés et les hypergraphes clos binarisables.

En pratique, notre algorithme peut être utilisé pour approximer un hypergraphe clos par intersection, qu'il soit totalement équilibré ou non, par un hypergraphe binaire, ce qui permet l'utilisation pour l'approximation d'un modèle quelconque par un modèle totalement équilibré. Le modèle permet de plus une représentation de l'hypergraphe à un moment donné de sa construction, comme une photo des liens entre les différentes hyperarêtes.

Ce chapitre est organisé comme suit. La Section 5.2 définit de manière formelle les hypergraphes binaires et binarisables. Ensuite, la Section 5.3 donne un algorithme de génération des hypergraphes binaires puis la Section 5.4 montre qu'un cas particulier de l'algorithme présenté peut être utilisé pour construire n'importe quel hypergraphe binaire donné, avant d'expliciter la caractérisation des hypergraphes clos et binaires proposée. Enfin, nous proposons d'utiliser l'algorithme de construction présenté pour approximer tout hypergraphe totalement équilibré par un hypergraphe binaire clos par intersection (Section 5.5), terminons la preuve d'équivalence entre les hypergraphes binarisables et les hypergraphes totalement équilibrés (Section 5.6) et étendons l'algorithme à l'approximation par un hypergraphe clos binaire de tout hypergraphe clos (Section 5.7).

5.2 Hypergraphes binaires

Nous définissons la notion d'hypergraphe binaire, équivalente à la notion de treillis binaire donnée dans la Section 4.2.

Définition 5.2.1. Un hypergraphe $\mathcal{H} = (V, E)$ est binaire ssi, pour tout $Y \in E$:

$$\begin{cases} |\{X \in E \mid X \prec Y \}| \leq 2 & (5.1) \\ |\{Z \in E \mid Y \prec Z \}| \leq 2 & (5.2) \end{cases}$$

Si seule la condition 5.1 est satisfaite, $\mathcal{H}$ est dit *couverture-binaire*.

En pratique, comme dans le Chapitre 4, nous exclurons $\emptyset$ de la définition et lui permettrons d'être couvert par plus de deux éléments.

Nous pouvons également définir la notion d'hypergraphe binarisable correspondant à celle de treillis binarisable introduite dans la Section 4.2.

Définition 5.2.2. Un hypergraphe $\mathcal{H} = (V, E)$ est *binarisable* s'il existe un hypergraphe binaire $\mathcal{H}' = (V', E')$ et $f : E \to E'$ tels que pour tout $E_1, E_2 \in E$:

$$E_1 \subseteq E_2 \iff f(E_1) \subseteq f(E_2).$$

Ces définitions sont identiques à celles données pour les treillis dans le chapitre précédent et les Figures 4.1 et 4.2 peuvent être considérées comme les diagrammes de Hasse d'un hypergraphe non binaire et de deux de ses binarisations possibles.

Nous utiliserons dans la construction proposée la notion d'arbre mixte.

Définition 5.2.3. Un *arbre mixte* $T = (V, E, \overrightarrow{E})$ est un graphe mixte tel que le graphe sous-jacent non orienté $G = (V, E \cup \overrightarrow{E})$ (obtenu en remplaçant toutes les arêtes orientées par des arêtes non orientées) est un arbre.

Dans un tel arbre $T = (V, E, \overrightarrow{E})$, nous appellerons *chemin* une séquence de sommets $X_1, \ldots, X_k$ telle que pour tout $i < k$, $(X_i, X_{i+1}) \in E$ ou $(X_i, X_{i+1}) \in \overrightarrow{E}$ ou $(X_{i+1}, X_i) \in \overrightarrow{E}$ *i.e.* X_i et X_{i+1} sont voisins dans le graphe sous-jacent non orienté.

5.3 Algorithme de construction d'un hypergraphe binaire par une séquence d'arbres

Nous proposons dans un premier temps un algorithme permettant de générer des hypergraphes clos couverture-binaires. En démarrant d'un ensemble de sommets V, on génère une séquence d'arbres mixtes $\mathcal{T} = (T_0, \ldots, T_n)$ avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$ telle que l'hypergraphe $\mathcal{H} = (V, E)$ avec $E = \bigcup_{0 \leq i \leq n} V_i$ est couverture-binaire et clos. L'Algorithme 11 détaille la procédure basée sur l'idée de contracter une arête (X, Y) de T_i pour créer un nouveau sommet $X \cup Y$ dans T_{i+1}. La Figure 5.1 donne un exemple de séquence d'arbres mixtes obtenue par l'Algorithme 11. Cette séquence d'arbres mixtes est associée à l'hypergraphe $\mathcal{H} = (\{1,2,3,4,5,6\}, \{\{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{4,5\},\{1,2\},$ $\{5,6\},\{3,4,5\},\{4,5,6\},\{1,2,3,4,5\},\{3,4,5,6\},\{1,2,3,4,5,6\}\})$.

Afin d'alléger le texte, nous dirons qu'un ensemble de sommets est une *partie connexe* d'un arbre mixte si le sous-graphe induit du graphe non orienté sous-jacent sur cet ensemble de sommets est connexe. Par exemple, $\{\{4,5\},\{5,6\},\{3,4,5\}\}$ est une partie connexe de l'arbre T_4 de la Figure 5.1.

Chaque arbre de la construction peut de plus être vu comme une coupe du diagramme de Hasse de l'hypergraphe. La Figure 5.2 illustre cette idée en représentant toutes les coupes correspondant aux arbres de la construction de la Figure 5.1. La coupe la plus basse correspond à T_0 et la plus haute à T_7. Les arbres montrent donc une représentation de chaque étage et une partition de plus en plus fine en partant du dernier arbre.

Comme le montrera le Théorème 5.3.7, l'hypergraphe obtenu par cette construction est clos par intersection binaire. Il est donc totalement équilibré (Théorème 4.4.3) et contient ainsi au maximum v^2 hyperarêtes [4] avec v le nombre de sommets. L'algorithme construit donc au maximum v^2 arbres. Chaque étape de la construction consiste à déplacer aléatoirement un certain nombre de voisins d'un nœud choisi. Ce nombre est borné par v. L'algorithme s'exécute donc en $\mathcal{O}(v^3)$.

Nous montrons dans un premier temps certaines propriétés des arbres mixtes construits qui seront utiles à nos résultats.

Lemme 5.3.1. *Soit $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 11. Pour tout $0 \leq i \leq n$:*

Algorithme 11 : Génération d'un hypergraphe couverture-binaire

Données : V un ensemble d'éléments
Résultat : $\mathcal{H} = (V, E)$ un hypergraphe couverture-binaire

$V_0 = V$
Choisir E_0 un ensemble d'arêtes formant un arbre sur V_0
$\overrightarrow{E_0} = \emptyset$
$T_0 = (V_0, E_0, \overrightarrow{E_0})$
$i = 0$
tant que $|V_i| > 1$ **faire**
  Choisir $(X, Y) \in E_i$
  $V_{i+1} = V_i \cup \{X \cup Y\}$
  $E_{i+1} = E_i \backslash \{(X, Y)\}$
  $\overrightarrow{E_{i+1}} = \overrightarrow{E_i} \cup \{(X, X \cup Y)\} \cup \{(Y, X \cup Y)\}$
  pour $Z \in \{X, Y\}$ **faire**
    si $\{Z' \in V_{i+1} \mid (Z, Z') \in E_{i+1}\} = \emptyset$ **alors**
      supprime = Vrai
    sinon
      supprime = `booleen_hasard` ()
    fin
    si *supprime* **alors**
      pour $(U, Z) \in E_i$ **faire**
        $E_{i+1} = E_{i+1} \cup \{(U, X \cup Y)\} \backslash \{(U, Z)\}$
      fin
      pour $(U, Z) \in \overrightarrow{E_i}$ **faire**
        $\overrightarrow{E_{i+1}} = \overrightarrow{E_{i+1}} \cup \{(U, X \cup Y)\} \backslash \{(U, Z)\}$
      fin
      $E_{i+1} = E_{i+1} \cup$ `arbre_elements_connexes`$(\{U \in V_i \mid (Z, U) \in \overrightarrow{E_{i+1}}\})$
      $\overrightarrow{E_{i+1}} = \overrightarrow{E_{i+1}} \backslash \{(Z, U) \in \overrightarrow{E_i}\}$
      $V_{i+1} = V_{i+1} \backslash \{Z\}$
    sinon
      Choisir $\mathcal{N}_Z \subsetneq \{U \in V_i \mid (U, Z) \in E_i\}$ au hasard
      pour $U \in \mathcal{N}_Z$ **faire**
        $E_{i+1} = E_{i+1} \cup \{(U, X \cup Y)\} \backslash \{(U, Z)\}$
      fin
    fin
    $T_{i+1} = (V_{i+1}, E_{i+1}, \overrightarrow{E_{i+1}})$
    $i = i + 1$
  fin
fin
retourner $\mathcal{H} = (V_0, \{U \in V_j, 0 \leq j \leq i\})$

Avec `booleen_hasard`() une fonction qui renvoie un booléen au hasard et `arbre_elements_connexes`(S) qui renvoie un arbre $T = (S, E)$ tel que pour tout $x \in \bigcup_{s \in S} s, \{s \in S \mid x \in s\}$ est une partie connexe de T.

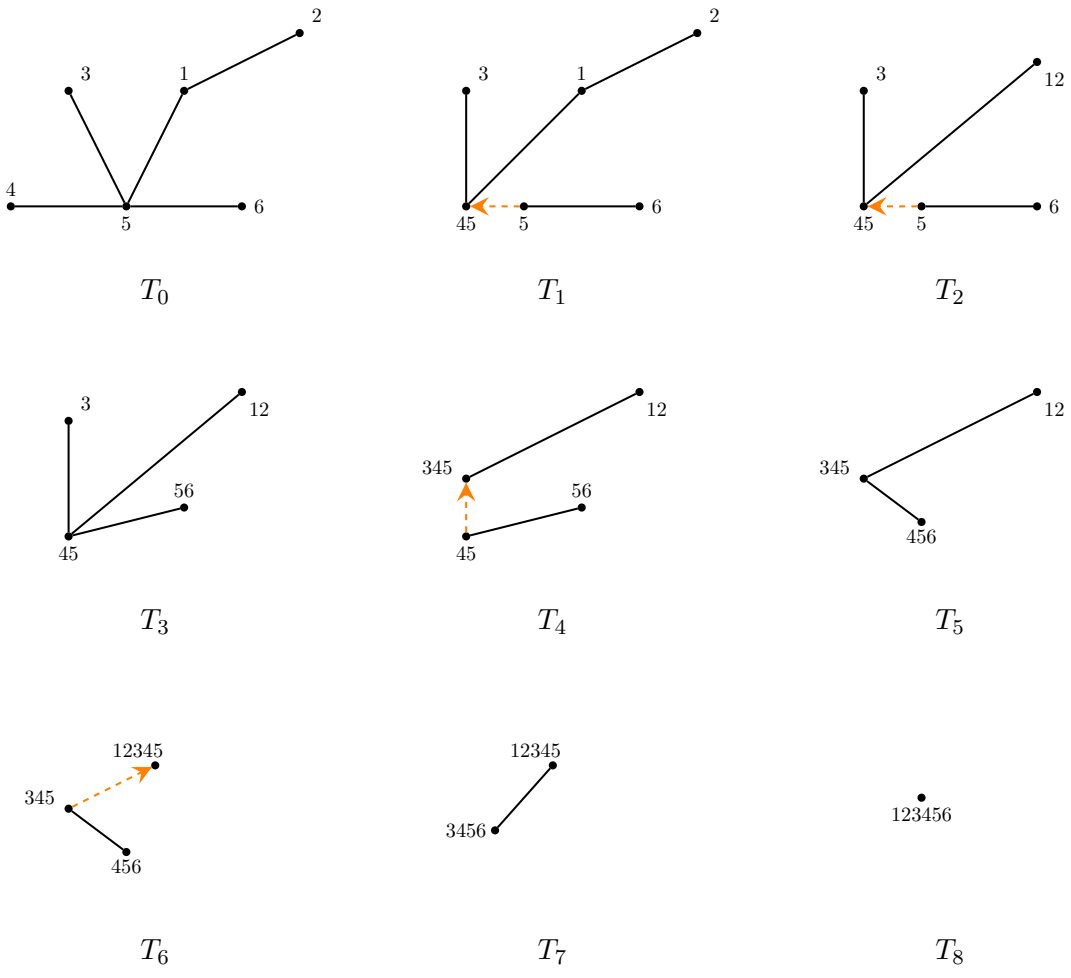


FIGURE 5.1 – Exemple d'une séquence d'arbres mixtes obtenue par l'Algorithme 11

Les arêtes orientées sont représentées par des flèches pointillées oranges.

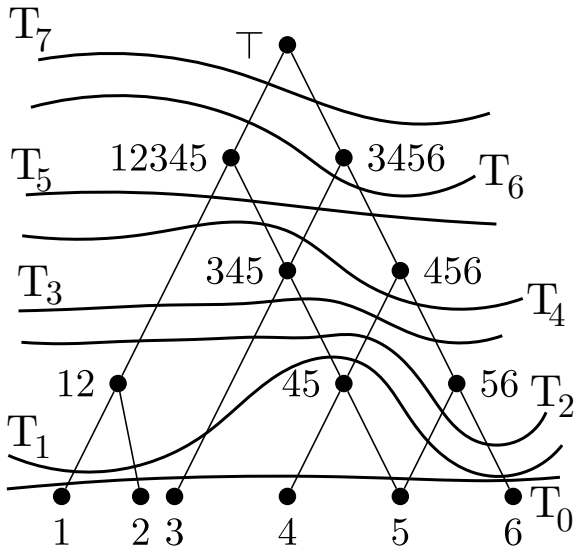


FIGURE 5.2 – Exemples de coupes du diagramme de Hasse de l'hypergraphe associé aux arbres de la Figure 5.1

(i) $\forall x \in V_0, \{X \in V_i \mid x \in X\}$ *est une partie connexe du graphe non orienté sous-jacent de* T_i *;*

(ii) $(X, Y) \in \overrightarrow{E_i} \implies X \subsetneq Y$ *;*

(iii) $(X, Y) \in E_i \implies X \parallel Y$.

Démonstration. On montre les trois résultats par récurrence. Pour T_0 :

(i) V_0 est un ensemble de singletons donc la propriété est vérifiée.

(ii) Par construction $\overrightarrow{E_0} = \emptyset$.

(iii) Les éléments de V_0 sont uniquement des singletons donc la propriété est vraie.

Supposons que les trois propriétés sont vérifiées pour un i donné et que $V_{i+1} \backslash V_i = \{X \cup Y\}$ *i.e.* on crée le sommet $X \cup Y$ dans T_{i+1} en contractant l'arête $(X, Y) \in E_i$. Montrons qu'elles sont vérifiées pour $i + 1$.

(i) Soit $u \in V_0$. Par hypothèse de récurrence (i), $\{U \in V_i \mid u \in U\}$ est une partie connexe de T_i.

Si $u \notin X$ et $u \notin Y$, alors, $\{U \in V_{i+1} \mid u \in U\} = \{U \in V_i \mid u \in U\}$ et les arêtes sont inchangées par construction. Donc $\{U \in V_{i+1} \mid u \in U\}$ est une partie connexe de T_{i+1}.

Si $u \in X$ (et symétriquement pour $u \in Y$), deux cas apparaissent :

— Si $X \in V_{i+1}$, pour tout $Z \in V_i$ tel que $u \in Z$, si $(Z, X) \in E_i$, par construction, soit $(Z, X) \in E_{i+1}$ et la connexité est préservée soit $(Z, X \cup Y) \in E_{i+1}$ et la connexité est également préservée car $(X, X \cup Y) \in \overrightarrow{E_i}$ et $u \in X \cup Y$. Pour tout $Z \in V_i$ tel que $u \in Z$ et $(Z, X) \notin E_i$ ou $(Z, X) \in \overrightarrow{E_i}$ ou $(X, Z) \in \overrightarrow{E_i}$, les arêtes sont inchangées donc la connexité est préservée.

— Si $X \notin V_{i+1}$, par construction l'algorithme construit un arbre avec pour sommets les voisins orientés de X tel que les sommets contenant un élément particulier forment une partie connexe. Cette construction est toujours possible. En effet, supposons qu'il est impossible de construire un tel arbre. Dans ce cas, il existe $(Z_1, \dots Z_k)$ tel que pour tout $1 \leq j \leq k$, $(X, Z_j) \in \overrightarrow{E_i}$ et pour tout $2 \leq j \leq k$, il existe v_j tel que $v_j \in Z_{j-1}, v_j \in Z_j$ et pour tout $j' \neq j-1, j, v_j \notin Z_{j'}$ et il existe $v_1 \in Z_1$ tel que $v_1 \in Z_k$. Par hypothèse de récurrence (i), tous ces éléments sont dans des parties connexes de T_i donc pour tout $1 \leq j \leq k, v_j \in X$. Pour tout j, on a $(X, Z_j) \in \overrightarrow{E_i}$ donc par hypothèse de récurrence (ii), $X \subsetneq Z_j$. Donc pour tout j, j', $v_j \in Z_{j'}$ ce qui est une contradiction.

(ii) S'il existe $Z, Z' \in V_{i+1}$ tels que $(Z, Z') \in \overrightarrow{E_{i+1}}$ mais $(Z, Z') \notin \overrightarrow{E_i}$, par construction, soit $Z' = X \cup Y$ et $Z = X$ soit $Z' = X \cup Y$ et $(Z, X) \in \overrightarrow{E_i}$. Dans le premier cas, par hypothèse de récurrence (iii) $X \neq X \cup Y$ et le résultat est bien vérifié. Dans le second cas, $(Z, X) \in \overrightarrow{E_i}$ donc par hypothèse de récurrence (ii), $Z \subsetneq X$ d'où $Z \subsetneq X \cup Y$.

(iii) Les nouvelles arêtes dans E_i peuvent être de deux types :
— $(Z, X \cup Y) \in E_{i+1}$ avec $Z \in V_i \cap V_{i+1}$ et $(Z, X) \in E_i$. Par hypothèse de récurrence (iii), il existe $z \in Z \backslash X$ et $x \in X \backslash Z$. Cependant, $(X, Y) \in E_i$ et T_i est un arbre donc par hypothèse de récurrence (i), $z \notin Y$. D'où $z \in Z \backslash (X \cup Y)$. De plus, $x \in X \backslash Z$ donc $x \in (X \cup Y) \backslash Z$. Donc $Z \parallel X \cup Y$.
— $(Z, Z') \in E_{i+1}$ quand $X \notin V_{i+1}$ avec $(X, Z) \in \overrightarrow{E_i}$ et $(X, Z') \in \overrightarrow{E_i}$ ou $Z' = X \cup Y$. Par hypothèse de récurrence (ii), $X \subsetneq Z$ et $X \subsetneq Z'$ donc il existe $z \in Z \backslash X$ et $z' \in Z' \backslash X$. Par hypothèse de récurrence (i) sur T_i, $z \notin Z'$ et $z' \notin Z$ d'où le résultat.

□

Ces propriétés peuvent être observées sur l'exemple de la Figure 5.1. Par exemple, l'objet 5 sur l'arbre T_4 appartient aux sommets $\{3, 4, 5\}, \{4, 5\}$ et $\{5, 6\}$ qui forment une partie connexe. De plus, toutes les arêtes non orientées de cet exemple sont bien entre deux sommets incomparables et toutes les arêtes orientées correspondent à une inclusion d'un sommet dans l'autre.

Nous montrons à présent que toute hyperarête U de l'hypergraphe $\mathcal{H}$ obtenu par l'Algorithme 11 est une partie connexe des arbres construits. Par cela nous entendons que l'ensemble des sommets d'un arbre T_i de la séquence d'arbres inclus dans l'hyperarête U et noté $T_i^U = \{U' \in V_i \mid U' \subseteq U\}$ est une partie connexe de T_i.

Lemme 5.3.2. *Soit $\mathcal{T} = (T_0, \dots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 11. Pour tout $0 \leq i \leq n$, pour tout $U \in V_i$ et pour tout $j < i$, le sous-graphe de T_j induit par $T_j^U = \{U' \in V_j \mid U' \subseteq U\}$ est connexe et $\bigcup_{S \in T_j^U} S = U$.*

Démonstration. Nous montrons dans un premier temps le résultat préliminaire suivant par récurrence :

$$\forall i, \forall (X,Y) \in E_{i+1}, X \cap Y = \emptyset \implies \forall j < i+1, \exists (X',Y') \in E_j, X' \subseteq X, Y' \subseteq Y.$$

De V_0 à V_1, un seul sommet de la forme $X \cup Y$ est ajouté avec $X \in V_0, Y \in V_0$. Pour tout $U \in V_1$, si $(X \cup Y, U) \in E_1$, par construction $(U, X) \in E_0$ ou $(U, Y) \in E_0$ donc le résultat est vérifié.

Supposons à présent que de T_i à T_{i+1} le sommet $X \cup Y$ est ajouté. Les seules arêtes dans E_{i+1} mais pas dans E_i sont soit de la forme $(U, X \cup Y)$ avec $(U, X) \in E_i$, soit de la forme (Z_j, Z_k) tel que $(X, Z_j) \in \overrightarrow{E_i}$ et $(X, Z_k) \in \overrightarrow{E_i}$ ou $Z_k = X \cup Y$. Dans le premier cas, $(X \cup Y) \cap U = \emptyset$ implique $X \cap U = \emptyset$. $(X, U) \in E_i$ vérifie le résultat pour $(X \cup Y, U) \in E_{i+1}$ et par hypothèse de récurrence sur $(X, U) \in T_i$, il existe $(X', U') \in E_j, X' \subseteq X \subseteq X \cup Y, U' \subseteq U$ pour tout $j < i$. Dans le second cas, d'après le Lemme 5.3.1(ii), $X \subsetneq Z_i$ et $X \subsetneq Z_j$ donc $Z_i \cap Z_j \neq \emptyset$.

Montrons maintenant le résultat principal par récurrence. Nous noterons T_k^U l'ensemble $\{U' \in V_k \mid U' \subseteq U\}$ et dirons que T_k^U est une partie connexe de T_k si le sous-graphe non orienté sous-jacent de T_k induit par T_k^U est connexe. Supposons que pour tout $j \leq i$, pour tout $U \in V_j$ et pour tout $k < j, T_k^U$ est une partie connexe de T_k et $\bigcup_{S \in T_k^U} S = U$. Dans T_{i+1}, le seul nouveau sommet est de la forme $X \cup Y$ donc pour tout $U \in V_{i+1}$, si $U \neq X \cup Y$ alors $U \in V_i$. Le résultat est donc vérifié par hypothèse de récurrence. Pour $X \cup Y$, dans T_i, $(X, Y) \in E_i$ et T_i^X, T_i^Y sont des parties connexes de T_i par hypothèse de récurrence donc $T_i^{X \cup Y}$ est une partie connexe de T_i. Par hypothèse de récurrence sur $X \in V_i$ et $Y \in V_i$, pour tout arbre T_j avec $j < i$, T_j^X et T_j^Y sont des parties connexes de T_j et $\bigcup_{S \in T_j^X} S = X$, $\bigcup_{S \in T_j^Y} S = Y$. Si $X \cap Y \neq \emptyset$, $T_j^{X \cup Y}$ est une partie connexe de T_j car les deux parties connexes ont une intersection non vide. Si $X \cap Y = \emptyset$, en appliquant le résultat préliminaire sur $(X, Y) \in E_i$, pour tout $T_j, j < i$, il existe $X' \in V_j, Y' \in V_j$ tel que $X' \subseteq X, Y' \subseteq Y$ et $(X', Y') \in E_j$ donc les deux parties connexes sont liées par une unique arête, ce qui donne le résultat. □

Prenons par exemple le sommet $\{1, 2, 3, 4, 5\}$ de l'arbre T_6 de la Figure 5.1. Dans T_5, les seuls sommets contenus dans cet ensemble sont $\{3, 4, 5\}$ et $\{1, 2\}$ qui forment une partie connexe. De même dans T_0, les sommets $\{1\}, \{2\}, \{3\}, \{4\}$ et $\{5\}$ forment une partie connexe. La même chose peut être observée pour tous les autres arbres.

Propriété 5.3.3. *Soit $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 11. Soit $i \leq n-1$. Pour tout $i' > i$:*

$$\begin{cases} V_{i+1} \backslash V_i \parallel V_{i'+1} \backslash V_{i'} \\ ou \\ V_{i+1} \backslash V_i \subsetneq V_{i'+1} \backslash V_{i'} \end{cases}$$

Démonstration. Soit $i \in \{0, n-1\}$. Soient $X, Y \in V_i$ tels que $V_{i+1} \backslash V_i = \{X \cup Y\}$.

Supposons qu'il existe $Z \in \bigcup_{0 \leq i \leq n} V_i$ tel que $Z \subsetneq X \cup Y$ et $\arg\min_{0 \leq i \leq n} Z \in V_i > i+1$ *i.e.* Z apparaît après $X \cup Y$ dans la construction mais est inclus dans $X \cup Y$. Par construction $(X, Y) \in E_i$ et T_i^Z est une partie connexe de T_i par le Lemme 5.3.2. $Z \subsetneq X \cup Y$ donc soit $X \notin T_i^Z$ soit $Y \notin T_i^Z$. Supposons sans perte de généralité que $Y \notin T_i^Z$. Par les Lemmes 5.3.1(i) et 5.3.2, pour tout $y \in Y \backslash X, y \notin Z$. Or $Z \subsetneq X \cup Y$ donc $Z \subseteq X$. Donc d'après le Lemme 5.3.1, T_i^Z est un chemin d'arêtes orientées. Donc par le Lemme 5.3.2, $Z \in V_i$ ce qui est une contradiction.

Soit i tel que $V_{i+1} \backslash V_i = \{X \cup Y\}$. Montrons que $Z = X \cup Y$ ne peut pas être ajouté à nouveau à la construction plus tard *i.e.* il n'existe aucun $i' > i$ tel que $V_{i'+1} \backslash V_{i'} = \{Z\}$. Par le Lemme 5.3.2, T_i^Z est une partie connexe de T_i. Or $X \subsetneq Z, Y \subsetneq Z$ et $X \cup Y = Z$ donc par le Lemme 5.3.1, $\{(U, U') \in T_i^Z \times T_i^Z \mid (U, U') \in E_i\} = \{(X, Y)\}$ et les autres arêtes dans T_i^Z sont des chemins d'arêtes orientées vers X ou vers Y. Par construction, T_{i+1}^Z est donc uniquement composé de chemins d'arêtes orientées vers $X \cup Y$. Par construction, cela est vrai pour tout $i' \geq i+1$. Aucune arête non orientée ne peut donc être contractée. □

Ce résultat montre que les sommets sont ajoutés dans les arbres mixtes en respectant un ordre partiel d'inclusion ce qui est aisément visible sur l'exemple de la Figure 5.1.

Théorème 5.3.4. *L'Algorithme 11 termine et le dernier arbre de la séquence est* $(\{V_0\}, \emptyset, \emptyset)$.

Démonstration. À chaque étape de la construction, un nouveau sommet de la forme $X \cup Y$ avec $X \in V_i, Y \in V_i, (X, Y) \in E_i$ est ajouté à V_{i+1}. D'après la Propriété 5.3.3, chaque sommet de la séquence d'arbres est ajouté à un arbre de la séquence exactement une fois. V_0 est un ensemble fini donc 2^{V_0} est fini et l'algorithme termine.

Par construction, si $X, X' \in V_i$ sont tels que $(X, X') \in \overrightarrow{E_i}$, il existe $Y \in V_i$ tel que $(X, Y) \in E_i$ (toutes les arêtes orientées $(X, X') \in \overrightarrow{E_i}$ telles qu'il n'existe aucun $Y \in V_i$ tel que $(X, Y) \in E_i$ sont supprimées). Donc pour tout $i \neq n, E_i \neq \emptyset$ donc l'algorithme termine sur un nœud unique. De plus, par construction, pour tout $x \in V_0$ et tout $1 \leq i \leq n$, il existe $U \in V_i$ tel que $x \in U$. □

Propriété 5.3.5. *Soit* $\mathcal{T} = (T_0, \ldots, T_n)$ *(avec* $T_i = (V_i, E_i, \overrightarrow{E_i})$ *pour tout* $0 \leq i \leq n$*) une séquence d'arbres mixtes obtenue par l'Algorithme 11. Soit* $0 \leq i \leq n$*. Pour toute séquence de sommets* $X_0, \ldots X_k \in V_i$ *telle que pour tout* $0 \leq j \leq k-1$*, soit* $(X_j, X_{j+1}) \in E_i$ *soit* $(X_j, X_{j+1}) \in \overrightarrow{E_i}$ *soit* $(X_{j+1}, X_j) \in \overrightarrow{E_i}$*,*

$$X_0 \cap X_k \subseteq X_0 \cap X_{k-1} \subseteq \cdots \subseteq X_0 \cap X_1.$$

Cette propriété est une conséquence immédiate du Lemme 5.3.1(i) car si $x \in X_i$ et $x \in X_j$ avec $i < j$, alors pour tout $i < k < j$, $x \in X_k$, car les graphes considérés sont

des arbres et le chemin entre deux sommets est donc unique. Cette formulation nous sera utile pour simplifier certaines preuves à venir.

Nous pouvons maintenant démontrer que l'hypergraphe généré est clos par intersection et couverture-binaire.

Lemme 5.3.6. *Soit $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 11. $\mathcal{S} = \{U \in V_i \mid 0 \leq i \leq n\}$ est clos par intersection.*

Démonstration. Montrons le résultat par récurrence. Soit $\mathcal{S}_i = \{U \in V_j \mid j \leq i\}$ pour tout $0 \leq i \leq n$. Pour tout $X, Y \in \mathcal{S}_0, X \cap Y = \emptyset$, car les éléments sont tous des singletons donc $\mathcal{S}_0$ est clos par intersection. Supposons que $\mathcal{S}_i$ est clos par intersection. Par construction, il existe $X, Y \in \mathcal{S}_i$ tels que $\mathcal{S}_{i+1} = \mathcal{S}_i \cup \{X \cup Y\}$. Montrons que pour tout $Z \in \mathcal{S}_i$, $Z \cap (X \cup Y) \in \mathcal{S}_i$:

- Si $Z \subseteq X \cup Y$ le résultat est évident.
- $X \cup Y \subseteq Z$ est impossible d'après la Propriété 5.3.3 car $X \cup Y \notin \mathcal{S}_i$ et $Z \in \mathcal{S}_i$.
- Si $Z \parallel X \cup Y$, d'après le Lemme 5.3.2, pour tout $j \leq i$, les ensembles de sommets contenant X, Y ou $X \cup Y$ sont des parties connexes de T_j. Soit $j \leq i$ tel que $Z \in V_j$. T_j est un arbre donc il existe $X_0 \in V_j$ tel que $X_0 \subseteq X$ et X_0 est sur le chemin de tout élément $X' \in T_i^X = \{U \in V_i \mid U \subseteq X\}$ à Z. Nous pouvons supposer sans perte de généralité que X_0 est sur le chemin de tout $Y' \in \{U \in V_i \mid U \subseteq Y\}$ à Z, car T_j est un arbre mixte et $T_j^{X \cup Y}$ est une partie connexe de T_j. D'après la Propriété 5.3.5, pour tout $Y' \in V_j$ tel que $Y' \subseteq Y$, $Y' \cap Z \subseteq X_0 \cap Z$ et $X_0 \cap Z \subseteq X \cap Z$ car $X_0 \subseteq X$. De plus, d'après le Lemme 5.3.2, $\bigcup_{Y' \in V_j, Y' \subseteq Y} Y' = Y$. Donc $Y \cap Z \subseteq X \cap Z$. Donc $(X \cup Y) \cap Z = X \cap Z$ et $X \cap Z \in \mathcal{S}_i$, car $\mathcal{S}_i$ est clos par intersection par hypothèse de récurrence.

□

Théorème 5.3.7. *Soit $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 11. $\mathcal{H} = (V_0, \{U \in V_i \mid 0 \leq i \leq n\})$ est un hypergraphe couverture-binaire et clos par intersection.*

Démonstration. D'après le Lemme 5.3.6, l'hypergraphe obtenu est clos par intersection. Soit $A \in \{U \in V_i \mid 0 \leq i \leq n\}$. Par construction, il existe $X, Y \in \{U \in V_i \mid 0 \leq i \leq n\}$ tels que $X \prec A, Y \prec A, A = X \cup Y$. Soit i tel que $A \in V_{i+1}, A \notin V_i$. Pour tout $Z \subsetneq A$, d'après la Propriété 5.3.3 il existe $j < i+1$ tel que $Z \in V_j$. Il y a alors deux cas pour T_i :

- si $Z \in V_i$, d'après la Propriété 5.3.5, comme T_i est un arbre mixte et $(X, Y) \in E_i$, soit $Z \cap X \subseteq Z \cap Y$ soit $Z \cap Y \subseteq Z \cap X$. Mais $Z \subsetneq X \cup Y$ et $Z \neq X, Z \neq Y$ par hypothèse donc $Z \subsetneq X$ ou $Z \subsetneq Y$ donc A est couverture-binaire.
- si $Z \notin V_i$, par construction, il existe $Z' \in V_i$ tel que $Z \subseteq Z'$. De manière semblable au cas précédent, en utilisant la Propriété 5.3.5 sur T_i, $Z' \cap X \subseteq Z' \cap Y$ ou l'inverse. Donc $Z' \subsetneq X$. Mais $Z \subsetneq Z'$ donc $Z \subsetneq X$ et A est couverture-binaire.

□

Il est possible de générer un hypergraphe binaire en effectuant un petit changement dans l'algorithme. La construction est telle que si $X \prec Y$ dans l'hypergraphe final, soit (X, Y) est une arête orientée d'un des arbres de la séquence, soit X a disparu des arbres au moment de la création de Y. Il suffit donc de modifier la ligne 15 de l'Algorithme 11 pour supprimer X dès qu'il crée sa deuxième arête sortante. Par exemple, la séquence d'arbres mixtes représentée sur la Figure 5.1 respecte cette condition et génère donc un hypergraphe binaire.

5.4 Caractérisation des hypergraphes binaires par une séquence d'arbres

Nous allons maintenant démontrer qu'un hypergraphe clos par intersection couverture-binaire donné peut être généré par une séquence d'arbres mixtes respectant la construction décrite dans l'Algorithme 11.

Ordre compatible

Nous définissons dans un premier temps la notion d'*ordre compatible* qui représente l'ordre dans lequel les hyperarêtes peuvent être ajoutées en tant que sommets des arbres mixtes de la construction.

Définition 5.4.1. Soit $\mathcal{H} = (V, E)$ un hypergraphe. Un ordre $\theta : E \to \{0, \ldots, |E| - 1\}$ est dit *compatible* avec $\mathcal{H}$ ssi θ est une fonction bijective et :

(i) $\forall S, T \in E, S \subsetneq T \implies \theta(S) < \theta(T)$

(ii) $\forall S, S', T, U \in E, S \prec T \prec U, S \prec S' \implies \theta(S') < \theta(U)$

Nous pouvons remarquer que la condition (i) de cette définition permet simplement d'ajouter les hyperarêtes de $\mathcal{H}$ en respectant un ordre partiel d'inclusion ce qui se justifie par la Propriété 5.3.3. La condition (ii) sera quant à elle utilisée pour démontrer la préservation de la connexité de $T_i^U = \{X \in V_i \mid X \subseteq U\}$ dans T_i pour tout $U \in E$ et pour tout i. Cette condition correspond, pour les arbres mixtes, à ne pas contracter une arête $(X, Y) \in E_i$ dans T_i s'il existe un sommet $Z \in V_i$ tel que $(Z, X) \in \overrightarrow{E_i}$ (ou $(Z, Y) \in \overrightarrow{E_i}$).

La Figure 5.3 représente le diagramme de Hasse de l'hypergraphe binaire correspondant à la séquence d'arbres de la Figure 5.1. Cet hypergraphe admet $\{4, 5\}, \{1, 2\}, \{5, 6\}, \{3, 4, 5\}, \{4, 5, 6\}, \{1, 2, 3, 4, 5\}, \{3, 4, 5, 6\}, \{1, 2, 3, 4, 5, 6\}$ pour ordre compatible (cet ordre est celui utilisé pour la construction par la séquence d'arbres de la Figure 5.1). De nombreux autres ordres sont compatibles avec cet hypergraphe. En effet, il est possible de remplacer la séquence $\{4, 5\}, \{1, 2\}$,

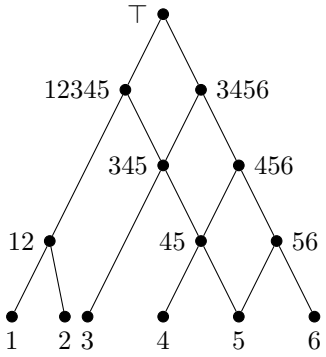


FIGURE 5.3 – Exemple d'hypergraphe binaire admettant pour ordre compatible $\{4,5\},\{1,2\},\ \{5,6\},\{3,4,5\},\{4,5,6\},\{1,2,3,4,5\},\{3,4,5,6\},\{1,2,3,4,5,6\}$

$\{5,6\}$ par $\{5,6\},\{4,5\},\{1,2\}$ par exemple. Par contre, afin de ne pas violer la condition (ii), $\{4,5,6\}$ sera toujours avant $\{1,2,3,4,5\}$.

Dans les démonstrations des résultats suivants, nous supposerons que θ est défini sur $\{U \in E \mid \forall v \in V, U \neq \{v\}\}$ afin de simplifier les résultats. En effet, dans ce cas, pour tout $U \in E$, $\theta(U) = i$ signifie que U n'est pas un sommet du $i^{\text{ème}}$ arbre mais est un sommet du $i+1^{\text{ème}}$.

Théorème 5.4.1. *Tout hypergraphe fini clos par intersection admet un ordre compatible.*

Démonstration. Soit $\mathcal{H} = (V, E)$ un hypergraphe totalement équilibré fini. Supposons qu'il n'existe pas d'ordre compatible avec $\mathcal{H}$. Soit $j \leq |E| - 1$ tel qu'il existe $\theta_j : E \to \{0, \ldots, j-1\}$ tel que θ_j est compatible avec $\mathcal{H}_j = (V, E_j)$ où $E_j = \{U \in E \mid \theta_j(U) \leq j\}$ mais tel qu'il n'existe aucun $U \in E$ tel que θ_{j+1} (défini par $\theta_{j+1}(U') = \theta_j(U)$ pour tout $U' \neq U$ et $\theta_{j+1}(U) = j+1$) est compatible avec $\mathcal{H}_{j+1}$. Pour simplifier les notations, si $U \in E \backslash E_j$, nous définissons $\theta_j(U) = -\infty$. Pour tout $U \in E \backslash E_j$:

— soit $\exists T \in E, T \prec U, \theta_j(T) = -\infty$ (condition (i) de la Définition 5.4.1 violée)

— soit $\exists S, S', T \in E, S \prec T \prec U, S \prec S', \theta_j(S) \neq -\infty, \theta_j(T) \neq -\infty, \theta_j(S') = -\infty$ (condition (ii) de la Définition 5.4.1 violée)

L'hypergraphe étant fini, la condition (ii) est violée au moins une fois ce qui signifie que pour tout $X_0^0, X_0^1, \ldots X_0^{l_0-2}$ tel que $X_0^{l_0-2} \prec X_0^{l_0-3} \prec \cdots \prec X_0^1 \prec X_0^0$ et $\theta_j(X_0^k) = -\infty$ pour tout $0 \leq k \leq l_0 - 2$, il existe $X_0^{l_0-1}$ et $X_0^{l_0}$ tels que $X_0^{l_0} \prec X_0^{l_0-1} \prec X_0^{l_0-2}$, $\theta_j(X_0^{l_0-1}) \neq -\infty, \theta_j(X_0^{l_0}) \neq -\infty$ et il existe X_1^0 tel que $X_0^{l_0} \prec X_1^0$ et $\theta_j(X_1^0) = -\infty$ (*i.e.* pour tout $0 \leq k \leq l_0 - 3$, X_0^k ne peut pas être ajouté à l'ordre car cela violerait la condition (i) et $X_1^0, X_0^{l_0}, X_0^{l_0-1}, X_0^{l_0-2}$ viole la condition (ii) donc $X_0^{l_0-2}$ ne peut pas être ajouté à l'ordre). Or $\theta_j(X_1^0) = -\infty$, ce qui permet de répéter le processus en partant de X_1^0.

Il existe $(X_0^0, X_0^1, \ldots, X_0^{l_0}), \ldots, (X_c^0, X_c^1, \ldots, X_c^{l_c})$ et X_{c+1}^0 tous distincts tels que :

— $\forall i \leq c, \forall k \leq l_c - 2, \theta_j(X_i^k) = -\infty$,

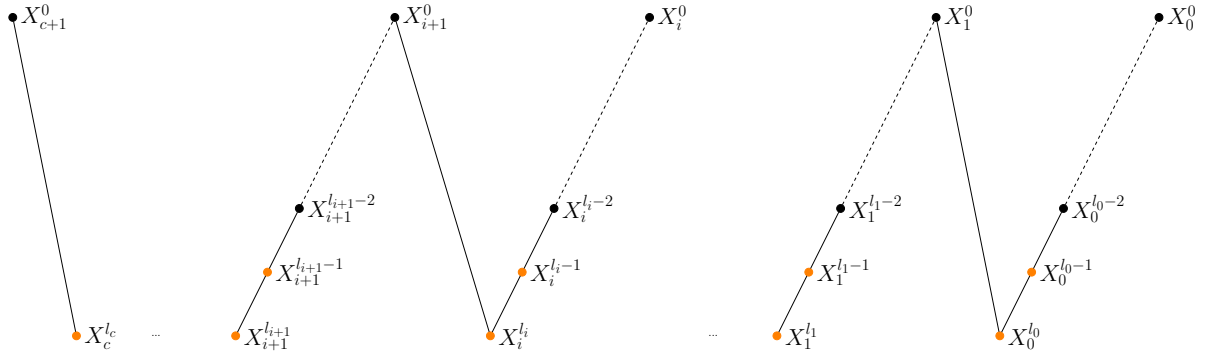


FIGURE 5.4 – Illustration de la preuve du Théorème 5.4.1

— $\forall i \leq c, \theta_j(X_i^{l_{i-1}}) \neq -\infty, \theta_j(X_i^{l_i}) \neq -\infty$,
— $\forall i \leq c, X_i^{l_i} \prec X_i^{l_{i-1}} \prec \cdots \prec X_i^1 \prec X_i^0$,
— $\forall i \ \leq c, X_i^{l_i} \prec X_{i+1}^0$,
— $\theta_j(X_{c+1}^0) = -\infty$.

La Figure 5.4 donne le diagramme de Hasse de la configuration que nous venons de décrire. Les nœuds oranges représentent les éléments déjà inclus dans l'ordre à ce stade de la construction. Les pointillés indiquent l'existence d'un chemin d'éléments entre deux éléments dans le diagramme de Hasse.

$\mathcal{H}$ est un hypergraphe fini et $X_{c+1}^0 = -\infty$, il est donc possible d'atteindre un élément c tel qu'il existe $i \leq c-2$ et $i' \leq l_i - 2$ tels que $X_i^{i'} \prec X_{c+1}^0$ (*i.e.* le schéma répété revient à un élément déjà vu durant le processus). Nous avons de plus $i \leq c-2$. En effet, $i = c$ donnerait $X_c^{i'} \prec X_{c+1}^0$ ce qui est impossible, car $X_c^{l_c} \subseteq X_c^{i'}$ et $X_c^{l_c} \prec X_{c+1}^0$. De plus, $i = c-1$ impliquerait $X_c^{l_c} \subseteq X_{c+1}^0 \cap X_c^0$ et $X_{c-1}^{l_c} \subseteq X_{c+1}^0 \cap X_c^0$, mais les séquences sont construites de manière à ce qu'il n'existe pas d'autre élément Z tel que $X_c^{l_c} \subseteq Z, X_{c-1}^{l_c} \subseteq Z$ et $Z \subseteq X_{c+1}^0 \cap X_c^0$ ce qui est impossible, car $\mathcal{H}$ est clos par intersection. Donc, soit la séquence $(X_i^{i'}, X_i^{l_i}, X_{i-1}^0, X_{i-1}^{l_i}, X_{i-2}^0, \ldots, X_c^{l_c}, X_c^0)$ forme un cycle ce qui est une contradiction, soit elle est telle qu'il existe $i \leq k \leq c$ tel que pour tout $i \leq j \leq c, X_j^0 \subsetneq X_k^0$ ce qui est impossible, car $X_k^0 \parallel X_{k-1}^0$ par hypothèse. □

Construction d'une séquence d'arbres mixtes correspondant à un hypergraphe clos par intersection couverture-binaire

L'Algorithme 13 détaille le processus de construction d'une séquence d'arbres mixtes correspondant à un hypergraphe totalement équilibré clos par intersection couverture-binaire. L'Algorithme 12, quant à lui, donne la manière de contracter une arête donnée en respectant l'hypergraphe. Nous pouvons remarquer que l'Algorithme 13 respecte la construction donnée par l'Algorithme 11. En effet, il contracte une arête à chaque étape en tenant compte des déplacements d'arêtes autorisés. Les seules différences entre les deux algorithmes sont liées au fait que l'ordre dans lequel les arêtes sont contractées ainsi

que le choix des arêtes qui se déplacent sont contraints par l'hypergraphe (et un de ses ordres compatibles). Ainsi, les arêtes $(Z, X) \in E_i$ qui deviennent $(Z, X \cup Y) \in E_{i+1}$ en contractant l'arête $(X, Y) \in E_i$ ne sont pas choisies au hasard, mais afin que T^U_{i+1} soit une partie connexe de T_{i+1} pour tout $U \in E$.

Nous pouvons remarquer que l'Algorithme 12 contrairement à l'Algorithme 11 ne comprend pas d'étape de redirection des arêtes entrantes dans un nœud à supprimer car par définition d'un ordre compatible, de telles arêtes n'existent pas.

En utilisant les diagrammes de Hasse, savoir quelles hyperarêtes sont dominées par une hyperarête donnée s'effectue en temps constant. L'Algorithme 13 a donc la même complexité que l'Algorithme 11, c'est-à-dire $\mathcal{O}(n^3)$.

Nous montrons à présent que toutes les hyperarêtes de l'hypergraphe considéré sont des parties connexes de tous les arbres mixtes créés.

Lemme 5.4.2. *Soit $\mathcal{H} = (V, E)$ un hypergraphe totalement équilibré clos par intersection et $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 13. Pour tout $U \in E$ et $0 \leq i \leq n$, $T^U_i = \{U' \in V_i \mid U' \subseteq U\}$ est une partie connexe de T_i.*

Démonstration. Montrons le résultat par récurrence. $\mathcal{H}$ est totalement équilibré, il admet donc un arbre support. Le résultat est donc vérifié pour T_0. Soit $0 \leq i \leq n$. Supposons que $V_{i+1} \backslash V_i = \{X \cup Y\}$.

Soient $U \in E$ et $0 \leq i \leq n$. Par hypothèse de récurrence $T^U_i = \{U' \in V_i \mid U' \subseteq U\}$ est une partie connexe de T_i. Si $X \nsubseteq U$ et $Y \nsubseteq U$, par construction, $T^U_{i+1} = T^U_i$ est inchangé dans T_{i+1}. La connexité est donc préservée.

Considérons à présent le cas où $X \subseteq U$ (ou/et $Y \subseteq U$). Si $X \in V_{i+1}$ alors $(X, X \cup Y) \in \overrightarrow{E_{i+1}}$ ce qui conserve la connexité de T^U_{i+1} dans T_{i+1}, car pour tout $X' \in V_i$ tel que $X' \subseteq U$ et $(X, X') \in E_i$, soit $(X, X') \in E_{i+1}$ soit $(X', X \cup Y) \in E_{i+1}$. Si $X \notin V_{i+1}$:

- si $X \cup Y \subseteq U$. Pour tout $X' \in V_i$ tel que $(X, X') \in E_i$, $(X', X \cup Y) \in E_{i+1}$. De plus, on construit un arbre support du système sur les sommets $\{X'' \in V_i \mid (X, X'') \in \overrightarrow{E_i}\}$ ce qui est possible car le système est totalement équilibré et tous ses sous-hypergraphes sont donc des hyperarbres et admettent un arbre support. T^U_{i+1} est donc une partie connexe de T_{i+1}.
- si $X \subseteq U$ et $Y \nsubseteq U$. Par construction, il n'existe aucun $X' \in V_i$ tel que $(X', X) \in \overrightarrow{E_i}$. De plus, il n'existe aucun $X'' \in V_i$ tel que $(X, X'') \in E_i$ et $X'' \subseteq U$. En effet, $(X, X'') \in E_i$, et $X \notin V_{i+1}$, il n'existe donc aucun $S \in E$ tel que $X'' \cup X \subseteq S$ et $Y \nsubseteq S$ par construction et par clôture par intersection de l'hypergraphe. Les seuls voisins $Z \in V_i$ de X inclus dans U sont donc de la forme $(X, Z) \in \overrightarrow{E_i}$. Par construction, le sous-graphe induit de T_{i+1} par $\overrightarrow{\mathcal{N}_{i+1}(X)} = \{Z \in V_i \mid (X, Z) \in \overrightarrow{E_i}\} \cup \{X \cup Y\}$ est un arbre support de l'hypergraphe dont les hyperarêtes sont $\{A \in E \mid A \subseteq \bigcup_{C \in \overrightarrow{\mathcal{N}_i(X)}} C, \exists B \in \overrightarrow{\mathcal{N}_{i+1}(X)}, B \subseteq A\}$ donc T^U_{i+1} est une partie connexe de T_{i+1}.

Algorithme 12 : Contraction d'une arête d'un arbre mixte

Données : $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection,
$T_i = (V_i, E_i, \overrightarrow{E_i})$ un arbre mixte,
$(X, Y) \in E_i$,
$supprime_X$ un booléen indiquant s'il faut supprimer X dans T_{i+1} ou non,
$supprime_Y$ un booléen indiquant s'il faut supprimer Y dans T_{i+1} ou non
Résultat : T_{i+1} un arbre mixte

Fonction `contracter_arete`$(\mathcal{H}, T_i, (X, Y), supprime_X, supprime_Y)$ **:**
  $V_{i+1} = V_i \cup \{X \cup Y\}$
  $\overrightarrow{E_{i+1}} = \overrightarrow{E_i} \cup \{(X, X \cup Y), (Y, X \cup Y)\}$
  $E_{i+1} = E_i \backslash \{(X, Y)\}$
  pour $Z \in \{X, Y\}$ **faire**
    si $supprime_Z$ **alors**
      pour $U \in V_i, (U, Z) \in E_i$ **faire**
        $E_{i+1} = E_{i+1} \cup \{(U, X \cup Y)\} \backslash \{(U, Z)\}$
      fin
      $E_{i+1} = E_{i+1} \cup$ `arbre_support`$(\mathcal{H}, \{(Z, U) \in \overrightarrow{E_i} \mid U \in V_i\})$
      $\overrightarrow{E_{i+1}} = \overrightarrow{E_{i+1}} \backslash \{U \in V_i \mid (Z, U) \in \overrightarrow{E_i}\}$
      $V_{i+1} = V_{i+1} \backslash \{Z\}$
    sinon
      pour $U \in \{U' \in V_i \mid (U', Z) \in E_i\}$ **faire**
        si $\nexists C \in E, U \cup Z \subseteq C, X \cup Y \nsubseteq C$ **alors**
          $E_{i+1} = E_{i+1} \cup \{(U, X \cup Y)\} \backslash \{(U, Z)\}$
        fin
      fin
    fin
  fin

Où `arbre_support`$(\mathcal{H}, S)$ est une fonction qui retourne un arbre sur les éléments de S tel que le nombre maximum d'hyperarêtes de $\mathcal{H}$ sont des parties connexes de l'arbre (ce qui peut être réalisé en calculant un arbre couvrant maximum d'un graphe support de $\mathcal{H}$ pondéré par le nombre d'hyperarêtes qui contiennent l'union des extrémités de chaque arête du graphe).

Algorithme 13 : Construction d'une séquence d'arbres mixtes correspondant à un hypergraphe totalement équilibré donné

Données : $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection
Résultat : $\mathcal{T} = (T_0, \ldots, T_n)$ une séquence d'arbres mixtes respectant la construction définie par l'Algorithme 11 telle que les hyperarêtes de $\mathcal{H}$ sont des sommets des arbres

1 $G =$ `graphe_support`$(\mathcal{H})$
2 $(V_0, E_0) =$ `mst`(G)
3 $T_0 = (V_0, E_0, \emptyset)$
4 $i = 0$
5 **tant que** $\mid V_i \mid > 1$ **faire**
6 Choisir $U \in E$ tel que $\forall j \leq i, U \notin V_j$, et $\forall T \in E, T \prec U \implies \exists j \leq i, T \in V_j$ et $\forall S, S', T, \in E, S \prec T \prec U, S \prec S' \implies \exists j \leq i, S' \in V_j$
7 **si** $\exists (X, Y) \in E_i, U = X \cup Y$ **alors**
8 $T_{i+1} =$ `contracter_arete`$(\mathcal{H}, T_i, (X, Y), (\nexists C \neq X \cup Y \in E, X \prec C, \forall j \leq i, C \notin V_j), (\nexists C \neq X \cup Y \in E, Y \prec C, \forall j \leq i, C \notin V_j))$
9 **sinon**
10 $T_{i+1} = T_i$
11 **tant que** $U \notin V_{i+1}$ **faire**
12 Choisir $(X, Y) \in \{(X, Y) \in E_i \mid X \cup Y \subseteq U\}$
13 **si** $\{X' \in V_{i+1} \mid (X', X) \in E_{i+1}\} = \{Y\}$ **alors**
14 $supprime_X = Vrai$
15 **sinon**
16 $supprime_X =$ `booleen_hasard` ()
17 **fin**
18 **si** $\{Y' \in V_{i+1} \mid (Y', Y) \in E_{i+1}\} = \{X\}$ **alors**
19 $supprime_Y = Vrai$
20 **sinon**
21 $supprime_Y =$ `booleen_hasard` ()
22 **fin**
23 $T_{i+1} =$ `contracter_arete`$(\mathcal{H}, T_{i+1}, (X, Y), supprime_X, supprime_Y)$
24 **fin**
25 **fin**
26 i = i +1
27 **fin**
28 $\mathcal{T} = (T_0, \ldots, T_{i+1})$

Où `graphe_support`$(\mathcal{H})$ est une fonction qui renvoie un graphe sur les sommets de $\mathcal{H}$ tel que chaque hyperarête de $\mathcal{H}$ est une partie connexe du graphe et où l'arête (X, Y) a un poids $|\{U \in E \mid X \cup Y \subseteq U\}|$, `mst`$(G)$ une fonction retournant un arbre couvrant maximum du graphe G et `booleen_hasard`() une fonction qui renvoie un booléen au hasard.

□

Lemme 5.4.3. *Soit $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection, totalement équilibré et couverture-binaire $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 13. Soit θ un ordre compatible avec $\mathcal{H}$. Soit $i \leq n$ et $X, Y \in E$ tels que $\theta(X \cup Y) = i$. Alors $(X, Y) \in E_i$.*

Démonstration. Soit $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection couverture-binaire et $X, Y \in E$ tels que $X \cup Y$ couvre X et Y. Par définition d'un ordre compatible, il existe $j_X, j_Y < i + 1$ tels que $X \in V_{j_X}$ et $Y \in V_{j_Y}$. Par construction et définition de l'ordre, $X \in V_i$ et $Y \in V_i$ car pour tout $j < i+1, X \cup Y \notin V_j$. Supposons qu'il existe un $Z \in V_i$ sur le chemin entre X et Y dans T_i. D'après le Lemme 5.4.2, $\{U \in V_i \mid U \subseteq X \cup Y\}$ est une partie de connexe de T_i donc $Z \subseteq X \cup Y$. $\mathcal{H}$ est couverture-binaire donc X et Y sont les seuls éléments couverts par $X \cup Y$ donc soit $Z \subsetneq X$ soit $Z \subsetneq Y$. Par construction, si $Z \in V_i$ alors il existe $T \in E$ tel que $Z \prec T$ et pour tout $k \leq i, T \notin V_k$ (d'où $T \neq X, T \neq Y$) ce qui est impossible par la condition (ii) de la définition d'un ordre compatible. Il n'existe donc aucun Z sur le chemin entre X et Y dans T_i. D'après le Lemme 5.3.1(ii), $(X, Y) \notin \overrightarrow{E_i}$ et $(Y, X) \notin \overrightarrow{E_i}$. Donc $(X, Y) \in E_i$. □

Ce résultat montre que si l'hypergraphe donné est binaire, la ligne 9 de l'Algorithme 13 n'est jamais atteinte car l'hyperarête considérée correspond toujours à une arête de l'arbre mixte en cours. Cette partie de l'algorithme sera utilisée pour approximer un hypergraphe non binaire par un hypergraphe binaire grâce à la construction.

Théorème 5.4.4. *Soit $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection, couverture-binaire totalement équilibré et $\mathcal{T} = (T_0, \ldots, T_n)$ une séquence d'arbres obtenue par l'Algorithme 13.*

$$\bigcup_{i=0}^{n} V_i = E$$

Ce résultat peut être déduit de manière immédiate du Lemme 5.4.3.

Caractérisation

L'Algorithme 13 est un cas particulier de l'Algorithme 11 et peut créer une séquence d'arbres mixtes représentant n'importe quel hypergraphe binaire totalement équilibré donné (Théorème 5.4.4). De plus, l'Algorithme 11 ne crée que des hypergraphes clos par intersection et couverture-binaires (Théorème 5.3.7) et peut facilement être adapté pour créer des hypergraphes clos et binaires. La combinaison de ces deux résultats nous permet de terminer la preuve de caractérisation des hypergraphes clos et binaires par la construction d'une séquence d'arbres mixtes.

Nous pouvons résumer ce résultat en donnant explicitement la caractérisation des hypergraphes clos binaires obtenue rappelant celle de Lehel [32] pour les hypergraphes totalement équilibrés.

Théorème 5.4.5. *Soit $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection. $\mathcal{H}$ est binaire ssi il existe une séquence d'arbres mixtes $\mathcal{T} = (T_0, T_1, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) telle que pour tout $0 \leq i \leq n$, $V_i \subseteq 2^V$ et :*

1. *(a)* $V_0 = V$

   (b) $\exists (X, Y) \in E_i, V_i \cup \{X \cup Y\} \backslash \{X, Y\} \subseteq V_{i+1} \subseteq V_i \cup \{X \cup Y\}$

2. $\forall S, T \in V_{i+1}, (S, T) \in E_{i+1} \implies \begin{cases} (S, T) \in E_i \\ ou \\ \exists X, Y \in V_i, S = X \cup Y, (X, T) \in E_i \\ ou \\ \exists X, Y \in V_i, S = X \cup Y, (X, T) \in \overrightarrow{E_i} \end{cases}$

3. *(a)* $\forall S, T \in V_{i+1}, (S, T) \in \overrightarrow{E_{i+1}} \implies \begin{cases} (S, T) \in \overrightarrow{E_i} \\ ou \\ \exists W \in V_i, (S, W) \in E_i, T = S \cup W \\ ou \\ \exists (W, X) \in E_i, (S, W) \in \overrightarrow{E_i}, T = W \cup X \end{cases}$

   (b) $\forall X \in V_i, |\{Y \in V_i \mid (X, Y) \in \overrightarrow{E_i}\}| \leq 1$

4. $E = \bigcup_{i=0}^{k} V_i$

5.5 Binarisation des hypergraphes totalement équilibrés par la construction

5.5.1 Processus de binarisation

Le Lemme 5.4.2 montre que chaque hyperarête d'un hypergraphe clos par intersection totalement équilibré est une partie connexe de tous les arbres construits par l'Algorithme 13 même si l'hypergraphe n'est pas couverture-binaire. Or l'algorithme produit un hypergraphe couverture-binaire. Nous pouvons donc l'utiliser pour transformer n'importe quel hypergraphe totalement équilibré en un hypergraphe couverture-binaire.

Théorème 5.5.1. *Soit $\mathcal{H} = (V, E)$ un hypergraphe totalement équilibré clos par intersection et $\mathcal{T} = (T_0, \ldots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 13. $\mathcal{H}' = (V, E')$ où $E' = \{U \in V_i \mid 0 \leq i \leq n\}$ est un hypergraphe couverture-binaire totalement équilibré clos par intersection avec $E \subseteq E'$.*

Démonstration. Soient $i \leq n$, θ un ordre compatible avec $\mathcal{H}$ et $U \in E$ tel que $\theta(U) = i$. D'après le Lemme 5.4.2, $T_j^U = \{U' \in V_j \mid U' \subseteq U\}$ est une partie connexe de T_j pour tout $j \leq i$ donc en particulier pour $j = \max_{S \in E, S \subsetneq U} \theta(S) + 1$. Supposons qu'il existe $X \in T_j^U, Y \in T_j^U$ tels que $(X, Y) \in \overrightarrow{E_j}$. Par construction, si $X \in V_j$ alors il existe $U' \neq U$ tel que $X \prec U'$ et pour tout $k \leq j, U' \notin V_k$. Mais $X \prec Y \prec U$ donc le résultat est incompatible avec la définition d'ordre compatible. Les éléments de T_j^U ne peuvent donc

être liés que par des arêtes non orientées. Nous pouvons donc appliquer la construction donnée par l'Algorithme 11 sur le sous-hypergraphe de $\mathcal{H}$ induit par $T_j^U \cup U$ ce qui donne un hypergraphe couverture-binaire clos par intersection (Théorème 5.3.7) et permet à l'hyperarête U de ne couvrir que deux éléments. □

Pour obtenir un hypergraphe couverture-binaire possédant le moins possible d'hyperarêtes à partir d'un hypergraphe totalement équilibré et clos donné $\mathcal{H}$, l'Algorithme 13 peut être légèrement modifié à la ligne 23 pour passer en argument `True` pour chaque booléen à chaque itération. Cela permet de supprimer les sommets des arbres le plus tôt possible afin de limiter le nombre de nouvelles hyperarêtes créées.

Ce résultat permet d'obtenir un hypergraphe couverture-binaire contenant toutes les hyperarêtes de n'importe quel hypergraphe totalement équilibré clos par intersection. Pour obtenir un hypergraphe binaire à partir d'un hypergraphe couverture-binaire $\mathcal{H} = (V, E)$, le processus suivant peut être appliqué :

1. Soit $\mathcal{R}_{\mathcal{H}}$ la relation d'ordre partiel d'inclusion de E.
2. Inverser la relation d'ordre partiel : pour tout $A, B \in E, A\mathcal{R}_{\mathcal{H}}^{-1}B$ ssi $B\mathcal{R}_{\mathcal{H}}A$.
3. Pour tout $B \in E$ tel qu'il existe un unique $A \in E$ tel que $A\mathcal{R}_{\mathcal{H}}^{-1}B$, étendre l'ordre à un nouvel élément A' tel que $A'\mathcal{R}_{\mathcal{H}}^{-1}B$.
4. Soit $\mathcal{H}^{-1}$ l'hypergraphe associé à la relation d'ordre étendue.
5. Appliquer l'Algorithme 13 sur $\mathcal{H}^{-1}$ pour obtenir $\mathcal{H}_b^{-1}$ un hypergraphe binaire.
6. Soit $\mathcal{R}_{\mathcal{H}_b^{-1}}$ l'ordre partiel associé à $\mathcal{H}_b^{-1}$.
7. Supprimer tous les éléments ajoutés à la relation d'ordre lors de l'étape 3 pour obtenir $\mathcal{R}_{\mathcal{H}_{b|}^{-1}}$.
8. Inverser $\mathcal{R}_{\mathcal{H}_{b|}^{-1}}$ pour obtenir $\mathcal{R}_{\mathcal{H}_{b|}^{-1}}^{-1}$.
9. Soit $\mathcal{H}_b$ l'hypergraphe associé à la relation $\mathcal{R}_{\mathcal{H}_{b|}^{-1}}^{-1}$. $\mathcal{H}_b$ est un hypergraphe binaire qui approxime $\mathcal{H}$.

Grossièrement, le processus consiste à considérer l'ordre inverse d'inclusion des hyperarêtes d'un hypergraphe couverture-binaire (étape 2) ce qui transforme les hyperarêtes couvertes par plus de deux hyperarêtes en hyperarêtes couvertes par deux hyperarêtes. L'Algorithme 13 peut ensuite être utilisé pour transformer les hyperarêtes non couverture-binaires en hyperarêtes couverture-binaires (étape 5), ce qui revient à transformer les hyperarêtes couvertes par plus de deux hyperarêtes de l'hypergraphe initial en hyperarêtes couvertes par deux hyperarêtes. Inverser une nouvelle fois l'ordre donne une approximation binaire de l'hypergraphe original (étape 8).

Le processus est plus intuitif dans le formalisme des treillis et revient à utiliser la même idée que pour la binarisation présentée dans la Section 4.3. Il s'agit d'appliquer la binarisation dans un sens au treillis associé à l'hypergraphe puis à appliquer le même traitement au dual de ce treillis pour revenir finalement au dual du treillis obtenu correspondant à l'approximation de celui de départ.

5.5.2 Comparaison avec l'approximation d'un treillis démontable par un treillis binaire

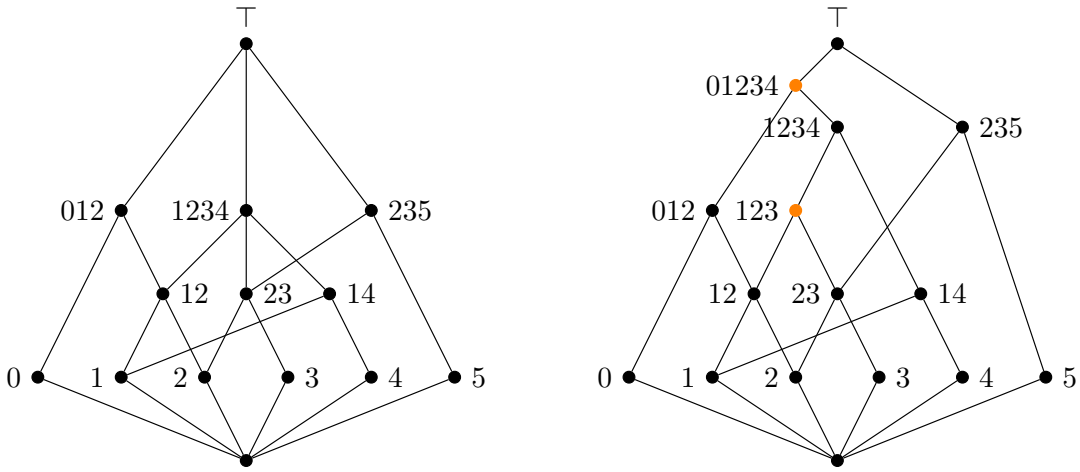
Si l'idée générale de cette approximation est identique à celle présentée pour les treillis démontables dans le Chapitre 4 (binariser la structure dans un sens puis appliquer la même méthode à la structure duale), la construction présentée dans ce chapitre est plus générale.

Proposition 5.5.2. *Soit $\mathcal{H} = (V, E)$ un hypergraphe totalement équilibré clos par intersection. Soit $\mathcal{B}(\mathcal{H})$ l'hypergraphe équivalent au treillis obtenu par l'Algorithme 10 appliqué sur le treillis associé à $\mathcal{H}$. $\mathcal{B}(\mathcal{H})$ peut être obtenu par l'Algorithme 13 appliqué à $\mathcal{H}$.*

Démonstration. Soit $\mathcal{H} = (V, E)$ un hypergraphe totalement équilibré clos par intersection. Si $\mathcal{H}$ est binaire, l'Algorithme 10 comme l'Algorithme 13 laissent $\mathcal{H}$ inchangé. Supposons à présent que $\mathcal{H}$ n'est pas binaire. Il existe au moins un $Z \in E$ tel que Z couvre l éléments, $l > 2$, $Z_1, Z_2, \ldots, Z_l$. On se place dans un arbre contenant tous les Z_i (existe par la Propriété 5.3.3). Pour tout i, j, k distincts tels que Z_j est sur le chemin de Z_i à Z_k, d'après la Propriété 5.3.5, $Z_i \cap Z_k \subseteq Z_i \cap Z_j$ et $Z_i \cap Z_k \subseteq Z_j \cap Z_k$. Si $Z_i \cap Z_k \subsetneq Z_i \cap Z_j$ ou $Z_i \cap Z_k \subsetneq Z_j \cap Z_k$, Z_i et Z_k ne sont pas d'intersection maximale. Nous avons dans ce cas Z_i et Z_k non voisins et Z_i, Z_k ne pas d'intersection maximale parmi les $\{Z_j\}_{\{1 \leq j \leq l\}}$.

Sinon, $Z_i \cap Z_j = Z_i \cap Z_k = Z_j \cap Z_k = Z_i \cap Z_j \cap Z_k$. Or $Z_i \cap Z_j \cap Z_k \subsetneq Z_i, Z_j, Z_k$, donc il existe Z'_i, Z'_j, Z'_k tels que $Z_i \cap Z_j \cap Z_k \prec Z'_i, Z'_i \subseteq Z_i, Z'_i \parallel Z_j, Z'_i \parallel Z_k$ (et symétriquement pour Z'_j et Z'_k). D'après la Propriété 5.3.3, il existe un arbre $T_m = (V_m, E_m)$ tel que $Z_i \cap Z_j \cap Z_k \in T_m$ mais $Z'_i \notin V_m, Z'_j \notin V_m, Z'_k \notin V_m$. Il existe $m' > m$ tel que pour tout $A \in E$, si $Z_i \cap Z_j \cap Z_k \prec A$, $A \in V_{m'}$. Par construction, pour $m' = \min\{0 \leq i \leq n \mid \forall A \in E, Z_i \cap Z_j \cap Z_k \prec A \implies A \in V_i\}$, un arbre couvrant minimum du graphe support de l'hypergraphe sera utilisé pour créer les arêtes entre les sommets $\{A \in T_{m'} \mid Z_i \cap Z_j \cap Z_k \prec A\}$. Z'_i, Z'_j et Z'_k sont tous les trois des successeurs de $Z_i \cap Z_j \cap Z_k$. De plus, pour tout $A \in E$, si $Z'_i \in E$ et $Z'_j \in E$ alors $Z'_k \in E$ (et symétriquement pour Z'_j, Z'_k et Z'_i, Z'_k). Il n'y a donc pas de contraintes sur les arêtes entre ces trois éléments. Il existe donc un arbre tel que Z'_k est sur le chemin de Z'_i à Z'_j et symétriquement pour ces trois éléments. Comme $Z'_i \parallel Z_j$ et $Z'_i \parallel Z_k$ (et symétriquement pour les trois éléments), par construction, il existe un arbre tel que Z_k est sur le chemin de Z_i à Z_j et symétriquement pour les deux autres éléments. Si deux d'entre eux sont d'intersection maximale parmi $Z_1, Z_2, \ldots, Z_l$ alors il existe une séquence d'arbres tels qu'ils sont voisins dans le premier arbre dont $Z_1, Z_2, \ldots, Z_l$ sont tous sommets. Pour toute paire d'éléments d'intersection maximale d'une antichaîne couverte par un même élément, il existe une séquence d'arbres telle que ces éléments sont voisins. □

Il est donc possible de reproduire la binarisation proposée dans le Chapitre 4 par cette construction. La Figure 5.5 représente le diagramme de Hasse d'un hypergraphe totalement équilibré et d'une de ses binarisations possibles. Cet exemple a déjà été utilisé



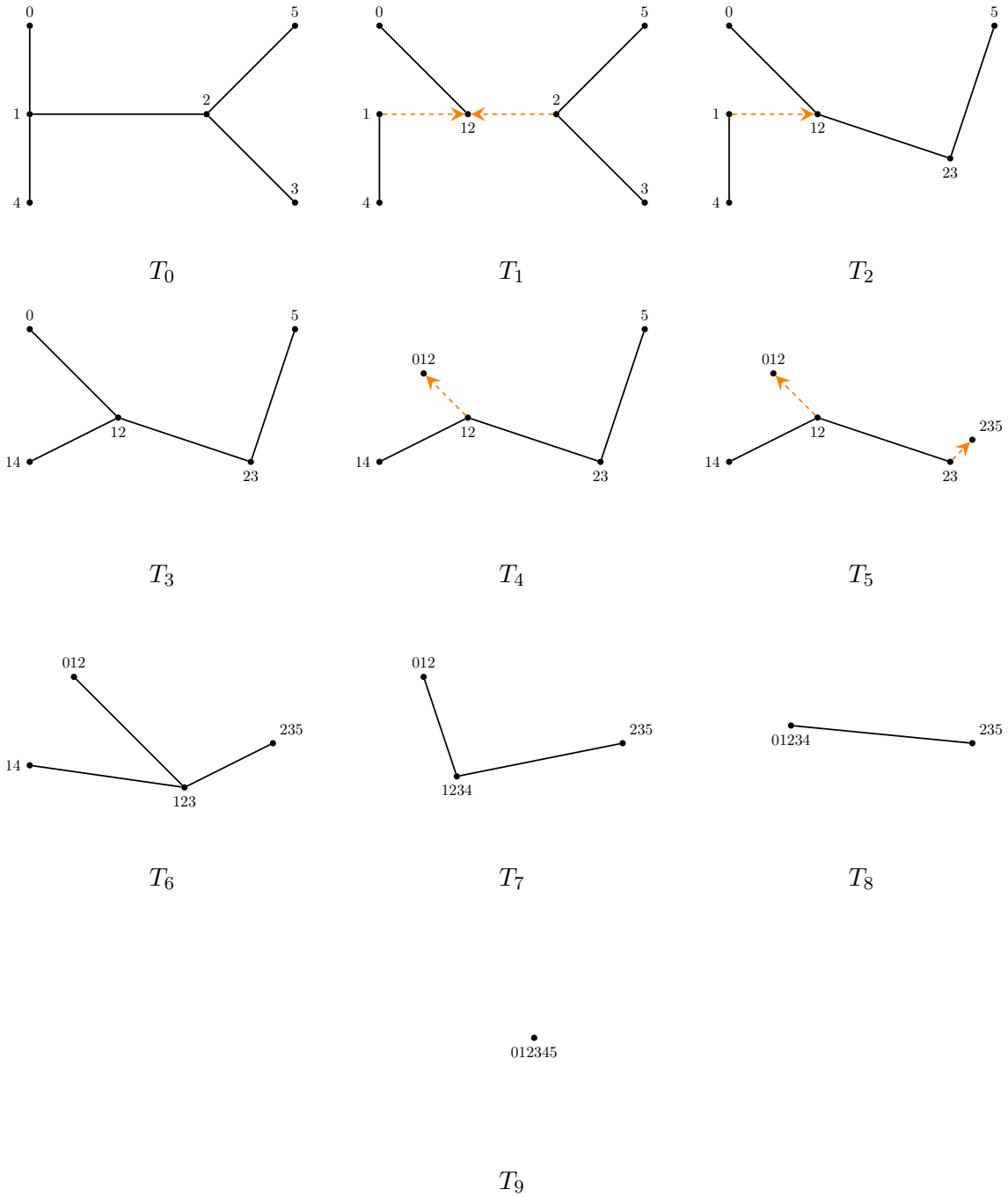
(a) Diagramme de Hasse d'un hypergraphe (b) Binarisation possible de l'hypergraphe

Les nœuds oranges sont ceux ajoutés par la binarisation.

FIGURE 5.5 – Exemple d'approximation d'un hypergraphe totalement équilibré par un hypergraphe binaire

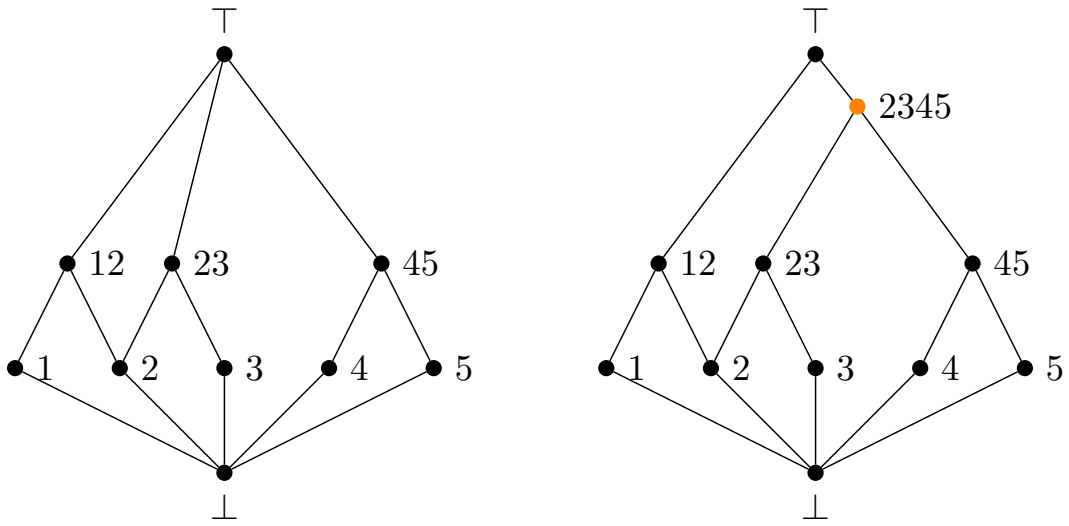
dans le Chapitre 4 pour illustrer la binarisation des treillis démontables. La Figure 5.6 représente une séquence d'arbres permettant une approximation de l'hypergraphe. Cette approximation est identique à celle présentée dans le Chapitre 4 et réalisée par l'algorithme proposé dans ce même chapitre pour les treillis démontables. La binarisation aurait pu être différente. En effet, sur l'arbre T_5, l'hyperarête $\{1, 2, 3, 4\}$ est une partie connexe de l'arbre composée des arêtes $(\{1, 2\}, \{1, 4\})$ et $(\{1, 2\}, \{2, 3\})$. Les deux arêtes peuvent donc être choisies pour approximer l'hypergraphe.

L'algorithme d'approximation proposé ici est plus général que celui du Chapitre 4. La Figure 5.7 présente le diagramme de Hasse d'un hypergraphe totalement équilibré mais non binaire ainsi que le diagramme de Hasse d'une approximation de cet hypergraphe par un hypergraphe binaire obtenu par l'algorithme de construction dont le résultat est représenté sur la Figure 5.8. Cette approximation ne peut pas être obtenue par l'algorithme proposé dans le Chapitre 4. En effet, l'hyperarête $\{1, 2, 3, 4, 5\}$ couvre les trois hyperarêtes $\{1, 2\}, \{2, 3\}, \{4, 5\}$. La binarisation proposée par les treillis créerait l'hyperarête $\{1, 2, 3\}$, car $\{1, 2\}$ et $\{2, 3\}$ sont les deux seules hyperarêtes d'intersection maximale parmi $\{1, 2\}, \{2, 3\}, \{4, 5\}$. Or, dans la construction proposée ici, c'est l'union des hyperarêtes $\{2, 3\}$ et $\{4, 5\}$ qui est choisie.



Les arêtes orientées sont représentées par des flèches pointillées oranges.

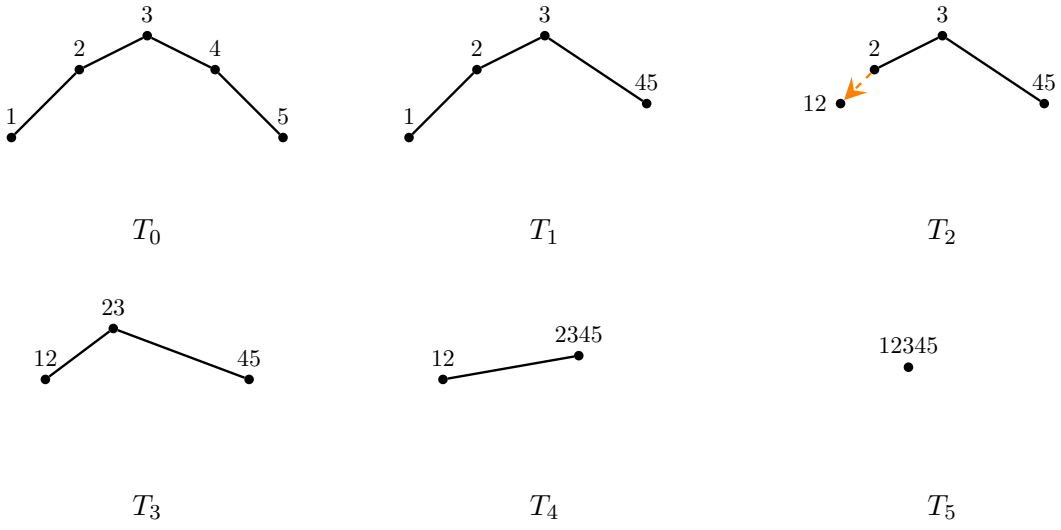
FIGURE 5.6 – Exemple de binarisation par la construction de l'hypergraphe dont le diagramme de Hasse est représenté sur la Figure 5.5



(a) Diagramme de Hasse d'un hypergraphe (b) Binarisation possible de l'hypergraphe

Le nœud orange est celui ajouté par la binarisation.

FIGURE 5.7 – Exemple d'approximation d'un hypergraphe totalement équilibré par un hypergraphe binaire



Les arêtes orientées sont représentées par des flèches pointillées oranges.

FIGURE 5.8 – Exemple de binarisation par la construction de l'hypergraphe dont le diagramme de Hasse est représenté sur la Figure 5.7

5.6 Équivalence entre les hypergraphes binaires et les hypergraphes totalement équilibrés

Les résultats présentés dans ce chapitre permettent d'obtenir l'équivalence entre les systèmes totalement équilibrés et les systèmes binarisables. Cette équivalence a déjà été démontrée pour les treillis dans la Section 4.4 (Théorème 4.4.3) dans le cas des hypergraphes. La preuve de la proposition suivante est strictement identique à celle présentée pour les treillis (Proposition 4.4.2), mais transposée dans le formalisme des hypergraphes.

Proposition 5.6.1. *Soit $\mathcal{H} = (V, E)$ un hypergraphe. Si $\mathcal{H}$ est binarisable alors $\mathcal{H}$ est totalement équilibré.*

Démonstration. Nous montrons dans un premier temps que si $(X_0, X'_0, \ldots, X_{n-1}, X'_{n-1})$ est un cycle d'un hypergraphe $\mathcal{H} = (V, E)$ alors $(X_0, \sup(X_0, X_1), \ldots, X_{n-1}, \sup(X_{n-1}, X_0))$ est un cycle de $\mathcal{H}$. Montrons que pour tout i et pour tout $j \neq i, i+1$, $\sup(X_i, X_{i+1}) \parallel X_j$. Or $X_i \parallel X_j$ et $X_{i+1} \parallel X_j$ par définition d'un cycle, donc $\sup(X_i, X_{i+1}) \nsubseteq X_j$. De plus, $X_j \parallel X'_i$ par définition d'un cycle. Mais pour tout $i \leq n-1$, $X_i \subseteq X'_i$ et $X_{i+1} \subseteq X'_i$ donc $\sup(X_i, X_{i+1}) \subseteq X'_i$. Donc $X_j \nsubseteq \sup(X_i, X_{i+1})$. D'où $X_j \parallel \sup(X_i, X_{i+1})$. Nous considérerons donc à partir de maintenant les cycles de la forme $(X_0, \sup(X_0, X_1), \ldots, X_{n-1}, \sup(X_{n-1}, X_0))$.

Nous prouvons maintenant par récurrence qu'un hypergraphe contenant un cycle ne peut pas être binaire.

Supposons $\mathcal{H} = (V, E)$ est un hypergraphe binaire possédant un 3-cycle. Soit $(X_0, \sup(X_0, X_1), X_1, \sup(X_1, X_2), X_2, \sup(X_2, X_0))$ un 3-cycle et $Y = \sup(X_0, X_1, X_2) = \sup(\sup(X_0, X_1), \sup(X_1, X_2), \sup(X_2, X_0))$. $\mathcal{H}$ est binaire donc Y couvre au plus deux éléments Y' et Y'' et $\{X_i \subseteq \sup(Y', Y'')\} = \{X_0, X_1, X_2\}$. Y est le supremum de $\sup(X_0, X_1)$, $\sup(X_1, X_2), \sup(X_2, X_0)$ et $Y' \prec Y, Y'' \prec Y$ donc, par le principe des tiroirs, nous pouvons supposer sans perte de généralité que $\sup(X_1, X_2) \subseteq Y'$ et $\sup(X_2, X_0) \subseteq Y'$. Donc $X_0 \subsetneq Y'$ et $X_1 \subsetneq Y'$. D'où $\sup(X_0, X_1) \subseteq Y'$ donc $Y' = Y$ ce qui est une contradiction.

Supposons que tout hypergraphe possédant un cycle de taille $n-1$ (avec $n > 3$) n'est pas binaire. Soit $\mathcal{H} = (V, E)$ un hypergraphe binaire contenant un n-cycle. Soit $(X_0, \sup(X_0, X_1), \ldots X_{n-1}, \sup(X_{n-1}, X_0))$ un cycle et $Y = \sup(X_0, \ldots X_{n-1})$. $\mathcal{H}$ est binaire donc Y couvre au plus deux éléments Y' et Y''. $Y = \sup(X_0, \ldots, X_{n-1})$ donc $1 \leq |\{X_i \subseteq Y' \mid i \leq n-1\}| < n$ et $1 \leq |\{X_i \subseteq Y'' \mid i \leq n-1\}| < n$. Y est binaire donc $\{X_i \subseteq Y' \mid i \leq n-1\} \cup \{X_i \subseteq Y'' \mid i \leq n-1\} = \{X_i \mid 1 \leq i \leq n-1\}$. Nous pouvons donc supposer sans perte de généralité que $|\{X_i \subseteq Y' \mid i \leq n-1\}| \geq 2$. Supposons de plus que $X_0 \nsubseteq Y'$. Soit $j = \min\{i \leq n-1 \mid X_i \subseteq Y'\}$ et $j' = \max\{i \leq n-1 \mid X_i \subseteq Y'\}$. Alors $(X_{j'}, \sup(X_{j'}, X_{j'+1 \mod n}), \ldots, X_0, \ldots, X_j, Y')$ est un cycle de taille inférieure ou égale à $n-1$ et supérieure à 3. En effet, pour tout $i < j$ et pour tout $i > j'$, $X_i \nsubseteq Y'$. Par hypothèse de récurrence, l'hypergraphe n'est pas binaire.

Par définition d'un hypergraphe binarisable et d'un cycle, pour tout hypergraphe non totalement équilibré $\mathcal{H} = (V, E)$, s'il existe un hypergraphe binaire $\mathcal{H}' = (V', E')$ tel

qu'il existe $f : E \to E'$ conservant l'ordre d'inclusion des hyperarêtes et $V \subseteq V'$, alors $(X_0, X'_0, \dots X_{n-1}, X'_{n-1})$ est un cycle dans $\mathcal{H}$ ssi $f(X_0), f(X'_0), \dots, f(X_{n-1}), f(X'_{n-1})$ est un cycle dans $\mathcal{H}'$. D'après le résultat précédent $\mathcal{H}'$ n'est pas binaire. $\mathcal{H}$ n'est donc pas binarisable. □

Théorème 5.6.2. *Soit $\mathcal{H} = (V, E)$ un hypergraphe. $\mathcal{H}$ est binarisable ssi $\mathcal{H}$ est totalement équilibré.*

Ce théorème se déduit immédiatement du Théorème 5.5.1, de la Proposition 5.6.1 et de la Propriété 1.3.1.

5.7 Approximation d'un hypergraphe quelconque par un hypergraphe binaire

Nous proposons à présent d'élargir le résultat d'approximation initié par le Théorème 5.5.1 aux hypergraphes quelconques et montrons que la construction de l'Algorithme 13, légèrement adaptée, appliquée à un hypergraphe clos par intersection donne un hypergraphe couverture-binaire (donc totalement équilibré) tel que le nombre d'hyperarêtes présentes dans l'hypergraphe initial mais pas dans l'hypergraphe obtenu est minimum. Notre construction permet donc de supprimer les cycles d'un hypergraphe en enlevant le moins possible d'hyperarêtes.

L'adaptation nécessaire consiste simplement à remplacer tous les "$U \in E$" par "$U \in E$ et U partie connexe de l'arbre mixte considéré". En effet, la suppression des cycles passe par la suppression de certaines hyperarêtes, ce qui se matérialise par le fait que ces hyperarêtes ne sont plus des parties connexes de l'arbre et ne peuvent donc plus être obtenues par contractions successives d'arêtes. Ces hyperarêtes ne doivent donc plus compter pour le choix d'une hyperarête (ligne 6) ainsi que pour la suppression ou non des nœuds selon l'existence d'hyperarêtes les contenant (ligne 8).

Théorème 5.7.1. *Soit $\mathcal{H} = (V, E)$ un hypergraphe clos par intersection et $\mathcal{T} = (T_0, \dots, T_n)$ (avec $T_i = (V_i, E_i, \overrightarrow{E_i})$ pour tout $0 \leq i \leq n$) une séquence d'arbres mixtes obtenue par l'Algorithme 13. $\mathcal{H}' = (V, E')$ où $E' = \{U \in V_i \mid 0 \leq i \leq n\}$ est un hypergraphe couverture-binaire, totalement équilibré et clos par intersection tel que $E \backslash E'$ est de taille minimale.*

Démonstration. Par construction, l'Algorithme 13 respecte la construction donnée par l'Algorithme 11 car il en est un cas particulier. L'hypergraphe $\mathcal{H}'$ est donc clos par intersection et couverture-binaire (Théorème 5.3.7). Il est donc totalement équilibré (Théorème 5.6.2).

Soit $U \in E \backslash E'$. Notons $T_i^U = \{U' \in V_i \mid U' \subseteq U\}$ pour tout $i \leq n$. Par construction, si $U \notin E'$, soit T_0^U n'est pas une partie connexe de T_0, soit il existe i tel que T_i^U est une partie connexe de T_i mais T_{i+1}^U n'est pas une partie connexe de T_{i+1}.

Dans le premier cas, $\mathcal{H}$ n'admet pas d'arbre support donc il n'est pas un hyperarbre et possède un cycle. L'algorithme utilise un arbre couvrant maximum du graphe support de $\mathcal{H}$, la connexité dans T_0 d'un nombre minimum d'hyperarêtes est donc supprimée.

Dans le second cas, il existe $X, Y \in V_i$ tel que $(X, Y) \in E_i$ et $X \cup Y \in V_{i+1}$. Si T_i^U est une partie connexe de T_i et T_{i+1}^U n'est pas une partie connexe de T_{i+1}, alors, par construction, $X \subseteq U, Y \nsubseteq U$ et $X \notin V_{i+1}$ (ou symétriquement pour Y) ce qui conduit à deux cas :

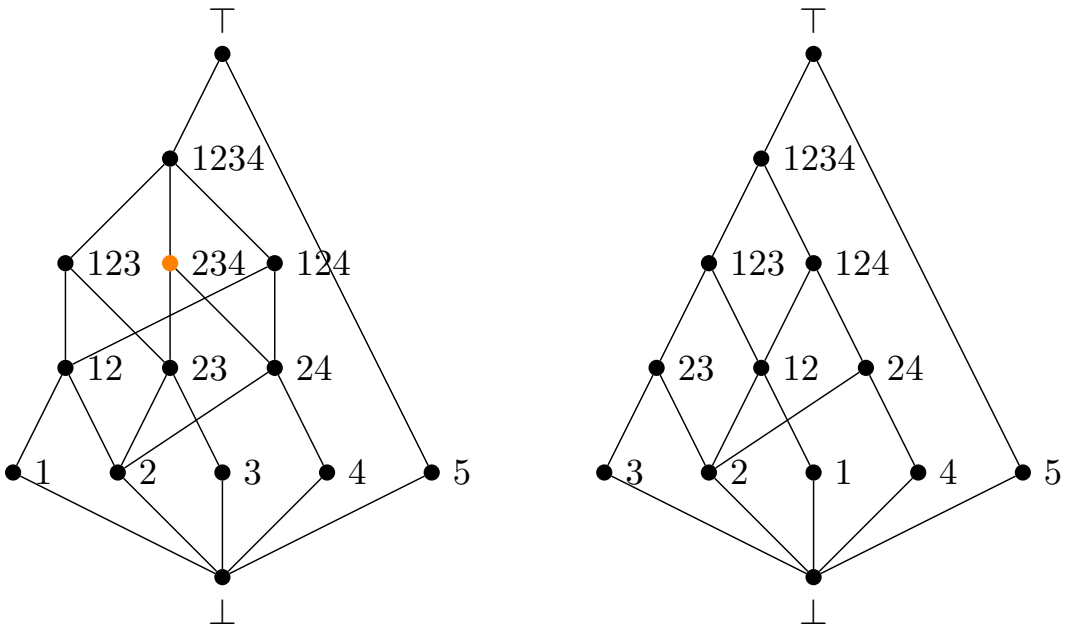
— $\exists! Z \in V_i, (X, Z) \in \overrightarrow{E_i}$. T_i^U est une partie connexe de T_i mais T_{i+1}^U n'est pas une partie connexe de T_{i+1} donc il existe un $W \in V_i, (W, X) \in E_i$ tel que $W \subseteq U$. Par construction, si $X \notin V_{i+1}$, les seuls éléments qui couvrent X sont $X \cup Y$ et Z. Or $(W, X) \in E_i$, nous obtenons donc, par construction, qu'il existe $U' \in E$ tel que $W \cup X \cup Y \subseteq U'$ et $Z \nsubseteq U'$. Or $U' \neq U$ car $Y \nsubseteq U$. $\mathcal{H}$ est clos par intersection et $W \cup X \subseteq U$ et $W \cup X \subseteq U'$ donc il existe $U'' \in E, U'' \subseteq U \cap U', W \cup X \subseteq U''$ et $Y \nsubseteq U'', Z \nsubseteq U''$ ce qui est impossible car Z et $X \cup Y$ sont les seuls éléments qui couvrent X dans $\mathcal{H}$.

— $\exists X_1, \ldots, X_k \in V_i, k > 1, \forall j \leq k, (X, X_j) \in \overrightarrow{E_i}$. Par construction, dans T_{i+1}, le sous-arbre induit par $\overrightarrow{\mathcal{N}_{i+1}(X)} = \{Z \in V_i \mid (X, Z) \in \overrightarrow{E_i}\} \cup \{X \cup Y\}$ est un arbre support de l'hypergraphe dont l'ensemble d'hyperarêtes est $\{A \in E \mid A \subseteq \bigcup_{C \in \overrightarrow{\mathcal{N}_i(X)}} C, \exists B \in \overrightarrow{\mathcal{N}_{i+1}(X)}, B \subseteq A\}$ donc si T_i^U est une partie connexe de T_i et T_{i+1}^U n'est pas une partie connexe de T_{i+1}, alors U fait partie d'un cycle. L'algorithme prend un arbre couvrant maximum du graphe support pondéré par le nombre d'hyperarêtes contenant chaque arête donc un nombre minimum d'hyperarêtes est supprimé.

□

À partir de ce résultat, le même processus que celui expliqué dans la Section 5.5 peut être appliqué afin d'obtenir une approximation par un hypergraphe binaire.

La Figure 5.9 présente le diagramme de Hasse d'un hypergraphe non totalement équilibré et une approximation par un hypergraphe binaire obtenue par la construction représentée sur la Figure 5.10. Cet hypergraphe n'est pas totalement équilibré mais est un hyperarbre. En effet, l'arbre T_0 de la Figure 5.10 est un arbre support de cet hypergraphe. L'hypergraphe approximé contient toutes les hyperarêtes de l'hypergraphe initial sauf $\{2, 3, 4\}$. Cette hyperarête est une partie connexe de T_2 mais pas de T_3. En effet, lors du passage de T_2 à T_3, le nœud 2 disparaît et on construit donc un arbre support des hyperarêtes sur les voisins sortants de 2 : $\{1, 2\}$, $\{2, 3\}$ et $\{2, 4\}$. L'hypergraphe contenant les hyperarêtes $\{1, 2, 3\}, \{2, 3, 4\}$ et $\{1, 2, 4\}$, il n'est pas possible de construire un arbre en conservant la connexité de toutes les hyperarêtes. En effet, le graphe support est un triangle.

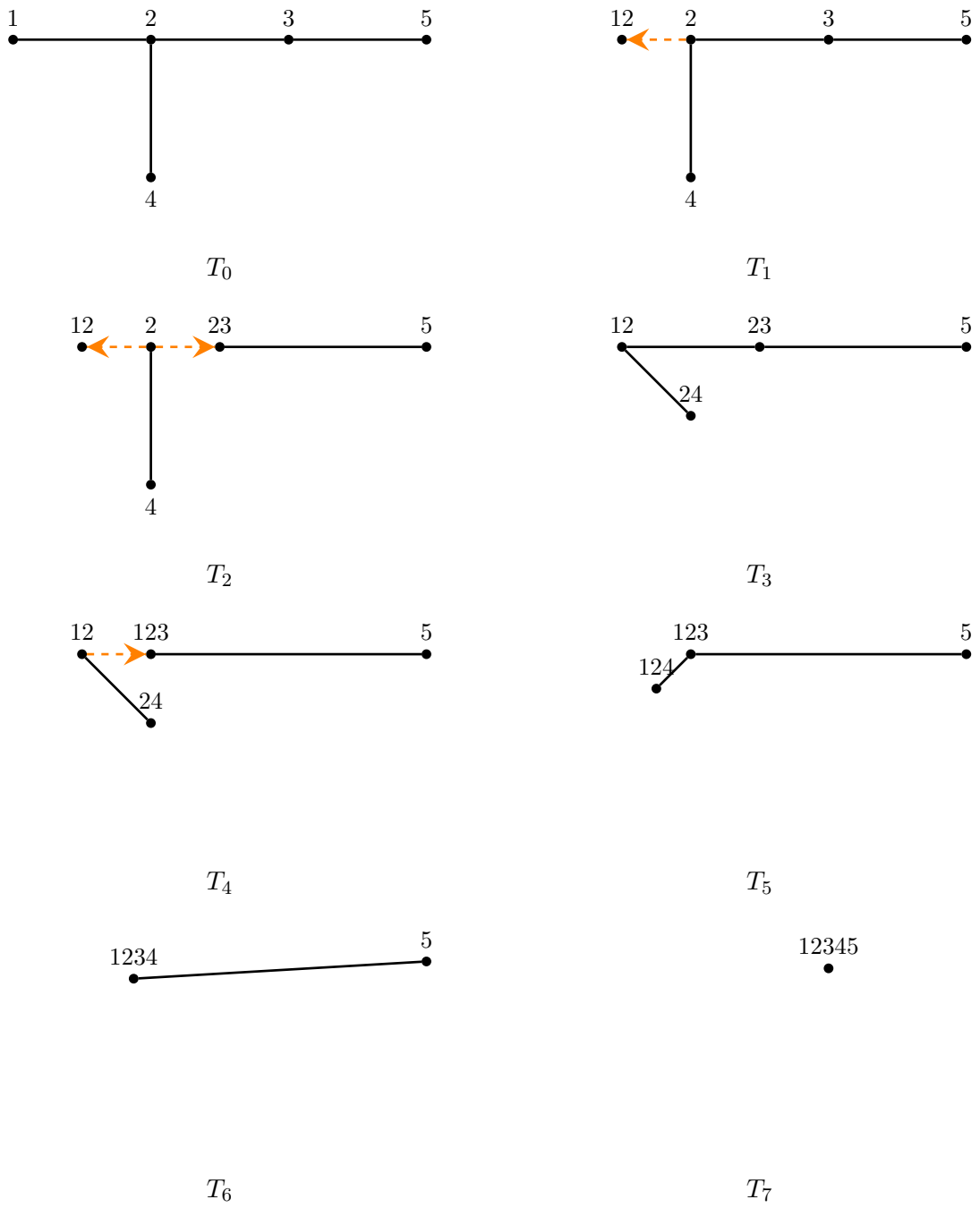


(a) Diagramme de Hasse d'un hypergraphe non totalement équilibré

(b) Approximation possible de l'hypergraphe par un hypergraphe binaire

Le nœud orange est celui supprimé par l'approximation.

FIGURE 5.9 – Exemple d'approximation d'un hypergraphe non totalement équilibré par un hypergraphe binaire



Les arêtes orientées sont représentées par des flèches pointillées.

FIGURE 5.10 – Exemple de binarisation de l'hypergraphe dont le diagramme de Hasse est représenté sur la Figure 5.9 par la construction de l'Algorithme 13

5.8 Conclusion

Ce chapitre définit les hypergraphes binaires et binarisables, équivalents aux treillis binaires et binarisables présentés dans le Chapitre 4. Nous proposons ensuite une construction permettant de caractériser les hypergraphes binaires par un procédé semblable à celui proposé par Lehel [32]. Le processus consiste en effet, à partir d'un arbre, à contracter des arêtes de cet arbre pour obtenir les différentes hyperarêtes de l'hypergraphe final.

L'algorithme présenté pour la construction de la séquence d'arbres associée à un hypergraphe binaire donné permet de plus d'approximer un hypergraphe totalement équilibré par un hypergraphe binaire. Ce résultat est une généralisation de la binarisation proposée dans le Chapitre 4. Notre algorithme permet également l'approximation d'un hypergraphe clos quelconque par un hypergraphe binaire, augmentant encore les possibilités d'approximation en un système binaire.

Enfin, cette construction permet de démontrer le résultat d'équivalence entre les systèmes binaires et les systèmes totalement équilibrés déjà démontrés dans le Chapitre 4 par une méthode différente formalisée par les hypergraphes.

Chapitre 6

Dissimilarités totalement équilibrées

6.1 Introduction

Ce chapitre aborde le problème de classification sous un angle métrique. Nous étudions ici la classification à partir d'une dissimilarité. Une dissimilarité est une représentation d'un ensemble d'individus par la distance qui les sépare. Chaque couple d'individus se voit associé une valeur mesurant la différence entre ces individus. Les dissimilarités permettent donc de représenter l'intensité d'un lien entre deux individus. Il est possible d'associer à une dissimilarité un système de classes. Il s'agit donc de construire un modèle global de classification à partir d'une description locale des données. Nous nous intéressons dans ce chapitre aux dissimilarités associées à un système de classes totalement équilibré et les définissons comme étant les dissimilarités totalement équilibrées.

En théorie, nous caractérisons les dissimilarités dont le système de classes associé est totalement équilibré et introduisons la notion de dissimilarité totalement équilibrée. Nous ajoutons donc une structure manquante aux équivalences entre tous les systèmes totalement équilibrés précédemment traités. Nous permettons de plus une approche locale de la classification en modèles totalement équilibrés. Nous donnons un algorithme polynomial permettant de calculer le système de classes associé à une dissimilarité totalement équilibrée et un algorithme polynomial de détection de dissimilarité totalement équilibrée.

En pratique, nous proposons un algorithme polynomial d'approximation d'une dissimilarité quelconque par une dissimilarité totalement équilibrée et permettons de produire un système de classes totalement équilibré à partir de données décrites par une dissimilarité.

Ce chapitre s'organise de la manière suivante. La Section 6.2 donne la définition des dissimilarités totalement équilibrées et montre leur équivalence avec les autres systèmes totalement équilibrés comme les hypergraphes. La Section 6.3 donne une manière efficace d'obtenir les classes du système de classes associé à une dissimilarité totalement équilibrée en utilisant un nouveau type de dissimilarités : les dissimilarités cordées. Nous décrivons ensuite un algorithme permettant de déterminer si une dissimilarité donnée est totalement équilibrée ou non (Section 6.4) et un algorithme d'approximation si ce n'est pas le cas

(Section 6.5). Enfin, nous donnons un exemple illustratif de l'utilisation de ces algorithmes sur un ensemble de données réelles dans la Section 6.6.

6.2 Dissimilarités totalement équilibrées

Nous définissons dans un premier temps deux nouveaux types de dissimilarités : les dissimilarités totalement équilibrées et les dissimilarités cordées, pouvant être associées respectivement aux autres systèmes totalement équilibrés déjà étudiés dans les chapitres précédents et aux graphes cordés.

Nous rappelons en premier lieu que l'hypergraphe des classes associé à une dissimilarité d est noté $\mathcal{C}(d)$ et l'hypergraphe des boules $\mathcal{B}(d)$ (Section 1.1.5). Nous rappelons de plus qu'un ordre $v_1 < \cdots < v_n$ est dit totalement équilibré pour un hypergraphe donné $\mathcal{H} = (V, E)$ si l'ensemble $\{X_{|v_i,\ldots,v_n} \mid v_i \in X, \}$ est une chaîne pour tout $1 \leq i \leq n$.

Définition 6.2.1. Une dissimilarité est dite *totalement équilibrée* si son hypergraphe des classes $\mathcal{C}(d)$ est totalement équilibré. Un ordre $v_1, \ldots, v_n$ sur V est dit *totalement équilibré* pour d si il est totalement équilibré pour $\mathcal{C}(d)$.

Définition 6.2.2. Étant donné une dissimilarité d sur V, un ordre $v_1, \ldots, v_n$ de V est appelé *ordre cordé* si

$$\forall i \leq j, k, d(v_j, v_k) \leq \max\{d(v_i, v_j), d(v_i, v_k)\}.$$

Une dissimilarité est dite *cordée* si elle admet un ordre cordé.

Les propositions suivantes remarquent que pour une dissimilarité d donnée, les ordres totalement équilibrés de d sont aussi des ordres cordés de d et caractérisent les ordre cordés qui sont totalement équilibrés.

Proposition 6.2.1. *Soit d une dissimilarité sur un ensemble V et $v_1, \ldots, v_n$ un ordre totalement équilibré pour d. $v_1, \ldots, v_n$ est un ordre cordé pour d.*

Démonstration. Supposons que l'ordre sur V est totalement équilibré pour $\mathcal{C}(d)$ mais pas cordé pour d. Il existe $i < j, k$ tel que

$$d(v_j, v_k) > \max\{d(v_i, v_j), d(v_i, v_k)\}.$$

Il existe donc deux cliques maximales X et Y du graphe seuil $G_{\max\{d(v_i,v_j),(d(v_i,v_k)\}}$ telles que $v_i \in X, v_j \in X, v_k \notin X$ et $v_i \in Y, v_k \in Y, v_j \notin Y$. L'ordre n'est donc pas totalement équilibré pour $\mathcal{C}(d)$. □

Proposition 6.2.2. *Soit d une dissimilarité sur un ensemble V. Un ordre $v_1, \ldots, v_n$ sur V est totalement équilibré pour la dissimilarité d ssi :*

1. *$v_1, \ldots, v_n$ est cordé pour d,*

2. *pour tout 5-tuple* (g,h,i,j,k) *avec* $g,h \leq i < j,k$:

$$\begin{cases} d(v_g,v_k) \leq \max\{d(v_g,v_i),d(v_g,v_j)\} \\ ou \\ d(v_h,v_j) \leq \max\{d(v_h,v_i),d(v_h,v_k)\}. \end{cases}$$

Démonstration. D'après la Proposition 6.2.1, si $v_1,\ldots,v_n$ est totalement équilibré pour d, alors la première condition est vérifiée.

Montrons à présent que la deuxième condition est aussi vérifiée. Supposons que l'ordre est totalement équilibré et cordé pour d mais qu'il existe $g,h \leq i \leq j,k$ tel que

$$\begin{cases} d(v_g,v_k) > \max\{d(v_g,v_i),d(v_g,v_j)\} \\ et \\ d(v_h,v_j) > \max\{d(v_h,v_i),d(v_h,v_k)\}. \end{cases}$$

Il existe alors une clique maximale X du graphe seuil $G_{\max\{d(v_g,v_i),d(v_g,v_j)\}}$ telle que :

— $v_g,v_i,v_j \in X$, car l'ordre est cordé,

— $v_k \notin X$,

et une clique maximale Y du graphe seuil $G_{\max\{d(v_h,v_i),d(v_h,v_k)\}}$ telle que :

— $v_h,v_i,v_k \in Y$, car l'ordre est cordé,

— $v_j \notin Y$.

L'ordre n'est donc pas totalement équilibré pour $\mathcal{C}(d)$. Nous avons donc montré que si $v_1 < v_2 < \cdots < v_n$ sur V est totalement équilibré pour $\mathcal{C}(d)$, alors les deux conditions sont satisfaites.

Montrons à présent l'autre implication. Supposons que l'ordre n'est pas totalement équilibré et montrons qu'au moins une des deux conditions n'est pas respectée. Soit i un indice tel que $\{Z \mid v_i \in Z, Z \in \mathcal{C}(d)_{|v_i,\ldots,v_n}\}$ n'est pas une chaîne. Il existe $X,Y \in \mathcal{C}(d)$ tels que $X_{|v_i,\ldots,v_n} \parallel Y_{|v_i,\ldots,v_n}$. Il existe donc $j,k > i$ tels que $v_j \in X\backslash Y$ et $v_k \in Y\backslash X$. Supposons sans perte de généralité que $diam(X) \geq diam(Y)$.

Si l'ordre n'est pas cordé alors la condition 1 n'est pas respectée et notre résultat est démontré. Supposons à présent que l'ordre est cordé et montrons que la condition 2 n'est pas vérifiée.

Nous avons montré que $v_k \in Y\backslash X$ et $v_j \in X\backslash Y$. De plus, X et Y sont des cliques maximales d'un graphe seuil de d donc il existe $v_g \in X$ et $v_h \in Y$ tels que $d(v_g,v_k) > diam(X)$ et $d(v_h,v_j) > diam(Y)$.

De plus, $i > g$, car si $i \leq g$ alors, par définition des ordres cordés, $d(v_g,v_k) \leq \max\{d(v_g,v_i),d(v_k,v_i)\} \leq diam(X)$ (car $v_i,v_g \in X$ et $v_i,v_k \in Y$) ce qui est une contradiction.

Si $h < i$, le 5-tuple (g,h,i,j,k) ne vérifie pas la condition 2. En effet, $v_g,v_i,v_j \in X$ donc $\max\{d(v_g,v_i),d(v_g,v_j)\} \leq diam(X)$ donc $d(v_g,v_k) \nleq \max\{d(v_g,v_i),d(v_g,v_j)\}$. Symétriquement pour $v_h,v_i,v_j \in Y$, $d(v_h,v_j) \nleq \max\{d(v_h,v_i),d(v_h,v_k)\}$.

Enfin, si $i < h$, $d(v_h,v_j) \leq \max\{d(v_i,v_j),d(v_i,v_h)\}$ car l'ordre est cordé. Or $d(v_i,v_h) \leq diam(Y) < d(v_h,v_j)$. Donc $d(v_h,v_j) \leq d(v_i,v_j)$ et $d(v_i,v_j) > diam(Y)$. Considérons alors

le tuple (g, i, i, j, k). On a $d(v_g, v_k) > diam(X)$ donc

$$d(v_g, v_k) \not\leq \max\{d(v_g, v_i), d(v_i, v_j)\},$$

car $v_i, v_j, v_g \in X$ donc $\max\{d(v_g, v_i), d(v_i, v_j)\} \leq diam(X)$. De plus,

$$d(v_i, v_j) \not\leq \max\{d(v_i, v_i), d(v_i, v_k)\} = d(v_i, v_k),$$

car $d(v_i, v_j) > diam(Y)$. La condition 2 n'est donc pas vérifiée. □

Comme évoqué dans la Section 1.1.5, un graphe peut être vu comme une dissimilarité. Dans ce cas, les ordres cordés pour les dissimilarités sont une généralisation des ordres simpliciaux et les dissimilarités admettant un ordre cordé sont une généralisation des graphes cordés. Dans le cas des dissimilarités associées à un graphe, [21] montre que $\mathcal{B}(d_G)$ totalement équilibré équivaut à $\mathcal{C}(d_G)$ totalement équilibré. Nous montrons ici que cela n'est plus vrai que dans un sens pour une dissimilarité quelconque grâce à la proposition suivante. Le contre-exemple donné dans la Section 1.1.5 sur la Table 1.2 illustre que l'autre implication n'est pas vraie.

Proposition 6.2.3. *Soit d une dissimilarité sur V et $v_1 < v_2 < \cdots < v_n$ un ordre totalement équilibré pour $\mathcal{B}(d)$. $v_1 < v_2 < \cdots < v_n$ est totalement équilibré pour $\mathcal{C}(d)$.*

Démonstration. Soit $v_1 < v_2 < \cdots < v_n$ un ordre totalement équilibré pour $\mathcal{B}(d)$. Soit v_i, v_j, v_k trois éléments de V tels que $i < j, k$. Si $d(v_j, v_k) > \max\{d(v_i, v_j), d(v_i, v_k)\}$ alors $v_k \notin B(v_j, d(v_i, v_j))$ et $v_j \notin B(v_i, d(v_i, v_k))$. Or v_i est dans les deux boules donc l'ordre n'est pas totalement équilibré pour $\mathcal{B}(d)$ ce qui est une contradiction. Donc pour tout $i < j, k$, $d(v_j, v_k) \leq \max\{d(v_i, v_j), d(v_i, v_k)\}$ donc l'ordre est cordée et donc la première condition de la Proposition 6.2.2 est vérifiée.

Soit (g, h, i, j, k) un 5-tuple qui ne vérifie pas la condition 2 de la Proposition 6.2.2. Nous avons $v_k \notin B(v_g, \max\{d(v_g, v_i), d(v_g, v_j)\})$ et $v_j \notin B(v_h, \max\{d(v_h, v_i), d(v_h, v_k)\})$. v_i est dans ces deux boules donc l'ordre n'est pas totalement équilibré pour $\mathcal{B}(d)$. □

La bijection entre les hypergraphes totalement équilibrés et les dissimilarités totalement équilibrés est un cas particulier d'une des bijections générales de Bertrand [10] pour le cas des hiérarchies faibles. Il est possible d'associer à tout hypergraphe $\mathcal{H} = (V, E)$ donné une valuation $f : E \to \mathbb{R}_+$. Le couple $(\mathcal{H}, f)$ est alors appelé un *hypergraphe valué*. La valuation f est alors appelée *index strict* si pour tout $X, Y \in E$, $X \subsetneq Y$ implique $f(X) < f(Y)$.

Théorème 6.2.4 (Bertrand [10])**.** *Soit $\mathcal{H} = (V, E)$ un hypergraphe contenant les singletons et $\{V\}$ comme hyperarêtes et tel que pour tout $X, Y, Z \in E$, $X \cap Y \cap Z \in \{X \cap Y, X \cap Z, Y \cap Z\}$. Soit f un index strict de $\mathcal{H}$ tel que $f(\{x\}) = 0$ pour tout $x \in V$. Alors la dissimilarité d définie pour tout $x, y \in V$ par $d(x, y) = \min\{f(X) \mid X \in E, x, y \in X\}$ est telle que*

— $\mathcal{C}(d) = E$,
— *pour tout* $X \in E, f(X)$ *est le diamètre de* X *pour* d.

Nous pouvons immédiatement déduire de ce théorème la proposition suivante :

Corollaire. *Soit* $\mathcal{H} = (V, E)$ *un hypergraphe contenant les singletons et* $\{V\}$ *comme hyperarêtes et tel que pour tout* $X, Y, Z \in E$, $X \cap Y \cap Z \in \{X \cap Y, X \cap Z, Y \cap Z\}$. *Soit* f *un index strict de* $\mathcal{H}$ *tel que* $f(\{x\}) = 0$ *pour tout* $x \in V$. *Alors la dissimilarité* d *définie pour tout* $x, y \in V$ *par* $d(x, y) = \min\{f(X) \mid X \in E, x, y \in X\}$ *est telle que* $\mathcal{C}(d) = E$.

Inversement, pour toute dissimilarité totalement équilibrée d *sur* V :
— $(V, \mathcal{C}(d))$ *est un hypergraphe totalement équilibré contenant les singletons et* $\{V\}$ *comme hyperarêtes,*
— *le diamètre est un index strict de* $(V, \mathcal{C}(d))$.

6.3 Algorithme de calcul des classes

Dans le cas général, une dissimilarité peut avoir un nombre exponentiel de classes. Nous montrons dans cette section qu'une dissimilarité cordée sur un ensemble V ne peut en avoir au maximum que $|V|^2$. Nous donnons également un algorithme en $\mathcal{O}(|V|^3)$ pour calculer les classes d'une dissimilarité. Cet algorithme est linéaire en la taille de la sortie dans le pire des cas ($\mathcal{O}(|V|^2)$ classes, toutes de taille $\mathcal{O}(|V|)$). Les dissimilarités totalement équilibrées étant un cas particulier des dissimilarités cordées, l'algorithme permet de calculer en temps linéaire les classes d'une dissimilarité totalement équilibrée.

Lemme 6.3.1. *Soit* d *une dissimilarité sur un ensemble* V *admettant* $v_1 < \cdots < v_n$ *pour ordre cordé. Soit* $\mathcal{C}(d)$ *l'ensemble des classes de* d. *Pour tout* $X \in \mathcal{C}(d), X = B(v^\star, \alpha)_{|v \geq v^\star}$ *avec :*
— $v^\star$ *le plus petit élément de* X *selon l'ordre cordé,*
— α *le diamètre de* X *selon la dissimilarité* d.

Démonstration. Pour tout $v \in X, v \geq v^\star$ par définition de $v^\star$ et $d(v, v^\star) \leq \alpha$ par définition de α. Donc $X \subseteq B(v^\star, \alpha)_{|v \geq v^\star}$.

Pour tout $v, w \in B(v^\star, \alpha)_{|v \geq v^\star}$, $v, w \geq v^\star$ donc par définition d'un ordre cordé, $d(v, w) \leq \max\{d(v^\star, v), d(v^\star, w)\}$. Or $d(v^\star, v) \leq \alpha$ et $d(v^\star, w) \leq \alpha$ car $v, w \in B(v^\star, \alpha)_{|v \geq v^\star}$. Donc $d(v, w) \leq \alpha$. Donc $v, w \in X$. Donc $B(v^\star, \alpha)_{|v \geq v^\star} \subseteq X$.

Donc $X = B(v^\star, \alpha)_{|v \geq v^\star}$. □

Le Lemme 6.3.1 montre que les cliques maximales d'une dissimilarité cordée d sont ce que nous appellerons à présent des *boules tronquées i.e.* des boules restreintes aux éléments plus grands que le centre de la boule selon un ordre donné. Nous avons donc $\mathcal{C}(d) \subseteq \{B(v, d(v, w))_{|x \geq v} \mid v, w \in V, v \leq w\}$. Le nombre de classes d'une dissimilarité cordée est donc borné par $\mathcal{O}(|V|^2)$.

d

	x	y	z	t	u
x	0				
y	3	0			
z	4	4	0		
t	5	2	5	0	
u	3	3	4	5	0

$B(a, d(a,b))_{\mid v \geq a}$

a	x	x	x	x	z	z	z	t	t	y
b	z	t	y	u	t	y	u	y	u	u
x	×	×	×	×						
z	×	×			×	×	×			
t		×			×			×	×	
y	×	×	×	×	×	×	×	×	×	×
u	×	×	×	×	×	×	×		×	×
diamètre	4	5	3	3	5	4	4	2	5	3

$\mathcal{C}(d)$

x	×	×	×	
z	×	×		
t		×		×
y	×	×	×	×
u	×	×	×	
diamètre	4	5	3	2

TABLE 6.1 – Exemple de dissimilarité admettant $x < z < t < y < u$ pour ordre cordé avec ses boules et classes.

La Table 6.1 donne un exemple de dissimilarité admettant un ordre cordé ainsi que toutes ses boules tronquées et ses classes. On remarque bien que toutes les classes sont des boules tronquées comme le montre le Lemme 6.3.1. Nous pouvons par contre remarquer que l'inverse n'est pas vrai sur cet exemple. La proposition suivante caractérise les boules tronquées qui sont des cliques maximales.

Théorème 6.3.2. *Soit d une dissimilarité sur un ensemble V de taille n admettant $v_1 < \cdots < v_n$ pour ordre cordé. La boule tronquée $B(v_i, d(v_i, v_j))_{\mid v \geq v_i}$ avec $i \leq j$ est une clique maximale de d de diamètre $d(v_i, v_j)$ si pour tout $k < i$:*

$$\max\{d(v_k, v_i), d(v_k, v_j)\} = d(v_i, v_j) \implies |B(v_k, d(v_i, v_j))_{\mid v \geq v_i}| < |B(v_i, d(v_i, v_j))_{\mid v \geq v_i}|$$

Démonstration. Soient i, j tels que $i \leq j$. Afin d'alléger les notations, nous noterons $\alpha = d(v_i, v_j)$. Montrons la contraposée de ce résultat :

$$B(v_i, \alpha) \notin \mathcal{C}(d) \implies \exists k < i, \max\{d(v_k, v_i), d(v_k, v_j)\} = \alpha \text{ et } |B(v_k, \alpha)_{\mid v \geq v_i}| \geq |B(v_i, \alpha)_{\mid v \geq v_i}|.$$

Soit $k < i$ tel que $\max\{d(v_k, v_i), d(v_k, v_j)\} = \alpha$. Remarquons tout d'abord que, comme l'ordre est cordé, pour tout $l > k$ tel que $v_l \in B(v_k, \alpha)_{\mid v \geq v_i}, d(v_i, v_l) \leq \max\{d(v_i, v_k), d(v_l, v_k)\} \leq \alpha$. Donc $v_l \in B(v_i, \alpha)_{\mid v \geq v_i}$. Donc $B(v_k, \alpha)_{\mid v \geq v_i} \subseteq B(v_i, \alpha)_{\mid v \geq v_i}$. Donc $|B(v_k, \alpha)_{\mid v \geq v_i}| \leq |B(v_i, \alpha)_{\mid v \geq v_i}|$. Nous pouvons donc simplifier le résultat à prouver :

$$B(v_i, \alpha) \notin \mathcal{C}(d) \implies \exists k < i, \max\{d(v_k, v_i), d(v_k, v_j)\} = \alpha \text{ et } |B(v_k, \alpha)_{\mid v \geq v_i}| = |B(v_i, \alpha)_{\mid v \geq v_i}|.$$

Supposons que $B(v_i, \alpha)_{\mid v \geq v_i}$ n'est pas une clique maximale. En utilisant une partie de la preuve du Lemme 6.3.1, on a $B(v_i, \alpha)_{\mid v \geq v_i}$ est une clique, car pour tout $v, w \in B(v_i, \alpha)_{\mid v \geq v_i}, d(v, w) \leq \alpha$. Donc $B(v_i, \alpha)_{\mid v \geq v_i}$ est inclus strictement dans une clique maximale. D'après le Lemme 6.3.1, cette clique maximale est une boule tronquée de centre strictement inférieur à i et de diamètre α :

$$\exists k < i, B(v_i, \alpha)_{\mid v \geq v_i} \subsetneq B(v_k, \alpha)_{\mid v \geq v_k}.$$

De plus, $v_i, v_j \in B(v_i, \alpha)_{|v \geq v_i}$, donc $v_i, v_j \in B(v_k, \alpha)_{|v \geq v_k}$. Donc $d(v_i, v_k) \leq \alpha$ et $d(v_j, v_k) \leq \alpha$.

L'ordre est cordé et $k < i \leq j$ donc $d(v_i, v_j) \leq \max\{d(v_k, v_i), d(v_k, v_j)\}$. Or $d(v_i, v_j) = \alpha$ par définition et $d(v_i, v_k) \leq \alpha$, $d(v_j, v_k) \leq \alpha$ donc $\max\{d(v_k, v_i), d(v_k, v_j)\} = d(v_i, v_j)$. Comme mentionné précédemment, $B(v_k, \alpha)_{|v \geq v_i} \subseteq B(v_i, \alpha)_{|v \geq v_i}$. De plus, $B(v_i, \alpha)_{|v \geq v_i} \subsetneq B(v_k, \alpha)_{|v \geq v_k}$ et $k < i$ donc $B(v_i, \alpha)_{|v \geq v_i} \subseteq B(v_k, \alpha)_{|v \geq v_i}$. Donc $B(v_i, \alpha)_{|v \geq v_i} = B(v_k, \alpha)_{|v \geq v_i}$. Donc $|B(v_i, \alpha)_{|v \geq v_i}| = |B(v_k, \alpha)_{|v \geq v_k}|$. □

Dans cette proposition, nous regardons des indices i, j, k tels que $k < i \leq j$. Par définition d'un ordre cordé, on a $d(v_i, v_j) \leq \max\{d(v_i, v_k), d(v_j v_k)\}$. La condition $\max\{d(v_i, v_k), d(v_j v_k)\} = d(v_i, v_j)$ revient donc à ajouter $\max\{d(v_i, v_k), d(v_j, v_k)\} \leq d(v_i, v_j)$ *i.e.* $v_i, v_j \in B(v_k, \alpha)_{|v \geq v_k}$, ce qui est nécessaire pour que cette boule contienne la clique entière (la boule $B(v_i, \alpha)_{|v \geq v_i}$). On cherche donc parmi les boules de même rayon contenant v_i et v_j, celles qui contiennent la clique entière. Le Théorème 6.3.2 indique qu'il suffit de s'intéresser aux nombres d'éléments d'une partie d'une boule.

L'Algorithme 14 utilise le Lemme 6.3.1 et le Théorème 6.3.2 pour calculer les classes d'une dissimilarité étant donné un ordre cordé. Il itère sur toutes les boules possibles et conserve celles qui sont effectivement des classes grâce au résultat du Théorème 6.3.2. Pour réaliser cette tâche efficacement, l'algorithme maintient à jour des nombres $S_k(t)$ pour $1 \leq k, t \leq n$ définis par :

— À l'étape i, s'il existe une boule tronquée $B(v_k, \alpha)_{|v \leq v_k}$ telle que $B(v_k, \alpha)_{|v \leq v_i}$ est de cardinal t alors $S_k(t) = \alpha$.

— À l'étape i, s'il n'existe aucune boule tronquée réalisant cette condition, $S_k(t) = -1$.

D'après le Théorème 6.3.2, étant donné une boule tronquée $C_{ij} = B(v_i, d(v_i, v_j))_{|v \geq v_i}$ avec $i \leq j$, C_{ij} est une clique maximale s'il n'existe aucun $k < i$ tel que $d(v_k, v_i) \leq d(v_k, v_j)$ et $S_k(|C_{ij}|) = d(v_i, v_j)$.

6.4 Reconnaissance

Nous donnons dans cette section un algorithme en $\mathcal{O}(|V|^3)$ permettant de déterminer si une dissimilarité donnée est cordée ou totalement équilibrée. Cet algorithme vérifie en premier lieu si la dissimilarité est cordée ou non en cherchant un de ses ordres cordés (Algorithme 15). Il calcule ensuite son hypergraphe des classes et sa matrice binaire associée. Il vérifie enfin en temps linéaire (Spinrad [42]) que la matrice réordonnée de manière doublement lexicale est Γ-free.

6.4.1 Ordres cordés

Pour trouver un ordre cordé pour une dissimilarité cordée, nous pouvons procéder récursivement grâce au Lemme 6.4.1.

Algorithme 14 : Algorithme de calcul des classes d'une dissimilarité

Données : Une dissimilarité d sur un ensemble V de taille n et un de ses ordres cordés $v_1 < \cdots < v_n$

Résultat : L'ensemble de classes $\mathcal{C}(d)$

```
1  C ← ∅
2  S_1(t) ← −1 for 1 ≤ t ≤ n
3  pour j = 1 à n faire
4  |  ajouter B(v_1, d(v_1, v_j)) à C
5  |  S_1(|B(v_1, d(v_1, v_j))| − 1) = d(v_1, v_j)
6  fin
7  pour i = 2 à n faire
8  |  V_i = {v_i, ..., v_n}
9  |  S_i(t) = −1 pour all 1 ≤ t ≤ n
10 |  pour j = i à n faire
11 |  |  C_ij = B(v_i, d(v_i, v_j)) ∩ V_i
12 |  |  si ∄k < i tel que d(v_k, v_i) ≤ d(v_i, v_j) et S_k(|C_ij|) = d(v_i, v_j) alors
   |  |  |  // C_ij n'est pas inclus dans une classe de même diamètre :
   |  |  |     c'est donc une classe
13 |  |  |  ajouter C_ij à C
14 |  |  |  S_i(|C_ij| − 1) = d(v_i, v_j)
15 |  |  fin
16 |  fin
17 |  pour j = 1 à i − 1 faire
18 |  |  pour t = 1 à n faire
19 |  |  |  si S_j(t) ≥ d(v_j, v_i) alors
20 |  |  |  |  si t > 1 et S_j(t − 1) = −1 alors
21 |  |  |  |  |  S_j(t − 1) = S_j(t)
22 |  |  |  |  fin
23 |  |  |  |  S_j(t) = −1
24 |  |  |  fin
25 |  |  fin
26 |  fin
27 fin
28 retourner C
```

Lemme 6.4.1. *Soit d une dissimilarité sur un ensemble V. Si d admet un ordre cordé alors pour tout $x \in V$ tel que, pour tout $y, z \in V, d(y,z) \leq \max\{d(x,y), d(x,z)\}$, la restriction de d à $V\backslash\{x\}$ admet un ordre cordé.*

Démonstration. Soit $v_1 < v_2 < \cdots < v_n$ un ordre cordé pour d. Si il existe i tel que pour tout $j, k \neq i, d(v_j, v_k) \leq \max\{d(v_i, v_j), d(v_i, v_k)\}$ alors $v_i < v_1 < \cdots < v_{i-1} < v_{i+1} < \cdots < v_n$ est un ordre cordé de d. Donc $v_1 < \cdots < v_{i-1} < v_{i+1} < \cdots < v_n$ est un ordre cordé de d restreint à $V\backslash\{v_i\}$. □

L'Algorithme 15 met en œuvre cette proposition de manière itérative pour trouver un ordre cordé d'une dissimilarité donnée.

Proposition 6.4.2. *Soit d une dissimilarité sur un ensemble V. L'Algorithme 15 renvoie un ordre cordé de d s'il en existe un en $\mathcal{O}(|V|^3)$ opérations.*

Démonstration. À chaque étape de l'algorithme, $N_i(v)$ est le nombre de paires (ordonnées) $\{x, y\} \subset V$ pour lesquelles $d(x,y) > \max\{d(v,x)d(v,y)\}$. D'après le Lemme 6.4.1, l'Algorithme 15 renvoie donc bien un ordre cordé s'il existe. La complexité est clairement $\mathcal{O}(|V|^3)$. □

La Table 6.2 montre la progression des différents N_i lors du calcul de l'ordre cordé $x < z < t < y < u$ pour la dissimilarité d de la Table 6.1.

6.4.2 Ordres totalement équilibrés et matrices Γ-free

La Section 1.1.2 donne certains faits sur les matrices et notamment le Théorème 1.3.5 dont on peut déduire la méthode suivante pour déterminer si un hypergraphe $\mathcal{H}$ est totalement équilibré :

— représenter $\mathcal{H}$ par sa matrice binaire associée $\mathcal{M}(\mathcal{H})$;
— réordonner $\mathcal{M}(\mathcal{H})$ doublement lexicalement ;
— vérifier si la matrice réordonnée est Γ-free.

Comme expliqué dans la Section 1.1.2, deux algorithmes de Spinrad permettent de déterminer un ordre doublement lexical pour une matrice donnée et de vérifier si la matrice est Γ-free en temps $\mathcal{O}(nm)$ pour une matrice de taille $n \times m$.

Cette procédure permet donc de déterminer si un hypergraphe $\mathcal{H} = (V, E)$ donné est totalement équilibré en $\mathcal{O}(|V| \times |E|)$. Comme remarqué dans la Section 1.1.1, un hypergraphe totalement équilibré a au maximum $\mathcal{O}(|V|^2)$ hyperarêtes, la complexité de l'algorithme est donc $\mathcal{O}(|V|^3)$.

Les Propositions 6.4.3 et 6.4.4 montrent que cela permet également de trouver les ordres totalement équilibrés d'un hypergraphe totalement équilibré. Ces deux propositions permettent de lier les ordres totalement équilibrés et les ordres Γ-free.

Proposition 6.4.3. *Soit $\mathcal{M}$ une matrice $n \times m$ Γ-free. L'ordre de ses lignes est totalement équilibré pour $\mathcal{H}(\mathcal{M})$.*

Algorithme 15 : Construction d'un ordre cordé associé à une dissimilarité

Données : Une dissimilarité d sur un ensemble V
Résultat : Un ordre cordé pour d si d en admet un

$V_1 = V$
pour $v \in V$ **faire**
  $N_1(v) = 0$
  pour $x \neq y \in V$ **faire**
    si $d(x,y) > \max\{d(v,x), d(v,y)\}$ **alors**
      $N_1(v) = N_1(v) + 1$
    fin
  fin
fin
$i = 1$
tant que $V_i \neq \emptyset$ **faire**
  Soit $v^* \in V_i$ tel que $N_i(v^*) = \min_{v \in V_i} N_i(v)$
  si $N_i(v^*) > 0$ **alors**
    Echec
  fin
  $v_i = v^*$
  $V_{i+1} = V_i \backslash \{v^*\}$
  pour $v \in V_{i+1}$ **faire**
    $N^* = 0$
    pour $w \in V_{i+1}$ **faire**
      si $d(v^*, w) > \max\{d(v, v^*), d(v, w)\}$ **alors**
        $N^* = N^* + 1$
      fin
    fin
    $N_{i+1}(v) = N_i(v) - 2N^*$
  fin
  $i = i + 1$
fin
retourner $v_1 < \cdots < v_n$

	N_1	N_2	N_3	N_4	N_5
x	0				
z	0	0			
t	0	0	0		
y	6	4	2	0	
u	0	0	0	0	0

TABLE 6.2 – Valeurs de N_i pendant le calcul de l'ordre cordé $x < z < t < y < u$ de la dissimilarité d de la Table 6.1.

Démonstration. Soit $\mathcal{M}$ une matrice Γ-free. Pour tout $1 \leq j \leq m$, on note $C_j = \{1 \leq i \leq n \mid \mathcal{M}_{i,j} = 1\}$.

Soient $1 \leq i \leq n$ et $1 \leq j < j' \leq m$ tels que $\mathcal{M}_{i,j} = \mathcal{M}_{i,j'} = 1$.

Par définition d'un ordre Γ-free, pour tout $i' > i$ tel que $\mathcal{M}_{i',j} = 1$, $\mathcal{M}_{i',j'} = 1$. Donc $C_j \cap \{i, \ldots, n\} \subseteq C_{j'} \cap \{i, \ldots, n\}$. Donc $1 < \cdots < n$ est un ordre totalement équilibré de $\mathcal{H}(\mathcal{M})$. □

Proposition 6.4.4. *Soit $\mathcal{H} = (V, E)$ un hypergraphe. Si $\mathcal{H}$ admet un ordre totalement équilibré $v_1 < \cdots < v_n$ alors il existe un ordre Γ-free de $\mathcal{M}(\mathcal{H})$ tel que la ligne i correspond au sommet v_i.*

Démonstration. Pour tout $X, Y \in E$, on note $V_{XY} = \{v_{i_{XY}}, v_{i_{XY}+1}, \ldots, v_n\}$ avec $i_{XY} = \min\{i \mid v_i \in X \cap Y\}$. Si $X \cap Y = \emptyset$, i_{XY} n'existe pas et nous définissons alors $V_{XY} = \emptyset$.

On considère le graphe dirigé G avec pour ensemble de sommets E et tel que pour tout $X, Y \in E, (X, Y)$ est un arc de G si et seulement si $X \cap V_{XY} \subsetneq Y \cap V_{XY}$. On note $X \to Y$ quand (X, Y) est un arc de G. On peut remarquer que si $X \to Y$ alors $V_{XY} \neq \emptyset$ (mais le contraire est faux).

Prouvons à présent que si $X \to Y$ et $Y \to Z$ alors $\neg(Z \to X)$ (il n'y a pas d'arc entre Z et X). Soient $X, Y, Z \in E$ tels que $X \to Y, Y \to Z$.

Si $i_{XY} < i_{YZ}$,

$$X \cap V_{XY} \subsetneq Y \cap V_{XY} \implies X \cap V_{YZ} \subsetneq Y \cap V_{YZ}.$$

Or $Y \cap V_{YZ} \subsetneq Z \cap V_{YZ}$. Donc $X \cap V_{YZ} \subsetneq Z \cap V_{YZ}$. Si i_{XZ} existe, $i_{XZ} \geq i_{YZ}$, car $i_{XY} < i_{YZ}$ et $X \to Y$ donc $X \cap V_{XZ} \subsetneq Z \cap V_{XZ}$ *i.e.* $X \to Z$ et $\neg(Z \to X)$(.

Sinon, si $i_{XY} \geq i_{YZ}$, alors $v_{i_{XY}} \in Z$. Donc i_{XZ} existe et $i_{XZ} \leq i_{XY}$. De plus, $X \to Y$ donc il existe $j > i_{XY}$ tel que $v_j \in Y \backslash X$. Or $Y \to Z$ et $i_{YZ} \leq i_{XY}$ donc $v_j \in Z \backslash X$. Or $v_1 < \cdots < v_n$ est un ordre totalement équilibré pour $\mathcal{H}$ donc soit $X \cap V_{XZ} \subsetneq Z \cap V_{XZ}$, soit $X \cap V_{XZ} = Z \cap V_{XZ}$, soit $Z \cap V_{XZ} \subsetneq X \cap V_{XZ}$. Or $v_j \in V_{XZ}$, car $j > i_{XY} \geq i_{XZ}$, donc $X \cap V_{XZ} \subsetneq Z \cap V_{XZ}$. Donc $X \to Z$ et $\neg(Z \to X)$.

Donc G est un graphe acyclique et admet ainsi un ordre topologique $<_T$ (*i.e.* un ordre $<_T$ sur les sommets de G tel que $X_i \to X_j$ implique $X_i <_T X_j$). En utilisant comme ordre

des lignes l'ordre totalement équilibré et comme ordre des colonnes $<_T$, on obtient une matrice Γ-free. □

6.4.3 Procédure de reconnaissance

Les deux sous-sections précédentes permettent d'obtenir un algorithme simple de reconnaissance des dissimilarités totalement équilibrées en $\mathcal{O}(|V|^3)$. L'Algorithme 16 détaille la procédure.

Algorithme 16 : Reconnaissance de dissimilarité totalement équilibrée

Données : Une dissimilarité d sur un ensemble V

1 Chercher un ordre cordé de d
2 **si** $\nexists$ ordre cordé **alors**
3 | **retourner** "d non cordée donc non totalement équilibrée"
4 **fin**
5 Calculer $\mathcal{M}(\mathcal{C}(d))$
6 Réordonner $\mathcal{M}(\mathcal{C}(d))$ selon un ordre doublement lexical
7 **si** $\mathcal{M}(\mathcal{C}(d))$ *est* Γ-*free* **alors**
8 | **retourner** "d est totalement équilibrée"
9 **sinon**
10 | **retourner** "d est cordée mais non totalement équilibrée"
11 **fin**

La ligne 1 correspond à l'Algorithme 15 tandis que la ligne 5 peut être réalisée par l'Algorithme 14. Comme la matrice $\mathcal{M}(\mathcal{C}(d))$ a $|V|$ lignes et $|V|^2$ colonnes, la ligne 7 s'exécute en $\mathcal{O}(|V|^3)$.

Un ordre doublement lexical de la Table 6.1 est représenté sur la Table 6.3. La matrice est Γ-free ce qui signifie que la dissimilarité de la Table 6.1 est totalement équilibrée.

a	t	x	x	x
b	y	y	z	t
t	×			×
z			×	×
x		×	×	×
u		×	×	×
y	×	×	×	×
diamètre	2	3	4	5

TABLE 6.3 – Réordonnancement doublement lexical de la Table 6.1

6.5 Approximation

Nous présentons dans cette section un algorithme en $\mathcal{O}(|V|^3)$ permettant d'approximer une dissimilarité d sur un ensemble V par une dissimilarité totalement équilibrée. Si la dissimilarité donnée est déjà totalement équilibrée, elle est inchangée. Cette procédure peut être résumée comme suit :

1. Approximer d par une dissimilarité cordée d' (Section 6.5.1).
2. Calculer $\mathcal{M}(d')$ et l'approximer par une matrice Γ-free (Section 6.5.2).
3. Associer une dissimilarité totalement équilibrée à la matrice $n \times m$ valuée $(M', (\alpha_i)_{1 \leq i \leq m})$ où les valuations α_i sont définies par d' (Section 6.5.3).

6.5.1 Approximation par une dissimilarité cordée

La procédure d'approximation d'une dissimilarité d donnée par une dissimilarité cordée consiste en deux étapes :

1. Trouver un ordre total sur V.
2. Approximer d par une dissimilarité cordée compatible avec l'ordre choisi.

Pour trouver un ordre potentiel, il est possible d'utiliser la méthode donnée dans la Section 6.4.1. La dissimilarité donnée peut ne pas être cordée et il convient donc d'adapter l'algorithme en supprimant le test de la ligne 13 de l'Algorithme 15 qui vérifie si $N_d(v^\star)$ est égal à 0 ou non. En conservant à chaque étape un élément $v^\star$ qui réalise un minimum de N_d, nous sommes assurés que l'algorithme retourne un ordre linéaire sur V et que cet ordre est cordé si d est cordée.

Soit d une dissimilarité sur V et $v_1 < \cdots < v_n$ un ordre sur V. L'Algorithme 17 peut être utilisé pour approximer d par une dissimilarité d' cordée admettant l'ordre donné comme ordre cordé.

Proposition 6.5.1. *Soit d une dissimilarité sur V et $v_1 < \cdots < v_n$ un ordre sur V. L'Algorithme 17 renvoie une dissimilarité d' admettant $v_1 < \cdots < v_n$ comme ordre cordé en $\mathcal{O}(|V|^3)$.*

De plus, si d admet $v_1 < \cdots < v_n$ comme ordre cordé, $d' = d$.

Démonstration. Pour tout $1 \leq i^\star < j^\star < k^\star \leq n$, le triplet $(i, j, k) = (i^\star, j^\star, k^\star)$ est atteint par l'algorithme. Si, à ce moment, $d'(v_{j^\star}, v_{k^\star}) \leq \max\{d'(v_{i^\star}, v_{j^\star}), d'(v_{i^\star}, v_{k^\star})\}$, l'algorithme ne fait rien. Sinon, il modifie d' pour vérifier la condition. Ensuite, seule $d'(v_{j^\star}, v_{k^\star})$ peut changer et uniquement diminuer donc la condition reste vérifiée. Donc pour tout $i < j, k, d(v_j, v_k) \leq \max\{d(v_i, v_j), d(v_i, v_k)\}$ et l'ordre est donc cordé.

Si l'ordre donné est déjà cordé, l'algorithme n'entre jamais dans le "si" et laisse donc d inchangée. □

Algorithme 17 : Approximation par une dissimilarité cordée d'une dissimilarité donnée relativement à un ordre

Données : Une dissimilarité d sur un ensemble V de taille n et un ordre linéaire $v_1 < \cdots < v_n$ sur V

Résultat : Une dissimilarité d' qui admet $v_1 < \cdots < v_n$ comme ordre cordé.

```
d' = d
pour i = 1 à n faire
    pour j = i + 1 à n faire
        pour k = j + 1 à n faire
            si d'(v_j, v_k) > max(d'(v_i, v_j), d'(v_i, v_k)) alors
                d'(v_j, v_k) = max(d'(v_i, v_j), d'(v_i, v_k))
            fin
        fin
    fin
fin
retourner d'
```

6.5.2 Approximation Γ-free de matrice binaire

Étant donnée une matrice binaire M de taille $n \times m$, le processus suivant permet de l'approximer par une matrice totalement équilibrée :

1. Réordonner les lignes et les colonnes de M de manière doublement lexicale.
2. Vérifier si la matrice obtenue est Γ-free et l'approximer par une matrice Γ-free si ça n'est pas le cas.

Chaque étape peut être réalisée en $\mathcal{O}(nm)$ opérations. En effet, l'étape 1 est un algorithme de Spinrad [41] et l'étape 2 un algorithme de Lubiw [36]. Spinrad [42] détaille ces algorithmes.

6.5.3 Dissimilarité depuis une matrice Γ-free

Étant donné une matrice binaire Γ-free $n \times m$ $\mathcal{M}$ et un vecteur $(\alpha_j)_{1 \leq j \leq m}$ de réels positifs, nous cherchons à créer une dissimilarité d telle que :

$$d(x, y) = \min\{\alpha_j \mid \mathcal{M}_{x,j} = \mathcal{M}_{y,j} = 1\}.$$

De plus, $\mathcal{M}$ étant Γ-free, d est totalement équilibrée.

Nous supposons sans perte de généralité que la dernière colonne de $\mathcal{M}$ est une colonne de 1. Le calcul de cette dissimilarité peut être fait en $\mathcal{O}(n^3)$ par l'Algorithme 18 comme le montre la Proposition 6.5.2.

Algorithme 18 : Calcul d'une dissimilarité totalement équilibrée à partir d'une matrice Γ-free et d'un index

Données : Une matrice $n \times m$ Γ-free binaire $\mathcal{M}$ dont la dernière colonne est une colonne de 1 et m nombres réels positifs α_j

Résultat : La dissimilarité d définie par $d(x, y) = \min\{\alpha_j \mid M_{x,j} = M_{y,j} = 1\}$

```
// Initialisation
```

pour $j \leftarrow 1$ **à** m **faire**
  $N_{n+1}[j] = 0$
fin
pour $i \leftarrow 1$ **à** n **faire**
  $d(i, i) = 0$
fin

```
// Boucle principale
```

pour $i = n$ **à** 1 **faire**

```
  // Initialisation de la boucle interne
```

  pour $j = 1$ **à** m **faire**
    $N_i[j] = N_{i+1}[j] + M_{i,j}$
  fin
  $P_i = [m]$
  $c = m$

```
  // Séparation des colonnes
```

  pour $j = m$ **à** 1 **faire**
    si $M_{i,j} = 1$ **et** $\alpha_j < \alpha_c$ **alors**
      si $N_i[c] > N_i[j]$ **alors**
        Ajouter j à la fin de P_i
      sinon
        Remplacer le dernier élément de P_i par j
      fin
      $c = j$
    fin
  fin

```
  // Mise à jour de la dissimilarité
```

  pour $j = i$ **à** n **faire**
    $d(i, j) = \min\{\alpha_k \mid k \in P_i \text{ et } M_{j,k} = 1\}$
  fin
fin
retourner d

Proposition 6.5.2. *Soit $\mathcal{M}$ une matrice binaire Γ-free de taille $n \times m$ dont la dernière colonne est une colonne de 1 et m réels positifs. L'Algorithme 18 calcule une dissimilarité d telle que $d(x, y) = \min\{\alpha_j \mid M_{x,j} = M_{y,j} = 1\}$ en $\mathcal{O}(nm + n^3)$.*

Démonstration. Remarquons que pour tout $i, j, N_i[j] = \sum_{k \geq i} \mathcal{M}_{i,j}$ et qu'il n'y a pas de répétitions dans P_i. On a donc $|P_i| \leq n$ pour tout $i \leq n$. La boucle de la ligne 23 fonctionne en $\mathcal{O}(n^2)$ et l'algorithme renvoie un résultat en $\mathcal{O}(nm + n^3)$.

$\mathcal{M}$ est Γ-free donc pour tout i, les ensembles $C_{ij} = \{i' \geq i \mid \mathcal{M}_{i',j} = 1\}$, les listes N_i et les listes P_i sont telles que :

- Si $M_{i,j} = M_{i,j'} = 1$, $N_i[j] < N_i[j'] \iff \{i\} \subseteq C_{ij} \subsetneq C_{ij'}$ (pour tout i, $\{C_{i,j} \mid 1 \leq j \leq m, M_{i,j} = 1\}$ est une chaîne). Il est possible de vérifier si $C_{ij} \subset C_{ij'}$ en comparant $N_i[j]$ et $N_i[j']$ (ligne 15).
- Si $j < j'$ et $M_{i,j} = M_{i,j'} = 1$, alors $C_{ij} \subseteq C_{ij'}$. Après la ligne 21, P_i contient donc tous les indices utiles de la colonne (et seulement eux) :
  - Les indices qui ne sont pas dans P_i sont inutiles : si, pour $j \notin P_i$, $M_{i,j} = M_{i',j} = 1$, alors il existe $j' \in P_i$ avec $M_{i,j'} = M_{i',j'} = 1$ et $\alpha_{j'} < \alpha_j$.
  - Tous les indices dans P_i sont utiles : $\forall j < j' \in P_i$, $\alpha_j > \alpha_{j'}$ et $C_{ij} \subsetneq C_{ij'}$.

d est donc telle que $d(x, y) = \min\{\alpha_j \mid \mathcal{M}_{x,j} = \mathcal{M}_{y,j} = 1\}$. □

En conservant dans $\mathcal{M}$ les colonnes C_j telles qu'il n'existe aucun $j' \neq j$ tel que $C_j \subsetneq C'_j$ et $\alpha'_j \leq \alpha_j$, nous obtenons une matrice $\mathcal{M}'$ dont les colonnes sont strictement indexées par les α. Nous avons donc que si $\mathcal{M}$ est Γ-free, la dissimilarité d renvoyée par l'Algorithme 18 est totalement équilibrée, que ses classes sont les colonnes de $\mathcal{M}'$ et que ses diamètres sont les α.

6.5.4 Dissimilarité totalement équilibrée

Nous pouvons à présent utiliser les résultats précédents pour décrire l'Algorithme 19 qui permet d'approximer une dissimilarité d donnée sur un ensemble V par une dissimilarité totalement équilibrée. Toutes les étapes de l'algorithme proviennent des sections précédentes :

- La ligne 1 est une variante de l'Algorithme 15 présenté dans la Section 6.5.1.
- La ligne 2 est l'Algorithme 17.
- La ligne 3 est l'Algorithme 14.
- La ligne 4 est un algorithme de Spinrad [41].
- Étant donné une classe C_j, la ligne 6 peut être réalisée en $\mathcal{O}(n)$ car $v_1 < \cdots < v_n$ est un ordre cordé donc le diamètre de C_j est $\max\{d(v^\star, w) \mid x \in C_j\}$ avec $v^\star$ le plus petit élément de C_j selon l'ordre cordé. Pour la matrice entière, cette ligne s'exécute donc en $\mathcal{O}(nm)$.
- La ligne 8 est un algorithme de Lubiw [36].
- La ligne 9 est l'Algorithme 18.

Algorithme 19 : Approximation d'une dissimilarité donnée par une dissimilarité totalement équilibrée

Data : Une dissimilarité d sur un ensemble V de taille n
Result : Une dissimilarité totalement équilibrée d' sur V

1 Trouver un ordre cordé possible $v_1 < \dots v_n$ de V
2 Approximer d par une dissimilarité cordée d' qui admet $v_1 < \dots v_n$ comme ordre cordé
3 Calculer $M' = \mathcal{M}(\mathcal{C}(d'))$
4 Réordonner M' selon un ordre doublement lexical
5 **pour** toutes les colonnes C_j de M' **faire**
6 | Calculer $\alpha_j = \text{diam}(C_j)$
7 **fin**
8 Approximer M' par une matrice Γ-free M''
9 Calculer la dissimilarité d' définie $d'(x, y) = \min\{\alpha_j \mid {M'}_{x,j} = {M''}_{y,j} = 1\}$
10 **retourner** d''

Toutes ces étapes s'exécutent en $\mathcal{O}(n^3)$ (car il y a au plus $\mathcal{O}(n^2)$ classes dans une dissimilarité cordée sur un ensemble de taille n) et nous obtenons donc la proposition suivante.

Proposition 6.5.3. *Étant donné une dissimilarité d sur un ensemble V. L'Algorithme 19 renvoie une approximation totalement équilibrée de d en $\mathcal{O}(|V|^3)$. De plus, si d est totalement équilibrée, l'algorithme renvoie d.*

6.6 Exemple

Nous appliquons à présent l'Algorithme 19 sur un jeu de données réelles. Ces données sont celles d'une expérience de psychologie dite du rappel libre, de Henley. L'expérience consiste à demander à des individus de donner le plus de noms d'animaux possible. A partir de ces listes, on conserve uniquement ceux cités par tous les participants et la dissimilarité entre deux animaux correspond à la distance moyenne entre ces deux animaux dans l'ordre dans lequel les participants les citent dans leurs listes. La dissimilarité correspondante est représentée sur la Table 6.4.

La Table 6.5 représente la dissimilarité approximée. Les valeurs modifiées par l'approximation apparaissent en italique. La Figure 6.1 représente les classes associées à la dissimilarité totalement équilibrée obtenue par l'Algorithme 19. La dissimilarité initiale présentait plus de 70 classes. L'approximation permet donc de la simplifier. Le résultat obtenu est lisible et intelligible. Par exemple, le chat et le chien sont immédiatement réunis en une classe, ce qui paraît naturel. On retrouve de plus des associations naturelles comme des classes contenant respectivement les animaux de la forêt, les animaux de la ferme, les

Ours	0	47.2	27.7	40.1	49.6	19.1	29.0	22.6	29.5	21.4	20.3	16.1
Chat		0	30.9	56.1	2.2	29.0	25.3	24.1	24.8	43.0	41.5	47.0
Vache			0	43.6	30.2	11.0	7.7	24.5	34.1	17.0	27.9	8.2
Cerf				0	50.9	44.5	43.0	44.7	39.9	41.1	19.9	53.1
Chien					0	17.0	24.0	26.9	27.5	45.0	39.4	46.8
Chèvre						0	7.2	23.1	39.6	19.5	21.8	1.8
Cheval							0	28.6	32.6	25.7	30.1	15.2
Lion								0	33.2	29.3	33.3	35.0
Souris									0	34.9	22.6	51.9
Cochon										0	25.9	19.6
Lapin											0	32.5
Mouton												0

TABLE 6.4 – Dissimilarité issue de l'expérience du rappel libre

animaux sauvages, etc. L'information contenue dans le résultat obtenu est suffisante et interprétable facilement.

La Figure 6.2 représente la même chose que la Figure 6.1 en positionnant chaque classe à une abscisse égale à son diamètre. Cette représentation permet de bien voir les classes regroupant des éléments très proches et celles regroupant des éléments éloignés. Le cerf est par exemple particulièrement mis à part par cette représentation. De nombreuses distances étant égales, certaines classes se retrouvent superposées sur ce diagramme, montrant la ressemblance de certaines d'entre elles. Cette représentation permet par exemple de montrer que la classe contenant uniquement le chat et le chien est très éloignée des autres les contenant tous les deux : le chat et le chien sont très associés dans l'esprit des participants à l'expérience mais peu à d'autres groupes d'animaux. Cette représentation met également en évidence des alignements verticaux de classes représentant des classes proches car de diamètres identiques.

6.7 Conclusion

Ce chapitre introduit les dissimilarités totalement équilibrées ainsi que les dissimilarités cordées pouvant être associées respectivement aux systèmes totalement équilibrés et aux graphes cordés. L'aspect métrique associé aux systèmes totalement équilibrés est ainsi développé.

Ce chapitre présente de plus trois algorithmes polynomiaux permettant respectivement de calculer les classes associées à une dissimilarité totalement équilibrée, de reconnaître une dissimilarité totalement équilibrée, mais également d'approximer une dissimilarité quelconque par une dissimilarité totalement équilibrée.

FIGURE 6.1 – Diagramme de Hasse du système de classes obtenu par l'Algorithme 19 sur la dissimilarité de la Table 6.4

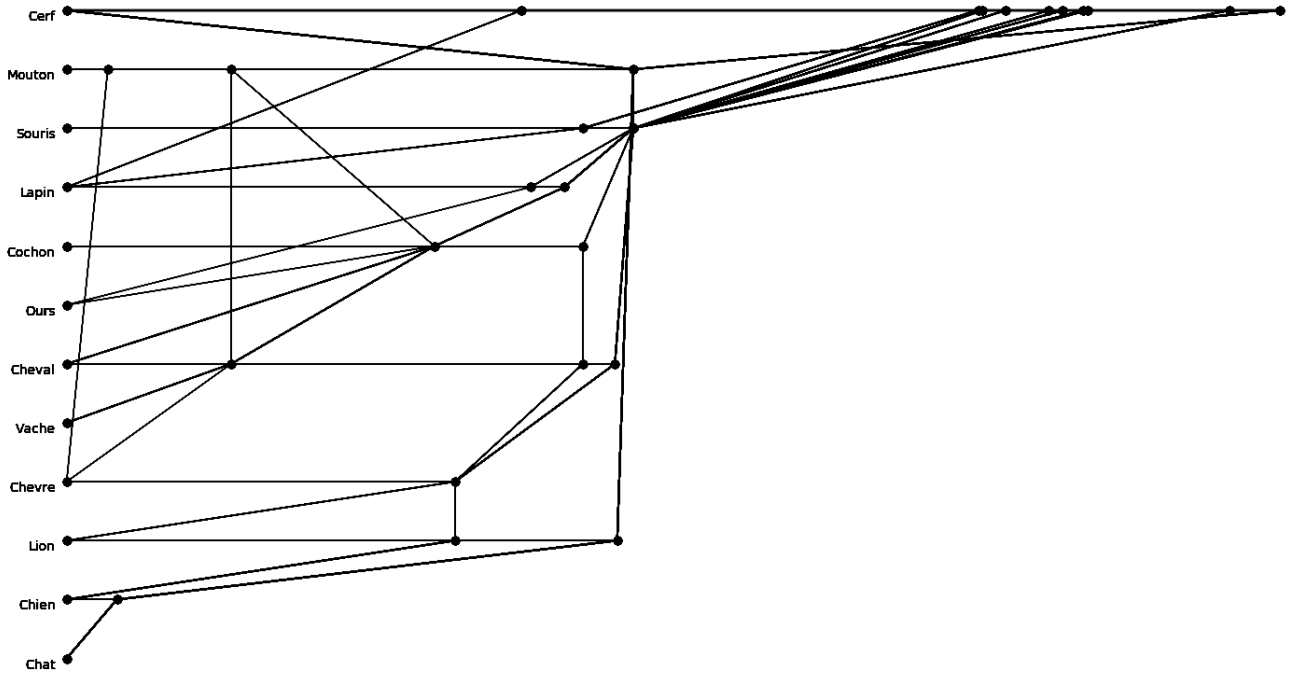


FIGURE 6.2 – Diagramme de Hasse du système de classes obtenu par l'Algorithme 19 sur la dissimilarité de la Table 6.4 tel que l'abscisse des classes est leur diamètre

Ours	0	*24.8*	*16.1*	40.1	*24.8*	*16.1*	*16.1*	22.6	*24.8*	*16.1*	20.3	16.1
Chat		0	*24.8*	56.1	2.2	*24.8*	*24.8*	24.1	24.8	*24.8*	*24.8*	*24.8*
Vache			0	43.6	*22.6*	*7.2*	*7.2*	*22.6*	*24.8*	*16.1*	*21.8*	8.2
Cerf				0	50.9	44.5	43.0	44.7	39.9	41.1	19.9	53.1
Chien					0	17.0	24.0	*17*	*24.8*	*24.8*	*24.8*	*24.8*
Chèvre						0	7.2	*17*	*24.8*	*16.1*	21.8	1.8
Cheval							0	*22.6*	*24.8*	*16.1*	*21.8*	15.2
Lion								0	*24.8*	*22.6*	*24.8*	*24.8*
Souris									0	*24.8*	22.6	*24.8*
Cochon										0	*21.8*	*16.1*
Lapin											0	*24.8*
Mouton												0

TABLE 6.5 – Dissimilarité approximée à partir de la dissimilarité de la Table 6.4

Nous montrons de plus l'intérêt de l'approximation d'une dissimilarité quelconque par une dissimilarité totalement équilibrée sur un exemple. La simplicité du modèle créé permet une interprétation aisée et représentable facilement.

Conclusion

Nous avons dans ce travail cherché à mettre en évidence des liens entre l'analyse de données et l'apprentissage automatique ainsi qu'à montrer ce que ces deux domaines peuvent s'apporter mutuellement, notamment à travers l'utilisation des structures totalement équilibrées. Cet objectif a été atteint à travers deux grands axes. Dans un premier temps, nous avons cherché à introduire des éléments d'analyse de données dans des méthodes classiques d'apprentissage automatique mais aussi de l'apprentissage automatique dans des modèles d'analyse de données. Ensuite, nous avons étendu la notion de binarité, souvent utilisée dans les modèles de décision aux treillis et aux hypergraphes. Cette extension a permis de montrer l'intérêt de l'utilisation des modèles totalement équilibrés dans ce cadre. Nous avons alors pu proposer divers algorithmes d'approximation de modèles par des modèles totalement équilibrés.

Plus précisément, nous avons tout d'abord mélangé apprentissage automatique et analyse de données dans le cadre de la prédiction en apprentissage automatique supervisé. Le Chapitre 2 présente un modèle de classification efficace en prédiction : les arbres de décision K-Means. Le modèle utilise le partitionnement K-Means, issu de l'analyse de données, pour diviser récursivement les données et créer une hiérarchie. Ce modèle crée donc une structure interprétable. L'utilisation du modèle à la manière d'un arbre de décision permet de plus d'obtenir des résultats de prédiction équivalents, voire supérieurs à ceux des arbres de décision classiques pour la majorité des ensembles de données réelles testés. Ces résultats permettent de valider l'approche consistant à utiliser majoritairement la structure des données (*i.e.* leur description dans l'espace donné) pour retrouver la structure en classes prédéterminées. En effet, ceci permet de mêler les objectifs des deux mondes : créer un modèle interprétable et obtenir de bons résultats en prédiction. Nous avons également étendu l'inspiration donnée par l'apprentissage automatique en créant les forêts d'arbres de décision K-Means. Ce modèle n'autorise néanmoins pas l'empiétance entre les classes et le travail concernant de tels modèles utilisables pour des tâches de prédiction reste à effectuer.

Nous avons alors voulu proposer une première approche de modèles avec classes empiétantes pour l'apprentissage automatique et avons cherché à introduire de l'apprentissage automatique dans une construction très inspirée de l'analyse de données. Pour cela, le Chapitre 3 introduit un modèle totalement équilibré autorisant l'empiétance des classes.

Ce modèle local, simple, peut être utilisé en apprentissage automatique mais ses résultats ne sont pas satisfaisants sur la plupart des ensembles de données. Les ensembles de petites tailles et dimensions présentent néanmoins des résultats équivalents aux arbres de décision classiques, ce qui permet de montrer que les modèles avec empiétance ne sont pas incompatibles avec l'utilisation en apprentissage automatique. Le chapitre détaille également le choix du modèle local et montre que le passage à un modèle global dans ce contexte ne permet pas de conserver les propriétés voulues.

Ces premiers essais montrent également que l'approche "bottom-up" n'est sûrement pas le meilleur choix dans le cadre de l'application en apprentissage automatique : séparer ce qui diffère permet d'obtenir des modèles plus simples (comme les arbres de décision K-Means) et plus efficaces que rassembler ce qui se ressemble (comme les systèmes de classes de centres de gravité, trop gros pour être efficaces). Le développement d'un modèle totalement équilibré autorisant l'empiétance et utilisable en apprentissage automatique reste donc une voie à explorer.

Dans le cadre de l'apprentissage automatique, les modèles binaires sont souvent préférés, principalement pour leur simplicité. Il est donc naturel de chercher à étendre cette propriété aux modèles de classification. Nous avons donc introduit les treillis binaires ainsi que les hypergraphes binaires mais également les treillis et hypergraphes binarisables. Nous avons ainsi pu étudier l'origine de la propriété de binarité. Nous sommes alors parvenus à montrer que les systèmes binarisables sont exactement les systèmes totalement équilibrés, confirmant l'intérêt de telles structures dans le cadre de la classification. Le Chapitre 4 définit les treillis binaires et binarisables puis montre l'équivalence entre les treillis binarisables et les treillis sans couronnes. Le Chapitre 5 pour sa part, se place dans le formalisme des hypergraphes pour définir les notions de binaires et binarisables. Il montre sous un autre angle l'équivalence entre les hypergraphes binarisables et les hypergraphes totalement équilibrés. Il présente également un des apports théoriques principaux de cette thèse : une caractérisation des hypergraphes binaires par la construction d'une séquence d'arbres, semblable à celle proposée par Lehel.

L'intérêt des modèles totalement équilibrés et binaires dans le cadre de la classification étant démontré par ce résultat, nous étendons la notion de système totalement équilibré aux dissimilarités dans le Chapitre 6. Nous explicitons également deux algorithmes polynomiaux capables de réaliser deux opérations utiles associées à ces dissimilarités : en calculer les classes et les reconnaître. Nous proposons de plus diverses manières d'approximer différentes structures par une version binaire et donc totalement équilibrée. Le Chapitre 4 présente ainsi un algorithme d'approximation d'un treillis totalement équilibré par un treillis binaire. Nous proposons également une application en analyse de concepts formels afin de montrer l'aspect interprétatif de l'approximation proposée. Le Chapitre 5 propose ensuite une approximation plus générale, par la construction par une séquence d'arbres, d'hypergraphes totalement équilibrés ou non par un hypergraphe binaire. Enfin, le Chapitre 6 propose un algorithme d'approximation d'une dissimilarité quelconque par

une dissimilarité totalement équilibrée.

Les treillis binaires (ou binarisables) semblent être intéressants dans le cadre de l'apprentissage automatique. En effet, ils constituent une généralisation naturelle des arbres de décision en permettant de reprendre l'idée générale de leur structure et de leur utilisation en prédiction tout en ajoutant la possibilité d'empiétance entre les classes créées. De tels modèles pourraient permettre à la fois l'interprétation des données et la prédiction. La prédiction pourrait de plus être améliorée par l'empiétance. En effet, autoriser l'empiétance des classes revient, lors de la prédiction, à autoriser une erreur lors de la descente de l'exemple dans la structure sans que cette erreur se répercute sur la classe prédite : l'exemple peut arriver au même endroit par plusieurs chemins différents. De même, la construction du Chapitre 5 pourrait être utilisée pour développer un modèle de classification utilisable en apprentissage automatique. Il s'agirait néanmoins d'inverser la construction proposée afin de construire le modèle de haut en bas comme remarqué dans le Chapitre 3. Cette inversion du sens de construction n'est néanmoins pas triviale.

Bibliographie

[1] Ali Rahimi's talk at NIPS(NIPS 2017 Test-of-time award presentation) https ://www.youtube.com/watch ?v=Qi1yry33tqe.

[2] UCI Machine Learning Repository https ://archive.ics.uci.edu/ml/index.php.

[3] Richard P. Anstee. Hypergraphs with no special cycles. *Combinatorica*, 3(2) :141–146, June 1983.

[4] Richard P. Anstee and Martin Farber. Characterizations of totally balanced matrices. *Journal of Algorithms*, 5(2) :215–230, 1984.

[5] David Arthur and Sergei Vassilvitskii. K-means++ : The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[6] H. J. Bandelt and A. W. Dress. Weak hierarchies associated with similarity measures–an additive clustering technique. *Bulletin of Mathematical Biology*, 51(1) :133–166, 1989.

[7] Marc Barbut and Bernard Monjardet. *Ordre et classification : algèbre et combinatoire*. Hachette, 1970.

[8] Radim Belohlavek, Bernard De Baets, Jan Outrata, and Vilem Vychodil. Inducing decision trees via concept lattices. *International Journal of General Systems*, 38(4) :455–467, May 2009.

[9] K. Bertet, M. Visani, and N. Girard. Dichotomic Lattices and Decision Tree. *Traitement du signal*, 26(5) :409–418, 2009.

[10] Patrice Bertrand. Set Systems and Dissimilarities. *European Journal of Combinatorics*, 21(6) :727–743, August 2000.

[11] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2) :123–140, August 1996.

[12] Leo Breiman. [Bias, Variance, and] Arcing Classifiers. Technical report, Statistics Department, University of California, Berkeley, University of California at Berkeley, Berkeley, California, February 1996.

[13] Leo Breiman. Random Forests. *Machine Learning*, 45(1) :5–32, October 2001.

[14] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and regression trees*. Chapman and Hall/CRC, 1984.

[15] François Brucker and Alain Gély. Crown-free Lattices and Their Related Graphs. *Order*, 28(3) :443–454, September 2010.

[16] François Brucker and Pascal Préa. Totally Balanced Formal Context Representation. In *Formal Concept Analysis*, number 9113 in Lecture Notes in Computer Science, pages 169–182. Springer International Publishing, June 2015.

[17] Alexander Böcker, Swetlana Derksen, Elena Schmidt, Andreas Teckentrup, and Gisbert Schneider. A Hierarchical Clustering Approach for Large Compound Libraries. *Journal of Chemical Information and Modeling*, 45(4) :807–815, July 2005.

[18] Victor Codocedo and Amedeo Napoli. Formal Concept Analysis and Information Retrieval – A Survey. In *International Conference in Formal Concept Analysis - ICFCA 2015*, volume 9113 of *Formal Concept Analysis - Lecture Notes in Computer Science*, pages 61–77, Nerja, Spain, June 2015. Springer.

[19] Edwin Diday. Une nouvelle méthode en classification automatique et reconnaissance des formes la méthode des nuées dynamiques. *Revue de Statistique Appliquée.*, page 16, 1971.

[20] Pierre Duchet. Propriété de Helly et problèmes de représentation. *Colloque International, Paris-Orsay*, pages 117–118, 1978.

[21] Martin Farber. Characterizations of strongly chordal graphs. *Discrete Mathematics*, 43(2–3) :173–189, 1983.

[22] Ronald Aylmer Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2) :179–188, September 1936.

[23] Claude Flament. Hypergraphes arbores. *Discrete Mathematics*, 21(3) :223–227, January 1978.

[24] E. W. Forgy. Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, 21 :768–769, 1965.

[25] Bernhard Ganter and Rudolf Ville. *Formal Concept Analysis - Mathematical Foundations*. Springer edition, 1997.

[26] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1) :3–42, April 2006.

[27] Nathalie Girard. *Vers une approche hybride mêlant arbre de classification et treillis de Galois pour de l'indexation d'images*. PhD thesis, Université de La Rochelle, 2013.

[28] Tin Kam Ho. The Random Subspace Method for Constructing Decision Forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8) :832–844, August 1998.

[29] David Kelly and Ivan Rival. Crowns, fences, and dismantlable lattices. *Canadian Journal of Mathematics*, 26(0) :1257–1271, January 1974.

[30] Pascale Kuntz. *Représentation euclidienne d'un graphe abstrait en vue de sa segmentation*. PhD thesis, Paris, EHESS, January 1992.

[31] Georges-Louis Leclerc comte de Buffon. *Histoire naturelle générale et particulière : avec la description du Cabinet du Roy.* 1749.

[32] Jenö Lehel. A characterization of totally balanced hypergraphs. *Discrete Mathematics*, 57(1–2) :59–65, November 1985.

[33] Stuart P Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2) :129–136, 1982.

[34] Gilles Louppe and Pierre Geurts. Ensembles on Random Patches. In *Machine Learning and Knowledge Discovery in Databases*, number 7523 in Lecture Notes in Computer Science, pages 346–361. Springer Berlin Heidelberg, September 2012.

[35] László Lovàsz. Graphs and set systems. In *Beiträge zur Graphentheorie*, pages 99–106. Beiträge zur Graphentheorie, Leipzig, 1968.

[36] Anna Lubiw. Doubly Lexical Orderings of Matrices. *SIAM Journal on Computing*, 16(5) :854–879, October 1987.

[37] Engelbert Mephu Nguifo and Patrick Njiwoua. Treillis de concepts et classification supervisée. *Techniques et sciences informatiques*, 24(4) :449–488, April 2005.

[38] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11) :559–572, November 1901.

[39] Arora Sanjeev. Toward Theoretical Understanding of Deep Learning : Slides and Bibliography http ://unsupervised.cs.princeton.edu/deeplearningtutorial.html.

[40] Gilbert Saporta. *Probabilités, analyse des données et statistique.* TECHNIP, 1990.

[41] Jeremy P. Spinrad. Doubly lexical ordering of dense 0–1 matrices. *Information Processing Letters*, 45(5) :229–235, April 1993.

[42] Jeremy P. Spinrad. *Efficient Graph Representations.* American Mathematical Soc., January 2003.

[43] Carl von Linné. *Systema naturæ per regna tria naturæ, secundum classses, ordines, genera, species, cum characteribus, differentiis, synonymis, locis.* 1758.

Résumé

Mots-clés: hypergraphes totalement équilibrés, treillis démontables, classification, apprentissage automatique

Le but des modèles traditionnels en classification (comme les partitions et les hiérarchies de parties) est de permettre de discriminer sans ambiguïté et donc de produire des classes non empiétantes (*i.e.* l'intersection de deux classes est vide ou une classe est incluse dans l'autre). Cependant, cette exigence de non ambiguïté peut conduire à occulter de l'information. Dans le cas des plantes hybrides en biologie par exemple ou encore de textes appartenant à plusieurs genres en analyse textuelle. Les modèles généraux comme les hypergraphes ou les treillis permettent de prendre en compte l'empiétance entre les classes.

Ce travail porte sur les hypergraphes totalement équilibrés clos par intersection et leurs équivalents. Ces hypergraphes sont définis comme étant des hypergraphes sans cycle (cycle spécial, aussi appelés alpha-cycle) et constituent une généralisation des arbres. Ils sont équivalents aux treillis démontables (*i.e.* treillis tels qu'il existe récursivement un élément doublement irréductible) et présentent des propriétés structurelles et algorithmiques qui leur permettent de bien se prêter à de nombreux domaines comme la phylogénie et de traiter différents types de données comme les dissimilarités, les matrices individus/attributs ou encore les graphes.

En apprentissage automatique, les arbres de décision sont un modèle très utilisé pour leur simplicité d'utilisation et de compréhension. Une partie de ce travail porte sur le développement de méthodes similaires aux arbres de décision mais s'appuyant principalement sur la structure des données et permettant l'empiétance des classes afin de fournir une représentation plus complète des données. Les modèles visés sont donc fortement interprétables et peuvent également être utilisés pour les tâches classiques d'apprentissage automatique comme la prédiction de classe. Cette thèse présente deux méthodes :

— les arbres de décision K-Means, une méthode d'apprentissage automatique utilisant la structure des données plutôt que les sorties attendues et présentant des résultats en classification équivalents aux arbres de décision classiques ;
— les treillis de centre de gravité, proposant une première approche pour un modèle en classes empiétantes.

Dans le cas des arbres de décision, l'usage veut que les arbres utilisés soient binaires. Nous définissons donc les hypergraphes binaires afin de conserver la simplicité propre aux arbres de décision. Nous proposons une caractérisation des hypergraphes binaires par une séquence d'arbres (similaire à celle donnée par Lehel en 1985 pour les hypergraphes totalement équilibrés) et prouvons l'équivalence entre les hypergraphes binarisables (*i.e.* tels

qu'ils peuvent être plongés dans un hypergraphe binaire) et les hypergraphes totalement équilibrés, faisant de ces hypergraphes particuliers un bon candidat à la classification inspirée des arbres de décision. Nous proposons également une binarisation des treillis démontables pouvant être appliquée dans le cadre de l'analyse de concepts formels. Ce travail présente de plus un aspect métrique en définissant les dissimilarités totalement équilibrées (les dissimilarités associées à un système totalement équilibré) et en donnant un algorithme de reconnaissance de dissimilarités totalement équilibrées, un algorithme d'approximation et enfin un algorithme permettant de calculer le système de classes associé à une dissimilarité totalement équilibrée de manière polynomiale.

Abstract

Keywords: totally balanced hypergraphs, dismantlable lattices, classification, machine learning

Traditionally, classification models (such as partitions and hierarchies) aim at separating without ambiguities and produce non-overlapping clusters (*i.e.* two clusters are either disjoint or one is included in the other). However, this non ambiguity may lead to mask information such as in the case of hybrid plants in biology or of texts which belong to two (or more) different genres in textual analysis for instance. General models like hypergraphs or lattices allow to take into account overlapping clusters.

This work focuses on closed under intersection totally balanced hypergraphs and their equivalents. These hypergraphs are defined as hypergraphs with no special cycles (also called alpha-cycle) and are a generalization of trees. They are equivalent to dismantlable lattices (*i.e.* lattices such that there recursively exists a doubly irreducible element) and have structural and algorithmic properties which allow them to fit many fields such that phylogenetics and deal with different data types such as dissimilarities, individuals/attributes matrices or graphs.

In machine learning, decision trees are a widely used model as they are simple to use and understand. A part of this work focuses on the development of similar methods which allow overlapping clusters in order to give a more complete representation of data. Hence, the aimed models are strongly interpretable and can be used for classic machine learning tasks such as class prediction. This thesis presents two methods : - K-Means decision trees, a classification method which builds the model on the structure of the data and gives practical results equivalent to decision trees ; - gravity decision lattices, which proposes a first approach to non-overlapping classification models.

Regarding decision trees, usage requires that the trees are binary. We thus define binary hypergraphs in order to keep the simplicity specific to decision trees. We propose a characterization of binary hypergraphs by a sequence of mixed trees (similar to the

characterization of totally balanced hypergraphs given by Lehel in 1985) and prove the equivalence between binarizable hypergraphs (*i.e.* such that they can be embedded into a binary hypergraph) and totally balanced hypergraphs which makes of these hypergraphs a perfect candidate for classification inspired from decision trees.

We also propose a binarization algorithm for dismantlable lattices which can be used in formal concept analysis.

This work also presents a metric angle : we define totally balanced dissimilarities (dissimilarities which are associated with a totally balanced system) and give a recognition algorithm, an approximation algorithm for these dissimilarities and an algorithm which computes the clusters associated with a totally balanced dissimilarity.