Normandie Université

INSA ROUEN

THÈSE

En vue de l'obtention du grade de

Docteur de l'INSA de Rouen

Spécialités : Mathématiques - Informatique

par

Zacharie Ales

Extraction et partitionnement pour la recherche de régularités : application à l'analyse de dialogues.

Soutenue le 28 Novembre 2014 devant le jury composé de :

Rapporteurs	
Mme. Martine Labbé	Professeur des Universités (Université Libre de Bruxelles)
M. Ridha Mahjoub	Professeur des Universités (LAMSADE, Université Paris Dauphine)
M. Olivier Pietquin	Professeur des Universités (LIFL, Université Lille 1)
Examinateur	
M. Colin de la Higuera	Professeur des Universités (LINA, Université de Nantes)
Directeurs de thèse	
M. Christian Gout	Professeur des Universités (LMI, INSA de Rouen)
M. Laurent Vercouter	Professeur des Universités (LITIS, INSA de Rouen)
Co-encadrants	
M. Arnaud Knippel	Maître de conférences (LMI, INSA de Rouen)
M. Alexandre Pauchet	Maître de conférences (LITIS, INSA de Rouen)





Thèse préparée à
L'Institut National des Sciences Appliquées (INSA) de Rouen
Laboratoire de Mathématiques de l'INSA de Rouen (EA 3226)
Laboratoire d'Informatique, du Traitement de l'Information et des Systèmes (EA 4108)
76 801 Saint Etienne du Rouvray

Remerciements

La réalisation de cette thèse est loin d'être un travail solitaire et il me paraît donc nécessaire d'exprimer ici ma gratitude à certains de ses principaux acteurs.

Je souhaiterais tout d'abord remercier Alexandre Pauchet qui le premier m'a motivé à me lancer dans la recherche en me décrivant les problématiques liées aux tableaux d'annotations et en me proposant de m'encadrer au cours d'une thèse. Son engagement sans faille tout au long du déroulement de ces trois années ainsi que son honnêteté ont grandement contribués à l'aboutissement de cette thèse. Je remercie aussi Martin d'avoir aidé Alexandre à relire et corriger mon manuscrit.

Je tenais également à remercier Arnaud Knippel qui a sû me guider avec patience et régularité au cours de ces années. Merci de m'avoir fait prendre goût à l'optimisation combinatoire et de m'avoir lancé dans la quête de facettes.

J'exprime ma gratitude à mes deux directeurs de thèse, Laurent Vercouter et Christian Gout pour leur soutien aussi bien administratif que scientifique et moral.

Je remercie tout particulièrement Martine Labbé, Ridha Mahjoub et Olivier Pietquin de m'avoir fait l'honneur d'accepter d'être rapporteurs de cette thèse. Je souhaitais remercier également Colin de la Higuera d'avoir accepté d'examiner ces travaux.

J'exprime tous mes remerciements à l'ensemble des doctorants et stagiaires du LITIS et du LMI que j'ai eu la chance de croiser et avec qui j'ai pu partager les joies et les déboires de nos expériences communes.

Je sais tout particulièrement gré à Brigitte Diarra, Sandra Hague et Fabienne Bocquet, secrétaires des laboratoires LITIS et/ou LMI, pour avoir toujours su me guider avec bonne humeur et efficacité dans les affres des démarches administratives.

Je souhaitais également témoigner ma reconnaissance à l'ensemble des Power Rouengers et des membres de l'orchestre d'harmonie de Forges-les-Eaux pour m'avoir permis respectivement de pratiquer l'Ultimate et la clarinette. Ces parenthèses bienvenues m'ont données l'occasion de me consacrer à deux de mes plus grandes passions.

Il m'est impossible de ne pas mentionner ici mes parents, mes deux frères ainsi que ma grand-mère. J'ai peine à exprimer à quel point leur présence et leur affection me sont chères. Je vous remercie de m'avoir donner l'envie, le soutien et les moyens de parvenir au terme de cette épreuve.

Enfin, je souhaitais terminer par remercier Diana qui a réussi l'exploit de me supporter au quotidien pendant toutes ces années. Son oreille attentive, ses encouragements ainsi que ses conseils ont fortement contribués à l'aboutissement de cette thèse.

Résumé

Dans le cadre de l'aide à l'analyse de dialogues, un corpus de dialogues peut être représenté par un ensemble de tableaux d'annotations encodant les différents énoncés des dialogues. Afin d'identifier des schémas dialogiques mis en œuvre fréquemment, nous définissons une méthodologie en deux étapes : extraction de motifs récurrents, puis partitionnement de ces motifs en classes homogènes constituant ces régularités.

Deux méthodes sont développées afin de réaliser l'extraction de motifs récurrents : LPCA-DC et SABRE. La première est une adaptation d'un algorithme de programmation dynamique tandis que la seconde est issue d'une modélisation formelle du problème d'extraction d'alignements locaux dans un couple de tableaux d'annotations.

Le partitionnement de motifs récurrents est réalisé par diverses heuristiques de la littérature ainsi que deux formulations originales du problème de K-partitionnement sous la forme de programmes linéaires en nombres entiers. Lors d'une étude polyèdrale, nous caractérisons des facettes d'un polyèdre associé à ces formulations (notamment les inégalités de 2-partitions, les inégalités 2-*chorded cycles* et les inégalités de clique généralisées). Ces résultats théoriques permettent la mise en place d'un algorithme de plans coupants résolvant efficacement le problème.

Nous développons le logiciel d'aide à la décision VIESA, mettant en œuvre ces différentes méthodes et permettant leur évaluation au cours de deux expérimentations réalisées par un expert psychologue. Des régularités correspondant à des stratégies dialogiques que des extractions manuelles n'avaient pas permis d'obtenir sont ainsi identifiées.

Mots-clefs : optimisation combinatoire, fouille de données, extraction de régularités, K-partitionnement, approche polyèdrale, analyse de dialogues.

Extraction and partitioning for regularity extraction: application to dialogue analysis

Abstract

In the context of dialogue analysis, a corpus of dialogues can be represented as a set of arrays of annotations encoding the dialogue utterances. In order to identify the frequently used dialogue schemes, we design a two-step methodology in which recurrent patterns are first extracted and then partitioned into homogenous classes constituting the regularities.

Two methods are developed to extract recurrent patterns: LPCA-DC and SABRE. The former is an adaptation of a dynamic programming algorithm whereas the latter is obtained from a formal modeling of the extraction of local alignment problem in annotations arrays.

The partitioning of recurrent patterns is realised using various heuristics from the literature as well as two original formulations of the K-partitioning problem in the form of mixed integer linear programs. Throughout a polyhedral study of a polyhedron associated to these formulations, facets are characterized (in particular: 2-chorded cycle inequalities, 2-partition inequalities and general clique inequalities). These theoretical results allow the establishment of an efficient cutting plane algorithm.

We developed a decision support software called VIESA which implements these different methods and allows their evaluation during two experiments realised by a psychologist. Thus, regularities corresponding to dialogical strategies that previous manual extractions failed to identify are obtained.

Keywords : combinatorial optimization, data mining, regularity extraction, K-partitioning, polyhedral approach, dialogue analysis.

Table des matières

Introduction **9**

I Extraction de motifs dans des tableaux d'annotations **17**

1 État de l'art des méthodes d'extraction de régularités **20**

1.1 Fouille de motifs fréquents 21

1.1.1 Fouille d'itemsets fréquents 21

1.1.2 Fouille de séquences d'itemsets fréquentes 25

1.1.3 Extraction de sous-structures fréquentes 31

1.2 Fouille de séquences de caractères 35

1.2.1 Algorithme de Needleman-Wunsch 38

1.2.2 Alignement local de séquences de caractères 40

1.2.3 Alignement local de tableaux de caractères 41

1.3 Discussion 45

2 Modélisation et résolution du problème d'extraction de motifs **48**

2.1 Algorithme $LPCA - DC$ 48

2.1.1 Description de l'algorithme 48

2.1.2 Discussion 51

2.2 Approche par extraction d'arborescences 52

2.2.1 Définitions et notations 53

2.2.2 Calcul du score d'un alignement 53

2.2.3 Modélisation du problème d'extraction d'alignements 57

2.2.4 Algorithme SABRE 58

2.3 Conclusion 62

II Classification non supervisée de motifs 65

3 État de l'art des méthodes de classification non supervisée 68

3.1 Heuristiques . . . 69

3.1.1 Méthodes hiérarchiques . . . 69

3.1.2 Méthodes de type partition . . . 72

3.2 Approches exactes . . . 76

3.2.1 Partitionnement non contraint . . . 79

3.2.2 Poids des parties contraint . . . 81

3.2.3 Nombre de sommets par partie fixé . . . 83

3.2.4 Nombre de parties contraint . . . 83

3.3 Discussion . . . 85

4 Approche polyèdrale pour le problème de K-partitionnement 86

4.1 Définitions et notations . . . 86

4.2 Formulations . . . 88

4.2.1 Formulation (F_1) . . . 88

4.2.2 Formulation (F_2) . . . 90

4.2.3 Comparaison des formulations . . . 91

4.3 Dimension et facettes . . . 94

4.3.1 Dimension de $P_{n,K}$ . . . 94

4.3.2 Facettes . . . 99

4.4 Conclusion . . . 119

5 Étude numérique 120

5.1 Évaluation de la qualité des inégalités définissant des facettes non triviales de $P_{n,K}$ . . . 120

5.1.1 Algorithmes de séparations . . . 121

5.1.2 Amélioration de la relaxation par famille d'inégalités . . . 124

5.2 Résolution numérique par un algorithme de plans coupants . . . 130

5.2.1 Description générale de l'algorithme . . . 130

5.2.2 Obtention d'une solution entière à partir d'une solution fractionnaire 131

5.2.3 Algorithmes de séparation . . . 132

5.2.4 Gestion des algorithmes de séparation et des coupes ajoutées . . . 134

5.2.5 Résultats numériques 135
5.3 Conclusion 137

III Mise en œuvre et conclusions 140

6 Expérimentations 141
6.1 Présentation des données 141
6.2 Logiciel VIESA 142
6.3 Évaluation de LPCA-DC et des heuristiques de partitionnement 143
6.3.1 Calcul de la dissimilarité inter-motifs avec LPCA-DC 143
6.3.2 Protocole d'expérimentation 145
6.3.3 Résultats et discussion 148
6.4 Evaluation de SABRE et l'algorithme de plans coupants 153
6.4.1 Calcul de la dissimilarité inter-motifs avec SABRE 153
6.4.2 Protocole d'expérimentation 154
6.4.3 Résultats 157
6.5 Discussion 161

Conclusion et perspectives 164

Index 166

Références bibliographiques 170

Introduction

Contexte applicatif

Le langage est un moyen de communication complexe dont l'origine remonte à plus de deux millions d'années. Au cours du temps, il n'a cessé d'évoluer et de s'affiner, entraînant ainsi la création des milliers de dialectes existant de nos jours. Dans le cadre de dialogues face à face, divers éléments communicatifs – tels que les expressions faciales, les gestes ou le ton – viennent enrichir le discours afin d'en faciliter la compréhension. En effet, différentes études [91, 40] ont montré que plus de la moitié des informations transmises dans une interaction interpersonnelle sont exprimées au travers de comportements non verbaux.

Ces différents aspects entraînent une grande difficulté à modéliser et comprendre les schémas dialogiques naturellement mis en œuvre par des interlocuteurs dans des situations données (*e.g.* : débat politique, dialogue entre un vendeur et un client, etc). Il existe, néanmoins, divers domaines applicatifs dans lesquels l'identification de tels schémas permettrait des progrès significatifs. Par exemple, l'étude de dialogues en contexte d'apprentissage serait susceptible de fournir des schémas récurrents efficaces permettant de guider la structuration des interactions entre un enseignant et ses élèves. Dans un cadre similaire, D'mello *et al.* [32] ont étudié des sessions de tutorat individuel qui leur ont, notamment, permis de constater que l'augmentation du nombre d'actions de nature collaborative entre le tuteur et l'étudiant permettait d'influencer positivement l'apprentissage. Un second exemple d'application qui pourrait bénéficier de l'identification de schémas dialogiques est la conception d'ACA (Agents Conversationnels Animés). Nous sommes de plus en plus amenés à communiquer avec des agents (*e.g.* : sites internet, jeux vidéos). Cependant, il est rare que ces interactions soient riches et naturelles pour l'humain [121]. Bien souvent, les réactions de l'agent – tant du point de vue verbal que non verbal – sont très limitées. La découverte de schémas dialogiques usuels pour les utilisateurs et efficaces pour un agent conversationnel donné est une piste sérieuse pour l'amélioration de ses capacités interactives.

Dans le cadre de cette thèse, nous cherchons à faciliter le travail d'identification des schémas dialogiques les plus couramment utilisés au sein d'un corpus de dialogues. Les schémas dialogiques les plus fréquents feront nécessairement apparaître des régularités dans le corpus. En d'autres termes, notre objectif est de fournir un outil d'aide à la décision paramétrable et simple d'utilisation permettant à un expert d'extraire et de visualiser des régularités susceptibles d'être généralisées en schémas dialogiques pertinents.

Modélisation de dialogues

L'extraction de régularités dans des données sous forme de texte en langage naturel n'est pas une tâche aisée. De nombreux travaux y ont été consacrés [109, 120, 128] et comme nous le voyons dans le chapitre suivant, les méthodes mises en place nécessitent généralement d'importants pré-traitements et ne permettent pas d'identifier des régularités s'étendant sur plusieurs phrases. De plus, nous avons mentionné que le dialogue est une activité dans laquelle de nombreux éléments non verbaux contribuent activement à la compréhension mutuelle des interlocuteurs. Ainsi, se pose la question de la représentation de ces éléments par rapport au texte et de leur prise en compte dans l'identification de régularités.

Pour représenter les dialogues, nous nous appuyons sur la notion de *schéma de codage* couramment employée dans le cadre de l'analyse de dialogue (*e.g.* : le codage DIT++ [20]). Un schéma de codage consiste, en premier lieu, à segmenter un dialogue en énoncés ordonnés chronologiquement. Un exemple de segmentation de ce type est présentée dans la seconde colonne de la table 1. Un dialogue est alors représenté en encodant chaque énoncé par un nombre fixé d'annotations. Le nombre d'annotations associées à un énoncé est appelé la *dimension du schéma de codage*. L'exemple représenté en table 1, correspond à un schéma de codage de dimension 3, tel que pour chaque énoncé :

- la première annotation représente le type de l'acte de dialogue ('E' : engageant, 'A' : assertif, 'D' : directif) ;
- la seconde annotation encode les émotions contenues dans l'énoncé ('J' : joie, 'T' : tristesse, 'S' : surprise, '-' : absence d'émotion) ;
- la dernière annotation décrit le regard des participants ('M' : contact visuel mutuel, 'U' : regard unilatéral du locuteur, 'I' : regard vers un objet identifié, '-' : absence de regard).

Locuteur	**Dialogue**	**Annotations**		
M	Bonjour monsieur, que puis-je faire pour vous ? <sourit><regarde le client>	E	J	U
C	Je souhaiterais acheter du thé noir. <contact visuel mutuel>	A	-	M
C	Avez-vous des thés spéciaux pour Noël ?	D	-	-
M	Oui nous en avons un, laissez-moi regarder... <regarde la vitrine>	E	-	I
M	J'ai bien peur que nous n'en ayons plus en stock. <air désolé>	A	T	I
C	Quoi ?	E	S	-
M	Cependant, nous devrions être fourni cet après-midi<regarde le client>	A	-	U
C	C'est parfait. <pause> Je repasserai demain.	A	J	-

TABLE *1 – Dialogue fictif entre un marchand et un client, annoté selon un schéma de codage associant trois annotations à chaque énoncé.*

Le choix du schéma de codage dépend du type de schémas dialogiques que l'on souhaite extraire du corpus. Par exemple, dans le cas du dialogue représenté en table 1,

si nous souhaitions obtenir des régularités comportant des structures interrogatives, il serait envisageable d'ajouter une colonne encodant le type d'énoncé ('Q' pour une question ou 'A' pour une affirmation).

L'extraction manuelle de régularités dans un corpus ainsi annoté est fastidieuse et atteint rapidement ses limites lorsque la taille du corpus et de ses dialogues augmente. Par exemple, dans le cas du corpus considéré durant nos expérimentations, seules des régularités identiques apparaissant sur une unique ligne d'annotations avaient pu être identifiées manuellement. L'automatisation du processus d'extraction permettrait donc un gain de temps significatif à un expert tout en lui offrant la possibilité d'obtenir des régularités plus variées (*i.e.* : s'étendant sur plusieurs lignes et pouvant apparaître de manière approchée à différents emplacements du corpus).

La présentation de la méthodologie que nous mettons en œuvre dans cette optique nécessite la définition de plusieurs notions et notations.

Formalisation du problème d'extraction de régularités

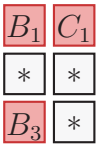
Dans le cadre de nos travaux, nous choisissons de représenter un dialogue annoté par un tableau d'annotations dont chaque colonne correspond à une dimension du schéma de codage considéré. Plus précisément, étant donné un dialogue d comportant l énoncés et encodé selon un schéma de codage comportant c dimensions, nous considérons un tableau d'annotations t de taille $l \times c$ (représenté $t[1..l][1..c]$) tel que pour tout $i, j \in \{1, \ldots, l\} \times \{1, \ldots, c\}$, $t[i][j]$ corresponde à la $j$ème annotation du $i$ème énoncé de d. A chaque colonne d'un tableau d'annotations est associé un *alphabet* Ω (*i.e.* : un ensemble de caractères) contenant l'ensemble des annotations susceptibles d'apparaître dans la dimension correspondante du schéma de codage. Par exemple, l'alphabet associé à la première colonne du schéma de codage utilisé en table 1 est $\{E, A, D\}$.

En vue d'obtenir les régularités contenues dans un corpus de tableaux d'annotations, nous cherchons à identifier des sous-parties de tableaux d'annotations apparaissant plusieurs fois, de manière exacte ou approchée. Pour ce faire, nous souhaitons identifier des *motifs* fréquents. Un *motif* est défini comme un tableau d'annotations pouvant contenir une ou plusieurs fois le caractère '*', appelé *joker* ou caractère *don't care*. Un joker peut être remplacé par n'importe quel autre caractère et permet simplement l'obtention de motifs dont la forme n'est pas nécessairement rectangulaire. Par exemple, le motif m_1 en figure 1a ne comporte que trois caractères, mais il est représenté par un tableau d'annotations de taille 3×2 grâce à l'ajout de trois jokers.

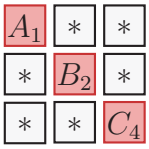
Un motif m *apparaît exactement* dans un tableau d'annotations t s'il est possible de remplaçer les caractères joker de m de telle sorte que m soit une sous-partie de t. Par exemple, le motif m_1 en figure 1a apparaît exactement dans le tableau en figure 1c (il suffit de remplacer les trois jokers de m_1 par B_2, C_2 et B_4). Un motif *apparaît de manière approchée* dans un tableau t s'il est similaire à un motif apparaissant exactement dans t (la notion de similarité entre deux motifs est précisée par la suite). Le motif m_2 (figure 1b) est similaire au motif en figure 1d (la différence se résume à l'insertion d'une ligne) qui apparaît de manière exacte dans t, m_2 apparaît donc de manière approchée dans t.

En vue d'obtenir les régularités les plus pertinentes pour un contexte dialogique donné, nous cherchons à identifier les sous-parties de tableaux d'annotations du corpus considéré qui apparaissent fréquemment. C'est la raison pour laquelle nous définissons la

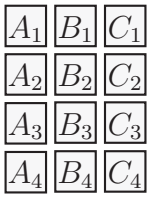
notion de *motifs récurrents*. Un motif est dit *récurrent* dans un corpus $\mathcal{C}$, s'il y apparaît, de manière exacte ou approchée, au moins deux fois (que ce soit dans deux tableaux d'annotations différents ou dans le même tableau à deux endroits différents). Chaque apparition d'un motif m dans un corpus est appelée une *occurrence de* m. Un motif m est donc récurrent si le corpus en comporte au moins deux occurrences o_1 et o_2. Ces deux occurrences sont nécessairement similaires puisqu'elles correspondent toutes deux à une apparition exacte ou approchée du même motif. Ainsi, pour détecter des motifs récurrents, nous identifions leurs occurrences (*i.e.* : les sous-parties similaires du corpus). Comme l'illustre la figure 1d, nous représentons une occurrence o d'un motif m dans un tableau d'annotations t par la sous-partie de t contenant les annotations de o et dans laquelle les autres annotations sont remplacées par des jokers.



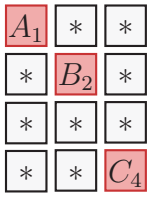
(a) Représentation d'un motif m_1.



(b) Représentation d'un motif m_2.



(c) Tableau d'annotations t dans lequel apparaît, de manière exacte, le motif m_1 et de manière approchée le motif m_2.



(d) Représentation de l'occurrence approchée du motif m_2 apparaissant dans le tableau t.

FIGURE *1 – Exemple de motif et d'occurrence de motif.*

Dans la suite, afin de simplifier la compréhension et d'utiliser des termes en cohérence avec les notations des domaines mentionnés, le terme "occurrence de motif récurrent" pourra simplement être remplacé – lorsque cela ne soulève aucune ambiguïté – par le terme "motif".

Nous définissons maintenant les deux formes que peut prendre l'approximation entre un motif et son occurrence dans le corpus :

- (V_{car}) variabilité des caractères ;
- (V_{ag}) variabilité de l'agencement des annotations.

L'approximation causée par (V_{car}) peut être illustrée par le motif m représenté en figure 2a. Les annotations en rouge dans le tableau t_A en figure 2b représentent une occurrence de m qui est approchée car les caractères E et I ont respectivement été substitués par les caractères F et J. La première raison pour laquelle ce type de variabilité doit être prise en compte est que plusieurs annotations d'un même alphabet d'une colonne d'annotations peuvent avoir des significations similaires et qu'il est donc intéressant de pouvoir les rapprocher (*e.g.* : des annotations correspondant à la présence d'émotions similaires mais non identiques). Le deuxième argument est que plusieurs facteurs sont susceptibles d'influencer la phase d'acquisition des tableaux d'annotations (*e.g.* : précision des outils d'annotations, accords inter-annotateurs, ...). La phase d'annotation ne

peut donc être supposée parfaite. Ainsi, se restreindre aux rapprochements de motifs dont les caractères sont identiques serait trop restrictif et empêcherait d'identifier des régularités intéressantes.



(a) Motif m. *(b) Tableau t_A.* *(c) Tableau t_B.* *(d) Tableau t_C.*

FIGURE *2 – Exemples d'occurrences approchées d'un motif m dans trois tableaux d'annotations t_A, t_B et t_C. Les annotations des trois tableaux correspondant au motif sont représentées en rouge.*

La variabilité (V_{ag}) concerne l'agencement (*i.e.* : la disposition) des annotations du motif et de l'occurrence. Nous souhaitons autoriser que des annotations puissent être décalées d'une ou plusieurs lignes. Par exemple, les tableaux t_B et t_C (figures 2c et 2d) représentent deux occurrences approchées du motif m en figure 2a. Dans t_B, les deux annotations de la seconde ligne de m sont décalées tandis que dans t_C, seule l'annotation E est déplacée. Les variations peuvent s'interpréter en rappelant que les lignes d'un tableau d'annotations sont ordonnées chronologiquement. Un décalage d'une ligne vers le bas d'une partie d'un motif signifie donc qu'une partie de l'évènement récurrent dénoté par ce motif a été retardée. Divers évènements peuvent causer ce type de retard, comme par exemple le fait que les interlocuteurs ne se connaissent pas ou qu'un orateur développe plus précisément qu'un autre un argument. Il est donc primordial de permettre la prise en compte de cette variabilité lors de la détection de motifs récurrents. Nous ne considérons pas dans ce contexte les décalages le long des colonnes car ils ne sont pas envisageables dans les tableaux d'annotations où chaque colonne possède un alphabet indépendant (une annotation représentant une émotion ne pourrait, par exemple, pas apparaître dans une autre colonne que celle relative aux émotions).

Méthodologie et problématique scientifique

Afin de réaliser l'extraction de motifs récurrents figurant dans un corpus de tableaux d'annotations, nous définissons une méthodologie comportant les deux étapes suivantes :

1. Extraction d'occurrences de motifs récurrents ;
2. Partitionnement des occurrences extraites.

Durant la première étape, les sous-parties de tableaux d'annotations similaires – qui correspondent à deux occurrences d'un seul et même motif récurrent – seront identifiées. La difficulté de cette phase d'extraction provient, d'une part, des données considérées qui sont des tableaux bidimensionnels et, d'autre part, du fait que deux occurrences d'un même motif peuvent différer selon les variabilités (V_{car}) et (V_{ag}).

Le but de la seconde étape est de partitionner les occurrences de motifs de telle sorte que chaque partie corresponde à l'ensemble des occurrences d'un seul et même motif récurrent. Le problème de partitionnement a été intensivement étudié et possède de nombreuses variantes dans lesquelles le nombre de parties et leur poids peuvent être contraint. Malheureusement, ces dernières sont généralement $\mathcal{NP}$-difficiles [45, 29].

Chacune de ces deux étapes nécessite un travail de modélisation du problème correspondant, ainsi que de sélection ou de définition d'une méthode de résolution perfor-

mante. Ce sont les raisons pour lesquelles ce sujet a été financé par le projet PRIMO [1] et constitue une collaboration entre les laboratoires LMI et LITIS de l'INSA de Rouen.

Pour chacune des deux étapes de la méthodologie, la meilleure façon de juger objectivement de la qualité de notre modélisation serait de s'appuyer sur une méthode de résolution exacte. En effet, lorsqu'une solution obtenue par une heuristique ne satisfait pas pleinement l'utilisateur, il est impossible de déterminer si cela est dû à la modélisation en elle-même ou à la méthode de résolution approchée employée. Ainsi, nous préférons, lorsque cela est envisageable (ce qui est le cas de la phase de partitionnement), nous orienter vers des méthodes de résolution exactes.

Le plan de ce manuscrit est représenté en figure 3. La première partie est consacrée à l'extraction de motifs. Le chapitre 1 est dédié aux méthodes de la littérature susceptibles d'y parvenir. Dans le chapitre 2 nous présentons tout d'abord la méthode d'extraction LPCA-DC que nous obtenons en adaptant un algorithme de programmation dynamique réalisant des alignements locaux dans des tableaux de caractères [83]. Afin de pallier les lacunes de cette méthode, nous réalisons ensuite une modélisation formelle du problème d'extraction de motifs sous la forme d'une recherche d'arborescences de poids maximal dans un graphe représentant deux tableaux d'annotations. Enfin, nous définissons une méthode d'extraction nommée SABRE permettant de résoudre plus efficacement ce problème.

Dans la seconde partie de ce manuscrit, nous traitons la phase de partitionnement de motifs. Le chapitre 3 est dédié à l'étude des méthodes de résolutions exactes et approchées de problèmes de partitionnement. Dans le chapitre 4 nous définissons et étudions deux formulations linéaires en nombres entiers originales du problème de K-partitionnement. Les conditions sous lesquelles trois familles d'inégalités non triviales définissent des facettes de l'enveloppe convexe des points entiers associés à une de ces formulations sont ainsi caractérisées. Enfin, dans le chapitre 5 nous étudions l'amélioration moyenne sur la solution de la relaxation lors de l'ajout de coupes provenant de ces trois familles d'inégalités, puis nous mettons à profit ces résultats dans le cadre d'un algorithme de plans coupants.

La dernière partie est consacrée à deux expérimentations (chapitre 6) réalisées sur un corpus de dialogues annotés qui nous ont permis d'évaluer l'efficacité de notre méthodologie et des méthodes définies. Pour terminer, les conclusions et perspectives de nos travaux sont discutées dans le dernier chapitre.

1. Projet de Recherche en Ingénierie des Méthodes d'Optimisation de l'INSA de Rouen qui finance des collaborations inter-laboratoires.

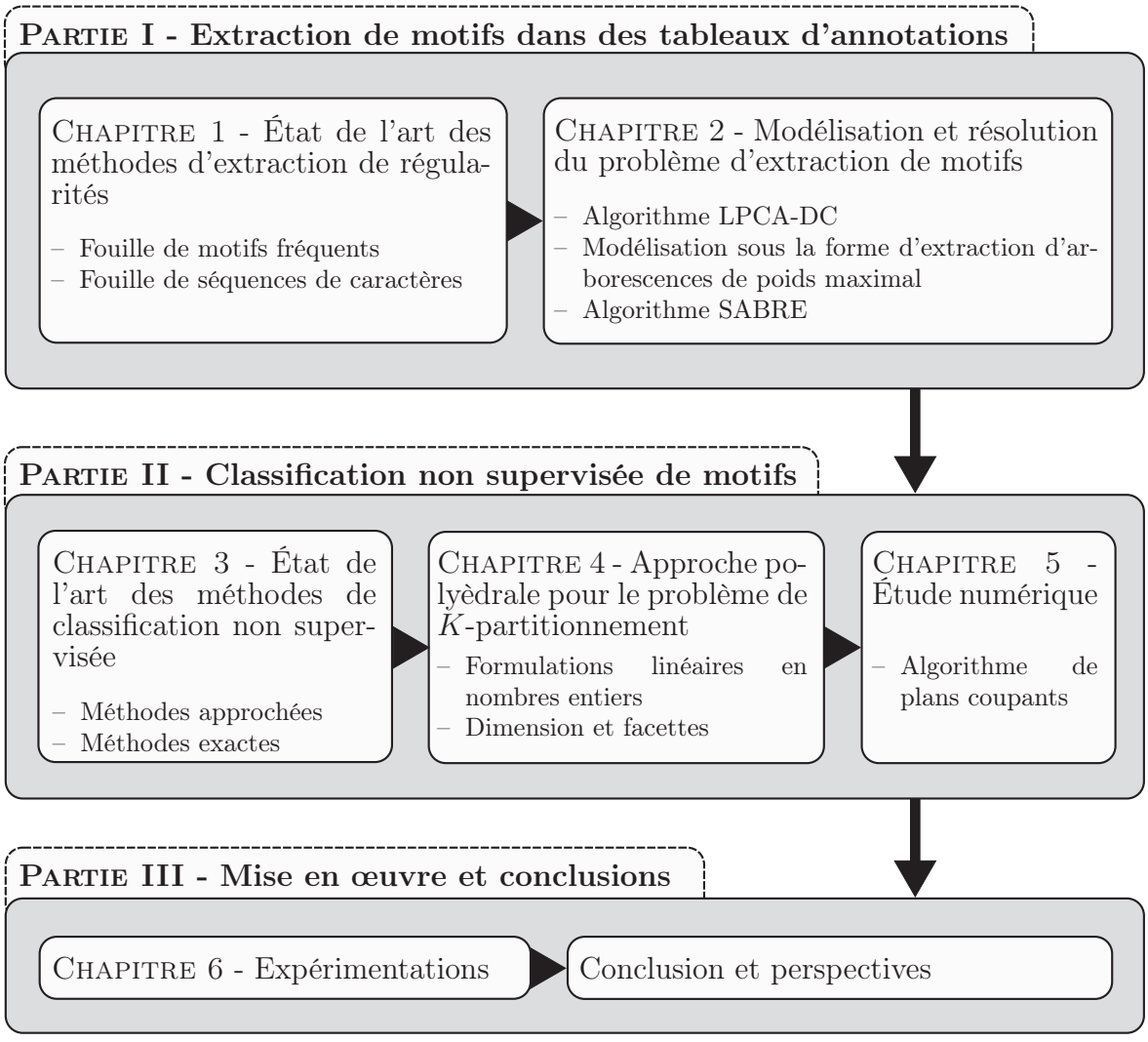


Figure *3 – Représentation graphique du plan.*

Première partie

Extraction de motifs dans des tableaux d'annotations

1 État de l'art des méthodes d'extraction de régularités 20

1.1 Fouille de motifs fréquents 21

1.1.1 Fouille d'itemsets fréquents 21

1.1.2 Fouille de séquences d'itemsets fréquentes 25

1.1.3 Extraction de sous-structures fréquentes 31

1.2 Fouille de séquences de caractères 35

1.2.1 Algorithme de Needleman-Wunsch 38

1.2.2 Alignement local de séquences de caractères 40

1.2.3 Alignement local de tableaux de caractères 41

1.3 Discussion 45

2 Modélisation et résolution du problème d'extraction de motifs 48

2.1 Algorithme $LPCA - DC$ 48

2.1.1 Description de l'algorithme 48

2.1.2 Discussion 51

2.2 Approche par extraction d'arborescences 52

2.2.1 Définitions et notations 53

2.2.2 Calcul du score d'un alignement 53

2.2.3 Modélisation du problème d'extraction d'alignements 57

2.2.4 Algorithme SABRE 58

2.3 Conclusion 62

L'objectif de cette première partie est de définir des techniques permettant l'extraction de motifs récurrents à partir de tableaux d'annotations.

Dans cette optique, nous passons en revue dans le chapitre 1 les différentes approches de l'état de l'art susceptibles d'y parvenir. Nous identifions deux domaines – la fouille d'itemsets fréquents (*frequent itemsets mining*) et la fouille de séquences de caractères (*string mining*) – susceptibles d'y parvenir. A la suite de cette étude bibliographique, il apparaît que la méthode de la littérature qui répond le mieux aux contraintes de notre problème est une méthode d'alignement bidimensionnel généralisant une méthode de programmation dynamique utilisée pour l'alignement local de séquences ADN.

Le chapitre 2 est consacré à la présentation des adaptations et des améliorations apportées à cette méthode lors de son application à l'extraction de motifs récurrents dans des tableaux d'annotations. Les résultats expérimentaux présentés en chapitre 6 montrent que la méthode permet une identification de motifs pertinents et interprétables par un expert. Cependant, nous constatons aussi que la forme des motifs extraits est relativement contrainte et dépendante de l'ordre dans lequel les colonnes du schéma de codage sont disposées tandis que dans notre objet d'étude, l'ordre des colonnes est arbitraire. Ces deux observations nous amènent à définir une modélisation du problème d'extraction de motifs récurrents dans des tableaux d'annotations et à proposer un algorithme d'extraction basé sur cette modélisation. Les performances de ce nouvel algorithme sont comparées à celle du précédent dans le chapitre 6 dédié aux expérimentations.

1 | État de l'art des méthodes d'extraction de régularités

Les variabilités (V_{car}) et (V_{ag}) présentées en introduction peuvent apparaître simultanément dans une occurrence d'un motif récurrent et rendent complexe leur extraction. C'est pourquoi nous cherchons à identifier des méthodes performantes de la littérature permettant d'extraire des motifs récurrents d'un corpus de tableaux d'annotations.

Dans ce chapitre, nous identifions les méthodes de l'état de l'art susceptibles de permettre une extraction de motifs récurrents à partir d'un corpus de tableaux d'annotations $\mathcal{C}$.

Les motifs que nous cherchons à extraire ne sont pas connus à l'avance. Nous ne possédons donc pas de connaissances a priori nous permettant d'utiliser des méthodes de types supervisées telles que les arbres de décision [111] ou les classifieurs bayésiens [110] pour effectuer cette tâche.

L'extraction non supervisée d'éléments récurrents est une activité courante aux multiples applications (compression de données, indexation de contenu, prédiction, ...) et c'est la raison pour laquelle des nombreuses recherches ont été effectuées dans cette optique. Vinciarelli [122] ainsi que Barzilay *et al.* [16] cherchent, par exemple, à identifier des régularités caractéristiques d'un orateur à partir de signaux audio. Dans le domaine de l'extraction d'information, de nombreux travaux ont été dédiés à la recherche de régularités dans des textes en langage naturel [109, 120, 128]. Nous pouvons aussi mentionner, Rao *et al.* [108] ainsi que Chowdhary *et al.* [26] qui étudient le problème de la conception de circuits intégrés de grande taille dans lesquels la détection de régularités permet de diminuer les coûts de conception.

Les techniques utilisées dans ces travaux sont, cependant, trop spécifiques aux types de données considérés pour nous permettre d'en envisager une adaptation à l'extraction de régularités dans des annotations de dialogues. Les méthodes d'extraction d'information, par exemple, se basent fortement sur le fait que les textes considérés sont structurés en phrases, ce qui n'est le cas d'un tableau d'annotations.

Nous avons néanmoins identifié deux domaines de recherche – la fouille de motifs fréquents (*frequent pattern mining*) et de séquences de caractères (*string mining*) – permettant l'extraction de régularités dans des types de données pouvant être utilisés pour représenter des tableaux d'annotations. Une section est dédiée à chacun de ces deux domaines, dans lesquelles nous précisons la ou les transformations permettant de faire le lien entre les types de données utilisés et les tableaux d'annotations, puis nous présentons leurs principales approches en soulignant leurs forces et leurs faiblesses vis-à-vis de notre problématique. Enfin, en section 1.3, les méthodes identifiées sont comparées

et la plus adéquate est identifiée.

1.1 Fouille de motifs fréquents

L'objectif de l'exploration de motifs est de détecter les structures les plus fréquentes apparaissant dans une base de données. Dans cette section nous n'avons pas pour objectif d'effectuer une présentation exhaustive des travaux de ce domaine, mais nous nous concentrons sur les techniques susceptibles d'être utilisées pour extraire des motifs récurrents dans des tableaux d'annotations et permettant la prise en compte des variabilités (V_{car}) et (V_{ag}) mentionnées en début de ce chapitre. Ainsi, certains aspects comportant peu de liens avec notre problématique – tels que les règles d'association [3], les motifs fermés [57], maximaux [126], multi-niveaux [57] ou multi-dimensionnels [105] – ne seront pas ici abordés.

Comme le présente Han *et al.* [56], les méthodes d'exploration de motifs fréquents peuvent être partitionnées en trois classes selon le type de structures fréquentes qu'elles permettent d'extraire : itemsets, séquences d'itemsets ou sous-structures. Dans la suite, une sous-section est consacrée à chacune de ces classes. Nous y décrivons comment modéliser un tableau d'annotations ainsi que les techniques principales qui ont été développées. Sauf indication contraire, le terme *motif* est par la suite utilisé pour désigner le type de structures que les algorithmes cherchent extraire (*i.e.* : respectivement itemsets fréquents, séquence d'itemsets fréquentes et sous-structures fréquentes dans les sous-sections 1.1.1, 1.1.2 et 1.1.3).

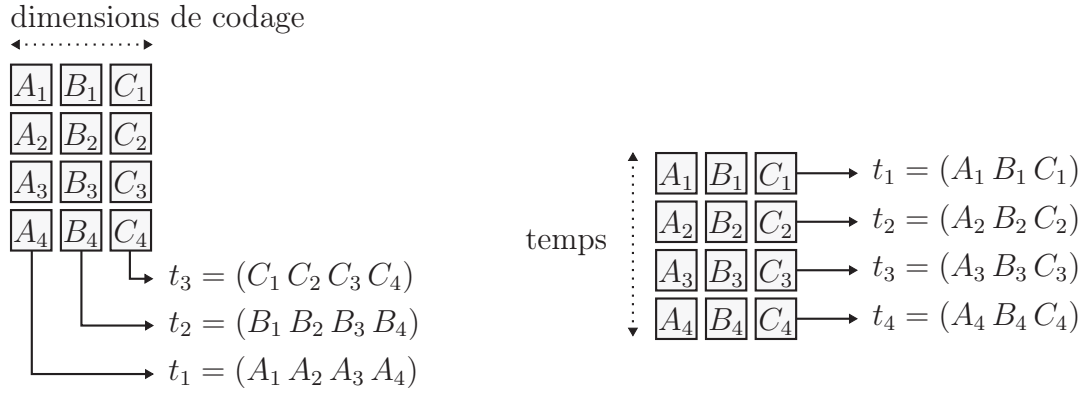
1.1.1 Fouille d'itemsets fréquents

La première évocation de cette problématique a été présentée par Agrawal *et al.* en 1993 [3]. Leur l'objectif était d'analyser les habitudes d'achat de clients via leurs paniers de consommation (*market basket*). Ceci explique que plusieurs des termes relatifs à cette problématique soient employés dans la fouille de motifs fréquents (*e.g.* : item, transaction, client).

Soit $I = \{i_1, \ldots, i_n\}$ un ensemble d'items. Une *transaction* est définie comme une collection non vide d'items de I à laquelle est associée un identifiant unique. Une transaction contenant deux items a et b est notée (ab) et une transaction ne contenant qu'un unique item c est simplement notée c afin d'alléger les notations. Considérons le cas du panier de consommation afin d'illustrer ces notations. Dans ce contexte les items sont des articles de magasins (*e.g.* : du lait, de la confiture), une transaction t correspond à un ensemble d'articles achetés en une fois par un client et l'identifiant associé à la transaction t permet, entre autres, de retrouver le client ainsi que la date à laquelle a eu lieu la transaction. Un *itemset* est un sous-ensemble d'items de I. L'itemset I_1 est dit inclus dans une transaction T si $I_1 \subseteq T$ (*i.e.* : si tous les items de I_1 figurent dans la transaction T). Étant donnée une *base de données* de transactions $\mathcal{D} = \{T_1, \ldots, T_{|\mathcal{D}|}\}$, le support de l'itemset I_1 est défini comme la proportion de transactions de $\mathcal{D}$ incluant I_1. Par exemple, le support des itemsets $\{a\}$ et $\{b, d\}$ dans la base de données $\mathcal{D} = \{(abc)(ad)(bd)c\}$ sont respectivement 50% et 25%. Un itemset est dit *fréquent* si son support est supérieur ou égal à un support minimum fixé et noté $min_support$. Enfin, un itemset contenant k

items est appelé un k-itemset.

Deux modélisations sont naturellement envisageables pour représenter un tableau d'annotations sous la forme d'itemsets comme l'illustre la figure 1.1. Dans la première (figure 1.1a), à chaque colonne est associée une transaction contenant un item pour chaque annotation de la colonne. Cette représentation a pour premier désavantage d'entraîner la perte de l'information de temporalité puisque tous les items d'une transaction sont considérés comme avoir été achetés en une seule fois. De plus, dans les tableaux d'annotations chaque colonne contient des caractères appartenant à un alphabet qui lui est propre. Le deuxième inconvénient de cette première représentation est donc qu'un caractère donné ne pourra figurer que dans un unique item correspondant au tableau d'annotations. Les motifs qu'il serait possible d'extraire avec cette modélisation seraient donc très limités. La deuxième représentation envisageable (figure 1.1b) consiste à associer à chaque ligne une transaction dont les items correspondent aux annotations qu'elle contient. Cette représentation ne possède pas les défauts mentionnés précédemment mais est néanmoins trop restrictive puisqu'un itemset fréquent correspondrait alors uniquement à des annotations figurant fréquemment sur une même ligne d'un tableau d'annotations.



(a) Une transaction équivaut à une colonne.

(b) Une transaction équivaut à une ligne.

FIGURE *1.1 – Deux transformations permettant de représenter un tableau d'annotations sous la forme d'une base de données de transactions.*

Ainsi, les méthodes de fouille d'itemset ne semblent pas appropriées à notre problématique. Néanmoins, certaines familles d'algorithmes classiques de fouille d'itemsets fréquents servent de base à de nombreux algorithmes de fouille de séquences d'itemsets ou de sous-structures fréquentes. C'est la raison pour laquelle, nous présentons ici ces principales familles d'algorithmes et décrivons brièvement leurs mécanismes.

Algorithme Apriori

Les algorithmes de type Apriori sont basés sur la propriété de la fermeture par le bas (*downward closure*) qui indique que tout itemset inclus dans un itemset fréquent est nécessairement fréquent. Par exemple, si le 4-itemset $\{a, b, c, d\}$ est fréquent, les 3-itemsets qu'il contient (*i.e.* : $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$ et $\{b, c, d\}$) le sont nécessairement eux aussi. Le premier algorithme de ce type a été proposé par Agrawal *et al.* [3] en 1993.

Dans ces algorithmes, les items fréquents – qui correspondent aux 1-itemsets fréquents – sont tout d'abord identifiés. Ensuite, pour tout entier k à partir de un et tant

qu'il existe des k-itemsets fréquents, les deux étapes suivantes sont réalisées :

1. génération de $(k+1)$-itemsets *candidats* (*i.e.* : les $(k+1)$-itemsets susceptibles d'être fréquents) ;
2. *élagage* (*i.e.* : suppression) des candidats qui ne sont pas fréquents par calcul de leur support.

Les diverses méthodes Apriori diffèrent par la façon dont ces deux étapes sont réalisées. Leur principal défaut est qu'elles nécessitent de nombreuses lectures de la base de données (lors de la génération et de l'élagage des candidats) qui peuvent s'avérer très coûteuses en temps d'exécution.

Croissance de motifs

Les méthodes de type croissance de motifs sont une alternative aux algorithmes Apriori et ont été développées en vue de pallier leurs défauts en s'affranchissant, notamment, de la phase de génération des candidats.

Afin d'illustrer leur fonctionnement, la description des principales étapes de ce type de méthodes est illustrée par l'algorithme *FP-Growth* (*Frequent Pattern-Growth*) [59] qui est l'un des plus emblématiques. Nous utilisons la base de données représentée en figure 1.2a et considérons un support minimal de 25%.

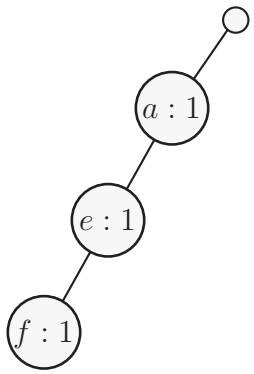
La première étape consiste à identifier les items fréquents contenus dans $\mathcal{D}$ et à les lister par fréquence décroissante. Dans l'exemple, la liste ainsi obtenue est : $(a, 3)$ $(e, 3)$, $(d, 2)$, $(f, 2)$, $(g, 2)$, $(h, 2)$, $(i, 2)$ et $(j, 2)$. Pour chaque transaction, les items fréquents qu'elle contient sont identifiés et triés en fonction de leur fréquence (*e.g.* : troisième colonne de la figure 1.2a). Ensuite, l'algorithme crée une structure de données permettant une compression significative des données. Dans le cas de *FP-Growth*, cette structure est un arbre $T(V, E)$, appelé *FP-tree*. A chaque sommet de V – à l'exception de la racine – est associé un item fréquent ainsi qu'une valeur entière que nous appelons un *compteur*.

Chaque branche de cet arbre correspond à des items figurant ensemble dans une ou plusieurs transactions. La valeur du compteur d'un sommet $v \in V$ est égale au nombre de transactions contenant, d'une part, l'item associé à v et, d'autre part, les items associés aux sommets se trouvant au-dessus de v dans la branche (*e.g.* : en figure 1.2d, le compteur de l'unique sommet associé à f est égal à deux car exactement deux transactions de la base contiennent les trois items a, e et f). Pour construire l'arbre relatif à la base qui nous sert d'exemple, une première branche (représentée en figure 1.2b), correspondante aux items fréquents de la première transaction est créée. Lors de la création d'un sommet, son compteur est toujours initialisé à un. La liste des items fréquents de la seconde transaction est a, e, h et i. Comme les deux premiers items de cette liste figurent aussi dans la première transaction, le compteur des deux sommets qui leur sont associés est incrémenté et une nouvelle branche, illustrée en figure 1.2c, est créée. L'arbre est complété de manière similaire pour chacune des transactions suivantes. Le résultat obtenu est représenté en figure 1.2d. Cette représentation est généralement très compacte lorsque de nombreuses transactions possèdent des items en commun.

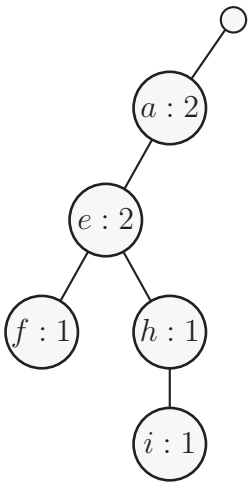
Ensuite, au lieu de rechercher et d'élaguer successivement des candidats dans l'ensemble de la base de données, l'espace de recherche est partitionné. Pour *FP-Growth* c'est le nombre de motifs fréquents qui détermine le nombre de parties. La base de données $\mathcal{D}$ précédemment considérée contient huit motifs fréquents qui donneront lieu à une division de l'espace de recherche en huit parties : (1) transactions contenant j, (2) tran-

Client	Transaction	Items fréquents
1	$(abefl)$	a, e, f
2	$(aehik)$	a, e, h, i
3	$(cdgh)$	d, g, h
4	$(dgijm)$	d, g, i, j
5	$(aefj)$	a, e, f, j

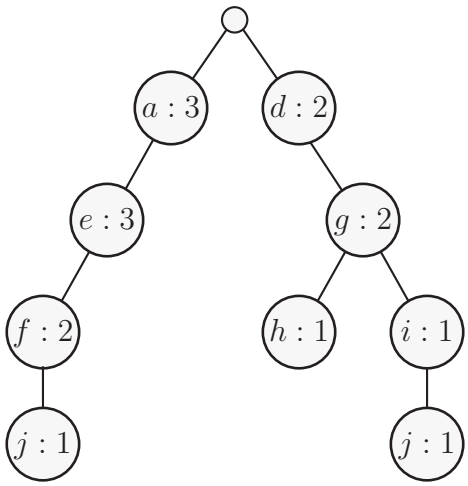
(a) Base de données $\mathcal{D}$ et liste des items fréquents (ordonnés par fréquence décroissante) de chaque transaction dans le cas où le support minimal est 25 %.



(b) Branche du FP-tree correspondant à la première transaction de la base de données $\mathcal{D}$.



(c) Branches du FP-tree correspondant aux deux premières transactions de la base de données $\mathcal{D}$.



(d) FP-tree correspondant à la base de données $\mathcal{D}$.

FIGURE *1.2 – Base de données $\mathcal{D}$ et FP-tree correspondants construit par l'algorithme FP-growth. A chaque sommet de l'arbre est associé un item fréquent. Chaque branche de l'arbre correspond à une liste d'items fréquents figurant dans une ou plusieurs transactions. A chaque sommet est, de plus, associé un item fréquent, un compteur correspondant au nombre de transactions contenant cet item ainsi que ceux qui le précèdent dans la branche de l'arbre.*

sactions contenant i et ne contenant pas j, (3) transactions contenant h et ne contenant ni i ni j, ..., (8) transactions contenant uniquement a. Pour chacune de ces classes, des branches du *FP-tree* sont identifiées. Dans le cas des motifs contenant h mais pas i et j, l'unique branche correspondante est $\{d, g, h\}$. Cette approche de type "partager pour régner" (*divide and conquer*) permet une réduction significative des temps d'exécution.

Ces deux familles d'algorithmes servent de base à de nombreuses méthodes de fouille de séquences d'itemsets et de sous-structures.

Dans les sous-sections suivantes, nous montrons notamment comment ces deux familles de méthodes peuvent se décliner afin de permettre l'extraction de motifs sous la forme de séquences d'itemsets ou de sous-structures.

1.1.2 Fouille de séquences d'itemsets fréquentes

Le problème de fouille de séquences d'itemsets a été en premier considéré par Agrawal et Srikant en 1995 [4] afin d'extraire des motifs dans des transactions ordonnées chronologiquement. Depuis, cette problématique a reçue beaucoup d'attention et a été appliquée à de nombreux jeux de donnés (transactions, flux, séries temporelles, ...), entraînant une grande diversité d'approches *ad hoc* comme l'attestent les divers rapports menés sur le sujet [95, 82, 88].

Une *séquences d'itemsets* $\alpha = \langle \alpha_1 \dots \alpha_n \rangle$ est une liste ordonnée de transactions effectuées par un seul et même client. Par exemple, dans la liste de transactions représentée en table 1.1, la séquence associée au client numéro 1 est $\langle (ab)c(de) \rangle$. La base de données de séquences d'itemsets complète correspondant à cet exemple est $\{\langle (ab)c(de) \rangle, \langle e(acd)(bf) \rangle, \langle (ae)c \rangle\}$. La séquence $\langle \beta_1, \dots, \beta_m \rangle$ est une *sous-séquence* de la séquence α s'il existe des entiers $j_1 < j_2 < \dots < j_n$ tels que pour tout $k \in \{1, \dots, n\}$ $\beta_k \subseteq \alpha_{j_k}$. Par exemple, $\langle a(cd) \rangle$ est une sous-séquence de $\langle (ab)(bc)(cd) \rangle$, ce qui n'est pas le cas de $\langle (bc)a \rangle$.

Client	Date	Transaction
1	2	(ab)
	4	c
	7	(de)
2	3	e
	5	(acd)
	6	(bf)
3	1	(ae)
	2	c

TABLE *1.1 – Liste de transactions auxquelles sont associées l'identifiant de leur client et de la date à laquelle elles ont été réalisées.*

Étant donnée une base de séquences $\mathcal{D}$, diverses définitions du support d'une séquence α ont été définies. Celle qui est la plus communément utilisée est la proportion de séquences de $\mathcal{D}$ contenant α. Tout comme dans le cas de la fouille d'itemsets fréquents, une séquence est dite fréquente si son support est supérieur ou égal au support minimal *min_supp*. Une séquence comportant k items est appelée une *k-séquence*. Un item est donc une 1-séquence tandis que $\langle (ab)c \rangle$ est une 3-séquence.

Un des inconvénients des méthodes de fouille d'itemsets pour l'extraction de motifs récurrents dans un corpus de tableaux d'annotations est qu'elles ne permettent pas de tenir compte de la temporalité représentée par les lignes des tableaux. La fouille de séquences fréquentes permet de pallier cela. En effet, comme l'illustre la figure 1.3, un tableau d'annotations pourra être représenté par une séquence d'itemsets dans laquelle chaque transaction correspond à une ligne et chaque item à une annotation. La temporalité des lignes est conservée en ordonnant de manière similaire les transactions dans la séquence.

Nous pouvons d'ores et déjà remarquer que dans la définition présentée d'une séquence fréquente, la durée entre deux transactions (*i.e.* : le nombre de lignes séparant

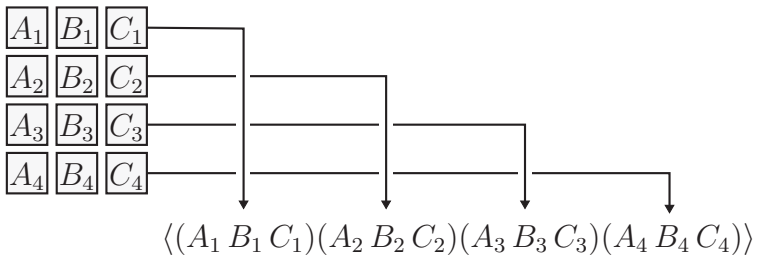


FIGURE *1.3 – Transformation permettant de convertir un tableau d'annotations en séquence d'itemsets.*

les annotations) n'a pas d'impact sur les motifs obtenus. Le seul aspect pris en compte est l'ordre dans lequel les itemsets apparaissent. Ceci est pour nous un inconvénient puisqu'avec cette définition, une partie des décalages permis par la variabilité (V_{ag}) ne pourront pas être pris en compte. Cependant, nous montrons que dans chacune des familles d'algorithmes présentées, des variantes permettent une modélisation plus fine de l'aspect temporel.

Tout comme pour la fouille d'itemsets, les méthodes sont ici regroupées par principales familles d'algorithmes.

Algorithme Apriori

Les trois premiers algorithmes de ce type permettant d'extraire des séquences fréquentes, nommés *AprioriAll*, *AprioriSome* et *DynamicSome*, ont été proposés par Agrawal et Srikant [4]. Dans ces derniers, les itemsets fréquents sont d'abord identifiés en utilisant un algorithme de fouille d'itemset Apriori. La seule différence est que le support d'un itemset i ne correspond plus désormais à la proportion de transactions contenant i mais à la proportion de clients ayant réalisé une transaction contenant i. Si nous considérons un support de 25% et la base $\mathcal{D}$ représentée en première colonne de la table 1.2, les itemsets fréquents obtenus sont $\{a\}$, $\{b\}$, $\{a, b\}$ et $\{c\}$. Puisqu'une séquence fréquente est nécessairement composée d'itemsets qui le sont aussi, il est possible d'obtenir une représentation plus compacte de la base de données de séquences en remplaçant chacune de ces transactions par l'ensemble des itemsets fréquents qu'elle contient (*e.g.* : deuxième colonne de la table 1.2). Les transactions et les clients ne contenant aucun itemset fréquent sont filtrés (*e.g.* : quatrième transaction en table 1.2). Cette phase de compression de la base permet une meilleure efficacité et est répétée à mesure que la taille des séquences fréquentes trouvées augmente.

Séquences	**Reformulation**
$\langle (ab)(bc)c\rangle$	$\langle \{a, b, \{ab\}\}\{bc\}\{c\}\rangle$
$\langle (cd)(ab)\rangle$	$\langle \{c\}\{a, b, \{ab\}\}\rangle$
$\langle ac\rangle$	$\langle \{a\}\{c\}\rangle$
$\langle d\rangle$	$\langle \emptyset\rangle$
$\langle cd(ab)\rangle$	$\langle \{c\}\{a, b, \{ab\}\}\rangle$

TABLE *1.2 – Base de données $\mathcal{D}$ illustrant les algorithmes AprioriAll, AprioriSome et DynamicSome. La première colonne correspond aux séquences de $\mathcal{D}$. La seconde colonne représente la reformulation des séquences en fonction des itemsets fréquents qu'elles contiennent, dans le cas où le support minimum est égal à 25%.*

La suite de ces algorithmes consiste à successivement générer à partir de séquences fréquentes contenant k itemsets une liste de séquences candidates possédant $k+1$ itemsets. Le support de chacun des candidats est ensuite vérifié afin de ne garder que ceux qui sont fréquents. Le processus s'arrête lorsqu'à une itération donnée, aucune séquence fréquente n'est obtenue. Ce sont les mécanismes employés dans ces étapes qui font la spécificité des algorithmes *AprioriAll, AprioriSome* et *DynamicSome*.

Ces trois approches ont pour inconvénient de ne considérer que l'ordre dans lesquelles apparaissent les transactions d'une séquence et pas la durée les séparant. Ainsi, deux transactions successives d'une sous-séquence fréquente pourront correspondre en réalité à des transactions très éloignées temporellement dans les séquences d'origine de la base $\mathcal{D}$. Cet aspect semble, cependant, primordial afin de permettre l'identification de motifs intéressants. Pour reprendre l'exemple présenté par Mooney et Roddick [95], le gérant d'un magasin de location n'aura que peu d'intérêt à savoir que ses clients louent le film "Les deux tours" plusieurs années après avoir loué "La communauté de l'anneau", mais l'information sera beaucoup plus pertinente si cela se produit après quelques semaines seulement. Pour pallier cela Srikant et Agrawal [118] proposent l'algorithme GSP (*Generalised Sequential Patterns*). Ce dernier améliore la prise en compte de la temporalité en imposant des durées maximales entre deux éléments consécutifs d'une séquence fréquente. De plus, cet algorithme permet aussi l'utilisation de taxonomies (*i.e.* : de hiérarchies d'items) entraînant la possibilité d'obtenir les motifs approchés (deux items proches dans la hiérarchie pourront être considérés identiques).

Diverses adaptations de GSP, permettant la prise en compte de contraintes sur les séquences obtenues ou améliorant les performances globales ont par la suite été développées (SPIRIT [47], MFS [133], PSP [90], etc.).

Croissance de motifs

De nombreuses méthodes basées sur la croissance de motifs ont été développées afin d'extraire des séquences fréquentes d'itemsets. Comme dans le cas de la fouille d'itemsets, les méthode Apriori ont pour défaut de nécessiter de nombreux parcours de la base de données lors des phases de génération des candidats et de leur élagage ce dont s'affranchissent les algorithmes à base de croissance de motifs.

FreeSpan (*FREquEnt pattern-projected Sequential PAtterN mining*) [58] utilise ce paradigme. Tout comme l'algorithme *FP-Growth*, les items fréquents (*i.e.* : les 1-séquences fréquentes) y sont identifiés puis triés en fonction de leur fréquence. Si nous considérons la base de données $\mathcal{D}$ représentée en table 1.3 et un support minimum égal à 25%, la liste des items fréquents obtenue est $L = \{a, b, c, d, e, f\}$ (l'item g n'étant pas fréquent). Pour partitionner l'espace de recherche, cet algorithme utilise la notion de base de données projetée. Étant donné un itemset I, la base de donnée projetée de I (noté base I-projetée) est la collection de séquences contenant I. De plus, certains des items de ces séquences ne sont pas conservés. Les items qui sont gardés sont ceux contenus dans I ou ceux de L qui apparaissent avant un item de I dans cette liste. Par exemple si I est égal à $\{a, e\}$, les items a et e seront conservés, ainsi que les items b, c et d qui apparaissent avant e dans L. La base $\{b\}$-projetée de la base de données $\mathcal{D}$ de la table 1.3 est donc $\{\langle a(ab)a\rangle, \langle aba\rangle, \langle(ab)b\rangle, \langle ab\rangle\}$. Étant donnée une séquence α la notion de base α-projetée est définie de façon similaire. FreeSpan utilise ensuite le principe de diviser pour régner – caractéristique des méthodes de croissance de motifs – et tente d'extraire simultanément les séquences et itemsets fréquents. Dans l'exemple en table 1.3, six sous-ensembles, cor-

respondant à chacun des éléments de L, seront considérés. Ensuite, les motifs fréquents de taille 2 de la base $\{a\}$-projetée seront identifiés (ici seul $\{a, a\}$ est obtenu). Puis, les bases projetées correspondant aux motifs obtenus seront elles-mêmes considérées afin d'obtenir, de manière similaire, des motifs de taille 3. Le processus est ainsi répété en augmentant la taille des motifs obtenus jusqu'à ce qu'aucun motif ne soit retourné. La procédure complète est répétée en considérant successivement les bases projetées correspondant aux items suivant a dans la liste L (*i.e.* : les bases $\{b\}$-projetée, $\{c\}$-projetée, ..., $\{f\}$-projetée).

Nom de la base	**Contenu de la base**			
$\mathcal{D}$	$\langle a(abc)(ac)d(cf)\rangle$	$\langle(ad)c(bc)(ae)\rangle$	$\langle(ef)(ab)(df)cb\rangle$	$\langle eg(af)cbc\rangle$
$\{a\}$-projetée	$\langle aaa\rangle$	$\langle aa\rangle$	$\langle a\rangle$	
$\{b\}$-projetée	$\langle a(ab)a\rangle$	$\langle aba\rangle$	$\langle(ab)b\rangle$	$\langle ab\rangle$
$\{c\}$-projetée	$\langle a(abc)(ac)c\rangle$	$\langle ac(bc)a\rangle$	$\langle(ab)c\rangle$	$\langle acbc\rangle$

TABLE *1.3 – Base de données $\mathcal{D}$ et trois exemples de bases projetées.*

La méthode *PrefixSpan* (*PREFIX-projected Sequential PAtterN mining*) [104] est basée sur *FreeSpan*. Les items fréquents de la liste L y sont considérés comme des préfixes afin de réduire la taille de l'espace de recherche. Pour un préfixe p donné, chaque séquence est tronquée afin qu'elle commence à la première occurrence de p. Par exemple pour le préfixe a et la séquence $\langle(ef)cba(df)\rangle$, seule la sous-séquence $\langle a(df)\rangle$ est considérée. Au cours de l'algorithme, la taille des préfixes augmente avec celle des motifs.

Seno et Karypis [112] ont proposé l'algorithme *SLPMiner* (Sequential pattern mining with Length-decreasing suPport) utilisant lui aussi la projection de bases de données et permettant, de plus, d'adapter le support minimal requis à la taille des motifs afin de ne pas obtenir uniquement de petits motifs ayant un grand support mais aussi des grands motifs ayant un support plus modeste.

Base de données verticales

La représentation classique d'une base de données de séquences consiste à faire correspondre à chaque client et à chaque date de transaction la liste d'items correspondante (*e.g.* : voir table 1.4a). Les méthodes présentées ici tirent parti d'une représentation alternative des données permettant d'accélérer le calcul du support des itemsets candidats.

Dans le cas de la méthode SPADE (*Sequential PAttern Discovery using Equivalence classes*) développée par Zaki [131], à chaque item est associé la liste des transactions dans lesquelles ce dernier est présent (les transactions sont identifiées par le client et la date à laquelle elles ont été effectuées), comme l'illustre la figure 1.4b qui correspond à la base de données en table 1.4a. La liste obtenue pour un item donné est appelé une *id-list*. L'algorithme SPADE utilise, en plus des id-lists, une décomposition de l'espace de recherche en sous-treillis pouvant être traités indépendamment. L'algorithme cSPADE (*constrained SPADE*) [130] complète la méthode SPADE en lui permettant la prise en compte de diverses contraintes : limitation de la taille des séquences, bornes sur les écarts temporels possibles entre deux transactions successives d'une séquence, fenêtre de temps contenant l'ensemble des transactions d'une séquence et exclusion de certains items dans les séquences.

Client	Date	Transaction
1	1	(a)
	3	(bc)
	4	(abd)
2	2	(bc)
	3	(b)
3	2	(ab)
	5	(cd)
	6	(abc)

(a) Table représentant une base de données $\mathcal{D}$ composée des trois séquences $\langle a(bc)(abd)\rangle$, $\langle(bc)b\rangle$ et $\langle(ab)(cd)(abc)\rangle$. Chaque transaction y est identifiée par son client et la date à laquelle elle a été effectuée.

Item	(Client, Temps)					
a	$(1,1)$	$(1,4)$	$(3,2)$	$(3,6)$		
b	$(1,3)$	$(1,4)$	$(2,2)$	$(2,3)$	$(3,2)$	$(3,6)$
c	$(1,2)$	$(2,2)$	$(3,5)$	$(3,6)$		
d	$(1,4)$	$(3,6)$				

(b) Base de données verticale correspondant à $\mathcal{D}$ obtenue dans la méthode SPADE. Une transaction est ici identifiée par le couple $(client, temps)$ correspondant.

TABLE *1.4 – Base de données de séquences de transactions $\mathcal{D}$ et exemple de représentation verticale correspondante.*

La méthode SPAM (*Sequential PAttern Mining using a bitmap representation*) [11] utilise elle aussi une représentation verticale des données puisqu'à chaque item i est associé un vecteur binaire b_i appelé *bitmap*. Comme l'illustre la table 1.5, chaque composante de b_i correspond à une transaction t de la base de données et a pour valeur 1 si l'item i figure dans la transaction t. L'algorithme tire grandement parti de la rapidité des opérations sur les bitmaps. Par exemple étant donnés deux items i et j, les transactions contenant ces deux items sont obtenues en utilisant l'opérateur "et" sur leurs bitmaps respectives b_i et b_j (*e.g.* : dernière colonne de la table 1.5).

La méthode SPAM a été adaptée en LAPIN-SPAM (*LAst Position INduction-SPAM*) [127]. Dans cette dernière, lors de la validation des candidats, une utilisation répétée de l'opérateur "et" sur les bitmaps est évitée par l'observation suivante : étant donné un item i et une k-séquence s, si la dernière position de i est inférieure ou égale à la position du dernier item figurant dans s, alors l'item i ne peut être ajouté à s pour former une $k+1$-séquence.

Enfin, Orlando *et al.* [103] ont développé CCSM (*Cache-based Constrained Sequence Miner*) qui est dérivé de SPADE. Leur principale différence réside dans le fait que

Client	Date	Transaction	b_a	b_b	b_c	b_d	b_a et b_b
1	1	(a)	1	0	0	0	0
	3	(bc)	0	1	1	0	0
	4	(abd)	1	1	0	1	1
2	2	(bc)	0	1	1	0	0
	3	(b)	0	1	0	0	0
3	2	(ab)	1	1	0	0	1
	5	(cd)	0	0	1	1	0
	6	(abc)	1	1	1	0	1

TABLE *1.5 – Table représentant une base de données $\mathcal{D}$ composée des trois séquences $\langle a(bc)(abd)\rangle$, $\langle(bc)b\rangle$ et $\langle(ab)(cd)(abc)\rangle$. Les colonnes 4 à 7 contiennent les vecteurs bitmap, utilisés par SPAM, des quatre items apparaissant dans $\mathcal{D}$. La dernière colonne contient le vecteur bitmap obtenu en appliquant l'opérateur "et" aux vecteurs b_a et b_b.*

cette méthode garde en mémoire (*i.e.* : en cache) des id-lists intermédiaires permettant d'accélérer le temps de calcul des supports.

Séquences approchées

La notion de séquence d'itemsets approchée semble une bonne approche pour prendre en compte la variabilité (V_{car}). En effet, le fait d'autoriser, sous certaines conditions, la correspondance de deux items distincts pourrait permettre de prendre en compte cette variabilité (*e.g.* : si les items b et c sont jugés proches, les séquences $\langle(ad)b(ef)\rangle$ et $\langle(ad)c(ef)\rangle$ pourraient être dites identiques).

Ceci est rendu possible, notamment par les méthodes de fouille de motifs séquentiels flous [41, 61, 62]. Nous illustrons le principe de la logique floue par la figure 1.4. Cet exemple indique que le même adjectif ne serait pas utilisé par tout le monde pour qualifier la température d'une eau. En effet, si la température est à 0°C, elle sera qualifiée de froide par n'importe qui tandis que si elle est à 15°C, la moitié des personnes la considérera tiède.

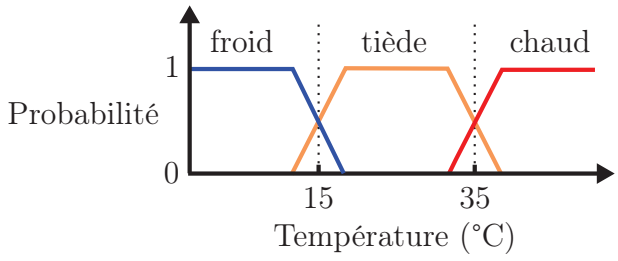


FIGURE *1.4 – Probabilité d'utiliser un adjectif pour qualifier une eau en fonction de sa température (données fictives).*

Ce type de logique peut être utilisé afin de modéliser la proximité entre différents itemsets. Soit $\mu(i,j)$ la proximité entre deux items i et j. Le support de l'item i peut être défini comme l'ensemble des transactions contenant un item j tel que $\mu(i,j)$ soit supérieur à un seuil $\omega \in [0,1]$. Le paramètre ω correspond ainsi au seuil de tolérance à partir duquel deux items sont considérés proches.

Une autre approche, nommée ApproxMAP (*Approximate Multiple Alignment Pattern mining*) a été proposée par Kum *et al.* [78]. Au lieu de trouver des motifs apparaissant de manière exacte, cette méthode identifie des motifs, appelés *consensuels*, apparaissant de manière approchée dans de nombreuses séquences. La première des deux étapes de cette méthode consiste à partitionner les séquences de la base de données. Pour ce faire, une distance dérivée de la distance d'édition (voir section 1.2 pour une description plus détaillée de cette distance) est calculée entre tous les couples de séquences de la base. A partir des distances obtenues, un algorithme à base de noyaux est ensuite utilisé afin de partitionner les séquences. Chacune des parties obtenue va ensuite entraîner la génération d'un motif consensuel. Ceci est réalisé en utilisant, en premier lieu, une heuristique permettant pour chaque partie d'aligner (*i.e.* : de juxtaposer) l'ensemble de ces séquences. De cet alignement est ensuite déduit le motif consensuel.

1.1.3 Extraction de sous-structures fréquentes

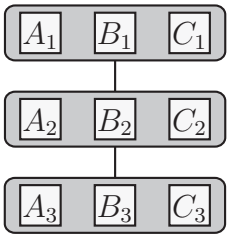
Dans le cadre de l'extraction de sous-structures fréquentes, chaque transaction est représentée par un graphe et la base de données considérée $GD = \{G_1, \ldots, G_n\}$ est un ensemble de graphes correspondant à des transactions. Les graphes étant des structures de données extrêment courantes, il n'est pas étonnant que diverses approches aient été dédiées à leur exploration comme le montre l'étude réalisée par Washio et Motoda en 2003 [124].

Dans ce contexte, le *support* d'un graphe est défini comme la proportion de graphes de GD qui l'incluent. Le problème consistant à identifier si un graphe $G_1(V_1, E_1)$ est inclus dans un graphe $G_2(V_2, E_2)$ (*i.e.* : s'il existe une bijection entre les sommets de V_1 et une sous-partie des sommets de V_2 telle que les arêtes soient conservées) est appelé l'isomorphisme de sous-graphes et est connu pour être $\mathcal{NP}$-complet [46]. C'est la raison pour laquelle les méthodes présentées dans cette sous-section sont parfois heuristiques et introduisent des limitations sur l'espace de recherche ou sur la structure des graphes.

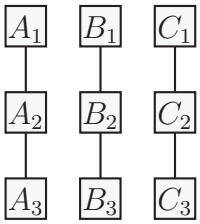
De nombreuses approches pourraient être considérées afin de modéliser des tableaux d'annotations sous forme de graphes, comme le montre la figure 1.5. Par exemple, dans la figure 1.5a à chaque ligne du tableau correspond un sommet et deux sommets sont reliés ensemble s'ils correspondent à des lignes successives. Cependant, chacune des modélisations de la figure 1.5 possède des lacunes en termes de forme des motifs qu'il serait possible d'en extraire. En effet, puisque des graphes connexes sont généralement recherchés, il ne sera pas possible avec la représentation précédemment décrite (*i.e.* : figure 1.5a) d'obtenir un motif ne comportant qu'une partie des annotations d'une ligne (*e.g.* : motif en figure 1.6a). Il en va de même pour les trois autres représentations :

- dans la modélisation figure 1.5b, il ne sera pas possible d'obtenir de motifs comportant des annotations sur plusieurs colonnes (*e.g.* : figure 1.6b) ;
- la représentation d'un tableau d'annotations selon la figure 1.5c ne permettra pas d'extraire de motifs comportant des annotations provenant de plusieurs colonnes (*e.g.* : figure 1.6c) ;
- si la modélisation de la figure 1.5d est utilisée, il ne sera pas possible d'obtenir de motifs comportant des lignes sans annotations (*e.g.* : figure 1.6d).

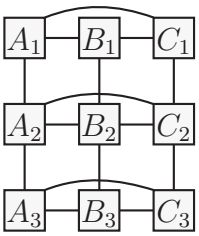
Ces différents contre-exemples montrent la grande flexibilité que nous souhaitons laisser à la forme des motifs extraits. Afin de prendre cet aspect en compte, nous consi-



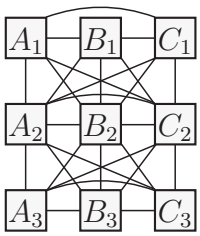
(a) A chaque ligne est associée un sommet et les sommets correspondant à des lignes successives sont reliés entre eux.



(b) A chaque annotation est associée un sommet et chaque couple de sommets correspondant à deux annotations de même colonne et dont les lignes sont consécutives sont reliées.



(c) A chaque annotation est associée un sommet. Une arête joint chaque couple de sommets correspondant à deux annotations de même ligne. Il en va de même pour les couples de sommets correspondant à des annotations de même colonne figurant sur des lignes consécutives.



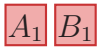
(d) A chaque annotation est associée un sommet et chaque couple de sommets correspondant à deux annotations de même ligne ou dont les lignes sont consécutives sont reliées.

FIGURE *1.5 – Exemples de représentations d'un tableau d'annotations sous forme de graphes.*

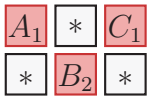
dérons donc une modélisation où à chaque annotation est associée un sommet et où il est possible de passer de n'importe quel sommet à un autre (*i.e.* : un graphe complet).



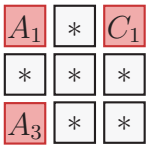
(a) Contre-exemple de la figure 1.5a



(b) Contre-exemple des figures 1.5a et 1.5b



(c) Contre-exemple des figures 1.5a, 1.5b et 1.5c



(d) Contre-exemple des figures 1.5a, 1.5b, 1.5c et 1.5d

FIGURE *1.6 – Exemples de motifs ne pouvant être obtenus à partir de sous-graphes connexes des graphes de la figure 1.5.*

Comme dans les sous-sections précédentes, nous nous concentrons ici sur les approches susceptibles d'être adaptées à notre problématique. Entre autres, les divers algorithmes développés afin d'extraire des motifs à partir de graphes comportant une structure particulière (*e.g.* : des arbres dans le cas de [132] et [100]) ne seront pas abordés.

Une des premières approches utilisées dans ce domaine est la méthode SUBDUE [74] qui se base sur le principe de la longueur de description minimale – notée MDL (*minimum description length*) – pour identifier les sous-structures permettant de compresser le plus efficacement possible une base de données structurelle. Étant donné un graphe orienté, SUBDUE recherche les meilleurs sous-graphes, au sens du principe MDL. Pour ce faire, l'algorithme commence par identifier les sommets fréquents. A chaque itération, l'algorithme sélectionne les meilleurs sous-structures parmi celles identifiées, puis les étend en ajoutant à chacune le sommet permettant de maximiser la longueur de description du sous-graphe correspondant. Lorsqu'aucune extension n'est plus possible, chaque sous-graphe est fusionné en un unique sommet et le processus est répété sur le graphe compressé obtenu.

Yoshida et Motoda ont élaboré la méthode GBI (*Graph Based Induction*) [129]. Tout comme SUBDUE, cet algorithme est glouton et fusionne progressivement des sommets du graphe d'origine. Une différence fondamentale réside dans le fait qu'à chaque itération, deux sommets adjacents y sont fusionnés. De plus, plusieurs fonctions d'évaluation de sous-structures basées sur la notion de fréquence peuvent y être utilisées (*i.e.* : plusieurs définitions du support d'une sous-structure).

Tout comme dans le cadre de la fouille d'itemsets et de la fouille de séquences d'itemsets, les techniques Apriori et à croissance de motifs ont été adaptées à l'exploration de graphes.

Algorithmes Apriori

De façon similaire aux algorithmes Apriori utilisés pour extraire des itemsets fréquents, la recherche de graphes fréquents débute ici par l'identification de graphes de taille faible avant de les étendre. La génération de candidats est ici effectuée en combinant des couples de graphes fréquents similaires, puis la fréquence de la nouvelle structure obtenue est vérifiée. La taille des graphes fréquents est ainsi augmentée jusqu'à ce qu'aucun graphe ne soit obtenu à une étape.

Dans la méthode AGM (*Apriori-based Graph Mining*) [63], la taille des sous-structures est augmentée d'un unique sommet à chaque étape et deux graphes fréquents de taille k (*i.e.* : contenant k sommets) ne seront combinés que s'ils partagent un sous-graphe de taille $k-1$ en commun. Dans l'exemple représenté en figure 1.7 – inspiré de [56] – les deux graphes de taille 5 ont en commun le sous-graphe encadré en pointillés qui est de taille 4. Ces derniers pourront donc être fusionnés ensemble. Comme le représente cette figure, la fusion de deux graphes de taille k peut mener à la génération de plusieurs candidats de taille $k+1$. Le second algorithme de ce type est FSG (*Frequent SubGraph discovery*) [79]. Contrairement à AGM, ce dernier utilise une stratégie de génération dans laquelle les candidats sont étendus en leur ajoutant uniquement une arête.

Une nouvelle fois, les algorithmes Apriori ont pour inconvénient la durée des phases de génération de candidats et d'élagage et les approches à base de croissance de motifs permettent, en partie, d'y pallier.

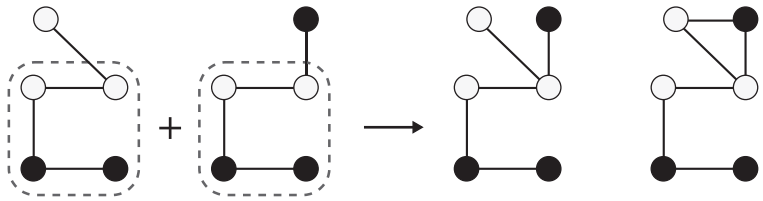


FIGURE *1.7 – Exemple de génération de candidats avec la méthode AGM (illustration inspirée de [56]). Les pointillés représentent deux sous-graphes de taille* 4 *identique dans deux graphes différents de taille* 5*. Leur fusion entraîne la création de deux graphes candidats de taille* 6*.*

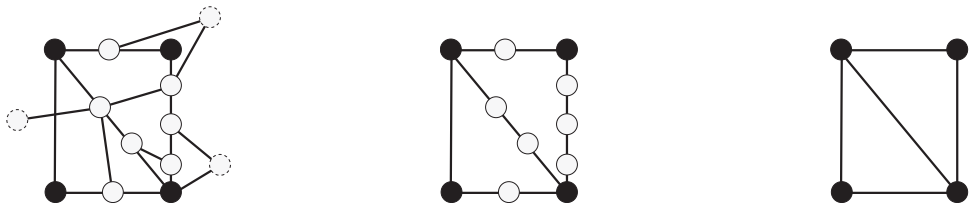
Croissance de motifs

Dans les algorithmes d'exploration de sous-structures de type croissance de motifs, les sous-graphes sont étendus d'une arête dans toutes les directions possibles. L'inconvénient de cette procédure est qu'un même graphe pourra être généré de nombreuses fois. Pour éviter cela, l'algorithme gSpan (*graph-based Substructure PAtterN mining*) [125] introduit la notion de *chemin le plus à droite*. Étant donnée une sous-structure fréquente G, son chemin le plus à droite est celui reliant le premier sommet ayant été utilisé pour créer G au dernier sommet qui y a été ajouté. La redondance lors de la génération des candidats est réduite en étendant les sous-structures exclusivement le long de ce chemin.

Wang *et al.* ont défini l'algorithme TSMiner (*Topological Structures Miner*) [68] qui se base sur la notion de chemins indépendants et qui permet d'extraire des motifs dans des structures de grandes tailles. Un chemin est défini comme une séquence de sommets v_1, v_2, ..., v_l d'un graphe $G = (V, E)$ telle que pour tout $k \in \{1, \ldots, l-1\}$, l'arête $(v_k v_{k+1})$ figure dans E. Les sommets v_2 à v_{l-1} sont appelés les *nœuds internes* de P tandis que les sommets v_1 et v_l en sont les *nœuds externes*. Un ensemble de chemins P est dit *indépendant* si tout sommet interne de $p \in P$ n'est inclus dans aucun chemin de $P \backslash \{p\}$. La notion de chemins indépendants va permettre de compresser la représentation des graphes sans pour autant compromettre les informations liées à leur topologie. Dans l'exemple représenté en figure 1.8, le graphe Y' qui est un sous-graphe de Y peut être compressé en ne conservant que ses sommets noirs qui correspondent aux nœuds externes d'un ensemble de chemins indépendants. De plus, cet algorithme définit la notion de *chaîne floue* permettant d'extraire de manière approchée des composés chimiques aux propriétés équivalentes. Ce mécanisme est cependant trop spécifique pour permettre une prise en compte de la variabilité (V_{car}) dans des graphes complets.

Nous pouvons enfin mentionner l'heuristique GREW développée par Kuramochi et Karypis [80] dont l'objectif est de permettre l'extraction de motifs fréquents dans des graphes de taille conséquente et qui ne sont pas nécessairement creux (contrairement aux méthodes présentées précédemment). Tout comme les heuristiques SUBDUE et GBI, GREW va fusionner successivement des sommets du graphe. L'originalité de cette approche réside dans le fait qu'à chaque itération, les sous-structures fréquentes pourront être étendues en leur ajoutant plusieurs sommets. Le caractère heuristique de l'approche entraîne l'identification d'un nombre plus réduit de motifs. Pour pallier cela, l'exécution de la méthode est répétée en modifiant l'ordre dans lequel les sommets sont considérés.

Cette section nous a permis d'obtenir un bon aperçu des techniques de fouille de motifs fréquents utilisées et de la façon dont ces dernières pourraient être adaptées à notre



(a) Graphe Y. *(b) Sous-graphe Y' de Y.* *(c) Compression du graphe Y'.*

FIGURE *1.8 – Exemple de compression de graphe effectuée par TSMiner (inspiré de [68]). Les sommets noirs représentent les sommets externes d'un ensemble de chemins indépendants tandis que les sommets blancs à bord plein en sont les sommets internes. Les trois sommets de Y dont les contours sont en pointillés ne font pas partie de l'ensemble de chemins indépendants.*

problématique. Nous allons maintenant nous intéresser aux techniques d'exploration de séquences avant de comparer les différentes approches en section 1.3 et de déterminer la plus pertinente.

1.2 Fouille de séquences de caractères

L'exploration de séquences de caractères est un domaine de recherche majeur de la bioinformatique qui comporte des techniques permettant l'étude de séquences biologiques telles que l'ADN, l'ARN et les protéines.

Afin de permettre l'application des méthodes de ce domaine à l'extraction de motifs récurrents d'un corpus de tableaux d'annotations, nos données doivent être représentées sous une forme adéquate. Pour ce faire, les deux modélisations suivantes, illustrées en figure 1.9 sont alors envisageables :

(M_1) Associer à chaque ligne d'un tableau d'annotations un caractère correspondant à la combinaison des annotations de cette ligne. Dans l'exemple représenté en figure 1.9, le caractère α est associé aux annotations A_1, B_1 et C_1. Les méthodes de fouille de séquences classiques peuvent ensuite être appliquées directement sur la séquence obtenue (*e.g.* : sur $\alpha\beta\alpha\gamma$ pour notre exemple).

(M_2) Utiliser des tableaux de caractères en associant simplement à chaque annotation le caractère qu'elle contient. L'inconvénient est que les méthodes traditionnelles de fouille de séquences ne peuvent pas être utilisées sur des tableaux. Cependant, certaines d'entre elles ont été généralisées dans cette optique [83, 76, 8, 12].

Ces deux représentations montrent que des données représentées sous la forme de tableaux d'annotations peuvent être utilisées dans le cadre de méthodes de fouille de séquences. Il reste à déterminer si certaines de ces approches sont adaptées pour l'extraction de motifs récurrents et la prise en compte des variabilités (V_{car}) et (V_{ag}) mentionnées en début de chapitre.

Dans la suite de ce chapitre ainsi que dans le suivant, une séquence de caractères s de taille m est représentée par un tableau unidimensionnel de caractères dont le premier indice est égal à 1 (noté $s[1..m]$). Étant donnés deux indices i et j tels que $1 \leq i < j \leq m$, le terme $s[i..j]$ désigne la sous-séquence de caractères de s allant des indices i à j compris.

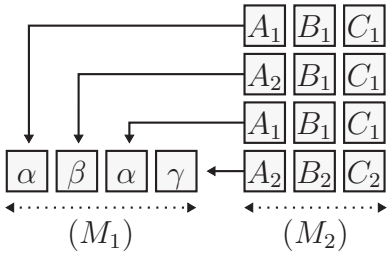


FIGURE *1.9 – Modélisations* (M_1) *et* (M_2) *envisagées afin de permettre d'appliquer les algorithmes du domaine de la fouille de séquences à des tableaux d'annotations. Dans* (M_1)*, chaque tableau d'annotations est modélisé par une séquence dont chaque caractère correspond à une ligne. La modélisation* (M_2) *associe à chaque annotation son caractère, ce qui produit un tableau de caractères.*

Un tableau de caractères t de taille $m \times n$ est de façon similaire noté $t[1..m][1..n]$ (*e.g.* : figure 1.10a). Le terme $t[i][j..k]$ pour tout $i \in \{1, \ldots, m\}$ et tout indices j et k vérifiant $1 \leq j < k \leq n$ est utilisé pour désigner la séquence contenant les caractères figurant en ligne i du tableau t et entre les colonnes j à k comprises (*e.g.* : figure 1.10b). De même, la séquence formée par les caractères en colonne i et entre les lignes j et k du tableau t est notée $t[j..k][i]$ (*e.g.* : figure 1.10c).

	1	2	3
1	C	A	G
2	A	C	T
3	G	T	A
4	A	T	C

(a) $t[1..4][1..3]$

	1	2	3
4	A	T	C

(b) $t[4][1..3]$

	1
1	C
2	A
3	G

(c) $t[1..3][1]$

FIGURE *1.10 – Exemple de tableau de caractères et de deux séquences qu'il contient.*

Les méthodes de fouille de séquences de caractères ont pour objectif d'identifier des zones similaires dans une ou plusieurs séquences. Il existe diverses façons de définir cette similarité. Celle qui est le plus communément utilisée pour l'exploration de séquences est dérivée de la distance d'édition. Étant données deux séquences de caractères $s_A[1..m_A]$ et $s_B[1..m_B]$, la *distance d'édition* entre s_A et s_B correspond au coût minimal des opérations d'édition permettant de transformer s_A en s_B. Les trois opérations d'édition autorisées sont :

- la *substitution* d'un caractère par un autre (*e.g.* : ATG peut être transformée en ACG en substituant le caractère T en seconde position par le caractère C, ce qui peut être représenté par $ATG \xrightarrow{sub(T,C,2)} ACG$) ;
- l'*insertion* d'un caractère (*e.g.* : ATG peut être transformée en $ATGA$ en insérant le caractère A en quatrième position, ce qui peut s'illustrer $ATG \xrightarrow{ins(A,4)} ATGA$) ;
- la *suppression* d'un caractère (*e.g.* : ATG peut être transformée en TG en supprimant le caractère A figurant en première position, ce qui peut se représenter par $ATG \xrightarrow{sup(A,1)} TG$).

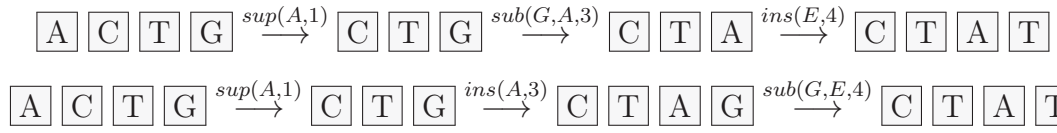
A chaque opération d'édition est associée un coût dans $\mathbb{R}^+$. Nous parlons de *distance de Levenshtein* [84] dans le cas où le coût des opérations d'édition est égal à un, excepté

les substitutions d'un caractère par lui-même dont le coût est nul. Par exemple, la distance de Levenshtein entre les séquences "*ACTG*" et "*CTAT*" est égale à 3. Ceci est illustré par la figure 1.11a qui représente deux séries de trois opérations d'édition permettant de transformer "*ACTG*" en "*CTAT*".

Lorsqu'une distance d'édition entre deux séquences de caractères s_A et s_B est calculée, les opérations d'édition permettant de transformer s_A en s_B peuvent être représentées en *alignant* ces deux séquences (voir exemple figure 1.11b), de telle sorte que :

- les caractères substitués sont alignés ensemble ;
- les caractères non impactés par les opérations d'édition sont alignés avec leur équivalent dans l'autre chaîne (*e.g.* : les caractères 'C' et 'T' de la figure 1.11b) ;
- les caractères insérés dans s_A ou supprimés dans s_B sont alignés avec le caractère '*'.

La recherche de séquences similaires peut donc être vue comme la recherche de la meilleure façon d'aligner ces séquences en question. C'est la raison pour laquelle nous parlons aussi de méthodes d'alignement de séquences.



(a) Représentation de deux séries de trois opérations d'édition permettant de transformer "ACTG" en "CTAT".

$$\begin{pmatrix} A & C & T & G & * \\ * & C & T & A & T \end{pmatrix} \begin{pmatrix} A & C & T & * & G \\ * & C & T & A & T \end{pmatrix} \qquad \begin{pmatrix} A & C & T & G \\ * & C & T & A \end{pmatrix} \begin{pmatrix} T & * & G \\ T & A & T \end{pmatrix}$$

(b) Alignements globaux correspondant aux séries d'opérations représentées en (a).

(c) Alignements locaux.

FIGURE *1.11 – Opérations d'édition et alignements relatifs aux séquences "ACTG" et "CTAT".*

Comme le remarquent Abouelhoda et Ghanem [1], les méthodes de fouille de séquences peuvent se décliner en deux catégories :

- (T_1) celles qui identifient des similarités à l'intérieur d'une unique séquence ;
- (T_2) celles qui s'intéressent à la similarité entre des séquences différentes.

Les méthodes de type (T_1) ne considèrent qu'une seule séquence s_A dans laquelle ils cherchent des sous-séquences similaires. Leur intérêt, dans le contexte de notre problématique, est donc limité puisqu'elles ne nous permettraient d'obtenir que des motifs apparaissant deux fois dans un seul et même tableau.

Les algorithmes de type (T_2) semblent donc plus appropriés et peuvent eux-même être partagés en trois catégories :

- ($T_{2,global}$) : les méthodes *globales* qui alignent l'ensemble des caractères de deux séquences s_A et s_B. Les alignements qu'elles permettent d'obtenir sont dit *globaux*. Deux exemples d'alignements globaux des séquences "ACTG" et "CTAT" sont représentés en figure 1.11b ;
- ($T_{2,semi-global}$) : les méthodes *semi-globales* qui alignent l'ensemble des caractères d'une séquence s_A avec des sous-parties d'une séquence s_B (*i.e.* : qui identifient les occurrences, exactes ou approchées, de s_A dans s_B) ;

– ($T_{2,local}$) : les méthodes *locales* qui alignent des sous-parties d'une séquence s_A avec des sous-parties d'une séquence s_B. Les alignements obtenus sont alors dit *locaux*. La figure 1.11c représente deux alignements locaux des séquences "ACTG" et "CTAT".

Dans notre contexte, les approches ($T_{2,global}$) ne peuvent convenir puisqu'elles ne permettraient que de calculer la similarité entre l'ensemble des annotations de deux tableaux d'annotations et pas d'en extraire des motifs récurrents.

Les méthodes ($T_{2,semi-global}$), sont plus intéressantes puisqu'elles pourraient permettre, une fois un motif m_1 identifié, de déterminer sa fréquence dans le corpus. Deux des approches de ce type les plus connues sont les heuristiques de recherche rapide BLAST [6] et FASTA [86]. Cependant, dans les méthodes ($T_{2,semi-global}$) c'est à l'utilisateur de préciser le motif qu'il souhaite rechercher dans le corpus. Elles ne fournissent pas d'outils permettant d'identifier en premier lieu ces motifs.

Enfin, les approches ($T_{2,local}$) permettent d'identifier des alignements locaux (*i.e.* : des sous-parties similaires de séquences de caractères). Ceci semble correspondre à notre problématique puisqu'un motif est une sous-partie d'un tableau d'annotations. De plus, notre objectif est d'obtenir des motifs récurrents (*i.e.* : des motifs apparaissant au moins deux fois, de manière exacte ou approchée, dans un corpus de tableaux d'annotations). Puisqu'un alignement local correspond à deux sous-parties similaires, il nous fournira donc deux occurrences d'un seul et même motif récurrent.

Dans la suite, nous présentons en sous-section 1.2.1 l'algorithme de programmation dynamique ($T_{2,global}$) de Needleman-Wunsch qui, bien que n'étant pas utilisable directement sur nos données, sert de base aux deux principales approches ($T_{2,local}$) présentées par la suite. Puis, la sous-section 1.2.2 décrit des approches prenant en entrée des séquences de caractères, en vue de les appliquer sur des tableaux d'annotations transformés selon la représentation (M_1). Un intérêt particulier est porté à l'algorithme de Smith-Waterman [114]. Enfin, en sous-section 1.2.3, nous envisageons l'utilisation de la représentation (M_2) et décrivons des approches considérant en entrée des tableaux de caractères (notamment l'algorithme LPCA [83]).

1.2.1 Algorithme de Needleman-Wunsch

L'algorithme de programmation dynamique dit de Needleman-Wunsch [98] est une approche classique permettant le calcul de la distance d'édition (*i.e.* : de l'alignement global) entre deux séquences de caractères $s_A[1..m_A]$ et $s_B[1..m_B]$. Il est ici décrit afin de permettre une meilleure compréhension des algorithmes présentés en sous-sections 1.2.2 et 1.2.3 qui se reposent sur des mécanismes similaires.

Dans la suite de cette section, pour deux caractères quelconques 'a' et 'b', le coût de substitution de 'a' par 'b' ainsi que les coûts de suppression et d'insertion de 'a' sont respectivement représentés par $sub(a, b)$, $sup(a)$ et $ins(a)$.

L'objectif de cet algorithme est de créer une table $T[0..m_A][0..m_B]$ telle que pour tout $(i, j) \in \{1, \ldots, m_A\} \times \{1, \ldots, m_B\}$, $T[i][j]$ soit égal à la distance d'édition entre $s_A[1..i]$ et $s_B[1..j]$ (voir exemple figure 1.12). Nous précisons que contrairement aux autres tableaux, la table T possède des indices commençant à 0 afin de simplifier les

notations et la compréhension. De part cette construction, la distance d'édition entre s_A et s_B est égale à $T[m_A][m_B]$.

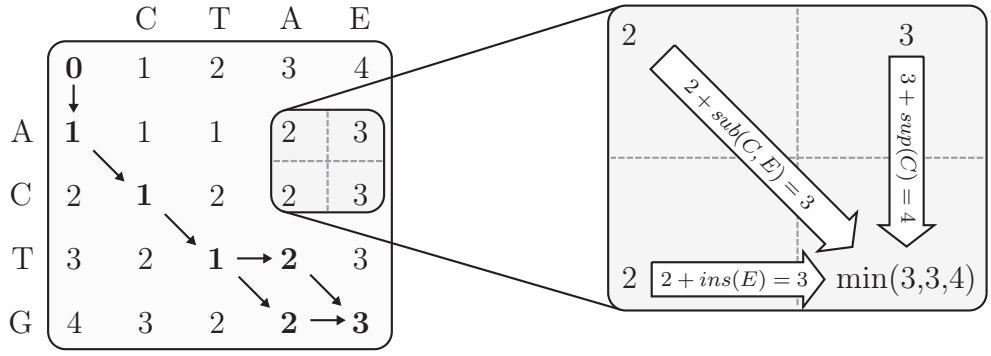


FIGURE *1.12 – Table T de l'algorithme de Needleman-Wunsch obtenue lors de l'alignement des séquences "ACTG" et "CTAE". Les arcs représentent les chemins suivis par l'algorithme fournissant les deux alignements globaux de coût minimal représentés en figure 1.11b. La formule de récurrence de l'algorithme consistant en la minimisation de trois termes est illustrée par un zoom sur une sous-partie du tableau.*

La première ligne et la première colonne de T sont initialisées de la manière suivante :

- $T[0][0]$ est égal à 0 (*i.e.* : si un alignement commence au début des séquences s_A et s_B, il n'est pas pénalisé) ;
- pour tout $i \in \{1, \ldots, m_A\}$, $T[i][0]$ est égal à $\sum_{k=1}^{i} sup(s_A[k])$ (*i.e.* : si un alignement débute en $s_A[i+1]$, les caractères $s_A[1]$ à $s_A[i]$ doivent être supprimés et l'alignement doit être pénalisé en conséquence) ;
- pour tout $j \in \{1, \ldots, m_B\}$, $T[0][j]$ est égal à $\sum_{k=1}^{j} ins(s_B[k])$ (*i.e.* : si un alignement débute en $s_B[j+1]$, les caractères $s_B[1]$ à $s_B[j]$ doivent être insérés).

Afin d'aligner les chaînes de caractères $s_A[1..i]$ et $s_B[1..j]$ pour tout $(i, j) \in \{1, \ldots, m_A\} \times \{1, \ldots, m_B\}$, une succession d'opérations d'édition est effectuée. La dernière de ces opérations peut être soit la substitution de $s_A[i]$ par $s_B[j]$, soit l'insertion de $s_B[j]$, soit la suppression de $s_A[i]$. Cette constatation mène naturellement à la définition d'un algorithme de programmation dynamique permettant de calculer la table T. En effet, à chaque position (i, j) de la table T, les trois opérations d'édition mentionnées précédemment sont considérées et celle entraînant un alignement de coût minimal est choisie. Par exemple, la distance d'édition entre les séquences "ACTG" et "CTAE" – notée $ed(ACTG, CTAE)$ – est égale à

$$min \begin{cases} ed(ACT, CTA) & + & sub(G, E) \\ ed(ACT, CTAE) & + & sup(G) \\ ed(ACTG, CTA) & + & ins(E) \end{cases},$$

ce qui donne, dans le cas général, la formule de récurrence

$$T[i][j] = min \begin{cases} T[i-1][j-1] & + & sub(s_A[i], s_B[j]) \\ T[i-1][j] & + & sup(s_A[i]) \\ T[i][j-1] & + & ins(s_B[j]) \end{cases} \quad (1.1)$$

pour tout $(i, j) \in \{1, \ldots, m_A\} \times \{1, \ldots, m_B\}$. Cette formule de récurrence est illustrée en figure 1.12 pour l'obtention de la valeur $T[2][4]$ de l'exemple correspondant.

La table T est donc complétée en utilisant (1.1) et $T[m_A + 1][m_B + 1]$ fournit la distance d'édition. L'alignement global correspondant est obtenu en réalisant une phase de tracé-arrière, très similaire à celle permettant de calculer T, qui permet d'inférer le chemin suivi par l'algorithme entre les positions $T[m_A][m_B]$ à $T[0][0]$ (*e.g.* : voir arcs de la figure 1.12).

1.2.2 Alignement local de séquences de caractères

La représentation (M_1) décrite page 38 associe à chaque ligne d'un tableau d'annotations un caractère. Ainsi, chaque tableau d'annotations est transformé en une séquence dans laquelle chaque caractère correspond à la concaténation des annotations d'une ligne (*e.g.* : voir exemple en figure 1.9). Il est ainsi possible d'appliquer les algorithmes d'alignement local de séquences de caractères afin d'identifier des motifs récurrents de notre corpus.

De nombreuses approches permettent l'alignement de sous-parties identiques de séquences de caractères comme le présente l'étude de Faro et Lecroq [37]. Cependant, leur utilité dans notre contexte est limitée puisque les caractères alignés ensemble sont nécessairement les mêmes et que les insertions et les suppressions ne sont pas envisagées. Ainsi, dans ces algorithmes les variabilités (V_{car}) et (V_{ag}) ne sont pas prises en compte.

Dans le cadre de la recherche d'alignements locaux approchés, l'algorithme de référence est celui de *Smith-Waterman* [114]. C'est la raison pour laquelle la suite de cette sous-section est dédiée à sa description. Son objectif est d'extraire l'alignement local de score maximal entre s_A et s_B. Les mécanismes qu'il utilise sont très similaires à ceux de l'algorithme de Needleman-Wunsch.

Dans ce dernier nous calculons la distance d'édition entre l'ensemble des chaînes s_A et s_B, c'est pourquoi la valeur de départ $T[0][0]$, qui est égale à 0, ne peut qu'augmenter en approchant de $T[m_A][m_B]$. Dans l'algorithme de Smith-Waterman, une table T aux dimensions similaires est calculée mais nous souhaitons au contraire que cette dernière fasse ressortir les zones similaires en leur associant un score élevé et que les zones peu similaires aient un score qui tende vers 0.

Pour ce faire, le score des opérations d'édition indiquant une similarité entre les deux chaînes (*i.e.* : les substitutions de caractères similaires) auront un score positif afin d'augmenter les valeurs de T lorsqu'elles sont utilisées. Par conséquent, les autres opérations d'édition *i.e.* : suppressions, insertions et substitutions de caractères non similaires), qui traduisent des différences entre s_A et s_B auront un score négatif afin d'entraîner vers 0 les valeurs de T dans ces zones.

Ainsi dans cet algorithme, la valeur de $T[i][j]$ est égale au score de l'alignement local se terminant en $s_A[i]$ et $s_B[j]$ – et non plus à la distance d'édition entre les séquences $s_A[1..i]$ et $s_B[1..j]$. La première ligne et la première colonne de T sont initialisées à 0 (*i.e.* : $\forall i \in \{0, \ldots, m_A\}\ T[i][0] = 0$ et $\forall j \in \{1, \ldots, m_B\}\ T[0][j] = 0$) car un alignement peut maintenant débuter sans pénalité à partir de n'importe quelle position de s_A et de s_B.

L'objectif étant de maximiser le score de l'alignement, la formule de récurrence de-

vient

$$T[i][j] = max \begin{cases} T[i-1][j-1] & + & sub(s_A[i], s_B[j]) \\ T[i-1][j] & + & sup(s_A[i]) \\ T[i][j-1] & + & ins(s_B[j]) \\ 0 & & \end{cases} \quad (1.2)$$

où *sub*, *sup* et *ins* désignent maintenant le score des opérations d'édition. La dernière ligne de ce système permet d'assurer que les valeurs de T ne deviennent jamais négatives. Si les trois première lignes du système (qui correspondent chacune à une opération d'édition) fournissent un score négatif, cela signifie qu'il n'existe aucun alignement local de score positif se terminant en $s_A[i]$ et $s_B[j]$. Dans ce cas, $T[i][j]$ est fixé à 0 afin de permettre qu'un nouvel alignement débute sans pénalité à partir de cette position.

En guise d'exemple, la figure 1.13 représente la table T obtenue pour les séquences "ACTG" et "CTAE" dans le cas où un score de 1 est attribué aux substitutions d'un caractère par lui-même et que le score des autres opérations d'édition est fixé à -1.

		C	T	A	E
	0	0	0	0	0
A	**0**	0	0	1	0
C	0	**1**	0	0	0
T	0	0	**2**	1	0
G	0	0	1	1	0

FIGURE *1.13 – Table T de l'algorithme de Smith-Waterman obtenue lors de l'alignement des séquences "ACTG" et "CTAE". Les arcs représentent le chemin suivi par l'algorithme fournissant l'alignement local de score maximal.*

Finalement, l'alignement local de score maximal est obtenu en effectuant un tracé-arrière à partir de la position de score maximal et en s'arrêtant lorsqu'une case de valeur 0 est atteinte. Dans le cas de l'exemple de la figure 1.13, le score du meilleur alignement local est égal à 2. Il correspond à un alignement contenant les caractères 'C' et 'T' des deux séquences.

1.2.3 Alignement local de tableaux de caractères

Nous considérons maintenant la représentation (M_2) dans laquelle un tableau d'annotations correspond à un tableau de caractères. Étant donnés deux tableaux de caractères $t_A[1..m_A][1..,n_A]$ et $t_B[1..m_B][1..n_B]$, nous souhaitons en extraire l'alignement local de score le plus élevé. Les méthodes classiques d'exploration de séquences ne sont pas directement applicables à des tableaux. Un certain nombre de travaux ont été réalisés en ce sens.

Les premières méthodes permettant l'exploration de tableaux de caractères sont de type ($T_{2,semi-global}$) et permettent donc uniquement de détecter les occurrences approchées d'un motif dans un tableau de plus grande taille. Krithivasan et Sitalakshmi [76] présentent une méthode permettant d'identifier des motifs rectangulaires comportant au

plus k erreurs (*i.e.* : insertions, suppressions et substitutions de caractères différents). L'approche proposée par la suite par Amir et Farach [8] permet de s'affranchir de la contrainte imposant la forme rectangulaire des motifs en introduisant la notion de caractère *don't care* capable s'aligner sans pénalité avec n'importe quel caractère. Ces caractères correspondent aux caractères '*' que nous utilisons dans la représentation de nos motifs. Enfin, plus récemment, Baeza et Yates [12] améliorent la complexité de ces méthodes en proposant une phase de filtrage réduisant significativement l'espace de recherche.

Ces trois méthodes sont, cependant, relativement restrictives sur les motifs qu'elles permettent d'obtenir puisqu'un motif t_A et un texte t_B sont comparés ligne par ligne. Les décalages engendrés par les insertions et les suppressions ne sont donc pris en compte que le long des lignes et pas le long des colonnes. Nous pouvons illustrer cet aspect par les deux motifs représentés en figure 1.14. Dans le motif m_A, les caractères 'd' et 'e' de la première ligne sont adjacentes tandis qu'un caractère les sépare dans m_B. Ce décalage peut être identifié par les méthodes présentées précédemment puisqu'elles apparaissent le long d'une ligne. Considérons maintenant les caractères 'c' et 'f' qui se trouvent dans une configuration similaire au niveau de la première colonne. Ce type de décalage ne peut pas être détecté puisqu'il intervient le long d'une colonne et que la distance d'édition entre les colonnes n'est pas prise en compte. La forme des motifs extraits dans ces approches est donc fortement contrainte. De plus, ces méthodes ne permettent pas d'extraire des motifs mais fournissent simplement la possibilité de chercher un motif connu dans des données.



(a) Motif m_A. *(b) Motif m_B.*

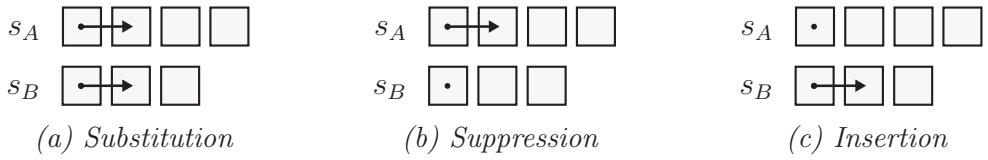
FIGURE *1.14 – Deux motifs m_A et m_B permettant d'illustrer une limitation des approches présentées de type ($T_{2,semi-global}$) explorant des tableaux d'annotations.*

Nous pouvons aussi mentionner une méthode plus récente permettant d'identifier les sous-parties d'un tableau de caractères similaires (*i.e.* : méthode de type T_1). Cette dernière a été développée par Apostolico *et al.* en 2008 [10]. Cette approche utilise elle aussi le caractère "don't care" afin de permettre l'extraction de motifs non rectangulaires. Cette dernière étant de type (T_1), elle ne peut cependant convenir à nos besoins.

A notre connaissance, il n'existe qu'une unique approche ($T_{2,local}$) permettant l'extraction d'alignements locaux dans des tableaux de caractères. Cette dernière a été développée par Lecroq *et al.* [83] et généralise l'algorithme de Smith-Waterman précédemment présenté à l'alignement de tableaux de caractères. Cette méthode se décline en deux algorithmes selon que des alignements locaux ou globaux sont recherchés. Étant donné notre problématique, nous ne présentons ici que sa version locale que nous appelons LPCA (d'après le nom des auteurs de l'article [83] correspondant : Lecroq, Pauchet, Chanoni, Ayala Solano). La suite de cette sous-section est dédiée à sa présentation.

Comme le représente la figure 1.15, dans l'algorithme de Smith-Waterman les séquences de caractères sont parcourues de gauche à droite (*i.e.* : de $s_A[1]$ et $s_B[1]$ à $s_A[m_A]$ à $s_B[m_B]$) en autorisant trois types de déplacements, chacun correspondant à

une opération d'édition. De manière similaire, LPCA considère les huit opérations d'édition représentées en figure 1.16.



(a) Substitution *(b) Suppression* *(c) Insertion*

FIGURE *1.15 – Représentation des opérations d'édition considérées lors de l'alignement de séquences de caractères dans l'algorithme de Smith-Waterman [114].*

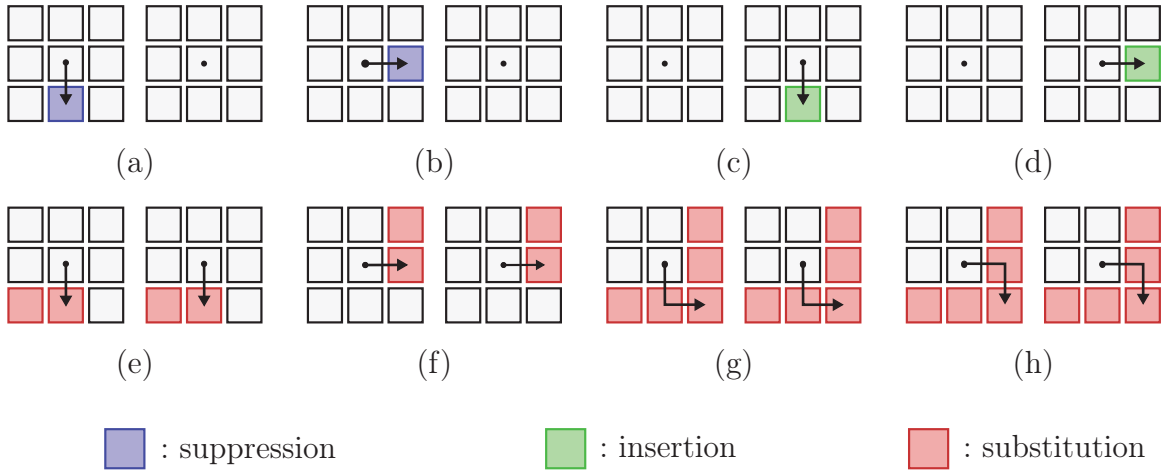


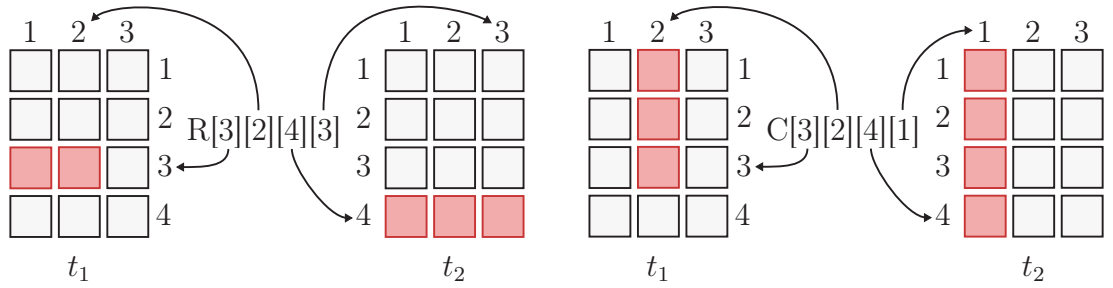
FIGURE *1.16 – Opérations d'édition considérées dans l'algorithme LPCA [83]. (a) Suppression horizontale. (b) Suppression verticale. (c) Insertion horizontale. (d) Insertion verticale. (e) Substitution de ligne. (f) Substitution de colonne. (g) et (h) Substitution de ligne et de colonne (dans un ordre différent).*

Lorsque les données sont unidimensionnelles, la table T construite par les algorithmes de programmation dynamique est de dimension 2. Puisque les données considérées sont ici des tableaux bidimensionnels, la table T est de dimension 4. Ainsi, l'algorithme crée une table $T[0..m_A][0..n_A][0..m_B][0..n_B]$ telle que pour tout $(i, j, k, l) \in \{1, ..., m_A\} \times \{1, ..., n_A\} \times \{1, ..., m_B\} \times \{1, ..., n_B\}$, $T[i][j][k][l]$ corresponde au score de l'alignement local se terminant en $t_A[i][j]$ et en $t_B[k][l]$.

Afin de faciliter le calcul des valeurs de T, deux tables R et C de dimensions $m_A \times n_A \times m_B \times n_B$, sont préalablement calculées. Pour toute position (i, j) de t_A et toute position (k, l) de t_B, $R[i][j][k][l]$ est égal au score de l'alignement local entre les séquences $t_A[i][1..j]$ et $t_B[k][1..l]$ se terminant en $t_A[i][j]$ et $t_B[k][l]$. La figure 1.17a illustre le lien entre les dimensions de R et les positions de t_A et t_B. La valeur de la distance d'édition locale entre les deux séquences représentées en rouge dans cette figure est fournie par la valeur de $R[3][2][4][3]$. Ainsi, lorsqu'est effectuée une opération d'édition bidimensionnelle entraînant une substitution de ligne (*i.e.* : opérations d'édition (e), (g) ou (h) de la figure 1.16), le score de cette substitution peut directement être lue dans la table R.

De manière similaire, $C[i][j][k][l]$ est égal au score de l'alignement local entre les séquences $t_A[1..i][j]$ et $t_B[1..k][l]$ se terminant en $t_A[i][j]$ et $t_B[k][l]$ (voir exemple figure 1.17b). Cette table est utilisée lorsque les opérations d'édition (f), (g) et (h) de la

figure 1.16, qui entraînent une substitution de colonnes, sont considérées. La valeur de cette substitution est directement fournie par la table C.



(a) Séquences considérées lors du calcul de $R[3][2][4][3]$.

(b) Séquences considérées lors du calcul de $C[3][2][4][1]$.

FIGURE *1.17 – Exemples de séquences d'annotations considérées lors du calcul des tables R et C. Les arcs représentent le lien entre les dimensions des tables R ou C et les positions dans t_A et t_B.*

Les tables R et C sont calculées en utilisant l'algorithme de Smith-Waterman. Si en position (i, j, k, l) de T, le score d'une opération d'édition bidimensionnelle faisant intervenir une substitution (*i.e.* : opérations (e), (f), (g) ou (h) de la figure 1.16) est nul, cela signifie que le meilleur alignement local consiste à ne prendre aucune annotation. Dans ce cas, il est nécessaire – pour que l'alignement se prolonge en $t_A[i][j]$ et en $t_B[k][l]$ – de supprimer le caractère $t_A[i][j]$ et d'insérer le caractère $t_B[k][l]$. C'est pourquoi, le cas échéant, le score de l'alignement est pénalisé par $sup(t_A[i][j]) + ins(t_B[k][l])$. Afin de simplifier les notations, étant donné un réel positif x et une position (i, j, k, l) de T, nous définissons la fonction suivante :

$$q_{i,j,k,l}(x) = \begin{cases} x & \text{si } x \neq 0 \\ sup(t_A[i][j]) + ins(t_B[k][l]) & \text{sinon} \end{cases}.$$

Tout comme pour l'algorithme de Smith-Waterman, les bornes de T sont initialisées à 0 afin qu'un alignement local puisse débuter à n'importe quelle position de t_A et t_B (*i.e.* : pour tout $(i, j, k, l) \in \{0, \ldots, m_A\} \times \{0, \ldots, n_A\} \times \{0, \ldots, m_B\} \times \{0, \ldots, n_B\}$, $T[0][j][k][l]$, $T[i][0][k][l]$, $T[i][j][0][l]$ et $T[i][j][k][0]$ sont égaux à 0). Ensuite, pour toute autre position (i, j, k, l) de T, la valeur $T[i][j][k][l]$ est obtenue à l'aide de l'expression suivante :

$$\max \begin{cases} T[i-1,j,k,l] + sup(t_A[i][j]) & (a) \\ T[i,j-1,k,l] + sup(t_A[i][j]) & (b) \\ T[i,j,k-1,l] + ins(t_B[k][l]) & (c) \\ T[i,j,k,l-1] + ins(t_B[k][l]) & (d) \\ T[i-1,j,k-1,l] + q_{i,j,k,l}(R[i][j][k][l]) & (e) \\ T[i,j-1,k,l-1] + q_{i,j,k,l}(C[i][j][k][l]) & (f) \\ T[i-1,j-1,k-1,l-1] + q_{i,j,k,l}(R[i-1][j][k-1][l] + C[i][j][k][l]) & (g) \\ T[i-1,j-1,k-1,l-1] + q_{i,j,k,l}(R[i][j][k][l] + C[i][j-1][k][l-1]) & (h) \\ 0 \end{cases}$$

Chacune des huit premières lignes de ce système correspond à une opération d'édition. Les lettres figurant dans la colonne de droite permettent d'identifier l'opération d'édition

correspondante selon les notations utilisées dans la figure 1.16. Tout comme dans l'algorithme de Smith-Waterman, la dernière ligne du système permet de débuter un nouvel alignement à partir de la position (i, j, k, l) dans le cas où les huit opérations d'édition fournissent un score négatif. La construction de la table T détermine la complexité de l'algorithme. Comme cela consiste à réaliser une maximisation sur chaque case de T, sa complexité est donc $\mathcal{O}(m_A \times n_A \times m_B \times n_B)$.

Tout comme pour l'algorithme de Smith-Waterman, le meilleur alignement local entre t_A et t_B est ensuite obtenu en effectuant un tracé-arrière à partir de la position de valeur maximale de T.

1.3 Discussion

Dans cet état de l'art, nous avons tout d'abord défini deux niveaux auxquels l'approximation entre deux occurrences d'un même motif pouvait intervenir : les caractères (V_{car}) et l'agencement (V_{ag}). Nous avons ensuite identifié diverses méthodes permettant d'extraire des motifs apparaissant plusieurs fois, de manière exacte ou approchée, dans un corpus de tableaux d'annotations $\mathcal{C}$. Pour ce faire, quatre représentations différentes d'un tableau d'annotations ont été définies, comme le montre la table 1.6. Ces dernières permettent l'utilisation des méthodes du domaine de la fouille de motifs et de la fouille de séquences de caractères afin d'extraire des motifs récurrents du corpus $\mathcal{C}$.

Diverses approches ont été présentées lorsque les représentations sous forme de séquence d'itemsets et de graphes complets sont considérées, tandis que dans le cadre de la fouille de séquences, deux méthodes ont été proposées : l'algorithme de Smith-Waterman dans le cas où un tableau d'annotations est représenté sous la forme d'une séquence de caractères et l'algorithme LPCA lorsqu'un tableau de caractères est considéré.

Dans cette sous-section, nous comparons les forces et les faiblesses de ces approches du point de vue de la prise en compte des variabilités (V_{car}) et (V_{ag}) ainsi qu'en terme de performances globales.

Domaine	**Représentation**	(V_{car})	(V_{ag})
Fouille de motifs	(M_3) Séquence d'itemsets	partiellement	partiellement
	(M_4) Graphe complet	non	oui
Fouille de séquences	(M_1) Séquence de caractères	oui	partiellement
	(M_2) Tableau de caractères	oui	oui

TABLE *1.6 – Liste des différentes approches envisagées pour extraire des motifs récurrents dans des tableaux d'annotations. Pour chaque approche, sa capacité à prendre en compte les deux types de variabilités souhaitées est précisée.*

Critère (V_{car})

Le critère (V_{car}) a pour but de permettre de la variabilité au niveau des caractères composant un motif (*e.g.* : faire correspondre le caractère B au caractère A dans deux occurrences d'un motif).

Nous avons vu que certaines méthodes de fouille de séquences d'itemsets, notamment GSP, permettaient une certaine prise en compte de cet aspect en autorisant l'utilisation de taxonomies (*i.e.* : hiérarchies de concepts). Ainsi, deux motifs proches dans la hiérarchie pourront être considérés identiques. Cette approche est cependant restrictive puisqu'elle ne permet pas de limiter le nombre d'approximations effectuées ou de favoriser certains rapprochement plutôt que d'autres (*e.g.* : prendre en compte le fait que le caractère A est légèrement similaire à C, tandis qu'il est très similaire à B).

Dans le cadre de la modélisation (M_4), très peu de méthodes permettent d'effectuer des approximations et lorsque cela est le cas, comme par exemple pour TSMiner, les mécanismes mis en œuvre sont très spécifiques aux types de graphes considérés qui sont généralement creux. Ce type d'approche ne peut donc pas raisonnablement être appliqué à des graphes complets.

En revanche, les méthodes de fouilles de séquences permettent une prise en compte fine de la similarité entre deux annotations. En effet, les scores de substitutions définis par l'utilisateur correspondent directement à la proximité associée aux différentes annotations. Ceci offre une grande flexibilité à l'utilisateur et lui permet aisément de paramétrer l'extraction en choisissant le poids qu'il accorde à chaque couple d'annotations. Par exemple, si les annotations d'une colonne ont plus d'importance aux yeux de l'utilisateur, ce dernier peut en tenir compte en augmentant le score des opérations de substitution des couples d'annotations appartenant à cette colonne. Ainsi, les alignements de score les plus élevés tendront à posséder des annotations de cette colonne.

Du point de vue de ce premier critère, les méthodes de fouille de séquences sont donc nettement plus prometteuses.

Critère (V_{ag})

Le deuxième critère (V_{ag}) a pour but d'autoriser des décalages temporels des annotations dans deux occurrences d'un même motif (*e.g.* : voir le motif m de la figure 2d en page 13 qui apparaît de manière approchée dans les tableaux t_B et t_C des figures 2c et 2d).

Nous avons vu que les méthodes de fouille de séquences d'itemsets permettaient de prendre en compte diverses contraintes temporelles telles que la durée maximale entre deux itemsets consécutifs d'un même motif ou la durée maximale sur laquelle peut se dérouler un motif. Ces mécanismes permettent d'extraire des motifs dont les transactions sont temporellement proches. Cependant, ils ne permettent pas d'obtenir des motifs dans lesquels des items peuvent être déplacés d'une transaction à une autre. Par exemple, les trois séquences suivantes $\langle(ab)(cd)\rangle$, $\langle a(bcd)\rangle$, $\langle(abc)d\rangle$ semblent se ressembler, mais elles ne pourront, néanmoins, pas être considérées comme des occurrences d'un seul et même motif. Ainsi, les méthodes de fouilles de séquences ne permettent de prendre que partiellement en compte les variabilités temporelles.

Dans le cadre de la modélisation d'un tableau d'annotations sous forme d'un graphe G, les variabilités temporelles sont toutes envisageables puisque le graphe est complet. Ainsi, il est toujours possible de passer d'une annotation quelconque du dialogue à une autre. Chaque motif possible correspond donc à un sous-graphe connexe de G.

Vis à vis de (V_{ag}), les méthodes basées sur la modélisation (M_1) sont similaires à celles de fouille de séquences d'itemsets. Les décalages entre les caractères de la séquence

sont envisageables (via les opérations d'insertion et de suppression), mais étant donné que chaque caractère de la séquence correspond à une ligne d'un tableau d'annotations, le décalage d'une partie des annotations d'une ligne ne pourra être détecté.

Enfin, la méthode LPCA (utilisant la représentation des tableaux d'annotations sous forme de tableaux de caractères) permet une très bonne prise en compte de cette variabilité. En effet, dans la représentation qu'elle considère, chaque caractère correspond à une unique annotation – et non plus à une ligne complète du tableau d'annotations. De ce fait, les opérations d'insertion et de suppression permettent maintenant d'identifier, non plus uniquement des décalages de l'ensemble des annotations d'une ligne, mais aussi ceux ne faisant intervenir qu'une partie des annotations d'une ligne.

Ainsi, seules deux représentations permettent de prendre en compte la variabilité (V_{ag}) de manière satisfaisante, le graphe complet et le tableau de caractères.

Conclusion

Afin que l'approche sélectionnée soit utilisable en pratique, il est primordial de s'assurer que ses performances (*i.e.* : ses temps d'exécution) soient raisonnables.

De ce point de vue, les méthodes de fouilles de motifs comportent une faiblesse. En effet, dans le cas de la fouille de séquences d'itemsets, les bases de données considérées comportent en général beaucoup d'items et peu de séquences. Dans notre cas ces proportions sont inversées. L'alphabet de chaque colonne comporte un nombre relativement faible de caractères tandis que le nombre de lignes du tableau d'annotations est susceptible d'être élevé. Les performances des méthodes de ce type seront donc peu raisonnables si nous les appliquons à nos données. De même, dans le cadre de la fouille de sous-structures, les méthodes sont basées sur le fait que les graphes considérés comportent une faible proportion d'arêtes par rapport au nombre de sommets. Par conséquent, leur application à des graphes complets correspondant à un corpus de tableaux d'annotations semble difficilement envisageable.

A l'inverse, le principal atout de l'algorithme d'alignement de séquences de caractères $s_A[1..m_A]$ et $s_B[1..m_B]$ est sa faible complexité ($\mathcal{O}(m_A m_B)$). L'algorithme LPCA, quant à lui, possède une complexité plus élevée ($\mathcal{O}(m_A m_B n_A n_B)$). Cette dernière est néanmoins plus raisonnable que l'explosion combinatoire qu'engendrerait l'utilisation de méthodes de fouille de motifs.

Les deux méthodes du domaine de la fouille de séquences semblent donc être les plus adaptées tant du point de vue de la complexité que de la prise en compte des variabilités. Néanmoins, chacune de ces méthodes comporte un inconvénient. L'alignement de séquences de caractères ne permet pas de prendre totalement en compte les variabilités temporelles tandis que l'algorithme LPCA possède une complexité non négligeable.

Deux raisons nous amènent à pencher en faveur de l'algorithme LPCA. La première est qu'il est pour nous plus important de permettre l'identification de motifs complexes dont les occurrences sont susceptibles de comporter des variations de leur agencement que d'avoir des temps de calculs de l'ordre de quelques secondes. La seconde est que nous montrons, entre autres, dans le chapitre suivant une adaptation originale de l'algorithme LPCA permettant de réduire sa complexité d'un facteur n_B.

2 | Modélisation et résolution du problème d'extraction de motifs

L'objectif de la première partie de ce manuscrit est de fournir une méthode permettant l'extraction de motifs récurrents dans des tableaux d'annotations. Dans ce contexte, le chapitre précédent est dédié à une étude des méthodes de la littérature susceptibles d'y parvenir.

Nous avons ainsi identifié l'algorithme LPCA qui nous semble être la meilleure alternative pour cette tâche. Cette méthode identifie des alignements locaux correspondant à deux sous-parties de tableaux (*i.e.* : deux occurrences d'un seul et même motif récurrent) similaires. Ainsi, chaque alignement permet d'identifier un motif récurrent. En détectant, dans tous les couples de tableaux d'annotations, les alignements dont la similarité est supérieure à un certain seuil τ, nous obtiendrons ainsi l'ensemble des motifs récurrents du corpus. Dans la suite, un alignement est représenté par les deux occurrences de motifs qu'il contient entourés par des parenthèses (*e.g.* : voir les alignements en figure 2.3d et 2.3e page 51). En section 2.1, nous présentons une adaptation de l'algorithme LPCA, réalisée en collaboration avec Rick Moritz [96], nommée *LPCA-DC*. La complexité de LPCA-DC est plus faible que celle de LPCA et sa phase de tracé arrière est adaptée afin de permettre l'obtention de plusieurs alignements locaux.

Comme nous le voyons dans le chapitre 6, LPCA-DC permet d'identifier des motifs pertinents et interprétables par un expert. Cependant, nous remarquons que les formes prises par les motifs sont contraintes et que l'ordre dans lequel se trouvent les colonnes d'annotations influence les résultats obtenus. Puisque la méthode la plus adaptée de la littérature ne nous donne pas entière satisfaction, nous proposons une modélisation originale du problème d'extraction de motifs récurrents dans des tableaux d'annotations et un algorithme d'extraction, appelé SABRE, basé sur cette modélisation. Ce travail est présenté en section 2.2.

2.1 Algorithme $LPCA-DC$

2.1.1 Description de l'algorithme

La première méthode que nous avons conçue afin d'extraire des motifs dialogiques à partir d'un corpus de tableaux d'annotations est une adaptation de l'algorithme LPCA, présenté en section 1.2.3 calculant le meilleur alignement local entre deux tableaux de caractères $t_A[1..m_A][1..n_A]$ et $t_B[1..m_B][1..n_B]$. Nous appelons cette méthode LPCA-

DC (LPCA Dependent Columns) puisqu'elle tire parti de la structure en colonne des tableaux d'annotations. Son algorithme est présenté en figure 2.1.

Données :
- deux tableaux d'annotations $t_A[1..m_A][1..n_A]$ et $t_B[1..m_B][1..n_A]$
- score minimal τ
- liste li contenant le score des opérations d'édition

Résultat : Un ensemble d'alignements

initialiserBords(T);
pour i *allant de* 1 *à* m_A **faire**
  pour j *allant de* 1 *à* m_B **faire**
    pour k *allant de* 1 *à* n_A **faire**
      $T[i][j][k] \leftarrow minReccurrence(T, i, j, k, li)$; `// Equation 2.1`
retourner *traceArriere*(T, τ);

FIGURE *2.1 – Algorithme LPCA-DC.*

Une observation nous permet, en premier lieu, de réduire d'un facteur n_A la complexité par rapport à l'algorithme LPCA. Chaque colonne d'un tableau d'annotations correspond à une dimension de codage indépendante des autres et possède un alphabet qui lui est propre. Ceci nous amène à postuler que deux annotations ne peuvent être alignées ensemble que si elles proviennent de la même colonne d'annotations. Ainsi, une annotation en position (i, j) de t_A ne pourra être substituée avec une annotation en position (k, l) de t_B uniquement si j est égal à l. De plus, l'insertion ou la suppression de colonnes (opérations d'édition (b) et (d)), qui créent un décalage au niveau des colonnes, sont maintenant interdites. Les dimensions 2 et 4 de la table T – qui correspondent aux colonnes de t_A et t_B – induisent maintenant une redondance dont il est possible de s'affranchir en réduisant à 3 la dimension de T. Pour tout $(i, j, k) \in \{1, ..., m_A\} \times \{1, ..., n_A\} \times \{1, ..., m_B\}$, $T[i][j][k]$ correspond donc maintenant au score de l'alignement local de t_A et t_B se terminant en $t_A[i][j]$ et $t_B[k][j]$. De manière similaire, la définition des tables R et C peut être adaptée afin que ces dernières ne contiennent plus que trois dimensions. La formule de récurrence obtenue à l'issue de ces modifications est alors :

$$T[i,j,k] = \max \begin{cases} T[i-1,j,k] + sup(t_A[i][j]) & (a) \\ T[i,j,k-1] + ins(t_B[k][j]) & (c) \\ T[i-1,j,k-1] + q_{i,j,k,j}(R[i][j][k]) & (e) \\ T[i,j-1,k] + q_{i,j,k,j}(C[i][j][k]) & (f) \\ T[i-1,j-1,k-1] + q_{i,j,k,j}(R[i-1][j][k-1] + C[i][j][k]) & (g) \\ T[i-1,j-1,k-1] + q_{i,j,k,j}(R[i][j][k] + C[i][j-1][k]) & (h) \\ 0 \end{cases} . \tag{2.1}$$

En conséquence, la complexité de l'algorithme LPCA-DC est réduite d'un facteur n_B par rapport à celle de LPCA et est donc égale à $\mathcal{O}(m_A \times n_A \times m_B)$.

La deuxième adaptation apportée à la méthode concerne la phase de tracé arrière.

Dans l'algorithme LPCA, seul l'alignement local de score maximal est obtenu en effectuant un tracé arrière à partir de la position de T de plus grande valeur. Cependant, il est tout à fait possible que deux tableaux de caractères partagent plusieurs sous-parties en commun susceptibles de donner lieu à plusieurs alignements de scores supérieurs au seuil τ. Dans l'exemple présenté en figure 2.2, les caractères en rouge (respectivement en bleu) dénotent une sous-partie de t_A similaire à une sous-partie de t_B. Si l'algorithme LPCA était utilisé sur ces deux dialogues, seul l'alignement dont le score est le plus élevé serait obtenu. Or nous souhaitons que dans ce cas deux alignements soient retournés (un correspondant aux annotations bleues et un correspondant aux annotations rouges).

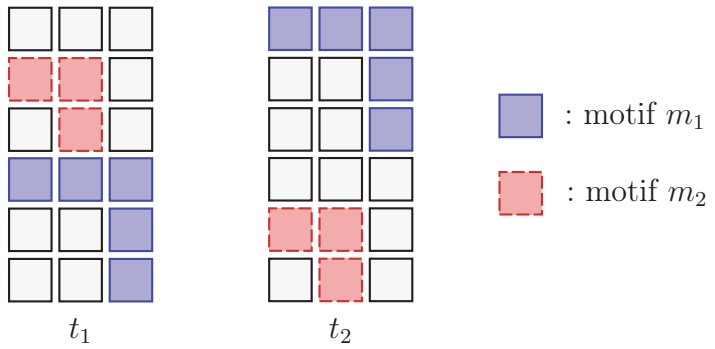
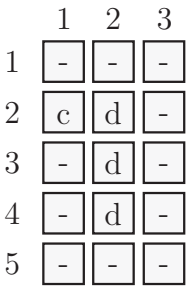


FIGURE *2.2 – Deux tableaux de caractères t_1 et t_2 possédant deux motifs distincts m_1 et m_2 en commun. L'ensemble des annotations d'une couleur représente un motif figurant à la fois dans t_1 et dans t_2.*

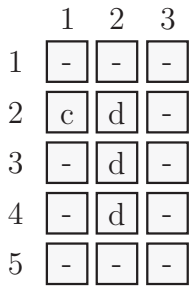
Les positions de T dont le score est supérieur ou égal à τ, que nous appelons *positions candidates*, dénotent un alignement de score supérieur ou égal à τ. Il serait envisageable dans l'algorithme LPCA-DC de réaliser un tracé arrière à partir de chacune de ces positions. Cependant, plusieurs positions candidates sont susceptibles de correspondre à un seul et même alignement, comme le montre la figure 2.3. Dans cet exemple, des alignements sont cherchés entre les tableaux d'annotations t_A (figure 2.3a) et t_B (figure 2.3b). Le score minimum τ est ici fixé à 3 et le score des opérations d'édition unidimensionnelles $sub(c, c)$ et $sub(d, d)$ est égal à 1 tandis que toutes les autres opérations ont un score de -1. Notamment, puisque le score de la substitution d'un tiret par lui même $sub(\text{-}, \text{-})$ est négatif, cela signifie qu'il n'est pas souhaitable de voir apparaître ce caractère dans un alignement. Étant donnés les scores mentionnés, la substitution du couple (c, c) et des trois couples (d, d) fournit un alignement de score 4. Le chemin suivi par l'algorithme dans la table T pour réaliser ces substitutions va de la position $(1, 1, 1)$ à la position $(4, 2, 4)$ et peut être représenté de la manière suivante :

$$(1,1,1) \overset{sub(c,c)}{\longrightarrow} (2,1,2) \overset{sub(d,d)}{\longrightarrow} (2,2,2) \overset{sub(d,d)}{\longrightarrow} (3,2,3) \overset{sub(d,d)}{\longrightarrow} (4,2,4)$$

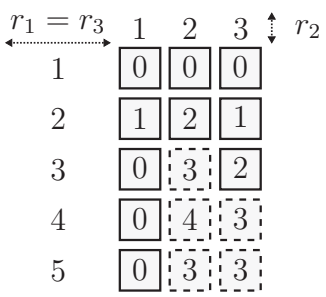
Le score de $T[4][2][4]$ est supérieur à τ, le meilleur alignement (voir figure 2.3d) est donc obtenu en effectuant un tracé arrière à partir de cette position. Cependant, d'autres alignements – qui sont des variations de l'alignement de score maximal – seront eux aussi retournés. La figure 2.3e montre un de ces alignements, obtenu en réalisant un tracé arrière à partir de $T[4][3][4]$. Afin d'éviter d'obtenir des alignements de ce type, les positions candidates sont filtrées et seules celles correspondant à des maximums locaux de T sont conservées. Un tracé arrière est ensuite effectué à partir de l'ensemble des positions candidates restantes.



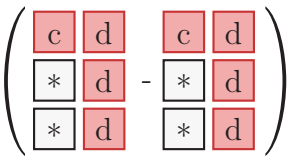
(a) Tableau d'annotations t_A.



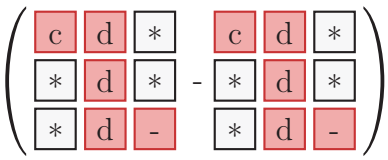
(b) Tableau d'annotations t_B.



(c) Extrait de la table $T[r_1][r_2][r_3]$ correspondante.



(d) Alignement de score 4 obtenu en effectuant un tracé arrière à partir de la position $T[4][2][4]$.



(e) Alignement de sore 3 obtenu en effectuant un tracé arrière à partir de la position $T[4][3][4]$.

FIGURE *2.3 – Deux tableaux d'annotations t_A et t_B ainsi que les valeurs de la table $T[r_1][r_2][r_3]$ correspondante dans l'intervalle $r_1 \in \{1,\ldots,5\}$, $r_2 \in \{1,2,3\}$, $r_3 \in \{1,\ldots,5\}$ et avec r_1 et r_3 égaux. Les positions de T dont le contour est discontinu représentent les positions candidates si un score minimum τ de valeur 3 est considéré.*

Un ensemble de motifs récurrents est obtenu en appliquant l'algorithme LPCA-DC à toutes les paires de dialogues du corpus considéré. Chaque alignement retourné fournit deux motifs récurrents.

2.1.2 Discussion

Comme nous le voyons dans la section 6.3, consacrée à notre première expérimentation (réalisée sur un corpus d'annotations de dialogue entre des parents et leur enfants), la méthode LPCA-DC permet l'obtention de régularités intéressantes que des extractions manuelles n'avaient jusqu'à présent pas permis d'obtenir.

Cependant, elle n'est pas exempte de défauts pour autant. En effet, il a été constaté une faible variabilité dans les formes prises par les motifs extraits. Ces derniers prenaient généralement une forme "d'escalier" (ou d'une partie d'un escalier) partant des premières colonnes et descendant vers celles de droite. Les motifs représentés en figure 2.4 sont de ce type. Ceci peut s'expliquer par le fait que les annotations qui composent un motif figurent sur le chemin suivi par l'algorithme pour construire la table T (ou à proximité de ce dernier). En effet, plus une annotation est éloignée de ce chemin, plus elle est susceptible de nécessiter d'opérations d'édition unidimensionnelles de coût négatif afin de faire partie de l'alignement. Ainsi, puisque les opérations d'édition bidimensionnelles considérées ne permettent de se déplacer dans les tableaux d'annotations que vers le bas et/ou la droite, les chemins suivis dans ces derniers – et par conséquent les motifs eux-mêmes – tendent à avoir une forme d'escalier.

Le type de motif qu'il est possible d'obtenir avec LPCA-DC est ainsi très limité.



FIGURE *2.4 – Exemple de forme de motifs qu'il est possible d'obtenir avec LPCA-DC. Les arcs représentent le chemin parcouru dans le tableau d'annotations pour obtenir ces motifs.*

De plus, de ce fait les motifs extraits dépendent grandement de l'ordre dans lequel apparaissent les colonnes d'annotations. Par exemple, le motif en figure 2.5a pourrait être obtenu à la place du motif figure 2.5b si l'ordre des colonnes était différent. Les annotations en rouge dans ces deux tableaux, représentent les motifs obtenus dans deux configurations différentes des colonnes. Or, l'ordre dans lequel figure les colonnes est arbitraire et ne devrait, dans l'idéal, pas avoir d'incidence sur les motifs extraits.



FIGURE *2.5 – Illustration de l'impact de l'ordre des colonnes d'annotations sur les motifs obtenus.*

Puisque l'approche de la littérature la plus pertinente dévoile des lacunes vis-à-vis de notre problématique, nous avons décidé de définir formellement le problème d'extraction de régularités dans des tableaux d'annotations et de nous appuyer sur cette modélisation pour obtenir un algorithme dont les performances sont meilleures que celles de LPCA-DC.

2.2 Approche par extraction d'arborescences

Afin de pallier les défauts de LPCA-DC, nous avons développé un nouvel algorithme, nommé SABRE, permettant d'extraire des alignements bidimensionnels. Ce dernier repose sur une modélisation formelle du problème d'extraction d'alignements locaux dans un couple de tableaux d'annotations. Nous présentons, en premier lieu, quelques définition et notations utiles à sa présentation.

2.2.1 Définitions et notations

Nous supposons qu'à tout couple de caractères c_A et c_B d'un alignement α est associé un score de similarité – noté $sim(c_A, c_B)$ – dont la valeur est positive ou nulle. Si la similarité entre deux caractères est nulle, cela signifie qu'ils ne pourront pas être alignés ensemble.

Étant donnée une annotation A figurant dans un tableau d'annotations, t, le numéro de la ligne et de la colonne de A dans t sont respectivement représentés par $l(A)$ et $c(A)$. La *position* de A, notée $p(A)$, correspond à $(l(A), c(A))$. Le caractère contenu dans une annotation A est noté $char(A)$.

Nous considérons deux dialogues annotés t_A et t_B. Tout comme en section 2.1, deux annotations ne pourront être alignées ensemble que si elles appartiennent au même alphabet (*i.e.* : à la même colonne). Un *couple d'annotations* (A, B) est un couple composé de deux annotations telles que $A \in t_A$, $B \in t_B$ et $c(A) = c(B)$. Le *score d'un couple d'annotations* (A, B) – noté $s(A, B)$ – est égal à la similarité entre $char(A)$ et $char(B)$. Deux couples d'annotations (A_1, B_1) et (A_2, B_2) sont dit *compatibles* si $p(A_1) \neq p(A_2)$ et $p(B_1) \neq p(B_2)$.

Un *alignement* est un ensemble de couples d'annotations $\{(A_i, B_i)\}_{i=1}^n$, deux à deux compatibles, et ayant tous un score strictement positif (*i.e.* : $\forall i \in \{1, \ldots, n\}$ $sim(char(A_i), char(B_i)) > 0$).

Un motif m *apparaît* dans un alignement $\{(A_i, B_i)\}_{i=1}^n$ si m est égal à $\{A_i\}_{i=1}^n$ ou à $\{B_i\}_{i=1}^n$. Un motif est donc dit récurrent s'il existe un alignement dans lequel il apparaît et dont le score est supérieur ou égal au seuil τ.

2.2.2 Calcul du score d'un alignement

Il semble naturel que le score $\mathcal{S}_\alpha$ d'un alignement $\alpha = \{(A_i, B_i)\}_{i=1}^n$ augmente avec la similarité des différents couples d'annotations qui le composent. Ainsi, le score de α dépendra de la somme des scores d'alignement de ses couples d'annotations $sim_\alpha = \sum_{i=1}^n s(A_i, B_i)$.

La valeur de cette somme n'est cependant pas suffisante pour définir, à elle seule, le score d'un alignement de manière satisfaisante. Les alignements β et γ représentés en figure 2.6 illustrent cette affirmation. Les sommes sim_β et sim_γ sont égales mais il ne semble pas souhaitable d'attribuer à ces deux alignements le même score puisque les agencements de leurs annotations diffèrent. Les deux motifs apparaîssant dans β étant identiques, nous souhaitons que le score de l'alignement β soit supérieur à celui de γ (*i.e.* : $\mathcal{S}_\beta > \mathcal{S}_\gamma$).

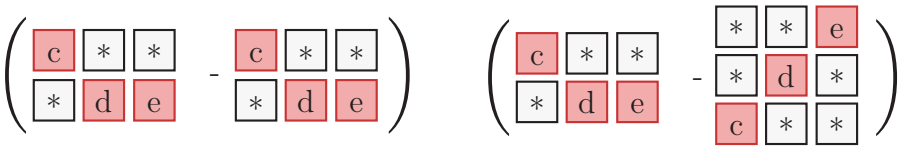
Afin de prendre en compte cet aspect, le score d'un alignement α est pénalisé par un coût relatif à la disposition des annotations au sein de ses deux motifs. Cette pénalité, appelée *coût d'agencement*, est notée ag_α. Le score de l'alignement α est, ainsi, obtenu par la relation

$$\mathcal{S}_\alpha = sim_\alpha - ag_\alpha. \tag{2.2}$$

Étant donné un alignement, le calcul de sim_α est aisé puisqu'il se résume à ajouter la

similarité des couples d'annotations le composant. En revanche, comme nous le voyons dans la suite, l'obtention du coût d'agencement est plus complexe.

Nous commençons par définir le coût d'agencement d'un alignement composé uniquement de deux couples d'annotations, puis nous nous appuyons sur cette définition pour introduire le coût d'agencement d'un alignement de taille quelconque.



(a) Alignement β. *(b) Alignement γ.*

FIGURE *2.6 – Représentation de deux alignements β et γ dont les motifs contiennent les même caractères mais dont l'agencement des annotations diffère.*

Coût d'agencement d'un alignement composé de deux couples d'annotations

La table 2.1 représente six alignements nommés α_1 à α_6, contenant exactement deux couples d'annotations, qui serviront à illustrer les notions présentées dans cette section. Étant donné 1 motif α composé de deux couples d'annotations (A_1, B_1) et (A_2, B_2), nous définissons les variables Δ_A et Δ_B comme respectivement le nombre de lignes séparant A_1 de A_2 et B_1 de B_2 :

$$\Delta_A = l(A_2) - l(A_1) \text{ et } \Delta_B = l(B_2) - l(B_1).$$

Nous rappelons que les lignes des tableaux d'annotations correspondent à des énoncés ordonnés chronologiquement. Les couples (A_1, B_1) et (A_2, B_2) sont dits

- *simultanés* si Δ_A et Δ_B sont nuls. Dans ce cas, A_1 et A_2 apparaîssent simultanément dans le tableau t_A (*i.e.* : ils figurent sur la même ligne) et il en va de même pour B_1 et B_2 dans t_B (*e.g.* : α_1) ;
- *successifs* si Δ_A et Δ_B sont de même signe et non nuls. Les couples sont donc successifs si A_1 apparaît avant A_2 dans t_A et si B_1 apparaît avant B_2 dans t_B (*e.g.* : α_2 et α_5) ou si l'inverse est vrai (*e.g.* : α_3) ;
- *consécutifs* s'ils sont successifs et si $|\Delta_A|$ et $|\Delta_B|$ sont égaux à un. En d'autres termes, les couples sont consécutifs s'ils sont successifs, si les lignes de A_1 et de A_2 dans t_A sont consécutives et si les lignes de B_1 et B_2 dans t_B le sont elles aussi (*e.g.* : α_2 et α_3) ;
- *désynchronisés* si $\Delta_A \neq \Delta_B$. Dans ce cas, soit le nombre de lignes séparant A_1 de A_2 diffère du nombre de lignes séparant B_1 de B_2 (*e.g.* : α_4), soit les couples ne sont pas successifs (*e.g.* : α_6).

Si les deux couples sont désynchronisés, la *taille de la désynchronisation* est définie par $|\Delta_A - \Delta_B|$ (voir exemples en cinquième colonne de la table 2.1). Des désynchronisations de taille élevée dans un alignement traduisent un décalage temporel important dans l'enchaînement des énoncés des dialogues correspondants et nous souhaitons donc les pénaliser. C'est la raison pour laquelle, le coût d'agencement est, tout d'abord, complété par un second terme appelé *coût de désynchronisation* qui est égal à la taille de la désynchronisation multipliée par un coefficient c_d. Le coefficient permet à l'utilisateur d'ajuster les pénalités en fonction des scores d'alignements fournis en entrée et des types de motifs qu'il souhaite obtenir.

Nom	Représentation	Δ_A	Δ_B	Taille de la désynchronisation	Taille de la pause
α_1	$\begin{pmatrix} A_1 & A_2 & - & B_1 & B_2 \end{pmatrix}$	0	0	0	0
α_2	$\begin{pmatrix} A_1 & * & - & B_1 & * \\ * & A_2 & & * & B_2 \end{pmatrix}$	1	1	0	0
α_3	$\begin{pmatrix} * & A_2 & - & * & B_2 \\ A_1 & * & & B_1 & * \end{pmatrix}$	-1	-1	0	0
α_4	$\begin{pmatrix} A_1 & A_2 & - & B_1 & * \\ & & & * & B_2 \end{pmatrix}$	0	1	1	0
α_5	$\begin{pmatrix} A_1 & * & & B_1 & * \\ * & * & - & * & * \\ * & A_2 & & * & B_2 \end{pmatrix}$	2	2	0	1
α_6	$\begin{pmatrix} A_1 & * & & * & B_2 \\ * & * & - & * & * \\ * & A_2 & & B_1 & * \end{pmatrix}$	2	-2	4	0

TABLE *2.1 – Représentation de six alignements composés de deux couples d'annotations et valeur des grandeurs suivantes qui leurs sont associées : nombre de lignes séparant A_1 de A_2 (noté Δ_A), nombre de lignes séparant B_1 de B_2 (noté Δ_B), taille de la désynchronisation entre ces deux couples, taille de la pause entre ces deux couples.*

La proximité temporelle des couples d'annotations d'un motif au sein d'un alignement est un aspect qui doit être pris en compte dans le calcul du coût d'agencement et que le coût de désynchronisation ne permet pas de traduire totalement. Par exemple, l'alignement α_5 représente deux couples qui sont temporellement distants mais qui sont néanmoins synchronisés. Pour intégrer cet aspect, le coût d'agencement est composé d'un deuxième terme appelé *coût de pause*. Deux couples d'annotations (A_1, B_1) et (A_2, B_2) successifs *contiennent une pause* si A_1 et A_2 sont séparés par une ou plusieurs ligne(s) vide(s) et s'il en va de même pour B_1 et B_2. La taille de la pause correspond au minimum du nombre de lignes vides séparant A_1 de A_2 et B_1 de B_2 (*i.e.* : à $\max(0, \min(|\Delta_A|, |\Delta_B|) - 1)$). Deux couples d'annotations simultanés ou consécutifs ne contiennent pas de pause et ne sont, ainsi, pas impactés par ce coût. Cette absence de pénalité est volontaire car ces deux types de couples d'annotations possèdent des agencements que nous ne souhaitons pas pénaliser. En effet :

- deux couples d'annotations simultanés représentent des comportements dialogiques similaires apparaissant dans un même énoncé sur plusieurs dimensions de codage (*i.e.* : dans plusieurs colonnes d'une même ligne).
- deux couples d'annotations consécutifs dénotent une activité qui se poursuit de manière continue dans le temps.

De plus, deux couples d'annotations non successifs auront un coût de pause nul puisque les éventuels décalages temporels de leur agencement sont déjà pris en compte dans le coût de désynchronisation (*e.g.* : α_6). Ceci est intégré à l'expression du coût de pause en ajoutant en facteur le terme ε égal à un si $\Delta_A \Delta_B$ est strictement positif (*i.e.* si les couples sont successifs) et à 0 dans les cas contraires. Tout comme pour les désynchronisations, un coefficient c_p en facteur du nombre de pauses permettra d'obtenir le coût total des pauses. Puisqu'une pause correspond à un décalage de deux lignes (*i.e.* : à deux désynchronisations), le coefficient c_p est égal au double de c_d.

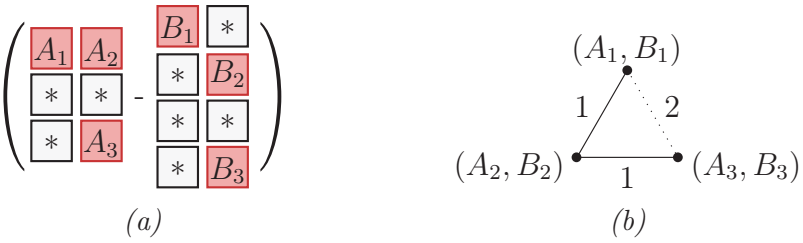
L'expression complète du coût d'agencement $\mathcal{C}_\mathcal{A}$ d'un alignement contenant deux couples d'annotations est ainsi défini par :

$$ag_{\,\alpha} = c_d\,(\,|\Delta_A - \Delta_B| + 2\,\varepsilon \max(0, \min(\,|\Delta_A|, |\Delta_B|) - 1). \tag{2.3}$$

Cette façon de calculer le coût d'agencement d'un alignement comportant uniquement deux couples d'annotations peut maintenant être utilisée comme base en vue de définir le coût d'agencement d'un alignement de taille quelconque.

Coût d'agencement d'un alignement de taille quelconque

Soient $m_A = \{A_i\}_{i=1}^n$ et $m_B = \{B_i\}_{i=1}^n$ les deux motifs figurant dans un alignement $\alpha = \{(A_i, B_i)\}_{i=1}^n$. Pour tout i et j distincts de $\{1, \ldots, n\}$, nous notons $ag_{i,j}$ le coût d'agencement entre les couples d'annotations (A_i, B_i) et (A_j, B_j). Afin de calculer le coût d'agencement de α, nous définissons un graphe complet $G = (V, E)$ composé de n sommets représentants les couples d'annotations et tel que le poids associé à chaque arête ij soit égal à $ag_{i,j}$. Les figures 2.7a et 2.7b illustrent respectivement un alignement composé de trois couples d'annotations et le graphe G correspondant (dans le cas où c_p et c_d sont égaux à un). Afin d'obtenir un agencement de coût minimal des n couples d'annotations les uns par rapport aux autres, nous cherchons un ensemble d'arêtes de poids total minimal permettant de connecter l'ensemble des sommets de V. Soit $T_G \subset E$ cet ensemble d'arêtes (*e.g.* : les traits pleins du graphe en figure 2.7b).



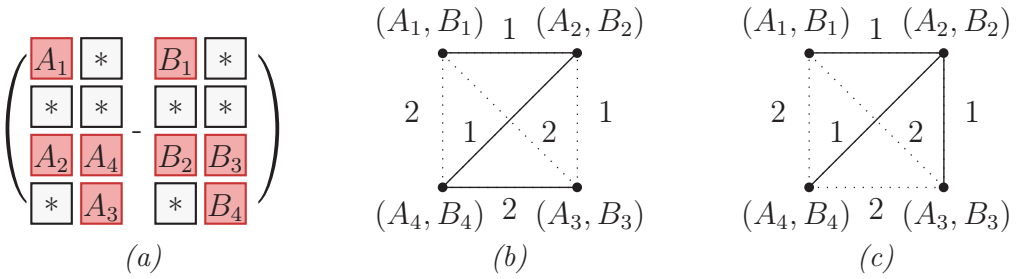
(a) *(b)*

FIGURE *2.7 – (a) Représentation d'un alignement contenant trois couples d'annotations. (b) Graphe G correspondant, lorsque les coefficients c_p et c_d sont fixés à un. Les arêtes figurant dans l'ensemble T_G de poids minimum connectant l'ensemble des sommets sont représentées en trait continu, tandis que les autres sont en pointillés.*

L'exemple en figure 2.8 montre qu'à cause des diverses désynchronisations susceptibles d'apparaître dans un alignement, l'ensemble T_G n'est pas nécessairement une chaîne. Les traits continus de la figure 2.8b, représentent la chaîne de poids minimal permettant de connecter l'ensemble des sommets du graphe G correspondant à l'alignement représenté en figure 2.8a. Le poids de cette chaîne est égal à 4. Cependant, nous

observons en figure 2.8c que le poids de la solution optimale est égal à 3. Néanmoins, il est possible de démontrer que dans le cas général T_G n'est pas pour autant un ensemble d'arêtes quelconques mais correspond à un arbre couvrant de poids minimal. En effet, T_G doit, d'une part, connecter l'ensemble des sommets du graphe et, d'autre part, il ne peut contenir de cycle puisque les poids $ag_{i,j}$ associés aux arêtes de G sont tous positifs et que nous cherchons à minimiser la somme du poids des arêtes de T_G.

Ainsi, le score $\mathcal{S}_\alpha$ d'un alignement α est égal à la somme des similarités des couples d'annotations qui le composent sim_α, moins le poids d'un arbre couvrant de poids minimal du graphe G associé à α.



(a) *(b)* *(c)*

FIGURE *2.8 – (a) Représentation d'un alignement contenant quatre couples d'annotations. (b) Chaîne de poids minimal reliant l'ensemble des sommets du graphe G correspondant. (c) Ensemble T_G de poids minimal connectant l'ensemble des sommets de G.*

Divers algorithmes permettent d'obtenir un arbre couvrant de poids minimal en temps polynomial [106, 77]. Le calcul du score d'un alignement α donné est donc un problème que nous savons résoudre aisément. Cependant, l'extraction d'alignements de score supérieur ou égal à τ à partir de deux dialogues annotés n'est pas une tâche aussi triviale.

2.2.3 Modélisation du problème d'extraction d'alignements

Nous sommes maintenant capables de calculer le coût d'agencement (et donc le score) d'un alignement donné. Etant donnés deux tableaux d'annotations t_A et t_B ainsi qu'un score minimum τ, il reste à modéliser le problème consistant à extraire de t_A et t_B les alignements locaux de score supérieur ou égal à τ.

Pour ce faire, nous considérons un graphe complet $G = (V, E)$ tel qu'à chaque couple d'annotations (A, B) de t_A et t_B soit associé un sommet $v_{A,B}$ de V de poids $s(A, B)$. A chaque arête ij de E est associé un poids égal à l'opposé du coût d'agencement des deux couples d'annotations reliés par l'arête. Le poids des arêtes est négatif puisque le coût d'agencement détériore le score d'un alignement.

Une idée intuitive serait, de manière similaire au calcul du coût d'agencement, de chercher des arbres de poids maximal (non nécessairement couvrant cette fois) dans le graphe G. Cependant, l'incompatibilité de certains couples d'annotations invalide cette approche. En effet, les couples d'annotations (A, B_1) et (A, B_2) contiennent tous deux l'annotation située en position $p(A)$ de t_A et par définition ils ne peuvent donc pas figurer dans un même alignement. Cependant, rien n'empêche ces deux sommets de faire partie d'un même arbre de G.

Afin de prévenir ces incompatibilités, nous ajoutons un nouvel ensemble de sommets V_2 et un ensemble d'arcs A au graphe G (voir exemple en figure 2.9). A toute annotation A de t_A ou t_B est associé un sommet $v_A \in V_2$. Puis, pour tout sommet $v_{A,B}$ de V, nous ajoutons deux arcs de poids nul allant de $v_{A,B}$ à v_A et v_B respectivement. Ainsi, chaque sommet de V est relié à exactement deux sommets de V_2 par deux arcs dont le sommet d'arrivée se trouve en V_2. Puisque G contient maintenant des arcs, nous ne souhaitons plus obtenir un arbre mais une arborescence que nous notons T. Si un sommet de V_2 figure dans T, il ne peut qu'en être une feuille puisqu'aucun arc ne permet de le quitter. Les incompatibilités sont évitées en imposant qu'un sommet $v_{A,B}$ ne puisse être ajouté à T que si c'est aussi le cas des deux arcs $(v_{A,B}, v_A)$ et $(v_{A,B}, v_B)$ qui le relient à des sommets de V_2. Ainsi, deux sommets ayant une annotation A en commun ne pourront faire partie de la même arborescence. Dans l'exemple représenté en figure 2.9, l'annotation B_1 est partagée par les sommets v_{A_1,B_1} et v_{A_3,B_1}, ces derniers ne pourront faire partie d'une même arborescence T, puisque les arcs (v_{A_1,B_1}, v_{B_1}) et (v_{A_3,B_1}, v_{B_1}) (dont l'ajout à T est imposé) créeraient un cycle.

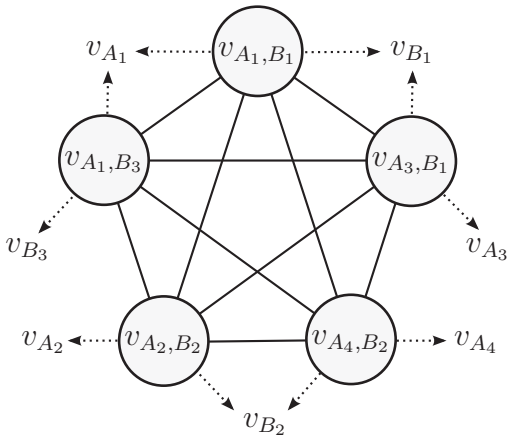


FIGURE *2.9 – Représentation du graphe $G = (V \cup V_2, E \cup A)$ correspondant aux couples d'annotations (A_1, B_1), (A_3, B_1), (A_4, B_2), (A_2, B_2) et (A_1, B_4). Chaque sommet de l'ensemble $V = \{V_{A_1,B_1}, V_{A_3,B_1}, V_{A_4,B_2}, V_{A_2,B_2}, V_{A_1,B_4}\}$ correspond à un de ces trois couples d'annotations tandis que les sommets de $V_2 = \{v_{A_1}, v_{A_2}, v_{A_3}, v_{A_4}, v_{B_1}, v_{B_2}, v_{B_3}\}$ correspondent aux annotations figurant dans ces couples. Les arêtes de E font le lien entre les sommets de V tandis que les arcs de E_2 relient les sommets de V à ceux de V_2.*

Une fois ce graphe G défini, l'alignement local de score maximum contenant un couple d'annotations (A, B) ($A \in t_A$ et $B \in t_B$) peut être obtenu en calculant l'arborescence de poids maximal dont la racine est le sommet $v_{A,B}$. Ainsi, les alignements locaux de score supérieur ou égal à τ peuvent être extraits en applicant cette procédure à tous les couples d'annotations de t_A et t_B et en ne conservant que les alignements dont le score est compatible (*i.e.* : supérieur ou égal à τ).

2.2.4 Algorithme SABRE

Nous pensons que le problème d'extraction d'alignements que nous avons modélisé en sections 2.2.2 et 2.2.3 est $\mathcal{NP}$-complet, bien que cela reste actuellement à démontrer. Dans cette partie, nous définissons une heuristique nommée SABRE (Seed and Arborescence Based Regularity Extraction) dont l'algorithme est représenté en figure 2.10. La

première étape de cet algorithme consiste à identifier des arborescences de G dont les arêtes possèdent un coût nul (*i.e.* : des ensembles de couples d'annotations simultanés ou successifs). Ces arborescences sont appelées des *graines*. Elles servent de points de départ à la détection d'alignements. Lorsque cela est possible, les graines proches et compatibles sont ensuite fusionnées, puis chaque graine est étendue. Cette dernière étape consiste à leur ajouter des arêtes de G de poids négatif (*i.e.* : des couples d'annotations dont l'agencement avec le reste de l'alignement n'est pas nul).

Données :
- deux tableaux d'annotations : $t_A[1..m_A][1..n_A]$ et $t_B[1..m_B][1..n_A]$
- score minimum d'une graine : g_{min}
- score minimum d'un alignement : τ
- table des scores entre annotations : $score$
- coût de désynchronisation : c_d

Résultat : Un ensemble d'alignements

pour i *allant de* 1 *à* m_A **faire**
  pour j *allant de* 1 *à* m_B **faire**
    $\mathcal{C}[i][j] \leftarrow 0$;
    pour k *allant de* 1 *à* n_A **faire**
      $\mathcal{C}[i][j] \leftarrow \mathcal{C}[i][j] + score[t_A[i][k]][t_B[j][k]]$;

$S \leftarrow extraireGraines(\mathcal{C}, g_{min}, sim, c_d)$;
$S \leftarrow fusionnerGraines(S, sim, c_d)$;
retourner $etendreGraines(S, \tau, sim, c_d)$;

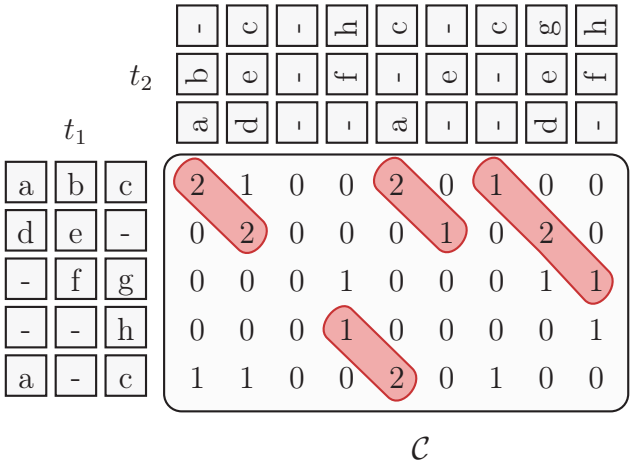
FIGURE *2.10 – Algorithme SABRE.*

L'algorithme SABRE adapte une idée utilisée dans la méthode BLAST [7] qui permet l'extraction d'alignements semi-globaux dans des séquences ADN. Étant donnés deux tableaux de caractères $t_A[1..m_A][1..n_A]$ et $t_B[1..m_B][1..n_A]$, SABRE construit une table $\mathcal{C}[1..m_A][1..m_B]$ telle que $\mathcal{C}[i][j]$ traduise la similarité entre la ligne i de t_A et la ligne j de t_B. Pour ce faire, $\mathcal{C}[i][j]$ est égal à la somme des similarités des annotations contenues dans ces deux lignes :

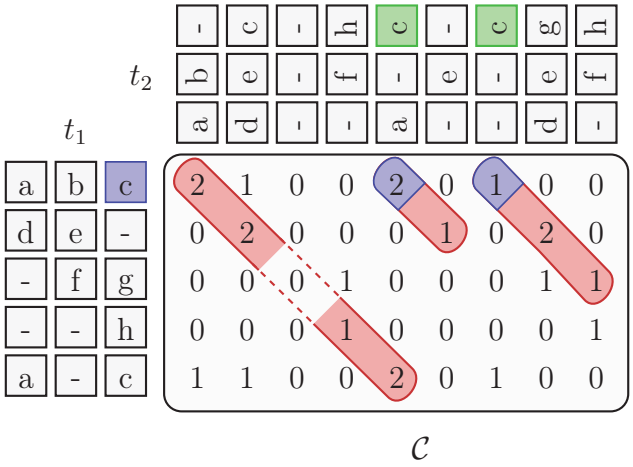
$$\mathcal{C}[i][j] = \sum_{k=1}^{n_A} sim(t_A[i][k], t_B[j][k]). \tag{2.4}$$

Cette table peut être vue comme une représentation simplifiée du graphe G dans laquelle les couples d'annotations simultanés en ligne i de t_A et j de t_B ont été fusionnés et fournissent la similarité $\mathcal{C}[i][j]$. Afin d'illustrer la construction de $\mathcal{C}$ ainsi que les différentes phases de l'algorithme permettant d'en extraire des alignements, nous nous appuyons sur l'exemple représenté en figure 2.11 page 60. Dans cet exemple, la similarité de tout caractère avec lui-même (excepté pour le caractère '-') est égale à un. Toutes les autres similarités entre deux caractères sont égales à 0. Ainsi, $\mathcal{C}[1][1]$ est égal à 2 puisque la première ligne de t_A et de t_B contiennent toutes deux les caractères 'a' et 'b'. Une fois la table $\mathcal{C}$ obtenue, notre algorithme se déroule en trois étapes.

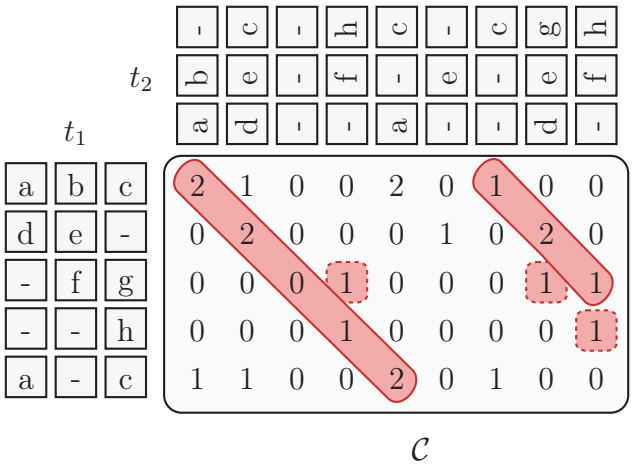
La première étape de SABRE consiste à extraire les *graines* de $\mathcal{C}$ dont le score est supérieur à un paramètre $s_{min} \in \mathbb{R}$. Nous définissons une *graine* g de $\mathcal{C}$ comme l'ensemble des éléments diagonaux $\{d_i\}_{i=1,n}$ d'une sous-matrice carrée $M_{n\times n}$ de $\mathcal{C}$ telle que pour tout



(a) Phase 1 - Détection des graines dans $\mathcal{C}$.



(b) Phase 2 - Fusion de graines compatibles. L'annotation bleue de t_1 apparaît dans les deux graines représentées partiellement en bleu ce qui les rend incompatibles. Les deux annotations 'c' de t_2 qui correspondent à l'annotation bleue sont représentées en vert.



(c) Phase 3 - Extension des alignements et filtrage de ceux dont le score est insuffisant.

FIGURE *2.11 – Présentation des trois phases du nouvel algorithme d'extraction d'alignements locaux sur les tableaux d'annotations t_A et t_B.*

$i \in \{1, \ldots, n\}$, $d_i > 0$ (*e.g* : voir les quatre graines représentées en rouge en figure 2.11a). Le *score de la graine* g, noté s_g, est égal à la trace de M (*i.e.* : $\sum_{i=1}^{n} d_i$). L'intérêt des graines est qu'elles correspondent à des couples d'annotations simultanés ou consécutifs (qui sont ceux dont le coût d'agencement est nul et que nous souhaitons privilégier dans les alignements que nous extrayons). En effet, des annotations simultanées en ligne i de t_A et j de t_B entraîneront une forte valeur $\mathcal{C}[i][j]$ (*e.g.* : $\mathcal{C}[1][1]$ figure 2.11a dont le score est égal à 2 grâce aux annotations 'A' et 'B' en premières lignes de t_A et t_B), tandis que deux couples d'annotations consécutifs entraîneront deux valeurs $\mathcal{C}[i][j]$ et $\mathcal{C}[i+1][j+1]$ non nulles (*e.g.* : $\mathcal{C}[1][1]$ et $\mathcal{C}[2][2]$ figure 2.11a). Dans l'exemple de la figure 2.11a, le score minimum s_{min} est fixé à 3. Quatre graines, représentées en rouge, sont identifiées. L'ensemble des couples d'annotations intervenant dans le score d'une graine g forment un alignement que nous notons α^g (*e.g.* : les alignements correspondant aux quatre graines de la figure 2.11a sont représentés en figure 2.12).

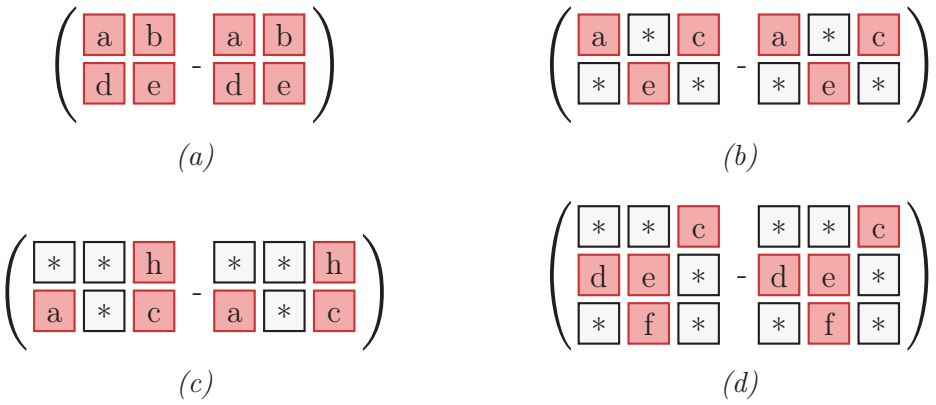


FIGURE *2.12 – Alignements correspondant aux graines représentées en figure 2.11a.*

Dans la seconde étape de SABRE, des graines sont fusionnées. Des alignements α^{g_1} et α^{g_2} correspondant à deux graines g_1 et g_2 sont dits *compatibles* si les couples d'annotations qui les composent sont tous compatibles entre eux. Dans la figure 2.11b, les alignements correspondant aux graines partiellement bleues ne sont pas compatibles car ils contiennent tout deux le caractère 'c' de t_A représenté en bleu. Le *coût d'agencement entre* α^{g_1} *et* α^{g_2} est égal au coût d'agencement minimal entre un couple d'annotations de α^{g_1} et un couple d'annotations de α^{g_2}. Durant cette deuxième étape, deux graines g_1 et g_2 sont fusionnées si elles sont compatibles et si le score de l'alignement α issu de la fusion est meilleur que le score de α^{g_1} et le score de α^{g_2}. Ce processus est répété jusqu'à ce qu'il ne reste qu'une seule graine ou que toutes les fusions envisageables aient été considérées. Dans la figure 2.11b, seules deux graines sont compatibles. Puisque le score de l'alignement obtenu en les fusionnant est meilleur que le score de l'alignement de chacune des deux graines, ces dernières sont fusionnées.

A ce stade, chaque alignement ne contient que des couples d'annotations issus de graines (*i.e.* : des couples d'annotations simultanés ou successifs). La dernière étape consiste à étendre les alignements en leur ajoutant successivement des couples d'annotations compatibles, ne figurant pas dans des graines, et dont l'agencement est proche. Soit $\alpha = \{(A_i, B_i)\}_{i=1}^{n}$ un alignement et soit (A_0, B_0) un couple d'annotations avec lequel il est compatible. Le couple (A_0, B_0) est dit *proche* de α si son ajout à α améliore le score de l'alignement (*i.e.* : si $\mathcal{S}_{\alpha \cup (A_0, B_0)} > \mathcal{S}_\alpha$). Pour chaque alignement, une liste L des couples d'annotations proches et compatibles est créée. Les éléments de L sont

classés en fonction de l'augmentation du score de l'alignement qu'ils engendrent (*i.e.* : plus l'ajout d'un couple d'annotations améliore le score de l'alignement, plus ce couple est situé en début de liste). Le premier couple $\{A_0, B_0\}$ de L est ensuite ajouté à l'alignement, puis la liste L est mise à jour (retrait des couples qui ne sont pas compatibles avec $\{A_0, B_0\}$, ajout de couples rendus proches par $\{A_0, B_0\}$ et nouveau classement des éléments de L). Ce processus est répété jusqu'à ce que la liste soit vide. Chaque alignement est ainsi étendu en commençant par ceux dont le score est le meilleur. Seuls les alignements dont le score final est supérieur ou égal au seuil τ sont conservés. Dans l'exemple de la figure 2.11c, les couples d'annotations correspondant aux valeurs de $\mathcal{C}$ qui figurent en pointillés sont ajoutés à un alignement. Le paramètre τ étant fixé à 4, un des trois alignements de l'étape précédente est filtré, ce qui permet d'obtenir les deux alignements représentés en figure 2.13.

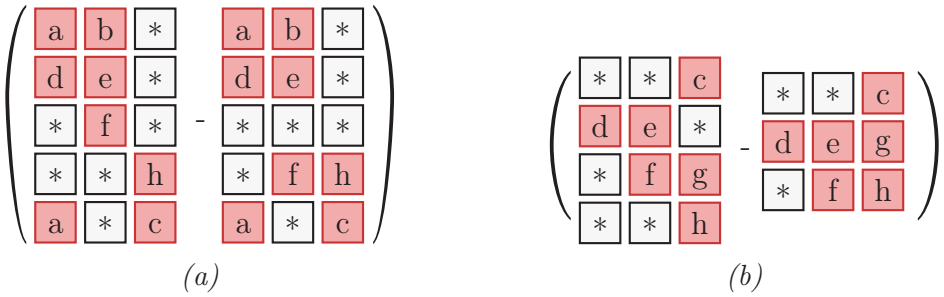


FIGURE *2.13 – Alignements obtenus dans l'exemple représentés en figure 2.11c.*

A chaque étape de cet algorithme, le score des couples d'annotations d'un alignement ainsi que le coût d'agencement des couples d'annotations ajoutés est pris en compte dans le calcul du score de l'alignement. Ainsi, le score d'un alignement α obtenu avec cette méthode correspond exactement au score de cet alignement tel que défini dans la modélisation que nous avons présentée en section 2.2.2.

2.3 Conclusion

Dans ce chapitre, nous présentons deux algorithmes permettant de résoudre le problème d'extraction de motifs récurrents dans des tableaux d'annotations.

Le premier, nommé LPCA-DC, est une adaptation de l'algorithme LPCA [83] permettant d'extraire l'alignement local de score maximal entre deux tableaux de caractères. La première adaptation de cet algorithme a permis de réduire sa complexité tandis que la seconde a rendu possible l'extraction de plusieurs alignements locaux par paire de tableaux d'annotations. Cet algorithme, bien qu'intéressant, comporte néanmoins des lacunes comme le fait de contraindre la forme des motifs obtenus et de fournir des résultats dépendants de l'ordre dans lequel les colonnes d'annotations du schéma de codage sont disposées.

Pour ces raisons, nous avons réalisé une modélisation formelle du problème d'extraction d'alignements locaux dans des tableaux d'annotations et avons montré qu'il revenait à extraire des arborescences de poids maximales dans un graphe associé à un couple de tableaux d'annotations. Nous nous sommes ensuite basé sur cette modélisation afin de définir l'heuristique nommée SABRE permettant de pallier les défauts de LPCA-DC.

La détermination de la complexité du problème d'extraction que nous avons modélisé reste à déterminer bien que nous supposons qu'il soit $\mathcal{NP}$-complet. Ceci constitue une perspective importante des travaux présentés dans ce chapitre.

Nous disposons maintenant de deux méthodes permettant de réaliser l'extraction de motifs récurrents dans un corpus de tableaux d'annotations. Afin de permettre l'obtention de régularités dialogiques, nous devons ensuite étudier le problème consistant à partitionner ces motifs en classes homogènes. Ce travail est décrit dans la seconde partie de ce manuscrit.

Deuxième partie

Classification non supervisée de motifs

3 État de l'art des méthodes de classification non supervisée 68
- 3.1 Heuristiques . . . 69
  - 3.1.1 Méthodes hiérarchiques . . . 69
  - 3.1.2 Méthodes de type partition . . . 72
- 3.2 Approches exactes . . . 76
  - 3.2.1 Partitionnement non contraint . . . 79
  - 3.2.2 Poids des parties contraint . . . 81
  - 3.2.3 Nombre de sommets par partie fixé . . . 83
  - 3.2.4 Nombre de parties contraint . . . 83
- 3.3 Discussion . . . 85

4 Approche polyèdrale pour le problème de K-partitionnement 86
- 4.1 Définitions et notations . . . 86
- 4.2 Formulations . . . 88
  - 4.2.1 Formulation (F_1) . . . 88
  - 4.2.2 Formulation (F_2) . . . 90
  - 4.2.3 Comparaison des formulations . . . 91
- 4.3 Dimension et facettes . . . 94
  - 4.3.1 Dimension de $P_{n,K}$ . . . 94
  - 4.3.2 Facettes . . . 99
- 4.4 Conclusion . . . 119

5 Étude numérique 120
- 5.1 Évaluation de la qualité des inégalités définissant des facettes non triviales de $P_{n,K}$ . . . 120
  - 5.1.1 Algorithmes de séparations . . . 121
  - 5.1.2 Amélioration de la relaxation par famille d'inégalités . . . 124
- 5.2 Résolution numérique par un algorithme de plans coupants . . . 130
  - 5.2.1 Description générale de l'algorithme . . . 130
  - 5.2.2 Obtention d'une solution entière à partir d'une solution fractionnaire 131
  - 5.2.3 Algorithmes de séparation . . . 132
  - 5.2.4 Gestion des algorithmes de séparation et des coupes ajoutées . . . 134
  - 5.2.5 Résultats numériques . . . 135
- 5.3 Conclusion . . . 137

La première partie nous a permis d'identifier des motifs récurrents dans un corpus de tableaux d'annotations. Dans cette seconde partie, nous ne considérons donc plus des tableaux d'annotations, mais un ensemble de motifs récurrents $V = \{m_1, \ldots, m_n\}$.

Nous souhaitons regrouper ces motifs en classes homogènes et pensons que chaque classe de taille conséquente ainsi obtenue constitue une régularité caractéristique du corpus de tableaux d'annotations étudié. Dans l'optique de fournir un outil d'aide à la décision souple et performant, nous souhaitons que cette phase de partitionnement (ou de *clustering*) puisse elle aussi être paramétrable simplement par l'utilisateur. Il nous apparaît que le paramètre le plus intuitif sur lequel il est possible d'influer est le nombre de classes de motifs obtenues que nous notons K.

Ainsi, nous présentons dans le chapitre 3 un état de l'art des méthodes – heuristiques (section 3.1) et exactes (section 3.2) – de partitionnement de graphes. A l'issue de cet état de l'art, cinq méthodes heuristiques, basées sur des mécanismes aussi variés que possible, sont sélectionnées en vue de comparer leurs performances sur un corpus de tableaux d'annotations. Cette expérience est présentée dans le chapitre 6.

Nous proposons en chapitre 4 deux formulations de ce problème que nous nommons K-partitionnement. Nous montrons qu'une des deux formulations est meilleure au sens de sa relaxation linéaire, puis étudions les conditions sous lesquelles des familles d'inégalités définissent des facettes de l'enveloppe convexe de ses solutions entières. Enfin, nous présentons en chapitre 5 une étude numérique de l'efficacité des trois familles d'inégalités non triviales identifiées ainsi qu'un algorithme de type plans coupants utilisant ces résultats pour résoudre efficacement le problème de K-partitionnement.

3 | État de l'art des méthodes de classification non supervisée

Les méthodes d'extraction présentées en chapitre 2 nous permettent d'obtenir un ensemble $\{m_1, \ldots, m_n\}$ de motifs récurrents d'un corpus de tableaux d'annotations. Afin de ne pas fournir à l'utilisateur de notre outil une simple liste de motifs récurrents, nous souhaitons maintenant partitionner ces motifs en classes homogènes. Cette deuxième étape de partitionnement devra permettre de faire apparaître les motifs les plus représentatifs du corpus dans des parties de grande taille et constitueront les régularités du corpus étudié.

Dans l'optique de fournir une méthode souple et ajustable, nous souhaitons laisser la possibilité à l'utilisateur de faire varier un ou plusieurs paramètres. Pour cela, différentes approches sont envisageables. De nombreuses méthodes proposent des bornes sur le nombre de points (*i.e.* : le nombre de motifs dans notre cas) contenu dans chaque partie. Ce type de contraintes est utilisé dans des applications où la taille des parties doit être relativement équilibrée (*i.e.* : répartition de charge sur des processeurs, gestion du personnel, etc.). Dans notre contexte, nous souhaitons au contraire permettre que les motifs puissent être regroupés en ensembles de tailles variables afin de mieux faire ressortir les régularités les plus significatives du corpus. Notre choix s'est porté sur un paramètre qui nous a paru la fois intuitif pour l'utilisateur et adapté à notre problématique, à savoir le nombre de parties K retournées par la méthode. Ainsi, dans ce chapitre nous présentons des méthodes de partitionnement de la littérature en mettant principalement l'accent sur celles qui permettent de fixer le nombre de parties.

Le problème de partitionnement est un problème à la fois naturel et fondamental qui est utilisé dans de nombreux domaines (*e.g.* : sciences naturels, psychologie, médecine, ingénierie, économie, marketing, etc.) et possédant une longue histoire. En effet, d'après Hansen et Jaumard [60], ce dernier était déjà bien étudié au 18ème siècle par des naturalistes telles que Buffon, Cuvier ou Linné et il a même été mentionné dans des écrits datant de l'époque d'Aristote. Il n'est donc pas étonnant que la littérature concernant ce sujet soit vaste et hétérogène comme le montre les diverses études réalisées sur le sujet [17, 60, 44, 64].

La forme que peuvent prendre les données à partitionner peut varier. Dans la représentation la plus courante, à chaque point x est associé un vecteur $(x_1, \ldots, x_d)$ de dimension d appartenant à un espace de recherche A. Chaque composante de ce vecteur correspond à un attribut qui peut être quantitatif ou qualitatif. Cependant, il ne semble pas pertinent de représenter un motif par un vecteur d'attributs. En effet, étant donné que le nombre d'annotations contenues dans un motif peut varier et que l'agencement de ces dernières les unes par rapport aux autres est peu contraint, il semble complexe,

voire réducteur, d'utiliser ce type de représentation. C'est la raison pour laquelle nous employons le point de vue utilisé par les méthodes de partitionnement de graphes et considérons un graphe complet $G = (V, E)$ dont les n sommets correspondent aux motifs extraits (*i.e.* : $V = \{m_1, \ldots, m_n\}$) et associant à chaque arête un poids égal à la dissimilarité entre les deux motifs qu'elle relie. Nous voyons, chapitre 6, comment les méthodes d'extractions LPCA-DC et SABRE peuvent être adaptées afin de permettre le calcul d'une telle dissimilarité.

Dans la suite, le nombre de sommets contenus dans le graphe est désigné par $|V|$ ou n. De même, l'arête reliant deux motifs distincts m_i et m_j pourra être dénotée par ij ou (i, j) et son poids est représenté par $w_{i,j}$.

Nous présentons, tout d'abord, en section 3.1 les méthodes heuristiques qui privilégient la rapidité, puis nous décrivons les approches exactes en section 3.2 qui permettent de fournir une solution optimale.

3.1 Heuristiques

Compte tenu de la représentation que nous considérons, les motifs à partitionner ne correspondent pas à des vecteurs d'attributs appartenant à un espace de recherche A, mais à des sommets d'un graphe complet. Ainsi, diverses familles de méthodes heuristiques utilisant des mécanismes liés à la présence d'un espace de recherche ne pourront être utilisées et ne seront donc pas décrites dans cette section. Nous pouvons, notamment, citer les approches basées sur la notion de densité dans l'espace de recherche A (*e.g.* : DBSCAN [35], OPTICS [9]) qui identifient des zones de A très peuplées, celles utilisant un découpage de A afin de réduire la complexité appelées *grid-based methods* (*e.g.* : CLIQUE [2] ou STING [123]) ou encore les méthodes probabilistes telles que l'espérance-maximisation (*Expectation-Maximisation*) [30] ou AUTOCLASS [23].

Malgré cela, nous ne pouvons prétendre à une présentation exhaustives des approches de la littérature et nous nous concentrons donc sur les méthodes qui nous ont semblé les plus pertinentes.

Dans la littérature, deux familles de méthodes de partitionnement sont généralement distinguées. Les approches hiérarchiques (que nous présentons en sous-section 3.1.1) renvoient un ensemble de partitions imbriquées tandis que les méthodes dites de type partition que nous traitons en sous-section (3.1.2) ont pour résultat une partition unique.

3.1.1 Méthodes hiérarchiques

Une *partition* $\pi = \{P_1, \ldots, P_K\}$ (ou *K-partition*) d'un ensemble V vérifie les trois conditions suivantes :

(i) $P_i \neq \emptyset \; \forall i \in \{1, \ldots, K\}$;

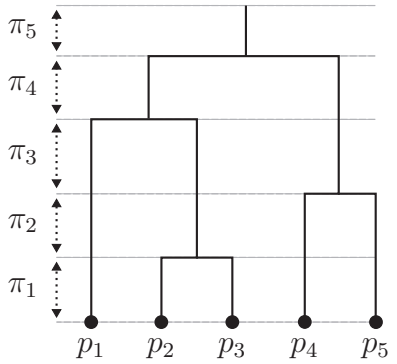
(ii) $P_i \cap P_j = \emptyset \; \forall i, j \in \{1, \ldots, K\}, i \neq j$;

(iii) $\cup_{i=1}^{K} P_i = V$.

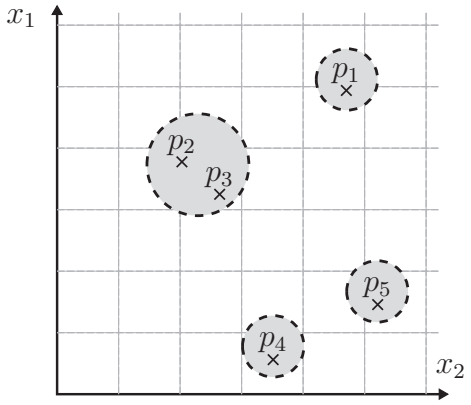
La majorité des méthodes de partitionnement permettent d'obtenir une partition des données. Cependant, d'autres types de structures peuvent aussi être considérés comme

les recouvrements (partition relaxant la contrainte (ii)) ou les pavages (partitions relaxant la contrainte (iii)).

Après les partitions, la structure ayant reçue le plus d'attention est la hiérarchie. Une *hiérarchie* H est composée de q partitions $\{\pi_1, \ldots, \pi_q\}$ $(q \leq n)$ telles que ($P_i \in \pi_k$, $P_j \in \pi_l$ et $k > l$) implique ($P_i \subset P_j$ ou $P_i \cap P_j = \emptyset$) pour tout $i \neq j$ et $k, l \in \{1, \ldots, n\}$). Une hiérarchie peut être visualisée par un diagramme appelé *dendrogramme*. La figure 3.1a illustre un exemple de dendrogramme, correspondant aux cinq points p_1 à p_5 représentés en figure 3.1b. Sur ce diagramme, cinq partitions π_1 à π_5, correspondant chacune à un intervalle vertical, peuvent être observées. Pour une hauteur donnée, chaque ligne verticale visible correspond à une partie. Par exemple, la partition π_2 (représentée figure 3.1b) contient quatre parties car dans son intervalle vertical quatre lignes (correspondant aux ensembles $\{p_1\}$, $\{p_2, p_3\}$, $\{p_4\}$ et $\{p_5\}$) peuvent être observées
.



(a) Dendrogramme correspondant aux cinq points représentés en (b).



(b) Cinq point dans un espace à deux dimensions regroupés selon la partition π_2 du dendrogramme figure (a).

FIGURE *3.1 – Exemple de dendrogramme obtenu en utilisant l'algorithme Single-Link.*

L'avantage des hiérarchies de partitions pour notre problématique est qu'elles permettent à l'utilisateur de visualiser des solutions correspondant à différentes valeurs de K (le nombre de parties) sans nécessiter de calcul supplémentaire.

Les méthodes hiérarchiques se décomposent elles-mêmes en deux catégories : les méthodes agglomératives et les méthodes divisives. Les approches agglomératives débutent avec une partition contenant n parties (*i.e.* : chaque sommet figure seul dans une partie) et vont itérativement fusionner les parties les plus similaires jusqu'à ce qu'un critère d'arrêt soit atteint. Le critère d'arrêt correspond en général à un nombre minimal de parties. Les méthodes hiérarchiques divisives, à l'inverse, considèrent initialement une unique partie contenant l'ensemble des sommets et à chaque itération une ou plusieurs parties seront divisées. Les méthodes divisives, qui sont plus rares, se basent généralement sur des propriétés géométriques pour déterminer la partie à diviser et la meilleure façon de le faire. Comme nous n'utilisons pas la représentation basée sur un espace de recherche, ces approches géométriques ne sont pas ici envisageables. Nous nous intéressons donc en particulier aux approches agglomératives.

En vue de fusionner les parties les plus similaires, les méthodes hiérarchiques agglomératives vont nécessiter la définition d'une distance entre des sous-parties P_1 et P_2

de V. Les méthodes les plus simples consistent à évaluer la proximité de P_1 et P_2 en prenant uniquement en compte la distance $w_{i,j}$ entre un élément i de P_1 et un élément j de P_2. La *métrique single-link* [42] définit la distance entre P_1 et P_2 comme la plus petite distance séparant un élément de P_1 d'un élément de P_2 (*i.e.* : $\min_{i \in P_1\ j \in P_2} w_{i,j}$). Le dendrogramme précédemment présenté en figure 3.1a correspond à cette métrique. Similairement, les métriques *average-link* [65] et *complete-link* [117] définissent la distance entre P_1 et P_2 comme respectivement la distance moyenne et la distance maximale entre deux éléments de P_1 et P_2.

Des métriques plus complexes ont aussi été envisagées. La méthode ROCK [53], par exemple, se sert de la notion de voisinage pour définir la proximité entre deux parties. Deux sommets de V v_i et v_j seront dits *voisins* si leur distance est inférieure à un seuil θ. Le lien entre les sommets m_i et m_j, noté $link(m_i, m_j)$ est défini comme le nombre de voisins que ces sommets ont en commun. De plus, le lien entre deux parties de V, P_i et P_j, noté $link(P_i, P_j)$ est égal à

$$\sum_{p_k \in P_i,\, p_l \in P_j} link(p_k, p_l).$$

Intuitivement cette notion de nombre de voisins partagés semble une bonne façon de mesurer la proximité de deux parties et il serait possible de fusionner à chaque itération les deux parties maximisant cette fonction. Cependant, cela aurait pour conséquence de souvent fusionner les sommets dans la même partie car plus une partie contient de sommets, plus elle contiendra de voisins. Pour pallier cela, les auteurs proposent de normaliser la fonction *link* par le nombre de sommets contenus dans les deux parties considérées. La qualité de deux parties P_i et P_j de tailles respectives n_i et n_j est ainsi égale à

$$\frac{link(P_i, P_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}},$$

où la fonction $f(\theta)$ est une fonction dépendante des données. En utilisant ce ratio, le risque que les sommets soient toujours fusionnés dans les même parties est évité.

Les méthodes que nous avons présentées jusqu'à présent qui généralisent la notion de distance entre sommets en une notion de distance entre des sous-parties sont dites de type *linkage*. Elles sont prédisposées à fournir des parties de formes convexes puisque les parties les plus proches sont systématiquement celles choisies pour être fusionnées.

Or, il n'est pas toujours souhaitable d'obtenir des parties de forme convexe. C'est pourquoi, des algorithmes hiérarchiques agglomératifs ne prenant pas uniquement en compte la proximité ont été développés en vue de permettre l'obtention de parties de formes quelconques. Les deux plus connus sont CURE [52] et CHAMELEON [72]. L'algorithme CURE a été développé par les même auteurs que ROCK et partagent certains de leurs mécanismes de voisinages. Cependant, CURE utilise pour représenter une partie, un nombre fixe de points répartis autour de cette dernière dans l'espace de recherche et utilise ces derniers pour déterminer la distance entre deux parties. Notre choix de représentation sous forme de graphe ne permet donc pas son utilisation.

La méthode CHAMELEON, en revanche, est utilisable sur des données représentées sous forme de graphes. Son déroulement est composé de trois étapes dont les deux premières permettent d'obtenir une partition initiale qui servira de point de départ à

la phase de fusion classique. L'idée derrière cet algorithme est que la proximité entre les parties ne devrait pas être le seul critère utilisé pour déterminer celles devant être fusionnées. Ceci peut être illustré par l'exemple en figure 3.2, inspiré de [72]. Dans ce dernier, les méthodes se basant uniquement sur la notion de proximité pour sélectionner les parties à fusionner décideront de regrouper les parties correspondant aux croix et aux cercles. Cependant, il paraît plus judicieux de regrouper celles représentées par des triangles et des carrés dont la disposition semble plus compatible. C'est la raison pour laquelle CHAMELEON évalue la pertinence de la fusion de deux parties P_i et P_j grâce, d'une part, à un facteur de proximité $RP(P_i, P_j)$ et, d'autre part, à un facteur $RI(P_i, P_j)$ représentant l'interconnectivité de P_i et P_j. L'expression finale d'évaluation de la qualité de la fusion de deux parties est

$$RI(P_i, P_j)RC(P_i, P_j)^{\alpha},$$

où le paramètre α permet d'accorder plus ou moins d'importance à un des deux facteurs.

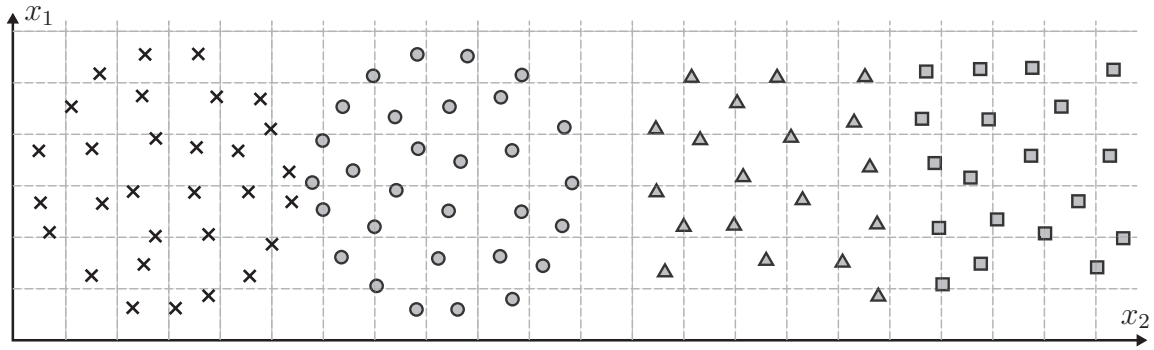


FIGURE *3.2 – Exemple de parties qui ne seraient pas fusionnées correctement par les méthodes hiérarchiques ne prenant en compte que la proximité entre les parties. Ces dernières fusionneraient ensemble les parties correspondant aux croix et aux cercles tandis qu'il semble plus judicieux de d'abord fusionner celles correspondant aux triangles et aux carrés. Image inspirée de [72].*

3.1.2 Méthodes de type partition

Nous présentons maintenant les approches heuristiques permettant l'obtention de partitions. Nous les regroupons en deux catégories : les méthodes de relocalisation et celles dites de partitionnement de graphes.

Méthodes de relocalisation

Étant donnés n points, le nombre total de K-partitions possibles est égal au *nombre de Stirling de seconde espèce* qui est défini de la manière suivante :

$$S(n, K) = \frac{1}{K!} \sum_{j=1}^{K} (-1)^{K-j} \binom{k}{j} j^n.$$

Cette expression donne un aperçu de la combinatoire inhérente à ce problème. En effet, pour $(n, K) = (10, 5)$, elle est supérieure à 10^{10} et pour $(n, K) = (100, 5)$ elle devient

supérieure à 10^{66}. L'évaluation de chaque solution n'est pas une option envisageable, c'est la raison pour laquelle des heuristiques gloutonnes de type relocalisation ont été définies. Ces dernières considèrent une partition initiale, puis itérativement réaffectent les sommets entre les K parties. Ainsi, contrairement aux approches hiérarchiques dans lesquelles une partition obtenue n'est jamais réexaminée, les algorithmes de relocalisation vont graduellement améliorer la solution courante.

Bien que vieille de plus de 50 ans, la méthode des k-moyennes (*k-means*) [89] est l'approche de partitionnement la plus célèbre et la plus utilisée. Elle consiste à choisir aléatoirement k points de l'espace de recherche, appelés *centroides* ou *noyaux*, et à assigner chaque sommet à la partie correspondant au centroide dont il est le plus proche. A chaque itération, les noyaux sont déplacés afin d'améliorer la partition obtenue. Cette méthode ainsi que ses nombreuses variantes nécessitent, néanmoins, une représentation des données dans un espace métrique afin de permettre la définition et l'évolution des centroides. C'est pourquoi nous ne pourront pas les utiliser.

Une méthode récente et originale de type relocalisation, nommée propagation d'affinité, a été développée par Frey et Dueck [43]. Dans cette dernière, chaque sommet du graphe envoie itérativement à ces voisins des *messages* (*i.e.* : des valeurs entières) permettant de faire émerger une partition. Les messages envoyés d'un sommet i à un sommet j peuvent être de deux types :

- responsabilité (noté $r(i,j)$) : quantifie la pertinence pour que i soit représenté par j ;
- disponibilité (noté $a(j,i)$) : quantifie la pertinence pour que i représente j.

Initialement, les disponibilités sont initialisées à 0. A chaque itération, la mise à jour des responsabilités est effectuée par la formule suivante :

$$r(i,j) = sim(i,j) - \max_{k \neq j}(a(i,k) + s(i,k)).$$

Ainsi plus i et j sont similaires, plus i est susceptible d'être représenté par j. En revanche, plus un i est proche d'un sommet k susceptible de le représenter, moins $r(i,j)$ est élevé.

La disponibilité de i envers j est, quant à elle, déterminée par la somme des responsabilités du nœud i envers tous les points excepté j :

$$a(j,i) = \min\left\{0, \sum_{k \in V \setminus \{j\}} \max\{0, r(k,i)\}\right\}$$

Cette méthode ne permet cependant pas de fixer le nombre de parties K obtenu puisque ce dernier émerge au cours des itérations. Cependant, elle a pour avantage d'attribuer un représentant à chaque partie qui figure parmi les sommets du graphe d'origine, ce qui pourrait être une information pertinente pour un utilisateur dans le cadre de notre outil d'aide à la décision.

Méthodes de partitionnement de graphe

Nous rappelons que nous considérons un graphe complet $G = (V, E)$ dont chaque arête $(i,j) \in E$ possède un poids $w_{i,j}$. Étant donnés deux sous-ensembles de V, V_1 et V_2, le terme $W(V_1, V_2)$ est utilisé pour désigner l'expression $\sum_{i \in V_1, j \in V_2} w_{i,j}$.

Un graphe complet peut être partitionné en K parties connexes en lui supprimant (*i.e.* : en lui coupant) des arêtes. Les arêtes de E ainsi coupées forment une K-coupe

(une 2-coupe est simplement appelée *coupe*). Les méthodes utilisant des fonctions objectif basées sur la notion de K-coupes sont dites de type partitionnement de graphes. L'approche la plus intuitive utilisant cette notion est *mincut*. Étant donné une nombre K de parties, elle consiste à trouver une partition $\pi = \{P_1, \ldots, P_K\}$ telle que le poids des arêtes de la K-coupe est minimisé :

$$\min \frac{1}{2} \sum_{i=1}^{K} W(P_i, V \backslash P_i).$$

Dans le cas particulier où K est égal 2 et où les poids sont positifs, mincut peut être résolu en un temps polynomial [119].

L'*algorithme de Kernighan-Lin* [73] utilise cette fonction objectif. Différentes variantes de cet algorithme existent et nous présentons ici celle correspondant à l'article cité précédemment. Les auteurs définissent en premier lieu un algorithme (que nous nommons KL_1) permettant de partitionner un graphe contenant $2q$ sommets en deux parties de taille identique. Ce dernier est ici présenté en figure 3.3 sous une forme légèrement différente de celle choisie par les auteurs et ceci pour deux raisons. La première est d'éviter au lecteur d'entrer dans des considérations techniques superflues et la seconde est de permettre une meilleure compréhension des adaptations de cet algorithme réalisées en section 5.2 dans le cadre de la séparation de familles d'inégalités. Comme le représente la figure 3.3, l'algorithme crée tout d'abord une bipartition et va tenter de l'améliorer dans des passes successives en diminuant le poids de sa coupe. Pour ce faire, des couples de sommets figurant dans des parties différentes vont successivement être échangés d'une partie à l'autre. Durant une passe, chaque sommet du graphe est échangé une unique fois et $|V|/2$ transformations seront donc nécessaires. Dans les autres variantes de cet algorithme, les transformations considérées ne sont pas toujours nécessairement des échanges entre deux sommets figurant dans des parties différentes. Par exemple, il est possible de réaliser des transformations dans lesquelles un unique sommet v est déplacé d'une partie à une autre. Une transformation est dite valide si elle implique deux sommets qui n'ont pas encore été échangés dans cette passe. La fonction $meilleureTransformation$ va effectuer l'échange le plus intéressant (*i.e.* : la transformation valide qui entraîne une coupe de poids minimal). Lorsqu'une transformation impliquant deux sommets a et b est effectuée, le coût des transformations est mis à jour en conséquence. L'algorithme est interrompu lorsque le coût de la meilleure coupe d'une passe est moins bon que celui de celle de la passe précédente. Une simple adaptation de cet algorithme permet de l'appliquer à des parties de taille quelconque (en ajoutant des sommets factices à la plus petite des deux parties).

Kernighan et Lin présentent ensuite un algorithme (que nous nommons KL_2), permettant de partitionner un graphe en K parties contenant au plus p sommets. Ce dernier débute par la génération d'une K-partition $\pi = \{P_1, \ldots, P_K\}$. Afin de faire évoluer π, l'algorithme KL_1 va être successivement appliqué à tous les couples de parties (P_i, P_j) pour tout i, j distinct de $\{1, \ldots, K\}$. Ce processus constitue une passe de l'algorithme KL_2. De manière similaire à KL_1, l'algorithme KL_2 est interrompu lorsque la K-partition d'une passe est moins bonne que celle de la passe précédente. Les expériences ont montré que le nombre de passes nécessaire est faible et que l'algorithme converge rapidement. La limitation sur le nombre de sommets est prise en compte en interdisant les transformations qui entraîneraient l'obtention d'une partie contenant plus de p sommets.

Données : Un graphe complet $G = (V, E)$
Résultat : Une bipartition de $G = (V, E)$

$\pi_{passe} \leftarrow$ créerBipartition(G); `// Meilleure bipartition de cette passe`

répéter
  $\pi_{prec} \leftarrow \pi_{passe}$; `// Meilleure bipartition de la passe précédente`
  $\pi \leftarrow \pi_{passe}$; `// Bipartition courante`
  $couts \leftarrow$ coutsTransformations(π_{passe});
  pour i *allant de* 1 *à* $|V|/2$ **faire**
    $\pi \leftarrow meilleureTransformation(\pi, couts)$;
    $couts \leftarrow miseAJour(\pi, couts)$;
    si $poids(\pi) < poids(\pi_{passe})$ **alors**
      $\pi_{passe} \leftarrow \pi$;
jusqu'à $poids(\pi_{passe}) \geq poids(\pi_{prec})$;
retourner π_{prec};

FIGURE *3.3 – Algorithme KL_1 permettant l'obtention d'une bipartition minimisant le poids de la coupe.*

Généralement, l'approche *mincut* ne permet pas d'obtenir des partitions satisfaisantes car la taille des différentes parties est très déséquilibrée (de nombreuses parties sont réduites à un unique sommet). Pour pallier cela, diverses fonctions objectif dérivées de *mincut* ont été définies. Les deux plus connues sont RatioCut [54] et Ncut [113]. Afin de permettre l'obtention des parties de tailles équilibrées, ces fonctions normalisent le poids des arêtes de la K-coupe par la taille des parties. Avec RatioCut, la taille d'une partie $P \subset V$ correspond au nombre qu'elle contient, tandis que Ncut considère le volume de P qui est défini par $vol(P) = \sum_{i \in P} \sum_{j \in V} w_{i,j}$. Les expressions de ces deux fonctions sont :

$$RatioCut(P_1, \ldots, P_K) = \frac{1}{2} \sum_{i=1}^{K} \frac{W(P_i, V \backslash P_i)}{|A_i|}$$

et

$$NCut(P_1, \ldots, P_K) = \frac{1}{2} \sum_{i=1}^{K} \frac{W(P_i, V \backslash P_i)}{vol(P_i)}.$$

De nombreux algorithmes utilisant sur ces fonctions objectifs ont depuis été développés. Notamment, les progrès les plus récents ont été réalisés par les approches dites spectrales. Ces méthodes sont basées sur des résultats théoriques faisant le lien entre des propriétés spectrales et le partitionnement de graphes [15, 33]. Elles réduisent la dimension du problème en travaillant dans l'espace formé par des vecteurs propres d'un laplacien de graphe L associé à la matrice de dissimilarité $W = (w_{i,j})_{i,j \in \{1,\ldots,n\}}$. Différents laplaciens de graphes peuvent être utilisés, notamment :

- le *laplacien de graphe non normalisé* : $L = D - W$, avec $D = (d_{i,j})_{i,j \in \{1,\ldots,n\}}$ une matrice diagonale telle que $d_{i,i} = \sum_{i=1}^{n} w_{i,j}$;
- le *laplacien de graphe normalisé dit symétrique* : $L_{sym} = D^{-1/2} L D^{-1/2}$;
- le *laplacien de graphe normalisé dit de marche aléatoire* : $L_{rw} = D^{-1} L$.

Par exemple, lorsque que l'on souhaite partitionner le graphe en deux et que la fonction

RatioCut est considérée, il a été montré que le problème équivalent à la minimisation du produit $f'Lf$ pour un vecteur f bien choisi dont les composantes discrètes peuvent prendre deux valeurs. Il est ensuite possible de prouver que la solution optimale du problème obtenu en relaxant la contrainte de discrétion des composantes du vecteur est obtenue lorsque f est égal à la deuxième valeur propre du laplacien L. Une très bonne approximation de la solution optimale peut ainsi être obtenue. Des propriétés similaires, permettent de résoudre des problèmes utilisant les diverses fonctions objectif précédemment mentionnées pour des valeurs de K quelconques.

Après cette description des méthodes heuristiques, nous présentons maintenant les travaux réalisés dans le cadre d'approches exactes pour différentes variantes du problème de partitionnement de graphes.

3.2 Approches exactes

Les approches exactes permettent de trouver la solution optimale de problèmes liés au partitionnement de graphes pour une fonction objectif donnée. La fonction objectif généralement choisie prend en compte le poids de la K-coupe ou, de manière équivalente, le poids de la partition (*i.e.* : somme du poids des arêtes ne figurant pas dans la K-coupe). Dans le cas où le poids de la K-coupe est considéré, l'objectif est de le minimiser lorsque les arêtes représentent une similarité et de le maximiser dans le cas contraire. Lorsque le poids de la partition est utilisé, c'est le contraire.

Dans cette section, nous mettons principalement l'accent sur les travaux similaires aux nôtres (*i.e.* : utilisant une approche polyèdrale pour la résolution d'un programme linéaire en nombres entiers). Nous remarquons néanmoins que diverses approches ont récemment été dédiées à l'étude de problèmes de partitionnement basés sur des formulations quadratiques [55, 18, 87].

Avant de décrire les travaux en eux-même, nous présentons quelques notions et techniques de programmation linéaire. Un *programme linéaire* est composé de variables $x \in \mathbb{R}^n$ soumises à des contraintes linéaires $Ax \leq b$ (avec $A \in \mathcal{M}_{m \times n}$ et $b \in \mathbb{R}^m$). Le but est de trouver le ou les vecteurs $x^* \in \mathbb{R}^n$ permettant d'optimiser (minimiser ou maximiser) une fonction objectif linéaire $c^T x$ ($c \in \mathbb{R}^n$). Lorsqu'un problème de ce type comporte un nombre raisonnable de variables, différentes techniques – telles que l'algorithme du simplexe [99] ou la méthode des points intérieurs [71] – permettent de le résoudre efficacement.

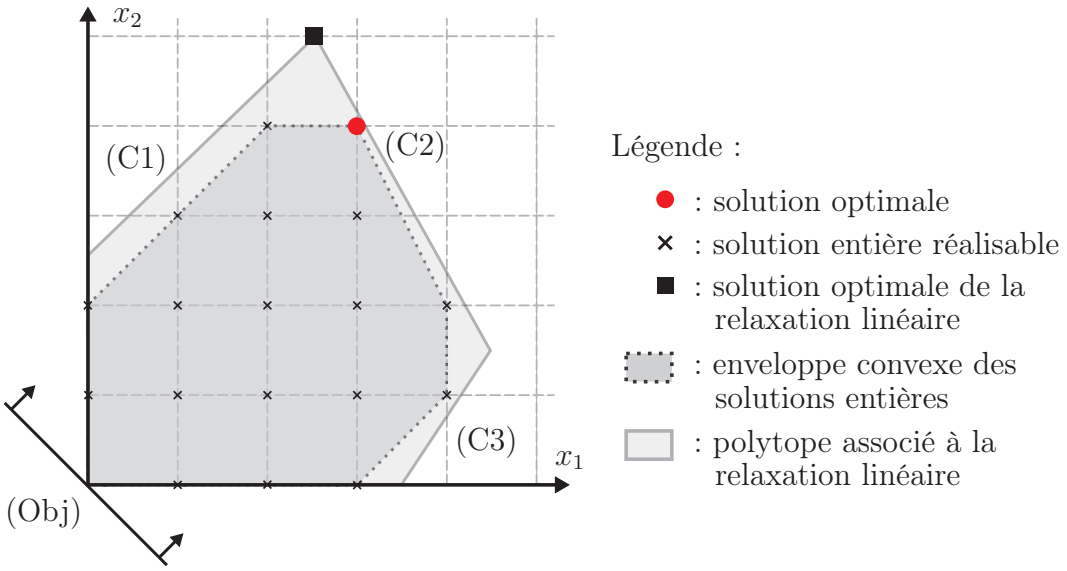
Le problème devient généralement plus complexe lorsqu'il est imposé qu'une partie ou que la totalité des variables soient entières. Afin d'illustrer les différentes notions présentées, nous considérons un problème (P_{ex}) et une formulation (F_{ex}) de ce dernier comportant deux variables entières x_1 et x_2 ainsi que trois contraintes (C_1), (C_2) et (C_3) :

$$(F_{ex}) \begin{cases} \text{maximiser} & x_1 + x_2 & & & (Obj) \\ \text{tel que} & -2x_1 + 2x_2 & \leq & 5 & (C1) \\ & 14x_1 + 8x_2 & \leq & 75 & (C2) \\ & 6x_1 - 4x_2 & \leq & 21 & (C3) \\ & x_1, x_2 \in \mathbb{Z} & & & \end{cases}$$

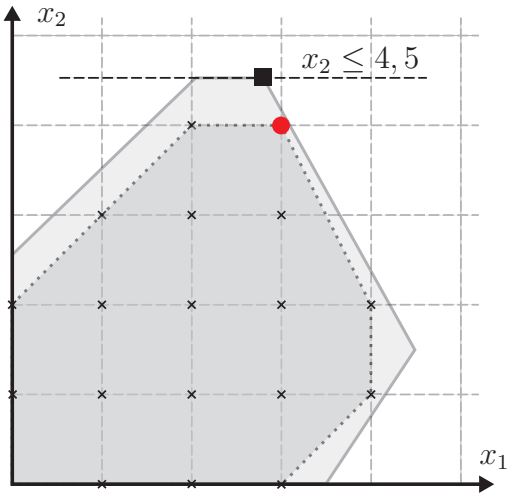
Nous pouvons, entre autres, observer en figure 3.4a une représentation des trois contraintes, de la fonction objectif (qui augmente dans le sens des flèches), des solutions réalisables (×) et de la solution optimale (●) de cette formulation.

Afin de résoudre de manière exacte un programme linéaire en nombres entiers (P), la majorité des techniques existantes reposent sur la notion de *relaxation linéaire*. La relaxation linéaire de (P) est le problème obtenu en relaxant les conditions d'intégrité (*i.e.* : en remplaçant $x \in \mathbb{Z}^n$ par $x \in \mathbb{R}^n$). Comme nous pouvons le voir dans l'exemple en figure 3.4a, la solution optimale de la relaxation linéaire (■) n'est généralement pas entière (dans le cas contraire, le problème serait résolu). Cependant, cette dernière fournit une borne sur la solution entière optimale (borne inférieure dans le cas d'une minimisation et borne supérieure lors d'une maximisation) qui constitue une information précieuse pour les techniques classiques de résolution listées ci-dessous :

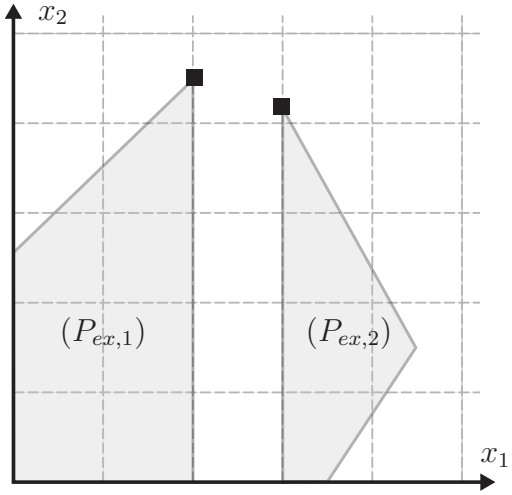
- les *méthodes de plans coupants* ajoutent itérativement des contraintes (appelées *coupes*) au problème afin d'améliorer la solution de sa relaxation linéaire, jusqu'à ce qu'une solution entière soit obtenue. Dans notre exemple, l'ajout de la coupe $x_2 \leq 4,5$ améliorerait la relaxation comme le représente la figure 3.4b. Afin que le problème ne soit pas altéré, il est nécessaire que les coupes ajoutées soient compatibles avec l'ensemble des solutions entières réalisables du problème (*e.g.* : dans notre exemple la coupe $x_2 \leq 2$ n'est pas compatible car elle exclurait des solutions entières). Les méthodes de ce type ont généralement des problèmes de convergence notamment dûs à la difficulté d'identifier à chaque itération une ou plusieurs coupes permettant une amélioration significative de la relaxation.
- les méthodes de type *branch-and-bound* utilisent une représentation sous la forme d'une arborescence dont la racine correspond au problème initial et les autres sommets à des sous-problèmes de ce dernier. Ces méthodes vont successivement diviser en 2 l'espace de recherche (créant ainsi deux branches). A un sommet donné (*i.e.* : à un sous-problème donné) la division est effectuée selon une des composantes fractionnaires de la solution de la relaxation linéaire. Dans notre exemple, la relaxation linéaire fournit la solution $(3,5; 5)$. Puisque la première composante est fractionnaire, elle peut être utilisée pour créer deux sous-problèmes correspondant tous deux au problème initial auquel a été ajouté une des inégalités suivantes $x_2 \leq 3$ ou $x_2 \geq 4$ (voir découpage en figure 3.4c et arborescence en figure 3.4d). Le processus peut ensuite être répété dans chacun des sous-problèmes jusqu'à ce que la solution optimale soit obtenue.
- les méthodes de type *branch-and-cut* combinent les deux précédentes approches. A chaque nœud de l'arborescence, des coupes pourront être utilisées afin d'améliorer la relaxation linéaire.
- les méthodes de *génération de colonnes* permettent la résolution de formulation mathématiques comportant un grand nombre de variables (*i.e.* : de colonnes). Elles sont basées sur le fait que la solution optimale d'un problème ne contiendra, en général, qu'un faible nombre de valeurs non nulles. Dans ces approches, un



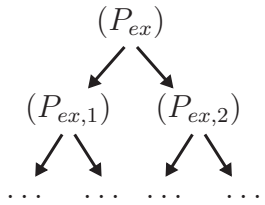
(a) Représentation du problème initiale et de sa relaxation.



(b) Effet de l'ajout de la coupe $x_2 \leq 4,5$ sur le polytope et la valeur optimale de la relaxation linéaire.



(c) Première division en deux sous-problèmes $(P_{ex,1})$ et $(P_{ex,2})$ réalisée dans le cadre d'un algorithme de branch-and-bound.



(d) Premiers niveaux d'une arborescence obtenue avec une méthode de type branch-and-bound.

FIGURE *3.4 – Représentations de l'exemple correspondant au problème (P_{ex}).*

ensemble de variables V_1 de petite taille est initialement considéré et le problème est, tout d'abord, résolu en ne considérant que les variables de V_1. La solution obtenue est ensuite utilisée afin d'ajouter une ou plusieurs variables à V_1. Le choix

de la meilleure variable à ajouter nécessite, lui aussi, la résolution d'un problème d'optimisation. Le problème initial est appelé *problème maître* tandis que celui permettant de sélectionner les variables est nommé *sous-problème* (ou *problème esclave*).
- les méthodes de *branch-and-price* combinent l'approche *branch and bound* et celle de la génération de colonnes.
- les méthodes de *branch-and-price-and-cut* sont des méthodes de *branch-and-price* dans lesquelles des coupes sont susceptibles d'être ajoutées à chaque sommet de l'arborescence.

Lorsque des méthodes utilisant des coupes sont considérées, il semble alors naturel de chercher à obtenir des coupes permettant de s'approcher le plus possible de l'enveloppe convexe des points entiers. Dans notre exemple, au lieu d'ajouter la coupe $x_2 \leq 4,5$, il serait plus efficace d'ajouter $x_2 \leq 4$. Nous présentons maintenant des définitions permettant de formaliser cette notion intuitive de proximité entre une coupe et l'enveloppe convexe des points entiers.

Un *polytope* P est l'enveloppe convexe d'un nombre fini de points de $\mathbb{R}^n$. L'exemple en figure 3.4a, représente deux polytopes, celui associé aux solutions entières du problème ( ) et celui correspondant aux solutions de la relaxation linéaire ( ). Une inégalité $\alpha^T x \leq \alpha_0$ est dite *valide* pour P, si P est inclus dans $\{x \in P \mid \alpha^T x \leq \alpha_0\}$. Si $\alpha^T x \leq \alpha_0$ est une inégalité valide de P, $F_\alpha = \{x \in P \mid \alpha^T x = \alpha_0\}$ est la *face* induite par cette inégalité. La dimension d'un ensemble E (notée $dim(E)$) est le nombre maximum de points affinements indépendants de E moins 1. Une *facette* de P est une face F_α non vide de ce polyèdre de dimension égale à $dim(P)-1$. De manière équivalente, une facette de P est une face non vide de P qui n'est incluse dans aucune autre face de P. Dans l'exemple en figure 3.4, l'inégalité $x_2 \leq 4$ définit une face du polyèdre entier de dimension 1, le polyèdre étant de dimension 2, cette inégalité définit une facette. Le problème consistant à trouver une ou plusieurs inégalités séparant la solution de la relaxation linéaire de l'enveloppe convexe des points entiers est appelé *problème de séparation*. Les algorithmes développés pour résoudre ce problème sont nommés *algorithmes de séparation*.

Ainsi, les inégalités définissant des facettes sont celles qui permettent de se rapprocher au plus prêt de l'enveloppe convexe des points entiers d'un problème. C'est la raison pour laquelle, un grand intérêt est porté à l'identification de familles d'inégalités de ce type dans les travaux que nous présentons par la suite.

Une liste non exhaustive de travaux basés sur une approche polyèdrale dans le cadre du partitionnement de graphes est présentée en table 3.1. Nous identifions, ainsi, quatre familles de variantes du problème en fonction du fait que le poids des parties et/ou leur nombre soit contraint ou pas. Dans la suite, une sous-section est consacrée à chacune de ces familles.

3.2.1 Partitionnement non contraint

Dans le cas général du problème de partitionnement de graphes, aucune contrainte n'est imposée sur le poids ou le nombre de parties.

Les premiers à avoir étudié ce problème sont Grötschel et Wakabayashi [49, 50, 50]. Ils

Variante du problème	**Référence**	**Contrainte sur le poids des parties**	**Contrainte sur le nombre de parties**
Partitionnement non contraint (sous-section 3.2.1)	Grötschel et Wakabayashi [49, 51, 50]	-	-
	Bandelt *et al.* [13]	-	-
	Oosten *et al.* [101, 102]	-	-
Poids des parties contraint (sous section 3.2.2)	Johnson *et al.* [69]	$[F_L, F_U]$	-
	Ferreira *et al.* [38, 39]	$\leq F_U$	$\in \{2, \ldots, K\}$
	Mehrotra et Trick [92]	$\leq F_U$	-
	Sørensen [115, 116]	$\leq F_U$	-
	Ji et Mitchell [66, 67]	$\geq F_L$	-
	Labbé et Özsoy [81]	$\in [F_L, F_U]$	-
Nombre de sommets par partie fixé (sous-section 3.2.3)	Conforti *et al.* [27, 28]	$\lfloor n \backslash 2 \rfloor$ et $\lceil n \backslash 2 \rceil$	$= 2$
	Mitchell [93, 94]	$n \backslash K$	$= K$
	Lisser et Rendl [87]	$n \backslash K$	$= K$
Nombre de parties contraint (sous-section 3.2.4)	Deza *et al.* [31]	-	$\leq K$ ou $\geq K$
	Chopra et Rao [24]	-	$\leq K$ ou $\geq K$
	Chopra et Rao [25]	-	$\in [3, K]$
	Kaibel *et al.* [70]	-	$\leq K$
	Fan et Pardalos [36]	-	$= K$

TABLE *3.1 – Liste non exhaustive d'articles traitant de la résolution exacte du problème de partitionnement graphes. Pour chaque article sont précisées les contraintes spécifiques à la variante de problème considérée par les auteurs. Les variables F_U, F_L et K correspondent à des entiers compris dans l'intervalle* $\{1, \ldots, n\}$.

considèrent un graphe complet $G = (V, E)$ et définissent le *clique partitioning problem*. Dans leurs travaux, le problème est formulé en associant à chaque arête (i, j) de E une variable discrète $x_{i,j}$, de telle sorte que $x_{i,j}$ vaut 1 si i et j sont dans la même partie et 0 sinon. Afin d'assurer la cohérence des parties obtenues, la famille d'inégalités suivante (appelées *inégalités triangulaires*) est considérée :

$$x_{i,j} + x_{i,k} - x_{j,k} \leq 1 \ \forall i \in V \, \forall j, k \in V \backslash \{i\}, \ j < k.$$

Ces inégalités permettent d'assurer que si le sommet i est dans la même partie que les sommets j et k (*i.e.* : $x_{i,j} + x_{i,k} = 2$) alors ces deux sommets doivent eux aussi être regroupés ensemble (*i.e.* : $x_{j,k} = 1$).

Grötschel et Wakabayashi introduisent différentes familles d'inégalités définissant des facettes, notamment les inégalités de 2-partitions et 2-*chorded cycles* que nous considéreront au chapitre suivant dans le cadre du problème de K-partitionnement. De plus, ils introduisent un algorithme de plans coupants et des algorithmes de séparation spécifiques aux familles d'inégalités qu'ils ont étudiés. Les expériences ont montré que la valeur de la relaxation linéaire de la formulation considérée était très bonne. Des graphes comportant jusqu'à 158 sommets ont put être partitionnés de manière optimale en des

temps de l'ordre de quelques minutes.

Oosten *et al.* [101, 102] considèrent eux aussi le problème de partitionnement en cliques mais ils s'intéressent à des graphes qui ne sont pas nécessairement complets. Afin de prendre en compte cet aspect, la famille de contraintes suivante est considérée :

$$x_{i,j} + x_{i,k} \leq 1 \; \forall i, j, k \in V, \; ij, ik \in E, \; jk \notin E.$$

Ainsi, le sommet i ne pourra pas être dans la même partie que deux sommets j et k s'il n'existe pas d'arête reliant ces derniers. Dans ces articles, les auteurs identifient des nouvelles familles d'inégalités définissant des facettes en utilisant, notamment, des techniques telles que le *lifting* (généralisation d'inégalités définissant des facettes d'un polyèdre de plus petite dimension) ou le *patching* (combinaison d'inégalités connues définissant des facettes).

3.2.2 Poids des parties contraint

Certaines applications telles que la répartition de charge sur des processeurs ou la gestion du personnel nécessitent que les sommets du graphe soient regroupés en parties équilibrées. Ainsi, différentes formulation mathématiques contraignant le poids de chacune des parties ont été considérées.

Dans le cas le plus simple, le poids d'une partie correspond au nombre de sommets qu'il contient et ce dernier doit être compris dans un intervalle $[F_L, F_U]$ tel que $1 \leq F_L \leq F_U \leq n$. Lorsqu'une borne inférieure n'est pas souhaitée, F_L peut être fixé à 1. De manière similaire, F_U peut être fixé à n. Les formulations utilisant des variables d'arêtes $x_{i,j}$ prennent cet aspect en compte en contraignant le nombre de voisins de chaque sommet dans une partition :

$$F_L - 1 \leq \sum_{j \in \delta(i)} x_{i,j} \leq F_U - 1 \; \forall i \in V,$$

où $\delta(i)$ correspond à l'ensemble des sommets j tels que $ij \in E$. Lorsqu'au plus K parties sont souhaitées, des variables x_i^k associant un sommet $i \in V$ à une partie $k \in \{1, \ldots, K\}$ sont définies de la façon suivante :

$$x_i^k = \begin{cases} 1 & \text{si } i \text{ se trouve dans la partie } k \\ 0 & \text{sinon} \end{cases}.$$

Afin d'assurer que chaque sommet ne soit assigné qu'à une unique partie, les formulations basées sur ce type de variables utilisent les contraintes :

$$\sum_{k=1}^{K} x_i^k = 1 \; \forall i \in V. \tag{3.1}$$

La restriction de la taille des parties peut alors simplement être prise en compte grâce aux K contraintes

$$F_L \leq \sum_{i \in V} x_i^k \leq F_U \; \forall k \in \{1, \ldots, K\}. \tag{3.2}$$

Johnson *et al.* [69] sont les premiers à avoir pris en compte une contrainte sur le poids des parties dans un problème qu'ils nomment *min-cut clustering*. Le contexte de leur application est la construction d'un compilateur dans lequel un ensemble de modules $i \in V$ de poids w_i doivent être partitionnés afin de minimiser les communications entre les différentes parties. Les bornes imposées sur le poids de chaque partie correspondent ici à des limitations physiques des composants utilisés. La formulation proposée utilise, en plus des variables x_i^k (qui lient un sommet à une partie), un ensemble de variables z_e^k, associant une arête $e \in E$ à une partie. Ils présentent différentes familles d'inégalités valides et proposent une approche par génération de colonnes permettant la prise en compte du nombre élevé de variables de leur formulation.

Le *capacitated graph partitioning problem*, introduit par Ferreira *et al.* [38, 39] considère, lui aussi, une borne supérieure sur le poids des parties. En plus de cela, le nombre de parties obtenu doit être compris entre 2 et un entier K. Les auteurs introduisent diverses familles d'inégalités valides et présentent plusieurs heuristiques de séparations de ces dernières qui sont utilisées dans un *branch-and-cut*.

Mehrotra et Trick [92] utilisent, quant à eux, une approche originale ou à chaque sous-partie $P \subset V$ est associée une variable x_P vaut 1 si la partie P figure dans la solution et 0 sinon. La formulation comporte une unique famille d'inégalité assurant que chaque sommet $i \in V$ n'apparaîsse que dans une unique partie

$$\sum_{P \subset V,\, i \in P} x_P = 1.$$

Les contraintes sur la taille des parties sont simplement prises en compte en ne considérant que les variables associées à des parties de taille adéquate. Le nombre de variables étant exponentiel, un algorithme de *branch-and-price* est utilisé pour la résolution.

Le *clique partitioning problem with minimum size requirement*, qui impose nombre minimal de sommet par partie, a été introduit par Ji et Mitchell [66, 67]. Dans [66], les auteurs proposent de nouvelles familles d'inégalités, présentent des conditions sous lesquelles elles définissent des facettes et utilisent un algorithme de *branch-and-cut* pour la résolution. Le second article [67] présente un algorithme de *branch-and-price-and-cut*.

Sørensen introduit le *simple graph partitioning problem* [115, 116] dans lequel une partie ne peut contenir plus de b sommets. Une très grande famille d'inégalitées appelée *b-tree* est définie dans [115]. L'idée à l'origine de ces inégalités est qu'un arbre contenant $b+1$ sommets possède nécessairement au moins une arête ne figurant pas dans la solution optimale (puisque chaque partie contient au maximum b sommets). Dans [116], d'autres familles d'inégalités sont étudiées.

A notre connaissance, le travail le plus récent réalisé sur cette variante du problème a été accompli par de Labbé et Özsoy [81]. Le problème considéré, appelé *size-constrained graph partitioning problem*, généralise plusieurs problèmes précédemment mentionnés puisqu'aucune contrainte n'est imposée sur les bornes F_L et F_U. Par exemple, le *clique partitioning problem* de Grötschel et Wakabayashi correspond au cas où $F_L = 1$ et $F_U = n$. Les auteurs étudient diverses familles d'inégalités et déterminent les conditions sous lesquelles elles définissent des facettes pour les différentes valeurs de F_L et F_U. Les résultats obtenus ont par la suite été utilisés dans un *branch-and-cut* afin de résoudre un problème de conception de réseau [22].

3.2.3 Nombre de sommets par partie fixé

Dans cette partie nous traitons des approches dans lesquelles le nombre de sommets par partie est contraint à être le même (à un sommet prêt).

Conforti et Rao [27, 28] introduisent le problème nommé *equipartition problem* consistant à minimiser le poids de la coupe séparant un graphe $G = (V, E)$ en deux parties de tailles respectives $\lfloor \frac{n}{2} \rfloor$ et $\lfloor \frac{n}{2} \rfloor$. Les auteurs s'intéressent à ce problème car il est équivalent à un problème quadratique apparaîssant dans le cadre de problématiques d'étude de la magnétisation d'alliage métallique. Des inégalités définissant des facettes y sont caractérisées et un algorithme de *branch-and-cut* les utilisant a par la suite été présenté [19].

Le *k-way equipartition problem* survient lorsque l'on souhaite obtenir un nombre de parties variable k contenant chacune $\frac{|V|}{k}$ sommets ($|V|$ est supposé être un multiple de k). Mitchell et John étudient notamment ce problème dans le cadre de la répartition d'équipes sportives en ligues afin de minimiser les distances parcourues lors des rencontres [93, 94]. Une étude polyèdrale est réalisée et un algorithme de *branch-and-cut* est, cette fois encore, utilisé pour la résolution.

3.2.4 Nombre de parties contraint

Nous considérons ici les variantes de partitionnement de graphe dans lesquelles le nombre de parties est contraint mais pas leur poids.

De nombreux travaux ont été réalisés pour le cas où le graphe doit être partitionné en deux. Par exemple, Barahona et Mahjoub étudient le polytope des coupes [14] qui correspond à l'enveloppe convexe des vecteurs d'incidence des coupes d'un graphe. Ils étudient la structure faciale de ce polytope et caractérisent l'adjacence de ses sommets.

Lorsque le nombre de parties à obtenir K est fixé et que le poids des arêtes du graphes sont tous positifs, Goldschmidt *et al.* [48] ont défini un algorithme polynomial permettant de trouver la K-coupe de poids minimal. La complexité de ce dernier est $\mathcal{O}(n^{\frac{K^2}{2}-\frac{3K}{2}+4}T(n,m))$, où $T(n,m)$ correspond au temps nécessaire à l'obtention d'un (s,t)-coupe dans un graphe comprenant n sommets et m arêtes. Bien que polynomial, cette complexité augmente rapidement avec la valeur de K. Plus récemment, des approches basées sur les fonctions sous-modulaires on aussi été utilisées pour résoudre efficacement ce problème [107, 134].

Chopra et Rao considèrent le polytope des k-partitions qui correspond à l'enveloppe convexe des vecteurs d'incidences des partitions comportant un nombre de parties r tel que $r \in \{3, \ldots, k\}$ [25]. Il utilisent des techniques de lifting afin de définir de nouvelles inégalités définissant des facettes.

Deza *et al.* [31] étudient les deux polytopes associés aux vecteurs d'incidence des multicoupes divisant un graphe en au plus K parties et en au moins K parties. Les auteurs présentent une large famille d'inégalités appelée *clique-web inequalities* et caractérisent les cas où ces dernières définissent des facettes. De plus, ils utilisent cette famille comme base pour générer de nouvelles inégalités définissant des facettes et présentent un algorithme polynomiale permettant de résoudre le problème de séparation d'une partie des inégalités *clique-web*. Chopra et Rao réalisent, eux aussi, une étude de

ces deux polytopes [24].

Dans les travaux de Fan et Pardalos [36], les partitions recherchées contiennent exactement K parties non vides. Les auteurs considèrent une formulation utilisant deux familles de variables précédemment présentées :

– $x_{i,j}$ $\forall (ij) \in E$ vaut 1 si i et j sont dans la même partie et à 0 sinon ;
– x_i^k $\forall i \in V \, \forall k \in \{1, \ldots, K\}$ qui vaut 1 si i est dans la partie k et à 0 sinon.

Le lien entre ces deux variables est effectué par la contrainte

$$x_{i,j} = \sum_{k=1}^{K} x_i^k x_j^k \; \forall ij \in E, \tag{3.3}$$

qui assurent que $x_{i,j}$ vaut 1 s'il existe une partie k contenant à la fois i et j. Chacune de ces contraintes peut ensuite être linéarisée et remplacée par les deux suivantes :

$$x_{i,j} - x_i^k + x_j^k \leq 1 \; \forall ij \in E \tag{3.4}$$

et

$$x_{i,j} + x_i^k - x_j^k \leq 1 \; \forall ij \in E. \tag{3.5}$$

En prenant F_L non nulle, les auteurs assurent via les équations 3.2 que le nombre de parties obtenues est exactement égal à K.

Cette formulation n'est cependant pas exempte de défauts puisque les variables x_i^k introduisent une forte symétrie dans le modèle. En effet, chaque solution réalisable de cette modélisation peut être utilisée pour obtenir $K!$ solutions équivalentes en permutant les indices des parties. Par exemple, si trois sommets et deux parties sont considérés, une solution dans laquelle les variables x_1^2, x_2^1 et x_3^1 valent 1 (qui correspond au fait de regrouper les sommets 2 et 3 ensemble) est équivalente à une solution où les variables x_1^1, x_2^2 et x_3^2 valent 1. Dans le premier cas, les sommets 2 et 3 sont dans la partie d'indice 1, tandis que dans le second, ils se trouvent dans la partie d'indice 2. Ces deux solutions correspondent à la même partition et sont donc équivalentes.

Les symétries peuvent grandement détériorer les performances d'algorithmes de résolution (*e.g.* : dans le cas d'un *branch-and-cut*, elles pourront entraîner l'exploration de nombreuses branches identiques). C'est la raison pour laquelle différents auteurs ont étudié comment casser la symétrie dans les formulations utilisant les variables x_i^k. Kaibel *et al.* [70] considèrent le cas dans lequel le poids des parties n'est pas contraint et le nombre de parties doit être inférieur ou égal à K. Afin de réduire le nombre de symétries, ils imposent que chaque sommet i se trouve dans une partie dont l'indice est inférieur ou égal à i :

$$x_i^j = 0 \; \forall i \in V \, \forall j \in \{i+1, \ldots, n\}. \tag{3.6}$$

Les symétries restantes ne sont pas prises en compte directement dans la formulation, mais progressivement au cours de la résolution par *branch-and-cut*. Pour ce faire, les auteurs appliquent à différents nœuds de l'arborescence un algorithme permettant d'identifier les branches correspondant à des solutions symétriques.

Plus récemment, Bonami *et al.* [18] ont montré qu'il était possible de retirer toute symétrie dans une formulation utilisant les variables x_i^k $\forall i \in V$ $\forall k \in \{1, \ldots, n\}$ en imposant que la partie d'indice i soit non vide si et seulement si le sommet de plus petit

indice qu'elle contient est i. Ceci peut être réalisé en ajoutant les contraintes 3.6 ainsi que les suivantes :

$$x_i^j \leq x_j^j \; \forall j \in V \; \forall i \in \{j+1, \ldots, n\}. \tag{3.7}$$

Ces dernières assurent que si le sommet j n'est pas dans la partie d'indice j, alors aucun sommet i plus grand que j ne peut être dans cette partie.

3.3 Discussion

Comme nous l'avons vu, la littérature concernant le problème de partitionnement de graphes est riche et variée. Elle comporte aussi bien des approches heuristiques qu'exactes chacune possédant ses avantages. Dans l'outil d'aide à la décision que nous définissons, nous souhaitons permettre à l'utilisateur de choisir entre ces deux approches en fonction du type de solution (rapide ou optimale) qu'il souhaite obtenir.

Comme nous l'avons présenté en introduction de ce chapitre, nous souhaitons utiliser des méthodes permettant à l'utilisateur de notre outil de fixer le nombre de parties K sans imposer de contraintes au nombre de sommets par parties. Ce problème peut apparaître dans la littérature sous différents noms (*e.g.* : K-coupes). Dans le cadre de ce document, nous utilisons le terme de K-partitionnement pour y faire référence.

De nombreuses méthodes heuristiques le permettent. Cependant, leur description section 3.1 ne peut prétendre suffire à en permettre une comparaison raisonnable. Les méthodes hiérarchiques ont pour avantage de fournir un ensemble de partitions permettant à l'utilisateur de parcourir des solutions correspondant à différentes valeurs de K sans nécessiter de calculs supplémentaires. Les approches de type partition, quant à elles, sont susceptibles de fournir de meilleures solutions grâce à des améliorations successives d'une partition initiale. Ainsi, nous avons décidé de sélectionner un échantillon représentatif d'approches des deux types et de comparer leurs performances (détail de l'expérience en section 6.3). Les algorithmes considérés sont les suivants Single-link (hiérarchique, basé sur la plus petite distance), ROCK (hiérarchique, basé sur le voisinage), CHAMELEON (hiérarchique basé sur l'interconnectivité), propagation d'affinité (type partition, utilisant l'envoi de messages) et trois méthodes spectrales utilisant les différents laplaciens présentés (type partition, basé sur la réduction de dimensions).

Concernant les approches exactes, l'algorithme proposé par Goldschmit *et al.* [48] ainsi que des algorithmes basés sur les fonctions sous-modulaires [107, 134] permettent de minimiser en temps polynomial le poids de la K-coupe (ce qui équivaut à maximiser le poids de la K-partition) dans le cas où l'ensemble des arêtes possède un poids positif. Ces approches correspondent bien au problème de K-partitionnement, cependant, comme nous le voyons en section 6, les méthodes utilisées pour calculer la proximité entre les motifs fournit des valeurs aussi bien positives que négatives. Les arcs de notre graphe ne seront donc pas tous positifs. La seule approche exacte permettant de fixer le nombre de parties K tout en laissant libre le poids des parties est celle proposée par Fan et Pardalos [36]. Bien qu'elle comporte beaucoup de symétrie, les inégalités proposées par Bonami *et al.* [18] peuvent être utilisée afin de s'en affranchir. Dans le chapitre suivant, nous présentons deux formulations alternatives du problème de K-partitionnement.

4 | Approche polyèdrale pour le problème de K-partitionnement

Afin d'offrir l'opportunité à l'utilisateur de notre application de partitionner les motifs extraits durant la première étape de notre méthodologie en un nombre de parties fixé, nous étudions le problème de K-partitionnement. Nous avons constaté dans le chapitre précédent que de nombreuses méthodes heuristiques permettaient de traiter ce problème. Cependant, comme nous l'avons mentionné en introduction, nous préférons nous orienter vers des méthodes de résolution exactes afin de pouvoir évaluer objectivement la qualité de notre modélisation. C'est la raison pour laquelle nous définissons et étudions deux formulations originales du problème de $K-$partitionnement, sous forme de programmes d'optimisation linéaire en nombres entiers. Ces formulations sont des alternatives à celle proposée par Fan et Pardalos [36] décrite précédemment.

Après quelques définitions en section 4.1, les formulations sont présentées et comparées en section 4.2. En section 4.3 nous étudions la dimension de l'enveloppe convexe du polyèdre entier associé à ces formulations et caractérisons les conditions sous lesquelles des familles d'inégalités en définissent des facettes. Nous nous intéressons notamment aux trois familles d'inégalités non triviales suivantes : inégalités de 2-partitions, inégalités 2-*chorded cycle* et inégalités de clique généralisées.

4.1 Définitions et notations

Dans cette section, nous rappelons quelques notations utilisées dans le chapitre précédent et en définissons des nouvelles.

Tout au long de ce chapitre, nous considérons un graphe $G = (V = \{m_1, \ldots, m_n\}, E)$ complet dont chaque sommet correspond à un des motifs récurrents précédemment extraits. Soient m_i et m_j deux sommets de V, m_i est dit *plus petit* (respectivement *plus grand*) que m_j si i est plus petit (respectivement plus grand) que j. Dans la suite, les sommets de V sont désignés par leur indice (*e.g.* : i fait référence au sommet m_i).

Soient V_1 et V_2 deux sous-ensembles de V et soit x un vecteur de $\mathbb{R}^{|E|}$ dont les composantes sont indicées en fonction de E (*i.e.* : $x^T = (x_{1,2}, x_{1,3}, \ldots, x_{n-1,n})^T$). Nous utilisons les expressions $x(V_1)$ et $x(V_1, V_2)$ pour dénoter respectivement $\sum_{i \in V_1, j \in V_1 \setminus \{i\}} x_{i,j}$ et $\sum_{i \in V_1} \sum_{j \in V_2} x_{i,j}$. Étant donné un ensemble $E_1 \subset E$, le terme $x(E_1)$ représente $\sum_{e \in E_1} x_e$.

Soit $\pi = \{P_1, \ldots, P_K\}$ une K-partition. Pour tout $i \in \{1, \ldots, K\}$, le sommet le plus petit de P_i est noté r_i. Le terme r'_i désigne le deuxième sommet le plus petit de P_i (*i.e.* : $r'_i = \min_{j \in P_i \setminus \{r_i\}} j$).

Soit B une expression booléenne. Nous définissons la *fonction indicatrice* $\mathbb{1}\{B\}$ qui vaut 1 si B est vrai et 0 sinon.

Transformations

Nous introduisons ici une notion originale, appelée *transformation*, permettant de simplifier la présentation et la compréhension des démonstrations de ce chapitre.

Afin d'étudier la dimension d'un polytope P dans un espace de dimension n (ou d'une face de P), nous identifions le nombre d'hyperplans affinement indépendants qui le contiennent. S'il en existe exactement n_1, alors la dimension de P est $n-n_1$. En particulier, pour montrer que P est de dimension pleine, nous prouvons qu'il n'est contenu dans aucun hyperplan. Ceci est réalisé en considérant un hyperplan $H = \{x \in \mathbb{R}^n \mid \alpha^T x = \alpha_0\}$ contenant P et en prouvant que l'ensemble des coefficients de H (*i.e.* : le coefficient α_0 et les n coefficients de $\alpha = (\alpha_1, \ldots, \alpha_n)^T$) sont nulles.

Pour ce faire, nous devrons découvrir des relations entre les coefficients de H. Nous obtiendrons ces relations en considérant différents points de $\mathbb{R}^n$ figurant dans P. Par exemple, supposons que n soit égal à 3 et que P contienne les points $x^a = (1,0,1)^T$ et $x^b = (1,1,0)^T$. Puisque P est inclus dans H, ces deux points doivent nécessairement vérifier la relation $\alpha^T x = \alpha_0$. Nous obtenons donc pour x^a, l'équation $\alpha_1 + \alpha_3 = \alpha_0$ et pour x^b : $\alpha_1 + \alpha_2 = \alpha_0$; ce qui permet d'obtenir :

$$\alpha_3 = \alpha_2. \tag{4.1}$$

Le membre gauche de cette équation correspond aux composantes non nulles de x^a qui sont nuls dans x^b et le membre droit aux composantes non nulles de x^b qui sont nulles dans x^a.

Ainsi, nous aurons régulièrement à considérer deux points de l'espace et à identifier leurs composantes qui diffèrent. Pour simplifier cette identification, nous définissons la notion de transformation permettant de représenter le passage d'une partition à une autre (*i.e.* : d'un point à un autre de notre espace de recherche) en modifiant uniquement deux de leurs parties.

Soient V_1 et V_2 deux sous-ensembles non vides disjoints de V et soit $R \subseteq V_1 \cup V_2$ avec $R_1 = R \cap V_1$ et $R_2 = R \cap V_2$. Nous notons $R = R_1 \cup R_2$. Nous appelons *transformation* la fonction $\mathcal{T} : \{V_1, V_2, R\} \mapsto \{V_1', V_2'\}$ telle que V_1' et V_2' correspondent aux ensembles V_1 et V_2 après échange des sommets contenus dans R (*i.e.* : $V_1' = (V_1 \backslash R_1) \cup R_2$ et $V_2' = (V_2 \backslash R_2) \cup R_1$). La transformation $\mathcal{T}$ est représentée en figure 4.1.

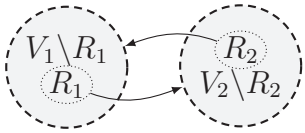


FIGURE *4.1 – Représentation de* $\mathcal{T}(V_1, V_2, R)$ *avec* $R_1 = R \cap V_1$ *et* $R_2 = R \cap V_2$.

Cette transformation est dite *valide* pour un polytope P, si :
- il existe une K-partition π, dont deux des parties sont V_1 et V_2 et dont le vecteur indicateur est inclus dans P ;
- la partition π' obtenue à partir de π en remplaçant V_1 et V_2 par V_1' et V_2', est elle-même incluse dans P.

Lorsque, pour une transformation $\mathcal{T}$ donnée, des partitions π et π' auront été explicitement déterminées, cette dernière pourra être représentée par : $\pi \rightarrow \pi'$.

L'utilisation de transformations, va nous permettre de représenter intuitivement le lien entre des paires de points de notre espace de recherche et d'en déduire aisément une relation sur les coefficients d'un hyperplan les contenant.

4.2 Formulations

Étant donné un graphe complet $G = (V, E)$, associant à chacune de ses arêtes $ij \in E$ un poids $w_{i,j}$, le problème de $K-$partitionnement consiste à partitionner les sommets de V en K parties non vides tout en minimisant le poids de la K-coupe. Ce problème est un cas particulier du problème de partitionnement en cliques présenté dans le chapitre précédent.

C'est la raison pour laquelle, nous utilisons une formulation basée sur celle introduite par Grötschell et Wakabayashi [51] associant à chaque arête (i, j) de E, une variable binaire $x_{i,j}$ telle que :

$$x_{i,j} = \begin{cases} 1 & \text{si i et j sont dans la même partie} \\ 0 & \text{sinon} \end{cases},$$

et comprenant les familles d'inégalités suivantes :

$$x_{i,j} \in \{0, 1\} \; \forall ij \in E, \tag{4.2}$$

et

$$x_{i,j} + x_{i,k} - x_{j,k} \leq 1 \; \forall i \in V \, \forall j, k \in V \backslash \{i\}, \; j < k. \tag{4.3}$$

Les inégalités (4.3), appelées *inégalités triangulaires*, assurent la cohérence des parties. En effet, si le sommet i se trouve dans la même partie que les sommets j et k (*i.e.* : $x_{i,j} + x_{i,k} = 2$), alors ces inégalités spécifient que les sommets j et k se trouvent nécessairement dans la même partie (*i.e.* : $x_{j,k} = 1$). Les solutions entières de $\mathbb{R}^{|E|}$ satisfaisant les équations (4.2) et (4.3) correspondent exactement aux solutions du problème de partitionnement en cliques.

Nous définissons, en section 4.2.1, une formulation (F_1) du problème de K-partitionnement qui est une adaptation originale de la formulation de Grötschell et Wakabayashi. En section 4.2.2, nous montrons qu'une formulation (F_2), étendant (F_1), peut être obtenue. Enfin en section 4.2.3 nous comparons les deux formulations et montrons que (F_2) est meilleure (au sens de la relaxation linéaire) que (F_1).

4.2.1 Formulation (F_1)

Nous présentons une extension de la formulation décrite précédemment permettant de fixer le nombre de parties. Pour ce faire, nous introduisons pour chaque sommet i de V, une variable x_i telle que

$$x_i = \begin{cases} 1 & \text{si i est le sommet le plus petit de sa partie} \\ 0 & \text{sinon} \end{cases}.$$

Ce type de variables a été utilisé pour la première fois par Campelo *et al.* [21] dans des problématiques de coloration de graphes. Elles ont pour avantage de permettre de fixer le nombre de parties souhaitées, sans pour autant ajouter de symétrie à la formulation.

Dans la suite, un sommet i est appelé *représentant* si la variable x_i vaut 1. Afin d'assurer que le sommet d'une partie dont l'indice est le plus petit indice soit un représentant, nous utilisons les contraintes :

$$0 \leq x_i \leq 1 \quad \forall i \in V \tag{4.4}$$

et

$$x_i + \sum_{j=1}^{i-1} x_{i,j} \geq 1 \quad \forall i \in V. \tag{4.5}$$

Ensuite, pour assurer que chaque partie ne contienne pas plus d'un représentant, nous ajoutons les contraintes :

$$x_i + x_{i,j} \leq 1 \quad \forall i \in V,\ \forall j \in \{1, \ldots, i-1\}. \tag{4.6}$$

Enfin, une $K-$partition peut être obtenue en fixant à K le nombre de représentants :

$$\sum_{i=1}^{n} x_i = K. \tag{4.7}$$

Nous avons pour l'instant défini une variable par arête et sommet du graphe. La dimension de l'espace de recherche correspondant est donc $|E| + |V|$. Cependant, il est possible de fixer trois variables, réduisant ainsi d'autant la dimension de l'espace de recherche :

- Le sommet 1 est toujours celui dont l'indice est le plus petit de sa partie, ainsi x_1 vaut toujours 1 ;
- Le second sommet est lui aussi le plus petit de sa partie, sauf s'il est dans la même partie que 1, en conséquence x_2 est égal à $1 - x_{1,2}$;
- En utilisant l'équation (4.7), x_3 peut être fixée à $K - 2 + x_{1,2} - \sum_{i=4}^{n} x_i$ (il n'est donc pas nécessaire que la contrainte (4.7) figure dans la formulation).

Le nombre de variables considérées est maintenant égal à $|E| + |V| - 3$. Afin de simplifier les notations, les variables x_1, x_2 et x_3 – qui sont maintenant des variables artificielles – ne sont pas substituées par leur expression, dans la suite du document. A partir de maintenant, tout vecteur $\alpha \in \mathbb{R}^{|E|+|V|-3}$ est indicé par les sommets de $V \backslash \{1, 2, 3\}$ et les arêtes de E (*i.e.* : $\alpha^T = (\alpha_4, \ldots, \alpha_n, \alpha_{1,2}, \ldots, \alpha_{n-1,n})^T$). De plus, nous associons à α trois composantes supplémentaires α_1, α_2 et α_3 qui valent toujours 0.

Nous obtenons une première formulation du problème de K-partitionnement :

$$(F_1)\begin{cases} \text{minimiser} & \displaystyle\sum_{ij\in E} w_{i,j}x_{i,j} & \\ \text{soumis à} & x_{i,j}+x_{i,k}-x_{j,k}\leq 1 \quad \forall i\in V\,\forall j,k\in V\backslash\{i\},\ j<k & (4.3)\\ & \displaystyle x_i+\sum_{j=1}^{i-1}x_{i,j}\geq 1 \quad \forall i\in V & (4.5)\\ & x_i+x_{i,j}\leq 1 \quad \forall i\in V,\ \forall j\in\{1,\ldots,i-1\} & (4.6)\\ & x_{i,j}\in\{0,1\} \quad \forall ij\in E & (4.2)\\ & x_i\in[0,1] \quad \forall i\in V & (4.4) \end{cases}$$

Il est important de noter qu'il n'est pas nécessaire de contraindre les variables de représentants x_i à être discrètes. En effet, l'intégrité des variables $x_{i,j}$ assure – via les contraintes (4.5) et (4.6) – que les variables x_i valent 0 ou 1.

4.2.2 Formulation (F_2)

Nous proposons maintenant une extension de cette formulation. Dans (F_1), le représentant de chaque partie est fixé grâce aux contraintes (4.5) et (4.6). Ces dernières pourraient, cependant, être remplacées par

$$x_j+\sum_{i=1}^{j-1}x_ix_{i,j}=1 \quad \forall j\in V. \tag{4.8}$$

Les contraintes (4.8), assurent qu'un sommet j est soit un représentant ($x_j=1$) soit qu'il se trouve dans la même partie qu'exactement un sommet – plus petit que lui – qui est un représentant ($x_ix_{i,j}=1$). Sans perte de généralité, dans la suite, i est supposé plus petit que j. Les contraintes (4.8) étant quadratiques, elles nécessitent d'être adaptées afin de pouvoir les utiliser dans une formulation linéaire. En conséquence, nous introduisons pour chaque arête ij de E une variable $\tilde{x}_{i,j}$ telle que

$$\tilde{x}_{i,j}=\begin{cases} 1 & \text{si i est le représentant de j}\\ 0 & \text{sinon}\end{cases}.$$

Les familles de contraintes (4.5) et (4.6) de (F_1) peuvent ainsi être remplacées par

$$x_j+\sum_{i=1}^{j-1}\tilde{x}_{i,j}=1 \quad \forall j\in V \tag{4.9}$$

et

$$0\leq\tilde{x}_{i,j}\leq 1 \quad \forall ij\in E. \tag{4.10}$$

Afin de vérifier que chaque variable $\tilde{x}_{i,j}$ est égale à $x_ix_{i,j}$, nous utilisons une technique de linéarisation classique qui consiste à ajouter les trois familles d'inégalités suivantes :

$$\tilde{x}_{i,j}\leq x_{i,j} \quad \forall ij\in E \tag{4.11}$$

$$\tilde{x}_{i,j}\leq x_i \quad \forall ij\in E,\ i<j \tag{4.12}$$

$$x_{i,j} + x_i - \tilde{x}_{i,j} \leq 1 \quad \forall ij \in E,\, i < j \tag{4.13}$$

Puisque le sommet 1 est toujours le représentant de sa partie, les variables $\tilde{x}_{1,j}$ et $x_{1,j}$ pour tout $j \in V\backslash\{1\}$ sont équivalentes. De manière similaire à x_1, x_2 et x_3, les variables $\tilde{x}_{1,j}$ sont donc artificielles mais ne sont pas substituées dans la suite, afin de simplifier les notations. En retirant ces $|V|-1$ variables, nous obtenons une formulation étendue de (F_1) comportant $2|E|-2$ variables :

$$(F_2)\begin{cases} \text{minimiser} & \displaystyle\sum_{ij\in E} w_{i,j}x_{i,j} & \\ \text{soumis à} & x_{i,j} + x_{i,k} - x_{j,k} \leq 1 \quad \forall i \in V\, \forall j,k \in V\backslash\{i\},\, j<k & (4.3) \\ & \displaystyle x_j + \sum_{i=1}^{j-1} \tilde{x}_{i,j} = 1 \quad \forall j \in V & (4.9) \\ & \tilde{x}_{i,j} \leq x_{i,j} \quad \forall ij \in E & (4.11) \\ & \tilde{x}_{i,j} \leq x_i \quad \forall ij \in E,\, i<j & (4.12) \\ & x_{i,j} + x_i - \tilde{x}_{i,j} \leq 1 \quad \forall ij \in E,\, i<j & (4.13) \\ & x_{i,j} \in \{0,1\} \quad \forall ij \in E & (4.2) \\ & x_i \in [0,1] \quad \forall i \in V & (4.4) \\ & \tilde{x}_{i,j} \in [0,1] \quad \forall ij \in E & (4.10) \end{cases}$$

Tout comme les variables de représentants, il n'est pas nécessaire de contraindre les variables $\tilde{x}_{i,j}$ à être entières.

L'ajout des familles d'inégalités (3.6) et (3.7) permet d'assurer que la partie d'indice i est non vide si et seulement si le sommet de plus petit indice qu'elle contient est i. Cela revient à dire que si la partie i est présente dans une solution, le sommet i en est le représentant. Les variables x_i $i \in V$ utilisées dans les formulations (F_1) et (F_2) jouent donc un rôle identique à celui des variables x_i^i de (F_3) (*i.e.* : pour une K-partition donnée, les variables x_i et x_i^i possèdent la même valeur dans leur formulation respective). Pour cette raison, de manière similaire aux formulations précédentes, les variables x_1^1, x_2^2 et x_3^3 peuvent être retirées de la formulation, ce qui amène à une formulation comportant $|E| + K|V| - 3$ variables.

4.2.3 Comparaison des formulations

La *relaxation linéaire* de (F_1) et (F_2) est la solution optimale de ces formulations dans lesquelles les contraintes d'intégrité (4.2) ont été substituées par

$$0 \leq x_{i,j} \leq 1 \quad \forall ij \in E. \tag{4.14}$$

Soit (R_1) (respectivement (R_2)) le polytope de l'ensemble des solutions réalisables de la relaxation linéaire de (F_1) (respectivement (F_2)). Ainsi,

$$(R_1) = \{x \in \mathbb{R}^{|E|+|V|-3} \mid (4.3) \text{ à } (4.6) \text{ et } (4.14)\}$$

et

$$(R_2) = \{x \in \mathbb{R}^{2|E|-2} \mid (4.3), (4.4) \text{ et } (4.9) \text{ à } (4.14)\}.$$

Des méthodes de résolution de programmes linéaires en nombres entiers telles que les méthodes de plans coupants ou de *branch-and-cut* sont basées sur le fait que leur relaxation linéaire peut être calculée rapidement. Dans les cas où la fonction objectif est à minimiser, la relaxation linéaire fournit une borne inférieure de la solution optimale. Plus le polytope de la relaxation linéaire est petit, meilleure sera la borne. Afin de comparer (R_1), et (R_2), nous considérons

$$proj(R_2) = \{x \in \mathbb{R}^{|E|+|V|-3} \mid \exists \tilde{x} \in \mathbb{R}^{|E|-|V|+1}, \begin{pmatrix} x \\ \tilde{x} \end{pmatrix} \in (R_2)\}.$$

La dimension de l'espace de recherche de (R_1), $proj(R_2)$ est $|E| + |V| - 3$. Les ensembles de solutions entières contenus dans chacun de ces polytopes se correspondent de façon bijective puisque (F_1) et (F_2). Le théorème suivant montre que – dans l'ensemble des cas non triviaux – $proj(R_2)$ est strictement inclus dans (R_1), prouvant ainsi que la formulation (F_2) est meilleure que (F_1).

Théorème 4.2.1 *$proj(R_2) \subset (R_1)$ si $n \geq 4$ et $K \in \{2, \ldots, n-2\}$.*

Démonstration Il est facile de constater que le polyèdre $proj(R_2)$ est inclus dans (R_1). Afin de prouver ce théorème, nous exhibons un point de (R_1) qui ne se trouve pas dans $proj(R_2)$.

Soit $x^1 \in \mathbb{R}^{|E|+|V|-3}$ le vecteur indicateur d'une K-partition $\{P_1, \ldots, P_K\}$ avec $P_1 = \{1, 2, 3\}$. Il est toujours possible de construire une telle partition lorsque $n \geq 4$ et $K \in \{2, \ldots, n-2\}$.

La table 4.1 représente les valeurs des composantes de x^1 relatives aux trois premiers sommets du graphe ainsi que celles d'un deuxième point x^2, identique à x^1 à l'exception des composantes $x_{1,3}$ et $x_{2,3}$ qui valent 0.5. Le point x^2 se trouve dans (R_1) puisqu'il satisfait les contraintes (4.3) à (4.6) et (4.14).

Composantes	x_1	x_2	x_3	$x_{1,2}$	$x_{1,3}$	$x_{2,3}$
Point initial x^1	1	0	0	1	1	1
Point modifié x^2	1	0	0	1	0.5	0.5

TABLE *4.1 – Valeurs des composantes relatives aux sommets* 1, 2 *et* 3 *pour deux vecteurs x^1 et x^2 de $\mathbb{R}^{|E|+|V|-3}$. Le premier correspond à une K-partition $\{P_1, \ldots, P_K\}$ telle que $P_1 = \{1, 2, 3\}$. Le second (qui est une modification de x^1) se trouve dans R_1 mais pas dans $proj(R_2)$.*

Pour que x^2 se trouve aussi dans $proj(R_2)$, il doit nécessairement exister un vecteur $\tilde{x}^2 \in \mathbb{R}^{|E|-2}$ tel que $\begin{pmatrix} x^2 \\ \tilde{x}^2 \end{pmatrix} \in (R_2)$. En particulier, les composantes $\tilde{x}_{1,3}$ et $\tilde{x}_{2,3}$ du vecteur $\tilde{x}^2$ doivent satisfaire les contraintes (4.9) et (4.12) qui imposent respectivement

$$x_3 + \tilde{x}_{1,3} + \tilde{x}_{2,3} = 1 \tag{4.15}$$

et

$$\tilde{x}_{2,3} \leq x_2. \tag{4.16}$$

Nous savons que x_2 et x_3 valent 0. Par conséquent, il en va de même pour $\tilde{x}_{2,3}$ (d'après (4.16)). Ainsi, pour satisfaire l'équation (4.15), $\tilde{x}_{1,3}$ doit valoir 1. Ceci est impossible puisque $\tilde{x}_{1,3}$ est nécessairement égal à $x_{1,3}$ (et ici $x_{1,3}$ vaut 0.5).

□

Afin de comparer numériquement l'amélioration apportée à la relaxation linéaire par (F_1) et (F_2), nous considérons des graphes complets dont le poids des arêtes $ij \in E$ est généré aléatoirement dans l'intervalle $[0, 1000]$. Soit o^* la valeur optimale de la fonction objectif du problème de K-partitionnement d'un de ces graphes et soit o_R la valeur obtenue via une des relaxations. Nous définissons l'*écart relatif* de la relaxation R sur ce graphe comme

$$\frac{|o^* - o_R|}{o^*}.$$

La table 4.2 représente l'écart relatif moyen des deux relaxations obtenues sur vingt graphes pour différentes valeurs du couple (n, K). L'amélioration apportée par l'utilisation de la formulation étendue, bien que limitée (inférieur à 14%) permet d'accélérer de manière significative la résolution du problème de K-partitionnement par des méthodes de plans coupants ou de *branch-and-cut*.

n	**K**						
	2	**3**	**4**	**5**	**6**	**7**	**8**
7	58 (-13)	31 (-13)	20 (-11)	3 (-1)	0 (-0)		
8	63 (-11)	43 (-7)	11 (-6)	1 (-0)	0 (-0)	0 (-0)	
9	68 (-11)	51 (-7)	29 (-6)	17 (-8)	15 (-6)	3 (-0)	0 (-0)
10	76 (-12)	61 (-8)	42 (-7)	22 (-6)	6 (-4)	5 (-5)	5 (-5)
11	79 (-11)	65 (-10)	48 (-8)	34 (-10)	22 (-9)	10 (-4)	11 (-5)
12	80 (-12)	68 (-12)	52 (-10)	35 (-8)	17 (-7)	8 (-4)	5 (-1)
13	81 (-11)	71 (-9)	59 (-7)	46 (-6)	30 (-6)	21 (-6)	17 (-2)
14	85 (-14)	76 (-13)	66 (-13)	55 (-14)	43 (-13)	26 (-10)	12 (-5)
15	88 (-11)	80 (-9)	70 (-9)	59 (-9)	46 (-6)	32 (-5)	18 (-4)
16	89 (-12)	82 (-9)	75 (-9)	64 (-9)	52 (-10)	38 (-10)	18 (-5)
17	89 (-8)	83 (-8)	77 (-10)	70 (-10)	60 (-9)	47 (-8)	30 (-10)
18	90 (-9)	84 (-9)	78 (-11)	72 (-11)	65 (-12)	55 (-12)	44 (-9)
19	91 (-11)	85 (-9)	80 (-10)	74 (-11)	66 (-11)	57 (-10)	48 (-11)
20	93 (-10)	89 (-9)	84 (-9)	79 (-9)	73 (-8)	63 (-6)	50 (-7)

TABLE *4.2 – Écart relatif moyen des relaxations (R_1) et (R_2) sur vingt graphes. Pour chaque couple (n, K), la première valeur correspond à (R_1) et l'amélioration apporté par (R_2) sur l'écart relatif moyen est représenté entre parenthèse.*

Soit $P_{n,K}$ l'enveloppe convexe de l'ensemble des solutions entières de la formulation (F_1) (*i.e. :* $P_{n,K} = conv(R_1 \cap \{0,1\}^{|E|+|V|-3})$. Compte tenu du fait qu'il existe une correspondance bijective entre les ensembles de solutions réalisables de (F_1) et (F_2), une

facette de $P_{n,K}$ définira aussi une facette de l'enveloppe convexe des points entiers de (F_2). Une partition π est dite incluse dans $P_{n,K}$, si son vecteur indicateur x^π est inclus dans $P_{n,K}$.

4.3 Dimension et facettes

La caractérisation de familles d'inégalités définissant des facettes de l'enveloppe convexe d'un polyèdre entier permet d'acquérir une meilleure connaissance du problème qui y est associé, et est susceptible d'améliorer les performances d'algorithmes de résolution exacte.

Nous nous intéressons ici au polyèdre $P_{n,K}$. Avant d'en déterminer des facettes en section 4.3.2, il est nécessaire de caractériser sa dimension.

4.3.1 Dimension de $P_{n,K}$

Afin de déterminer la dimension de $P_{n,K}$, nous dénombrons le nombre d'hyperplans linéairement indépendants $H = \{x \in \mathbb{R}^{|E|+|V|-3} \mid \alpha^T x = \alpha_0\}$ qui le contiennent. Si le polytope est inclus dans exactement p hyperplans linéairement indépendants, sa dimension est égale à la taille de l'espace des solutions moins p (dans notre cas $|E| + |V| - 3 - p$). La dimension du polytope est donc maximale s'il n'est inclus dans aucun hyperplan.

Pour effectuer le dénombrement de ces hyperplans, nous déterminons successivement des relations sur les composantes de H. Chacune de ces relations est obtenue en appliquant une transformation valide pour $P_{n,K}$. Puisque le polytope dont nous souhaitons connaître la dimension est ici $P_{n,K}$, une transformation $\mathcal{T}(P_1, P_2, R) \mapsto \{P'_1, P'_2\}$ est valide s'il existe une K-partition π contenant P_1 et P_2 et si la partition π' obtenue à partir de π en remplaçant P_1 et P_2 par P'_1 et P'_2 contient elle aussi K parties (*i.e.* : si $P'_1 \neq \emptyset$ et $P'_2 \neq \emptyset$). Compte tenu du fait que les partitions π et π' sont incluses dans $P_{n,K}$, elles figurent aussi dans H. Ainsi, la multiplication de α^T par le vecteur indicateur de π ou de π' est égale à α_0, ce qui nous permet d'obtenir la relation : $\alpha^T x^\pi = \alpha^T x^{\pi'}$.

Dans la suite, étant donnée une K-partition $\pi = \{P_1, P_2, \ldots, P_K\}$ et un entier $i \in \{1, \ldots, K\}$, le terme r_i correspondra au plus petit sommet contenu dans P_i (*i.e.* r_i est le représentant de r_i). De manière similaire, r'_i est utilisé pour représenter le plus petit sommet de $P_i \backslash \{r_i\}$ (*i.e.* : le deuxième sommet de plus petit indice contenu dans P_i).

La démonstration de la dimension de $P_{n,K}$ que nous présentons s'appuie sur quatre lemmes résumés en table 4.3. Si P_1 et P_2 vérifient les conditions représentées dans la deuxième colonne de la table 4.3 et que les transformations figurant dans la colonne suivante sont valides, alors l'égalité présentée en dernière colonne est vraie pour les composantes de tout hyperplan incluant $P_{n,K}$.

Lemme 4.3.1 *Étant données les quatre K-partitions suivantes :*

(i) $\pi = \{P_1, P_2, P_3, \ldots, P_K\}$ *avec* $p_1 \in P_1 \backslash \{r_1\}$ *et* $p_2 \in P_2 \backslash \{r_2\}$;

(ii) $\pi^1 = \{C_1^{(1)}, C_2^{(1)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(1)}, C_2^{(1)}\} = \mathcal{T}(P_1, P_2, \{p_1\})$;

(iii) $\pi^2 = \{C_1^{(2)}, C_2^{(2)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(2)}, C_2^{(2)}\} = \mathcal{T}(P_1, P_2, \{p_2\})$;

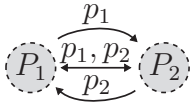
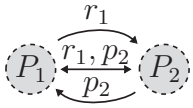
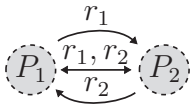
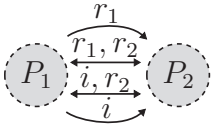
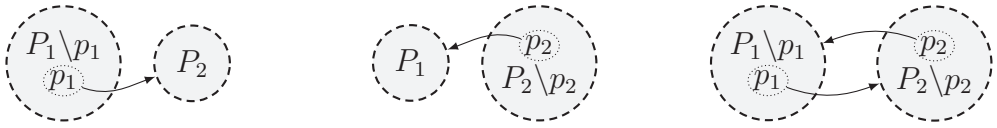
Lemme	Conditions	Transformations valides	Résultat
4.3.1	$p_1 \in P_1 \backslash \{r_1\}$, $p_2 \in P_2 \backslash \{r_2\}$		$\alpha_{p_1,p_2} = 0$
4.3.2	$r_1 < r_2$ $p_2 \in P_2 \backslash \{r_2\}$		$2\alpha_{r_1,p_2} + \alpha_{r'_1} = \alpha_{\min(r'_1,p_2)}$
4.3.3	$r_1 < r_2$ $r_2 < r'_1$ $\lvert P_2 \rvert \geq 2$		$2\alpha_{r_1,r_2} + \alpha_{r'_1} + \alpha_{r'_2} = 2\alpha_{r_2}$
4.3.4	$i \in P_1 \backslash \{r_1\}$ $i < r_2$		$\alpha_{r_1,r_2} = \alpha_{r_2,i}$

TABLE *4.3 – Résumé des quatre lemmes utilisés dans les démonstrations. Étant donnée une $K-$partition $\pi = \{P_1, \ldots, P_K\}$ dont le vecteur indicateur x^π est inclus dans l'hyperplan $H = \{x \in \mathbb{R}^{|E|+|V|-3} \mid \alpha^T x = \alpha_0\}$, si les conditions figurant en deuxième colonne sont vérifiées et que les $K-$partitions obtenues en appliquant chacune des transformations de la troisième colonne sont incluses dans H alors, la relation sur les composantes de H figurant en dernière colonne est nécessairement vraie.*

(iv) $\pi^3 = \{C_1^{(3)}, C_2^{(3)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(3)}, C_2^{(3)}\} = \mathcal{T}(P_1, P_2, \{p_1, p_2\})$.
Si x^π, x^{π^1}, x^{π^2}, x^{π^3} *vérifient* $\alpha^T x = \alpha_0$ *alors* $\alpha_{p_1,p_2} = 0$.



(a) $\pi \to \pi^1$ *(b)* $\pi \to \pi^2$ *(c)* $\pi \to \pi^3$

FIGURE *4.2 – Représentation des transformations permettant de passer de π à π^1, π^2 et π^3.*

Démonstration

Nous rappelons, tout d'abord, que pour tout sous-ensemble de V, V_1 et V_2 et tout vecteur $x \in \mathbb{R}^{|E|}$, $x(V_1)$ et $x(V_1, V_2)$ dénotent respectivement $\sum_{i \in V_1, j \in V_1 \backslash \{i\}} x_{i,j}$ et $\sum_{i \in V_1} \sum_{j \in V_2} x_{i,j}$.

Les transformations permettant de passer de la K-partitions π à π^1, π^2 et π^3 sont représentées en figure 4.2. Notons m_2 le minimum de r_2 et p_1. En utilisant le fait que $\alpha^T x^\pi$ et $\alpha^T x^{\pi^1}$ sont égaux à α_0, nous obtenons :

$$\sum_{i=1}^{K} (\alpha(P_i) + \alpha_{r_i}) = \sum_{i=3}^{K} (\alpha(P_i) + \alpha_{r_i}) + \alpha(P_1 \backslash \{p_1\}) + \alpha(P_2 \cup \{p_1\}) + \alpha_{r_1} + \alpha_{m_2}. \quad (4.17)$$

Ceci peut être simplifié en supprimant les composantes apparaissant des deux côtés de l'égalité :

$$\alpha(\{p_1\}, P_1) + \alpha_{r_2} = \alpha(\{p_1\}, P_2) + \alpha_{m_2}. \tag{4.18}$$

Notons m_1 le minimum de r_1 et p_2. Nous obtenons de manière similaire de π^2 :

$$\alpha(\{p_2\}, P_2) + \alpha_{r_1} = \alpha(\{p_2\}, P_1) + \alpha_{m_1}. \tag{4.19}$$

Grâce à π^3 nous obtenons :

$$\alpha(\{p_1\}, P_1) + \alpha(\{p_2\}, P_2) + \alpha_{r_1} + \alpha_{r_2} = \alpha(\{p_2\}, P_1\backslash\{p_1\}) + \alpha(\{p_1\}, P_2\backslash\{p_2\}) + \alpha_{m_1} + \alpha_{m_2} \tag{4.20}$$

Enfin, les équations (4.18), (4.19) et (4.20) permettent de conclure. □

Lemme 4.3.2 *Étant données les quatre K-partitions suivantes :*
(i) $\pi = \{P_1, P_2, P_3, \ldots, P_K\}$ *avec* $r_1 < r_2$, $p_2 \in P_2\backslash\{r_2\}$ *et* $|P_1| \geq 2$;
(ii) $\pi^1 = \{C_1^{(1)}, C_2^{(1)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(1)}, C_2^{(1)}\} = \mathcal{T}(P_1, P_2, \{r_1\})$;
(iii) $\pi^2 = \{C_1^{(2)}, C_2^{(2)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(2)}, C_2^{(2)}\} = \mathcal{T}(P_1, P_2, \{p_2\})$;
(iv) $\pi^3 = \{C_1^{(3)}, C_2^{(3)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(3)}, C_2^{(3)}\} = \mathcal{T}(P_1, P_2, \{r_1, p_2\})$.
Si x^π, x^{π^1}, x^{π^2}, x^{π^3} *vérifient* $\alpha^T x = \alpha_0$ *alors* $2\alpha_{r_1,p_2} + \alpha_{r_1'} = \alpha_{\min(r_1', p_2)}$.

Démonstration La transformation permettant de passer de π à π^1 est représentée en figure 4.3. Sachant que $\alpha^T x^\pi = \alpha^T x^{\pi^1}$ nous déduisons :

$$\alpha(\{r_1\}, P_1) + \alpha_{r_2} = \alpha(\{r_1\}, P_2) + \alpha_{r_1'}. \tag{4.21}$$

Le sommet r_2 n'est pas un représentant de π^1, puisqu'il se trouve dans la même partie que r_1 qui est plus petit que lui. Ainsi, α_{r_2} n'apparaît que dans la partie gauche de l'équation. A l'inverse, r_1' n'est pas un représentant de π, mais en devient un lorsque r_1 est déplacé dans P_2. C'est la raison pour laquelle, $\alpha_{r_1'}$ apparaît dans la partie droite de (4.21).

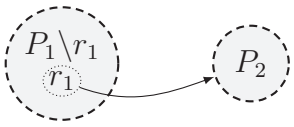


FIGURE 4.3 – *Représentation de $\pi \to \pi^1$ lorsque $p_1 = r_1$.*

Nous déduisons de $\alpha^T x^\pi = \alpha^T x^{\pi^2}$:

$$\alpha(\{p_2\}, P_2) = \alpha(\{p_2\}, P_1). \tag{4.22}$$

De plus $\alpha^T x^\pi = \alpha^T x^{\pi^3}$ nous donne

$$\alpha(\{r_1\}, P_1) + \alpha(\{p_2\}, P_2) + \alpha_{r_2} = \alpha(\{r_1\}, P_2\backslash\{p_2\}) + \alpha(\{p_2\}, P_1\backslash r_1) + \alpha_{\min(r_1', p_2)}. \tag{4.23}$$

Enfin, ce dernier résultat peut être simplifié en $\alpha_{r_1,p_2} + \alpha_{r_2} = \alpha_{\min(r_1', p_2)}$ en utilisant les équations (4.21) et (4.22). □

Lemme 4.3.3 *Étant données les quatre K-partitions suivantes :*
(i) $\pi = \{P_1, P_2, P_3, \ldots, P_K\}$ *avec* $r_1 < r_2 < r'_1$ *et* $|P_2| \geq 2$ *;*
(ii) $\pi^1 = \{C_1^{(1)}, C_2^{(1)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(1)}, C_2^{(1)}\} = \mathcal{T}(P_1, P_2, \{r_1\})$ *;*
(iii) $\pi^2 = \{C_1^{(2)}, C_2^{(2)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(2)}, C_2^{(2)}\} = \mathcal{T}(P_1, P_2, \{r_2\})$ *;*
(iv) $\pi^3 = \{C_1^{(3)}, C_2^{(3)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(3)}, C_2^{(3)}\} = \mathcal{T}(P_1, P_2, \{r_1, r_2\})$.
Si x^π, x^{π^1}, x^{π^2}, x^{π^3} *vérifient* $\alpha^T x = \alpha_0$ *alors* $2\alpha_{r_1,r_2} + \alpha_{r'_1} + \alpha_{r'_2} = 2\alpha_{r_2}$.

Démonstration L'égalité $\alpha^T x^\pi = \alpha^T x^{\pi^1}$ nous amène une nouvelle fois à (4.21).

De $\alpha^T x^\pi = \alpha^T x^{\pi^2}$ nous déduisons :

$$\alpha(\{r_2\}, P_2) + r_2 = \alpha(\{r_2\}, P_1) + r'_2. \tag{4.24}$$

Le sommet r_2 n'est pas un représentant de π^1 puisqu'il se trouve dans la même partie que r_1. Le sommet r'_2, quant à lui, devient un représentant puisque r_2 ne se trouve plus dans sa partie.

Enfin, $\alpha^T x^\pi = \alpha^T x^{\pi^3}$ donne

$$\alpha(\{r_1\}, P_1) + \alpha(\{r_2\}, P_2) = \alpha(\{r_1\}, P_2 \backslash \{r_2\}) + \alpha(\{r_2\}, P_1 \backslash \{r_1\}). \tag{4.25}$$

Cette dernière relation peut être simplifiée en $2\alpha_{r_1,r_2} + \alpha_{r'_1} + \alpha_{r'_2} = 2\alpha_{r_2}$ via les équations (4.21) et (4.24). □

Lemme 4.3.4 *Étant données les cinq K-partitions suivantes :*
(i) $\pi = \{P_1, P_2, P_3, \ldots, P_K\}$ *avec* $i \in P_1 \backslash \{r_1\}$, $i < r_2$ *;*
(ii) $\pi^1 = \{C_1^{(1)}, C_2^{(1)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(1)}, C_2^{(1)}\} = \mathcal{T}(P_1, P_2, \{r_1\})$ *;*
(iii) $\pi^2 = \{C_1^{(2)}, C_2^{(2)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(2)}, C_2^{(2)}\} = \mathcal{T}(P_1, P_2, \{i\})$ *;*
(iv) $\pi^3 = \{C_1^{(3)}, C_2^{(3)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(3)}, C_2^{(3)}\} = \mathcal{T}(P_1, P_2, \{r_1, r_2\})$ *;*
(v) $\pi^4 = \{C_1^{(4)}, C_2^{(4)}, P_3, \ldots, P_K\}$ *avec* $\{C_1^{(4)}, C_2^{(4)}\} = \mathcal{T}(P_1, P_2, \{i, r_2\})$.
Si x^π, x^{π^1}, x^{π^2}, x^{π^3}, x^{π^4} *vérifient* $\alpha^T x = \alpha_0$ *alors* $\alpha_{r_1,r_2} = \alpha_{r_2,i}$.

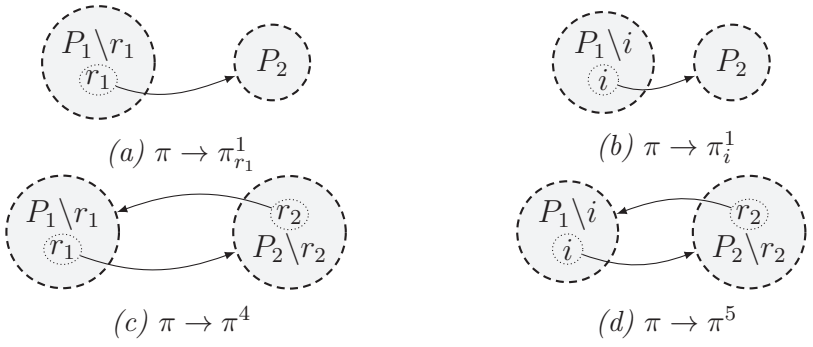


FIGURE *4.4 – Représentation des transformations utilisées dans le lemme 4.3.4.*

Démonstration Comme présenté dans les démonstrations précédentes, nous obtenons respectivement de $\alpha^T x^\pi = \alpha^T x^{\pi^1_{r_1}}$ et $\alpha^T x^\pi = \alpha^T x^{\pi^1_i}$ les équations (4.21) et (4.18) (avec $p_2 = r_2$ dans le dernier cas).

De l'égalité $\alpha^T x^\pi = \alpha^T x^{\pi^4}$ nous déduisons :

$$\alpha(\{r_1\}, P_1) + \alpha(\{r_2\}, P_2) + \alpha_{r_2} = \alpha(\{r_1\}, P_2 \backslash \{r_2\}) + \alpha(\{r_2\}, P_1 \backslash \{r_1\}) + \alpha_{r'_1}. \quad (4.26)$$

L'ensemble $P_1^{(4)}$ correspond à $\{P_1 \backslash p_1\} \cup \{r_2\}$. Son représentant est r'_1 puisque $r'_1 < r_2$.

L'égalité $\alpha^T x^\pi = \alpha^T x^{\pi^5}$ donne

$$\alpha(\{i\}, P_1) + \alpha(\{r_2\}, P_2) + \alpha_{r_2} = \alpha(\{i\}, P_2 \backslash \{r_2\}) + \alpha(r_2, P_1 \backslash \{i\}) + \alpha_i. \quad (4.27)$$

Des équations (4.21) et (4.26) nous déduisons

$$\alpha(\{r_2\}, P_2) + \alpha_{r_1, r_2} = \alpha(\{r_2\}, P_1 \backslash \{r_1\}), \quad (4.28)$$

et des équations (4.18) et (4.27) nous obtenons

$$\alpha(\{r_2\}, P_2) + \alpha_{i, r_2} = \alpha(\{r_2\}, P_1 \backslash \{i\}). \quad (4.29)$$

Ces deux dernières équations nous mènent au résultat souhaité. □

Ces différents lemmes nous amènent au théorème suivant qui caractérise la dimension de $P_{n,K}$ dans tous les cas non triviaux.

Théorème 4.3.5 *Suivant K, la dimension de $P_{n,K}$ est :*

(i) $dim(P_{n,2}) = |E| + |V| - 4$ *;*

(ii) $dim(P_{n,K}) = |E| + |V| - 3$, *pour* $K \in \{3, 4, \ldots, n-2\}$ *(i.e. : le polytope est de dimension pleine) ;*

(iii) $dim(P_{n,n-1}) = |E| - 1$.

Démonstration Démontrons tout d'abord les deux premiers cas (*i.e.* : $K \in \{2, 3, \ldots, n-2\}$). Supposons que $P_{n,K}$ soit inclus dans un hyperplan $H = \{x \in \mathbb{R}^{|E|+|V|-3} \mid \alpha^T x = \alpha_0\}$. Nous prouvons que H est unique si $K = 2$ et que toutes ses composantes valent 0 si $K \in \{3, 4, \ldots, n-2\}$ (*i.e. :* il n'existe aucun hyperplan H contenant $P_{n,K}$). Sauf indication contraire, la seule contrainte imposée aux $K-$partitions considérées dans la suite de cette preuve est que deux de leurs parties contiennent plus d'un sommet. Cette condition est toujours vérifiable si K est compris entre 2 et $n-2$.

Nous appliquons, tout d'abord, le lemme 4.3.1 avec $r_1 = 1$, $r_2 = 2$ et $p_1, p_2 \in \{3, 4, \ldots, n\}$ pour obtenir : $\alpha_{p_1,p_2} = 0$. De façon similaire, si $r_2 = 3$ et $p_1 = 2$, nous déduisons : $\alpha_{2,p_2} = 0$ pour tout $p_2 \geq 4$.

Nous rappelons que les variables x_1, x_2 et x_3 sont artificielles et que, par conséquent, l'hyperplan H ne possède pas de composantes qui leur sont associées. Ainsi, en

prenant $\{1,3\} \subset P_1$, $\{2\} \subset P_2$ et $p_2 \in \{4,5,\ldots,n\}$, le lemme 4.3.2 assure que α_{1,p_2} vaut 0.

Le lemme 4.3.4 pour $(r_1,i,r_2) = (1,2,3)$ montre que $\alpha_{1,3}$ et $\alpha_{2,3}$ sont égaux. Soit β cette valeur. Si $(r_1,r_2,p_2) = (1,2,3)$ et $p_1 \in \{4,5,\ldots,n\}$, le lemme 4.3.2 peut être utilisé afin de montrer que : $\alpha_s = -2\beta$, $\forall s \in \{4,5,\ldots,n\}$.

Nous utilisons maintenant le lemme 4.3.3 avec $\{1,3\} \subset P_1$, $r_2 = 2$ et $r'_2 \in \{4,5,\ldots,n\}$ pour obtenir : $2\alpha_{1,2} + \alpha_{r'_2} = 0$. Nous en déduisons : $\alpha_{1,2} = \beta$.

Nous avons prouvé que $\alpha_{1,2}$, $\alpha_{1,3}$ et $\alpha_{2,3}$ sont égaux à β, que pour tout $s \in \{4,5,\ldots,n\}$ $\alpha_s = -2\beta$ et que toutes les autres composantes valent 0. Puisque l'ensemble des composantes de H sont fixées par un unique coefficient β, le seul hyperplan susceptible de contenir $P_{n,K}$ est

$$\beta(x_{1,2} + x_{1,3} + x_{2,3} - 2\sum_{s=4}^{n} x_s) = \alpha_0. \tag{4.30}$$

Si $K = 2$, l'équation (4.30) est vérifiée si et seulement si α_0 est égal à β. En effet, soit les sommets 1, 2 et 3 sont tous dans la même partie, soit seuls deux de ces sommets sont ensemble et dans les deux cas l'expression $x_{1,2} + x_{1,3} + x_{2,3} - 2\sum_{s=4}^{n} x_s$ vaut 1. En conséquence, il existe un unique hyperplan contenant $P_{n,2}$. La dimension de ce dernier est donc $|E| + |V| - 4$.

Si $K \in \{3,4,\ldots,n-2\}$ le lemme 4.3.3 peut être utilisé avec $r_1 = 1$, $r_2 = 2$ et $3 \in P_3$ pour montrer que β vaut 0. Dans le cas général, il n'existe donc aucun hyperplan contenant $P_{n,K}$.

Dans le cas (*iii*), nous savons que $P_{n,n-1}$ contient exactement $|E|$ solutions entières qui correspondent aux partitions dans lesquelles une unique arête est contenue dans une partie (*i.e.* : $\sum_{i\in V, j\in V\setminus\{i\}} x_{i,j} = 1$). Ces solutions étant indépendantes, $dim(P_{n,n-1}) = |E| - 1$. □

Dans la suite, nous étudions les conditions sous lesquelles des familles d'inégalités définissent des facettes de $P_{n,K}$ lorsque ce dernier est de dimension pleine (*i.e.* : $K \in \{3,\ldots,n-2\}$).

4.3.2 Facettes

Toute face de $P_{n,K}$ est incluse dans au moins une de ses facettes. Pour démontrer qu'une face $F = \{x \in P_{n,K} \mid \omega^T x = \omega_0\}$ est une facette, nous considérons $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ incluant F, puis nous montrons que les vecteurs ω et α sont égaux à un coefficient multiplicateur près (*i.e.* : F_α est unique).

Pour ce faire, de manière similaire aux preuves de la section précédente, nous obtenons des relations sur les composantes du vecteur α en utilisant des transformations valides pour F. Une transformation $\mathcal{T}(P_1,P_2,R) \mapsto \{P'_1,P'_2\}$ est valide pour F s'il existe deux partitions $\pi = \{P_1,P_2,P_3,\ldots,P_K\}$ et $\pi' = \{P'_1,P'_2,P_3,\ldots,P_K\}$ se trouvant, non plus seulement dans $P_{n,K}$, mais aussi dans F.

Une facette est dite *triviale* si elle est engendrée par une inégalité figurant dans la formulation du problème considéré.

4.3.2.1 Bornes des variables d'arêtes

Soit uv une arête de E, les inégalités considérées ici sont :

$$x_{u,v} \leq 1 \tag{4.31}$$

et

$$x_{u,v} \geq 0. \tag{4.32}$$

Remarque 4.3.6 *L'inégalité* (4.31) *ne définit pas une facette de* $P_{n,K}$ *car elle est induite par les inégalités* $x_{u,v} + x_{u,i} - x_{v,i} \leq 1$ *et* $x_{u,v} + x_{v,i} - x_{u,i} \leq 1$ *qui figurent dans la formulation.*

Les inégalités 4.32, quant à elles, définissent généralement des facettes de $P_{n,K}$ comme le présente le théorème suivant.

Théorème 4.3.7 *Si* $K \in \{3, \ldots, n-2\}$, *l'inégalité* (4.32) *définit une facette de* $P_{n,K}$ *si et seulement si* $uv \notin \{12, 13, 23\}$.

Démonstration Montrons tout d'abord que si $uv \in \{12, 13, 23\}$ alors l'inégalité ne définit pas une facette. Si $x_{1,2}$ est nulle, deux configurations sont possibles pour le sommet 3 : soit il se trouve dans la même partie que exactement un des deux premiers sommets, soit il est dans une autre partie. Dans le premier cas, la somme des variables de représentants $\sum_{i=4}^{n} x_i$ vaut $K-2$. Dans le second cas, cette somme vaut $K-3$. Ainsi, lorsque $x_{1,2}$ vaut 0, l'égalité $\sum_{i=4}^{n} x_i - x_{1,3} - x_{2,3} = K-3$ est toujours vérifiée. La face de $P_{n,K}$ considérée est incluse dans l'hyperplan défini par cette égalité et n'est donc pas une facette. Des raisonnements symétriques permettent d'obtenir la même conclusion pour (13) et (23).

Nous considérons maintenant une arête $uv \in E$ telle que $v \geq 4$. Soient $F_{u,v} = \{x^\pi \in P_{n,K} \mid x^\pi_{u,v} = 0\}$ la face de $P_{n,K}$ engendrée par cette inégalité et $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette du polytope incluant $F_{u,v}$. Nous prouvons que $F_{u,v}$ est une facette en montrant que les composantes du vecteur α sont nulles exceptée $\alpha_{u,v}$.

Les transformations utilisées dans la première partie de la démonstration du théorème 4.3.5 peuvent de nouveau être appliquées ici, en posant $P_3 = \{v\}$. Ceci est toujours possible puisque le nombre de parties K est supérieur à 2. La validité des transformations pour $F_{u,v}$ est assurée puisque la composante $x_{u,v}$ est nulle, entraînant une inclusion des partitions considérées dans $F_{u,v}$.

Nous obtenons :
- $\alpha_{i,j} = 0 \quad \forall ij \in E \backslash \{12, 13, 23\}$, $i \neq v$, $j \neq v$;
- $\alpha_{1,2} = \alpha_{1,3} = \alpha_{2,3} = \beta$;
- $\alpha_i = -2\beta \quad \forall i \neq v$.

Nous appliquons ensuite le lemme 4.3.1 avec r_1 et r_2 dans $\{1,2,3\}\backslash\{u\}$, $v \in P_1, P_3 = \{u\}$ et $i \in P_2$ avec $i \in \{4,5,\ldots,n\}\backslash\{u,v\}$, ce qui donne : $\alpha_{i,v} = 0$.

Il reste à démontrer que $\alpha_{a,v}$, α_v et β pour tout $a \in \{1,2,3\}\backslash\{u\}$ valent 0. Soit $b \in \{1,2,3\}\backslash\{a,u\}$. Les transformations $\mathcal{T}(\{a,b,u\},v,a)$ et $\mathcal{T}(\{a,v\},u,a)$ (figure 4.5 et 4.6) donnent $\beta = 0$ et $\alpha_{a,v} = \alpha_v$. Enfin, $\mathcal{T}(\{a,b,v\},u,\{a,b\})$ (figure 4.7) permet de déduire $2\alpha_{a,v} = \alpha_v$ et de conclure.



FIGURE *4.5* – $\mathcal{T}(\{a,b,u\},v,a)$.



FIGURE *4.6* – $\mathcal{T}(\{a,v\},u,a)$.



FIGURE *4.7* – $\mathcal{T}(\{a,b,v\},u,\{a,b\})$.

□

4.3.2.2 Borne des variables de représentants

Soit v un sommet de $V\backslash\{1,2,3\}$, nous considérons ici les inégalités

$$x_v \leq 1 \tag{4.33}$$

et

$$x_v \geq 0. \tag{4.34}$$

Remarque 4.3.8 *L'inégalité* (4.33) *ne définit pas une facette de* $P_{n,K}$ *puisque la face qu'elle induit est incluse dans les faces* $\{x \in P_{n,K} \mid x_{u,v} = 0\}$ *pour tout* $u \in \{1,2,\ldots,v-1\}$.

Le théorème suivant montre que dans la plupart des cas, les inégalités 4.34 définissent des facettes de $P_{n,K}$.

Théorème 4.3.9 *Si* $K \in \{3,\ldots,n-2\}$, *les inégalités* $x_v \geq 0$ *pour tout* $v \in \{4,5,\ldots,n\}$, *définissent des facettes de* $P_{n,K}$ *si et seulement si* $K \neq n-2$.

Démonstration Prouvons d'abord que si K vaut $n-2$, l'inégalité ne définit pas de facette. Dans ce cas, seuls deux sommets ne sont pas des représentants et il est possible de vérifier que l'inégalité $\sum_{i=4}^{n} x_i - x_{1,2} - x_{1,3} - x_{2,3} = K-3$ est toujours vraie. En effet, la somme des représentants $\sum_{i=4}^{n} x_v$ peut varier de $K-3$ (si 1, 2 et 3 sont chacun dans une partie réduite à un sommet) à $K-1$ (si ces sommets sont dans la même partie).

Soit F_v la face de $P_{n,K}$ induite par (4.34) et soit $\pi = \{P_1, P_2, \ldots, P_K\}$ une K-partition dont le vecteur indicateur est dans F_b. Puisque K est inférieur à $n-2$, nous savons qu'au moins trois sommets de V ne sont pas des représentants. En conséquence, les transformations considérées dans la première partie de la démonstration du théorème 4.3.5 sont toutes valides si nous ajoutons le sommet v à P_1 ou P_2 de façon à ce qu'il ne soit jamais le représentant de sa partie (*i.e.* : en le mettant toujours dans une

partie contenant au moins un des sommets 1, 2 et 3). Ainsi, en suivant le même raisonnement, nous obtenons que les composantes d'une facette $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ incluant F_v sont nulles à l'exception de α_v. □

4.3.2.3 Inégalité de maximum de représentants par partie

Les inégalités limitant le nombre de représentants par partie sont

$$x_{u,v} + x_v \leq 1 \quad \forall v \in V\backslash\{3,\ldots,n\}\ \forall u \in \{1,\ldots,v-1\}. \tag{4.35}$$

Les conditions sous lesquelles ces inégalités définissent des facettes de $P_{n,K}$ sont présentées dans le théorème ci-dessous.

Théorème 4.3.10 *Si $K \in \{3,\ldots,n-2\}$, les inégalités* (4.35) *définissent des facettes de $P_{n,K}$ si et seulement si $n \geq 6$ ou $\{u,v\} \neq \{4,5\}$.*

Démonstration Soit $F_{u,v}$ la face induite par (4.35). Montrons tout d'abord que cette équation ne définit pas de facette si $(n,u,v) = (5,4,5)$, en prouvant qu'alors $F_{4,5}$ est incluse dans l'hyperplan induit par

$$2(x_4 + x_5) + \sum_{i\leq 3} x_{i,5} - (x_{1,2} + x_{1,3} + x_{2,3}) = 1. \tag{4.36}$$

Selon les hypothèses du théorème, si n vaut 5, K vaut 3. Une 3-partition de cinq sommets contient des parties de taille $\{2,2,1\}$ ou $\{3,1,1\}$. Le nombre de solutions entières figurant dans $P_{5,3}$ et $F_{4,5}$ est donc limité. La table 4.4 liste l'ensemble des valeurs que peuvent prendre les différentes expressions figurant dans (4.35) et (4.36). Il est aisé de vérifier que dans chacun des cas, l'équation (4.36) est satisfaite.

Expression	$x_{4,5}$	x_4	x_5	$\sum_{i\leq 3} x_{i,5}$	$\sum_{i\leq 3} x_{i,2}$
Configuration entière	1	1	0	0	1
	1	0	0	1	0
	0	1	1	0	3
	0	0	1	0	1

TABLE *4.4 – Liste exhaustive des valeurs pouvant être prises par cinq expressions dans une solution entière de $P_{5,3}$ figurant dans $F_{4,5}$. Chaque ligne représente une configuration.*

Dans les autres cas, soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ incluant $F_{u,v}$. Nous montrons qu'à l'exception de $\alpha_{u,v}$ et α_v qui sont égales, l'ensemble des composantes de α sont nulles. Tout d'abord nous déduisons, de manière similaire à la preuve du théorème 4.3.7, que $\alpha_{i,j} = 0\ \forall ij \in E\backslash\{12,13,23\}$ avec $i \neq v$ et $j \neq v$; $\alpha_{1,2} = \alpha_{1,3} = \alpha_{2,3} = \beta$; $\alpha_i = -2\beta\ \forall i \neq v$. Ensuite, la transformation $\mathcal{T}(\{u,v,i\},\{j\},\{i\})$ avec $i \in V\backslash\{u,v\}$ et $j \in \{1,2,3\}\backslash\{i,u\}$ donne : $\alpha_{i,v} = 0$.

Considérons une partition $\pi = \{P_1, P_2, \ldots, P_K\}$ telle que :

– $P_1 = \{a, b\}$, avec a et b deux sommets distincts parmi $\{1, 2, 3\}$;
– $P_2 = \{i\}$, avec $i \in \{4, \ldots, n\} \backslash \{u, v\}$ (ce qui est toujours possible si $n \geq 6$ ou $\{u, v\} \neq \{4, 5\}$) ;
– $\{u, v\} \subset P_3$.

La transformation $\mathcal{T}(\{a, b\}, i, a)$ (figure 4.8) montre que β vaut 0. Enfin, $\mathcal{T}(\{a, u\}, v, u)$ nous donne $\alpha_v = \alpha_{u,v}$.



FIGURE *4.8* – $\mathcal{T}(\{a, b\}, i, a)$.

□

Théorème 4.3.11 *Si $K \in \{3, \ldots, n-2\}$, les inégalités $x_{1,2} + x_{a,3} - \sum_{i=4}^{n} x_i \leq 3 - K$ pour $a \in \{1, 2\}$ - qui correspondent à $x_{a,3} + x_3 \leq 1$ - définissent des facettes de $P_{n,K}$.*

Démonstration Soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ incluant $F_u = \{x \in P_{n,K} \mid x_{1,2} - \sum_{i=4}^{n} x_i + x_{a,3} = 3 - K\}$. Nous prouvons que $\alpha^T x$ est égal à $\alpha_{1,2}x_{1,2} + \alpha_{a,3}x_{a,3} - \sum_{i=4}^{n} \alpha_i x_i$. Soit $b \in \{1, 2\} \backslash \{a\}$. En utilisant le lemme 4.3.1 avec $(r_1, r_2) = (a, 3)$ et $b \in P_3$, nous montrons que $\alpha_{i,j}$ vaut 0 pour tout $i, j \geq 4$. Le lemme 4.3.3 avec $(r_1, r_2) = (a, 3)$, et $b \in P_3$ nous permet d'obtenir : $2\alpha_{a,3} + \alpha_i + \alpha_j = 0$ $\forall i, j \geq 4$. Nous déduisons ainsi que les composantes α_i sont toutes égales à $-\alpha_{a,3}$. Nous démontrons ensuite que $\alpha_{1,2} = \alpha_{a,3}$ en utilisant le lemme 4.3.3 avec $1 \in P_1$, $2 \in P_2$ et $3 \in P_3$.

Il reste à prouver que $\alpha_{c,i}$ et $\alpha_{d,i}$ sont nuls pour tout $c, d \in \{1, 2, 3\}$ et $i \geq 4$. La transformation $\mathcal{T}(\{c, i\}, d, i)$ donne $\alpha_{c,i} = \alpha_{d,i}$. Ensuite $\mathcal{T}(\{a, 3\}, i, a)$ avec $i \geq 4$ nous permet de déduire $\alpha_{a,i} = 0$.

Enfin, nous prouvons que $\alpha_{b,3}$ vaut 0 en considérant la transformation $\mathcal{T}(\{1, 2, 3\}, i, 3)$ avec $i \geq 4$. □

4.3.2.4 Inégalités de minimum de représentants par partie

Les inégalités assurant que chaque partie contienne au moins un représentant sont :

$$x_u + \sum_{i=1}^{u-1} x_{i,u} \geq 1, \; u \geq 3 \; \forall u \in \{4, \ldots, n\} \tag{4.37}$$

Comme le montre le théorème suivant, elles définissent des facettes de $P_{n,K}$.

Théorème 4.3.12 *Si $K \in \{3, \ldots, n-2\}$, les inégalités* (4.37) *définissent des facettes de $P_{n,K}$.*

Démonstration Soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ incluant $F_u = \{x \in P_{n,K} \mid x_u + \sum_{i=1}^{u-1} x_{i,u} = 1\}$. Montrons que $\alpha^T x$ est égal à $\alpha_u x_u + \sum_{i=1}^{u-1} \alpha_{i,u} x_{i,u}$.

Une nouvelle fois, nous déduisons de la preuve du théorème 4.3.7 que $\alpha_{i,j} = 0$ $\forall i, j \in \{ij \in E \backslash \{12, 13, 23\} \mid i \neq u, j \neq u\}$ et $\alpha_{1,2} = \alpha_{1,3} = \alpha_{2,3} = \beta$, $\alpha_i = -2\beta$ $\forall i \neq u$.

Soient c et d deux sommets d'indice plus petit que u. Il est alors possible, grâce à $\mathcal{T}(\{c, u\}, d, u)$ (figure 4.9), de montrer que $\alpha_{c,u}$ et $\alpha_{d,u}$ sont égaux à une valeur que nous appellons γ. Les transformations $\mathcal{T}(\{1, 2\}, u, 2)$ et $\mathcal{T}(\{1, 2, 3\}, u, 2)$ (figures 4.10 et 4.11) prouvent respectivement $\beta + \alpha_u = \gamma$ et $2\beta + \alpha_u = \gamma$. Ainsi, $\beta = 0$ et $\alpha_u = \gamma$.



FIGURE *4.9* – $\mathcal{T}(\{c, u\}, d, u)$.



FIGURE *4.10* – $\mathcal{T}(\{1, 2\}, u, 2)$.



FIGURE *4.11* – $\mathcal{T}(\{1, 2, 3\}, u, 2)$.

Finalement, nous prouvons que $\alpha_{i,u}$ est nul pour tout i supérieur à u en utilisant $\mathcal{T}(\{c, u\}, \{d, i\}, u)$. □

Théorème 4.3.13 *Si* $K \in \{3, \ldots, n-2\}$, *l'inégalité* $x_{1,2} + x_{1,3} + x_{2,3} - \sum_{i=4}^{n} x_i \geq 3 - K$ *pour* $a \in \{1, 2\}$ *- qui correspond à* $x_3 + \sum_{i=1}^{2} x_{i,3} \leq 1$ *- définit une facette de* $P_{n,K}$.

Démonstration La preuve est similaire à celle du théorème 4.3.11 en considérant que $a \in \{1, 2\}$. L'unique différence est que la transformation finale n'est pas nécessaire $\mathcal{T}(\{1, 2, 3\}\ \{i\}\{3\})$. □

4.3.2.5 Inégalités triangulaires

Les inégalités triangulaires, assurant la cohérence de la partition obtenue, sont définies par :

$$x_{i,j} + x_{i,k} - x_{j,k} \leq 1,\ \forall i \in V, \forall j, k \in V \backslash \{i\},\ j < k. \tag{4.38}$$

Dans le polyèdre correspondant au cas général du problème de partitionnement, étudié par Grötschel et Wakabayashi [51], les inégalités triangulaires définissent toujours des facettes. Le théorème suivant montre que le fait de considérer une formulation avec des variables de représentants entraîne que plus d'un tiers d'entre elles ne définissent pas de facette du polyèdre correspondant.

Théorème 4.3.14 *Si* $K \in \{3, \ldots, n-2\}$, *l'inégalité* (4.38) *définit une facette de* $P_{n,K}$ *si et seulement si les conditions suivantes sont satisfaites :*

(i) $i < j$ *ou* $i < k$ *;*

(ii) $\{i, j, k\} \neq \{1, 2, 3\}$.

Les inégalités triangulaires étant un cas particulier des inégalités de deux partitions (en prenant $S = \{i\}$ et $T = \{j, k\}$) le lecteur peut se référer au théorème 4.3.20 page 108 pour la preuve du théorème précédent.

4.3.2.6 Inégalités 2−*chorded cycles*

Les *inégalités* 2−*chorded cycles* ont été introduites par Grötschell et Wakabayashi [51]. Soit $C = \{e_1, \ldots, e_{|C|}\}$ un cycle de E tel que $e_i = c_i c_{i+1}$ pour tout $i \in \{1, 2, \ldots, |C| - 1\}$ et $e_{|C|} = c_1 c_{|C|}$. Soient $V_C = \{c_1, c_2, \ldots, c_{|C|}\}$ et $U = V \backslash V_C$. Soient $u_1, u_2, \ldots, u_{|U|}$ les sommets de U ordonnés de telle sorte que $u_1 < u_2 < \ldots < u_{|U|}$. Dans la suite, les indices des sommets de V_C sont donnés modulo $|C|$ (*e.g.* $c_{|C|+2}$ correspond à c_2). L'ensemble des 2-cordes de C est définie par $\overline{C} = \{c_i c_{i+2} \in E \mid i = 1, \ldots, |C|\}$. L'inégalité 2-*chorded cycle* induite par un cycle C de taille supérieure ou égale à 5 est défini par

$$x(C) - x(\overline{C}) \leq \lfloor \frac{1}{2}|C| \rfloor. \tag{4.39}$$

La preuve du lemme suivant est volontairement omise. Pour de plus amples détails, le lecteur peut se référer à [51].

Lemme 4.3.15 *L'inégalité (4.39), induite par un cycle C de taille supérieure ou égale à 5, est valide pour $P_{n,K}$. La face correspondante F_C ne définit pas de facette si $|C|$ est pair.*

Le théorème suivant présente trois conditions suffisantes sous lesquelles les inégalités 2-*chorded cycles* définissent des facettes de $P_{n,k}$.

Théorème 4.3.16 *La face F_C induite par un cycle C de taille $2p + 1$ ($p \in \mathbb{N}\backslash\{0, 1\}$) définit une facette de $P_{n,K}$ si les conditions suivantes sont satisfaites :*

(i) $K \in \{4, \ldots, n - 2\}$;

(ii) $|U \cap \{1, 2, 3\}| \geq 2$;

(iii) $2 \leq p \leq n - K - |U \cap \{1, 2, 3\}|$.

Démonstration Nous exhibons tout d'abord des $K-$partitions $\pi_i = \{P_1, P_2, \ldots, P_K\}$ se trouvant dans F_C lorsque les conditions (i) à (iii) sont satisfaites.

Soit c_i un sommet de V_C. La $K-$partition π_i est construite comme suit :
- $P_1 = \{c_i, u_1, u_2\}$;
- Les $2p$ sommets restants de V_C sont répartis dans les parties suivantes de telle sorte que exactement p arêtes de C soient activées et aucune de $\overline{C}$. Si la partie P_K est atteinte, les sommets restants sont répartis dans P_{K-1} et P_K (voir exemple en figure 4.12) :
  - $P_q = \{c_{i+2q-1}, c_{i+2q}\}\ \forall q \in \{2, \ldots, \min(p, K)\}$;
  - $P_{K-1} \supset \{c_{i+K+2q-1}, c_{i+K+2q}\}$, pour tout entier naturel impair q tel que $K + 2q \leq 2p$;
  - $P_K \supset \{c_{i+K+2q-1}, c_{i+K+2q}\}$, pour tout entier naturel pair q tel que $K+2q \leq 2p$.
- Les sommets de $U\backslash\{u_1, u_2\}$ sont répartis dans les parties suivantes ou dans P_K s'il a été atteint :
  - $P_{p+q-1} = \{u_q\}\ \forall q \in \{3, K - p + 1\}$;
  - $P_K \supset \{u_q\}\ \forall q \in \{K - p + 2, |U|\}$.

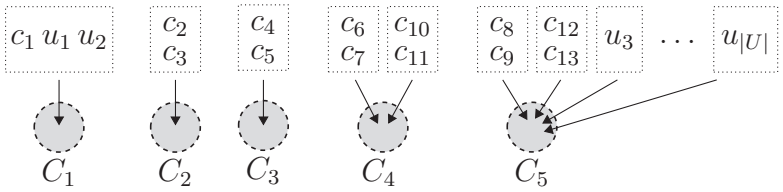


FIGURE *4.12 – Construction de π_1 si $K = 5$ et $|C| = 13$.*

De part cette construction, π_i est une $K-$partition. Puisqu'aucune arête $e \in \overline{C}$ n'est incluse dans une partie et qu'exactement p arêtes de C le sont, π_i appartient à F_C.

L'ensemble des transformations considérées dans la suite de cette preuve peuvent être appliquées soit directement à π_i, soit à une K-partition obtenue à partir de π_i via des transformations valides pour F_C. De plus, les transformations ne modifient ni le nombre de parties, ni la valeur de l'expression $x(C) - x(\overline{C})$, ce qui assure leur validité pour F_C.

Soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ contenant F_C. Nous montrons que pour tout i dans $\{1, 2, \dots, |C|\}$, les composantes $\alpha_{c_i,c_{i+1}}$ (respectivement $\alpha_{c_i,c_{i+2}}$) sont égales à une constante appelée β (respectivement $-\beta$) et que tout autre composante de F_α vaut 0.

Nous savons, par la condition (ii), que u_1 et u_2 appartiennent à $\{1, 2, 3\}$.

Nous prouvons tout d'abord que pour tout i dans $\{1, 2, \dots, |C|\}$, les expressions $\alpha_{c_i,c_{i+1}}$ et $-\alpha_{c_i,c_{i+2}}$ sont égales à β. Pour ce faire, nous considérons les transformations représentées figures 4.13 et 4.14. Ces transformations sont valides puisqu'elles n'altèrent ni le nombre de parties, ni la somme $x(C) - x(\overline{C})$. Durant ces transformations, aucune variable de représentant n'est modifiée puisque u_1 et u_2 sont plus petits ou égaux à 3. Nous obtenons $\alpha_{u_2,c_i} = \alpha_{c_i,c_{i+1}} + \alpha_{c_i,c_{i+2}} + \alpha_{u_1,c_i}$ et $\alpha_{u_1,c_i} = \alpha_{c_i,c_{i+1}} + \alpha_{c_i,c_{i+2}} + \alpha_{u_2,c_i}$, dont nous déduisons :

$$\lambda_i = \alpha_{u_1,c_i} = \alpha_{u_2,c_i}, \tag{4.40}$$

et

$$\beta_i = \alpha_{c_i,c_{i+1}} = -\alpha_{c_i,c_{i+2}}. \tag{4.41}$$

Si nous substituons c_i et c_{i+2} dans les transformations, nous obtenons : $\beta_{i+1} = -\alpha_{c_i,c_{i+2}} = \beta_i$. En appliquant de manière similaire ces deux transformations pour toute valeur possible de i, nous concluons que l'ensemble des coefficients β_i sont égaux à une valeur que nous appellons β.

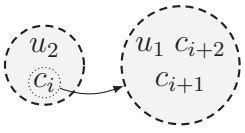


FIGURE *4.13 –* $\mathcal{T}(\{c_i, u_2\}, \{c_{i+1}, c_{i+2}, u_1\}, c_i)$.

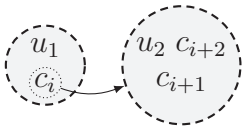


FIGURE *4.14 –* $\mathcal{T}(\{c_i, u_1\}, \{c_{i+1}, c_{i+2}, u_2\}, c_i)$.

Nous montrons maintenant que les autres composantes de α sont nulles.

Les transformations $\mathcal{T}(\{u_1, c_i\}, \{c_{i+1}, c_{i+2}\}, c_{i+1})$ et $\mathcal{T}(\{u_1, u_2, c_i\}, \{c_{i+1}, c_{i+2}\}, \{c_{i+1}\})$ permettent respectivement d'obtenir $\alpha_{c_{i+1}} \mathbb{1}(c_{i+1} < c_{i+2}) = \lambda_{i+1} + \alpha_{c_{i+2}} \mathbb{1}(c_{i+1} <$

c_{i+2}) et $\alpha_{c_{i+1}}\mathbb{1}(c_{i+1} < c_{i+2}) = 2\lambda_{i+1} + \alpha_{c_{i+2}}\mathbb{1}(c_{i+1} < c_{i+2})$. Nous en déduisons que pour tout $i \in \{1, \ldots, |C|\}$

$$\lambda_i = 0. \tag{4.42}$$

Donc, $\alpha_{c_{i+1}}\mathbb{1}(c_{i+1} < c_{i+2}) = \alpha_{c_{i+2}}\mathbb{1}(c_{i+1} < c_{i+2})$. Ainsi, toutes les variables de représentants de V_C sont égales à une constante que nous appelons γ. La transformation $\mathcal{T}(c_i, \{u_1, u_2, c_{i+1}, c_{i+2}\}, u_1)$ donne ensuite $\alpha_{u_1,u_2} + \gamma = 0$.

- Si $|C \cap \{1, 2, 3\}| = 1$, nous obtenons directement $\gamma = 0$ et donc $\alpha_{u_1,u_2} = 0$.
- Si $|C \cap \{1, 2, 3\}| = 0$, il existe u_3 appartenant à $\{1, 2, 3\} \cap (U \backslash \{u_1, u_2\})$ et la transformation $\mathcal{T}(\{u_1, u_2, u_3, c_i\}, \{c_{i+1}, c_{i+2}\}, u_1)$ fournit le même résultat.

Soit t un entier naturel dans $\{3, 4, \ldots, 2p-3\}$. Nous prouvons maintenant que pour tout i dans $\{1, 2, \ldots, |C|\}$ les composantes $\alpha_{c_i,c_{i+t}}$ et $\alpha_{c_i,c_{i+t+1}}$ sont nulles. Les transformations représentées en figure 4.15 et 4.16 donnent respectivement

$$\alpha_{c_i,c_{i+t}} + \alpha_{c_i,c_{i+t+1}} = 0, \tag{4.43}$$

et

$$\alpha_{c_i,c_{i+t}} + \alpha_{c_{i-1},c_{i+t}} = 0. \tag{4.44}$$

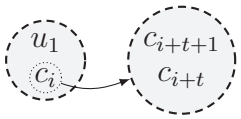


FIGURE *4.15* – $\mathcal{T}(\{c_i, u_1\}, \{c_{i+t}, c_{i+t+1}\}, c_i)$.

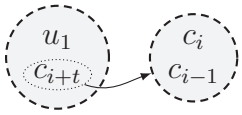


FIGURE *4.16* – $\mathcal{T}(\{c_{i+t}, u_1\}, \{c_{i-1}, c_i\}, c_{i+t})$.

De ces deux équations, nous déduisons qu'il existe un coefficient α_{odd} tel que pour tout $i \in \{1, 2, \ldots, |C|\}$ et tout $t \in \{3, 4, \ldots, 2p-2\}$

$$\alpha_{c_i,c_{i+t}} = \begin{cases} \alpha_{odd} & \text{si } t+i \text{ est impair} \\ -\alpha_{odd} & \text{sinon} \end{cases}. \tag{4.45}$$

En conséquence, $\alpha_{c_{2p-2},c_{2p+1}}$ est égal à α_{odd}. Cependant, $c_{2p+1} = c_1$ puisque $|C| = 2p+1$. Ainsi, $\alpha_{c_{2p-2},c_{2p+1}} = \alpha_{c_1,c_{2p-2}}$ et du fait que $2p-2$ est pair, nous obtenons aussi $\alpha_{c_{2p-2},c_{2p+1}} = -\alpha_{odd}$. Le coefficient α_{odd} vaut donc 0.

Pour conclure cette preuve, nous montrons que pour tout sommet u dans $U \cap (V \backslash \{1, 2, 3\})$, l'ensemble des composantes relatives à u sont nulles. Les transformations présentées en figure 4.17, 4.18 et 4.19 donnent respectivement : $\alpha_{c_i,u} = 0$, $\alpha_{u_1,u} = \alpha_u$ et $\alpha_u\mathbb{1}(c_i < u) = 0$. S'il existe $c_i < u$ nous obtenons que α_u et $\alpha_{u_1,u}$ valent 0. Dans le cas contraire la transformation $\mathcal{T}(\{c_i, u_1, u_2, u\}, \{c_{i+1}, c_{i+2}\}, u_1)$ fournit le résultat souhaité.

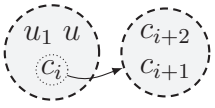


FIGURE *4.17* – $\mathcal{T}(\{u_1, u, c_i\}, \{c_{i+1}, c_{i+2}\}, c_i)$.

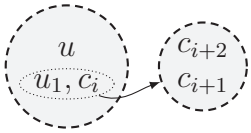


FIGURE *4.18* – $\mathcal{T}(\{u_1, u, c_i\}, \{c_{i+1}, c_{i+2}\}, \{u_1, c_i\})$.

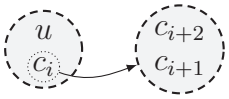


FIGURE 4.19 – $\mathcal{T}(\{c_i, u\}, \{c_{i+1}, c_{i+2}\}, c_i)$.

□

4.3.2.7 Inégalités de 2-partitions

Étant donnés deux sous-ensembles non vides de V, S et T, les *inégalités de 2-partitions* introduites dans [51], sont définies par

$$x(S,T) - x(S) - x(T) \leq \min(|S|, |T|). \tag{4.46}$$

Soit $F_{S,T}$ la face de $P_{n,K}$ définie par l'équation 4.46. Les preuves des trois lemmes suivants sont omises. Pour de plus amples détails, le lecteur peut se référer à [51] pour les lemmes 4.3.17 et 4.3.18 et à [31] pour le lemme 4.3.19.

Lemme 4.3.17 *L'inégalité* (4.46) *est valide pour* $P_{n,K}$.

Lemme 4.3.18 *Si* $|S| = |T|$, $F_{S,T}$ *n'est pas une facette de* $P_{n,K}$.

Lemme 4.3.19 *Étant donnés deux sous-ensembles disjoints de* V, S *et* T *tels que* $|S| < |T|$. *Le vecteur indicateur d'une* $K-$*partition* $\pi = \{P_1, P_2, \ldots, P_K\}$ *est inclus dans* $F_{S,T}$ *si et seulement si pour tout* $i \in \{1, 2, \ldots, K\}$, $|T \cap P_i| - |S \cap P_i| \in \{0, 1\}$

Soit $U = \{u_1, u_2, \ldots, u_{|U|}\}$ l'ensemble des sommets qui ne sont ni dans S ni dans T tels que $u_1 < u_2 < \ldots < u_{|U|}$ (*i.e.* : $U = V \backslash (S \cup T)$). Les sommets de S et T - $\{s_1, s_2, \ldots, s_{|S|}\}$ et $\{t_1, t_2, \ldots, t_{|T|}\}$ respectivement - sont triés de manière similaire.

L'ensemble des cas dans lesquels les inégalités de 2-partitions définissent des facettes de $P_{n,K}$ sont caractérisés dans le théorème suivant.

Théorème 4.3.20 *Si* $K \in \{3, \ldots, n-2\}$, *l'inégalité de 2-partitions (4.46) correspondant à deux ensembles disjoints* S *et* T *de* V, *définit une facette de* $P_{n,K}$ *si et seulement si :*

(i) $|T| - |S| \in \{1, 2, \ldots, K-1\}$;
(ii) $|S| \leq n - (K+2)$;
(iii) $\forall s \in S \ \exists t \in T,\ t > s$;
(iv) if $|S| = 1 \ \exists u \in U \cap \{1, 2, 3\}$.

La preuve de ce théorème est précédée par deux lemmes. Le premier a pour but d'assurer que toute transformation considérée dans cette dernière est bien incluse dans $F_{S,T}$.

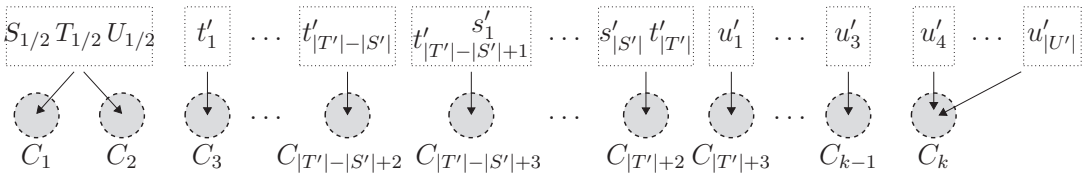


FIGURE *4.20 – Construction de π si $K = |T'| + 6$ et $|U'| \geq 4$.*

Lemme 4.3.21 *Soient P_1 et P_2 deux sous-ensembles disjoints de V. Il existe une $K-$partition contenant P_1 et P_2 dont le vecteur indicateur figure dans $F_{S,T}$ si :*

(i) $|P_1 \cap T| - |P_1 \cap S| = 0$ *;*

(ii) $|P_2 \cap T| - |P_2 \cap S| = 1$ *;*

(iii) $|(P_1 \cup P_2) \cap (T \cup U)| \leq 4$

(iv) S et T vérifient les conditions du théorème 4.3.20.

Démonstration

Soient $S_{1/2}$, $T_{1/2}$ et $U_{1/2}$ les sommets de P_1 et P_2 se trouvant respectivement dans S, T et U (*i.e. :* $S_{1/2} = S \cap (P_1 \cup P_2)$). Soit S' (respectivement T' et U') les sommets de S (respectivement T et U) qui ne sont ni dans P_1 ni dans P_2 (*i.e. :* $S' = S \backslash S_{1/2}$). Nous définissons les sommets de S', T' et U' de la manière suivante : $S' = \{s'_1, \ldots, s'_{|S'|}\}$, $T' = \{t'_1, \ldots, t'_{|T'|}\}$ et $U' = \{u'_1, \ldots, u'_{|U'|}\}$.

Soit π la partition définie par (voir exemple en figure 4.20) :
- Les deux premières parties de π sont P_1 et P_2 ;
- Les $|T'| - |S'|$ parties suivantes sont réduites à un sommet de T' : $P_i = \{t'_{i-2}\}$ $\forall i \in \{3, 4 \ldots, |T'| - |S'| + 2\}$;
- Les sommets restants de T' et S' sont répartis dans les parties suivantes ou dans P_K si cette partie est atteinte :
  - $P_i = \{s'_{i-|T'|+|S'|+2}, t'_{i-2}\}$ $\forall i \in \{|T'| - |S'| + 3, \ldots, \min(|T'| + 2, K - 1)\}$;
  - $P_K \supset \{s'_{i-|T'|+|S'|+2}, t'_{i-2}\}$ $\forall i \in \{K, |T'| + 2\}$;
- Les sommets de U' sont répartis de manière similaire dans les parties suivantes ou dans P_K :
  - $P_i = \{u'_{i-|T'|-2}\}$ $\forall i \in \{|T'| + 3, \ldots, \min(|T'| + |U'| + 2, K - 1)\}$;
  - $P_K \supset \{u'_{i-|T'|-2}\}$ $\forall i \in \{K, |T'| + |U'| + 2\}$;

Si la partition π contient exactement K parties, le lemme 4.3.19 assure que $x^\pi \subset F_{S,T}$. Nous montrons, pour ce faire, que $|T'| + |U'| + 2$ est supérieur ou égal à K.

Selon la seconde condition du théorème 4.3.20, K est inférieur ou égal à $n-(|S|+2)$. En remarquant que $n-|S|$ est égal à $|T|+|U|$, nous déduisons que K doit être inférieur ou égal à $|T_{1/2}|+|T'|+|U_{1/2}|+|U'|-2$. Grâce à (iii), nous déduisons que $|T'|+|U'|+2$ est supérieur ou égal à K. □

Une transformation $\mathcal{T} : \{P_1, P_2, R\} \mapsto \{P'_1, P'_2\}$ est valide pour $F_{S,T}$ si :
- il existe une K-partition π dans $F_{S,T}$ contenant P_1 et P_2 ($\pi = \{P_1, P_2, \ldots, P_K\}$) ;
- la K-partition $\pi' = \{P'_1, P'_2, P_3, \ldots, P_K\}$ se trouve, elle aussi, dans $F_{S,T}$.

Le lemme 4.3.21 fournit des conditions suffisantes pour que la première condition soit vérifiée. Pour nous assurer que le deuxième point est satisfait, nous devons tout d'abord vérifier que π' est une $K-$partition (ce qui est le cas si P'_1 et P'_2 sont non

vides). Puis, nous vérifions que π' est bien dans $F_{S,T}$. Pour ce faire, en accord avec le lemme 4.3.19, il est suffisant que P_1' et P_2' vérifient les deux premières conditions du lemme 4.3.21. Chacune des transformations que nous considérons dans les deux preuves suivantes vérifient l'ensemble de ces conditions et sont donc valides pour $F_{S,T}$.

Afin d'alléger la preuve du théorème 4.3.20, nous présentons un lemme assurant que pour tout $u \in U$ et tout $v \in V\backslash\{u\}$ la composante $\alpha_{u,v}$ d'une facette $F_\alpha\{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ de $P_{n,K}$ incluant $F_{s,t}$ est nulle.

Lemme 4.3.22 *Si S et T vérifient les conditions du théorème 4.3.20 :*

(i) $|T| - |S| \in \{1, 2, \ldots, K-1\}$;
(ii) $|S| \leq n - (K+2)$;
(iii) $\forall s \in S \ \exists t \in T,\ t > s$;
(iv) if $|S| = 1 \ \exists u \in U \cap \{1,2,3\}$;

alors pour toute facette $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ incluant $F_{S,T}$ les coefficients $\alpha_{u,v}$ sont nuls pour tout $u \in U$ et tout $v \in V\backslash\{u\}$.

Démonstration Pour prouver ce lemme, nous considérons les cas suivants :
- cas 1 : $\{t_1, t_2\} \subset \{1,2,3\}$;
- cas 2 : $\{s_1, s_2\} \subset \{1,2,3\}$;
- cas 3 : $\{u_1, u_2\} \subset \{1,2,3\}$;
- cas 4 : $\{s_1, t_1, u_1\} = \{1,2,3\}$.

Dans chacun de ces cas, nous prouvons que pour tout $s \in S,\ t \in T$ et tout $u' \in U$ les composantes $\alpha_{s,u},\ \alpha_{t,u}$ et $\alpha_{u,u'}$ sont nulles.

Cas 1 : $\{t_1, t_2\} \subset \{1,2,3\}$

Dans ce cas, les résultats présentés table 4.5 prouvent la nullité de la plupart des composantes. Il reste à montrer que $\alpha_{t,u}$ vaut 0 pour tout $t \in T$.

Lemme	P_1	P_2	**Résultats**		
4.3.1	$\{s, t_1\}$	$\{t_2, u\}$	$\alpha_{s,u} = 0$	$\forall s \in S$	(4.47)
4.3.1	$\{s, t_1, u\}$	$\{t_2, u'\}$	$\alpha_{u,u'} = 0$	$\forall u' \in U$	(4.48)

TABLE *4.5 – Résultats obtenus dans le cas $\{t_1, t_2\} \subset \{1,2,3\}$.*

Des conditions (i) et (ii) du théorème 4.3.20, nous déduisons que $|T| \leq n-3$. Ainsi, soit $|S| \geq 2$ soit $|U| \geq 2$. Pour tout $t \in T\backslash\{t_1\}$, posons $P = \{s_2, t'\}$ avec $t' \in T\backslash\{t_1, t\}$ si $|S| \geq 2$ et $P = \{u'\}$ avec $u' \in U\backslash\{u\}$ si $|U| \geq 2$. Les transformations $\mathcal{T}(\{t_1, u\}, P, \{u\})$ et $\mathcal{T}(\{s_1, t_1, t, u\}, P, \{u\})$ nous permettent d'obtenir deux égalités identiques à l'exception des composantes $\alpha_{t,u}$ et $\alpha_{s_1,u}$ qui apparaissent dans le membre gauche de la deuxième. Ainsi, puisque $\alpha_{s_1,u}$ est nul, nous déduisons qu'il en va de même pour $\alpha_{t,u}$.

Enfin nous montrons que $\alpha_{t_1,u}$ vaut 0 via la transformation $\mathcal{T}(\{s, t_1\}, \{t_2, u\}, \{u\})$.

Cas 2 : $\{s_1, s_2\} \subset \{1, 2, 3\}$

Selon (i), puisque $|S| \geq 2$, alors $|T| \geq 4$. La première ligne de la table 4.6 montre que $\alpha_{t,u}$ vaut toujours 0 pour tout $t \in T$. Pour tout $s \in S \backslash \{s_1\}$, les transformations $\mathcal{T}(\{s_1, t, u\}, \{t'\}, \{u\})$ et $\mathcal{T}(\{s_1, s, t, t'', u\}, \{t'\}, \{u\})$ montrent que $\alpha_{s,u}$ vaut 0. Un raisonnement similaire en substituant respectivement s_1 et s par s_2 et s_1 amène à : $\alpha_{s_1,u} = 0$. Enfin, les équations (4.50) et (4.51) nous permettent de déduire que les composantes $\alpha_{u,u'}$, pour tout u, u' distincts dans U, valent 0.

Lemme	P_1	P_2	**Résultat**		
4.3.1	$\{s_1, t, t'\}$	$\{s_2, t'', u\}$	$\alpha_{t,u} = 0$	$\forall t \in T$, $t', t'' \in T \backslash \{t\}$	(4.49)
4.3.4	$\{u, t\}$	$\{u'\}$	$\alpha_{u,u'} = \alpha_{t,u'}$	$\forall u' \in U$, $t < u'$, $u < u'$	(4.50)
4.3.4	$\{s_1, t, u\}$	$\{t', u'\}$	$\alpha_{s_1,u'} = \alpha_{u,u'}$	$\forall u' \in U$, $u < u' < t'$	(4.51)

TABLE *4.6 – Résultats obtenus dans le cas* $\{s_1, s_2\} \subset \{1, 2, 3\}$.

Cas 3 : $\{u_1, u_2\} \subset \{1, 2, 3\}$

Soit u un sommet de U et $a \in \{u_1, u_2\} \backslash \{u\}$. Les transformations $\mathcal{T}(\{a\}, \{t, u\}, \{u\})$ et $\mathcal{T}(\{a, s, t\}, \{t, u\}, \{u\})$ permettent d'obtenir : $\alpha_{s,u} + \alpha_{t,u} = 0$. Ensuite, les transformations $\mathcal{T}(\{s, t\}, \{a, t'\}, \{a\})$ et $\mathcal{T}(\{s, t\}, \{a, t', u\}, \{a, u\})$ montrent que $\alpha_{t,u}$ vaut 0. Il en va donc de même pour $\alpha_{s,u}$.

Il reste à montrer que $\alpha_{u,u'}$ est nul pour tout u, u' distincts dans U. Supposons, sans perte de généralité, que u est inférieur à u'. Si $\min(s_1, t_1) \leq 3$, la transformation $\mathcal{T}(\{u, u'\}, \{s_1, t_1, t_2\}, \{u'\})$ fournit le résultat. Dans le cas contraire, U contient au moins trois sommets. Soit $a \in U \backslash \{u, u'\}$. Nous obtenons le même résultat grâce aux transformations $\mathcal{T}(\{a, u\}, \{t\}, \{a\})$ et $\mathcal{T}(\{a, u, u'\}, \{t\}, \{a, u'\})$.

Cas 4 : $\{s_1, t_1, u_1\} = \{1, 2, 3\}$

La table 4.7 permet de conclure pour l'ensemble des composantes, à l'exception de α_{s,u_1} pour tout $s \in S$ et α_{t,u_1} pour tout $t \in T$.

La transformation $\mathcal{T}(P \cup \{t_1\}, \{t\}, \{t_1, t\})$ avec P successivement égal à $\{s_1\}$ et $\{s_1, u_1\}$ nous indique que α_{t_1,u_1} et α_{t,u_1} sont égaux. Ensuite, $\mathcal{T}(\{t_1, u_1\}, \{s_1, t\}, \{u_1\})$ permet de conclure que α_{s_1,u_1} est nul.

Comme nous l'avons remarqué précédemment, au moins un des ensembles S et U contient plus d'un sommet. Si U contient au moins deux sommets, la transformation $\mathcal{T}(P \cup \{s_1, t\}, \{u_2\}, \{s_1, t\})$ avec P égal à $\{t_1\}$ puis $\{t_1, u_1\}$ permet de montrer que α_{t,u_1} est nul. Si S contient plus d'un sommet, nous considérons la transformation $\mathcal{T}(P \cup \{s, t\}, \{s', t'\}, \{s, t\})$ avec $\{s, s'\} \subset S$ et $\{t, t'\} \subset T$ et P un ensemble de sommets. En prenant, tout d'abord, s égal à s_1 et en considérant P égal à $\{t_1\}$ puis $\{t_1, u_1\}$, nous montrons que α_{t,u_1} est nul. Ensuite, en procédant de même mais en prenant $s \in S \backslash \{s_1\}$, nous obtenons : $\alpha_{s_1,u_1} = 0$.

Lemme	P_1	P_2	**Résultat**		
4.3.1	$\{s_1, t, u\}$	$\{t_1, u'\}$	$\alpha_{u,u'} = 0$	$\forall u' \in U$	(4.52)
4.3.1	$\{t, u_1, u\}$	$\{t_1, s\}$	$\alpha_{s,u} = 0$	$u \neq u_1$	(4.53)
4.3.1	$\{u_1, u\}$	$\{s_1, t, t'\}$	$\alpha_{t,u} = 0$	$\forall t, t' \in T$, $u \neq u_1$	(4.54)

TABLE *4.7 – Résultats obtenus dans le cas* $\{s_1, t_1, u_1\} = \{1, 2, 3\}$.

□

Démonstration du théorème 4.3.20

Nous commençons par prouver que l'ensemble des conditions sont nécessaires.

(i) Les lemmes 4.3.18 et 4.3.19 montrent respectivement que $|T| - |S| > 0$ et $|T| - |S| \leq K$. Il reste à prouver que si $|T| - |S| = K$, l'équation (4.46) ne définit pas une facette de $P_{n,K}$. Dans ce cas, chaque partie P vérifie, $|P \cap T| = |P \cap S| + 1$ et $F_{S,T}$ est ainsi incluse dans les $|T|$ hyperplans induits par $\sum_{i \in T \setminus \{t\}} x_{i,t} = \sum_{i \in S} x_{i,t}$ pour tout $t \in T$.

(ii) Nous savons grâce au lemme 4.3.19 que chaque partie P contient au moins autant de sommets provenant de T que de S. Ainsi, au moins $|S|$ sommets ne sont pas des représentants de leur partie, et donc $K \leq n - |S|$. Si $K = n - |S|$, $x(S, T)$ vaut nécessairement $|S|$ puisque les seules arêtes dans la partition sont celles assurant que chaque sommet $s \in S$ est relié à un sommet de T. Enfin, si $K = n - |S| - 1$ nous déduisons, en énumérant l'ensemble des configurations possibles, que $F_{S,T}$ est incluse dans l'hyperplan défini par : $x(U) + x(U, T) + x(S, T) - x(S) = 3$.

(iii) Supposons qu'il existe un sommet s de S supérieur à tous les sommets figurant dans T. Puisque chaque sommet de S se trouve dans une partie contenant au moins un sommet de T, il est impossible que s soit un représentant. Nous avons donc nécessairement x_s vaut 0.

(iv) Si S est réduit à un sommet et que u_1 est plus grand que 4, alors il existe au moins deux sommets de T – notés t_1 et t_2 – , dans l'ensemble $\{1, 2, 3\}$. Le troisième sommet de cet ensemble se trouve soit dans T, soit dans S :
– s'il est dans T alors $\sum_{i=4}^{n} x_i = x_{1,2} + x_{1,3} + x_{2,3} + K - 3$;
– s'il est dans S alors $\sum_{i=4}^{n} x_i = x_{s,t_1} + x_{s,t_2} + K - 3$.

Soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ incluant $F_{S,T}$.

Nous considérons trois cas :
– cas 1 : $|S| = 1$;
– cas 2 : $|S| \geq 2$ et $|U| = 0$;
– cas 3 : $|S| \geq 2$ et $|U| \geq 1$.

Dans chacun de ces cas, nous prouvons qu'il existe un coefficient β tel que $\alpha^T x$ est égal à $\beta(x(S, T) - x(S) - x(T))$.

Cas 1 : $|S| = 1$

Dans ce cas, selon (iv), u_1 appartient à $\{1, 2, 3\}$. De (i) et (ii) nous déduisons que $|T|$ est plus petit que $n - 3$. En conséquence, puisque S ne contient qu'un sommet, U en contient nécessairement au moins deux.

Soient $t' \in T$ et $t \in \{t_1, t_2\} \backslash \{t'\}$. En remarquant que $\min\{s_1, t, u_2\}$ est inférieur ou égal à 3, nous déduisons de $\mathcal{T}(\{s_1, t, u_2\}, \{t', u_1\}, \{u_1\})$ que : $\alpha_{t'} = 0$.

Grâce à (iii) nous savons que s_1 est plus petit que $t_{|T|}$. De plus, puisque seuls u_1 et s_1 sont susceptibles d'être plus petits que $\min(t_1, u_2)$, nous savons que $\alpha_{\min(t_1,u_2)}$ est nul. Ainsi, $\mathcal{T}(\{t_1, u_2\}, \{s_1, t_{|T|}, u_1\}, \{u_1\})$ donne : $\alpha_{s_1} = 0$. Pour tout $u \in U \backslash \{u_1\}$, la transformation $\mathcal{T}(\{u_1, u\}, \{t_{|T|}\}, \{u_1\})$ prouve que les composantes α_u sont nulles.

Enfin, pour tout $t, t' \in T$ distincts, nous prouvons que les expressions $\alpha_{s_1,t}$ et $-\alpha_{t,t'}$ sont égaux en utilisant $\mathcal{T}(\{s_1, t\}, \{t'\}, \{s_1\})$ et $\mathcal{T}(\{s_1, t, t'\}, \{u_1\}, \{t\})$.

Cas 2 : $|S| \geq 2$ et $|U| = 0$

Les conditions (i) et (ii) donnent $|T| \leq n - 3$, ce qui assure que S contient au moins trois sommets. Puisque K est plus grand que 2, (ii) implique : $|T| \geq 4$.

Dans la suite de la preuve de ce cas, soient $\bar{t}$ et $\tilde{t}$ tels que $\{\bar{t}, \tilde{t}\} = \{t_1, t_2\}$. Similairement, $\{\bar{s}, \tilde{s}\}$ est égal à $\{s_1, s_2\}$. Puisque U est un ensemble vide, $\min(\bar{t}, \bar{s})$ appartient à $\{1, 2, 3\}$ et $\alpha_{\min(\bar{t},\bar{s})}$ est donc nul.

Nous prouvons tout d'abord que pour tout $s \in S \backslash \{s_1, s_2\}$ et tout $t, t' \in T \backslash \{t_1, t_2\}$, $\beta = \alpha_{s,t} = -\alpha_{t,t'} = \alpha_{\bar{s},t_{|T|}} = -\alpha_{\bar{t},t_{|T|}}$.

Nous obtenons que $\alpha_{s,t} + \alpha_{t,t'}$ est nul grâce aux transformations représentées figures 4.21 et 4.22. Le reste est déduit des transformations $\mathcal{T}(\{s, t, t_{|T|}\}, \{s', t'\}, t_{|T|})$ et $\mathcal{T}(\{\bar{s}, \bar{t}, s, t, t_{|T|}\}, \{s', t'\}, t_{|T|})$.

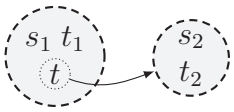


FIGURE *4.21* – $\mathcal{T}(\{s_1, t_1, t\}, \{s_2, t_2\}, t)$.

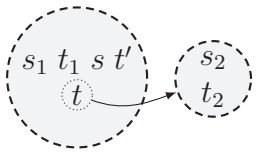


FIGURE *4.22* – $\mathcal{T}(\{s_1, t_1, s, t, t'\}, \{s_2, t_2\}, t)$.

Pour tout couple $(s, t) \in S \times T$, nous considérons les transformations $\mathcal{T}(\{\bar{s}, s, \bar{t}, t\}, \{t_{|T|}\}, \{s\})$ et $\mathcal{T}(\{\bar{s}, s, \bar{t}, \tilde{t}\}, \{t\}, \{\bar{t}, t\})$ qui amènent respectivement à

$$\alpha_{s,\bar{s}} + \alpha_{s,\bar{t}} + \alpha_{s,t} + \alpha_{t_{|T|}} = \alpha_s + \beta \tag{4.55}$$

et

$$\alpha_{\bar{t},\tilde{t}} + \alpha_{\tilde{s},\bar{t}} + \alpha_{s,\bar{t}} + \alpha_t = \alpha_{\bar{t}} + \alpha_{s,t} + \alpha_{\tilde{s},t} + \alpha_{\tilde{t},t}. \tag{4.56}$$

L'équation (4.55) montre que pour chaque $s \in S$, la valeur de $\alpha_{s,t}$ est la même pour tout $t \in T \backslash \{t_{|T|}\}$. Puisque α_{s,t_3} est égal à β, nous déduisons :

$$\alpha_{s,\bar{t}} = \beta \ \ \forall s \in S. \tag{4.57}$$

Après remplacement de $\alpha_{s,\bar{t}}$ par β dans l'équation (4.56), un raisonnement similaire peut être appliqué pour prouver que $\alpha_{\bar{s},t} = \beta \ \ \forall t \in T \backslash \{t_1, t_2\}$.

La transformation $\mathcal{T}(\{\bar{s}, \tilde{s}, s, \bar{t}, \tilde{t}, t\}, \{t_{|T|}\}, \{s\})$ et l'équation (4.55) donnent : $\alpha_{s,\bar{s}} = -\beta \ \ \forall s \in S \backslash \{s_1, s_2\}$.

Si t_3 est égal à 3, nous prouvons par symétrie que $\alpha_{\overline{t},t} = \alpha_{t_3,t}\ \forall t \in T\backslash\{t_1,t_2,t_3\}$. Ainsi, $\alpha_{\overline{t},t} = \beta$. Dans le cas contraire, s_1 appartient à $\{1,2,3\}$ le même résultat est obtenu via l'équation (4.55) et les transformations $\mathcal{T}(\{t,s_1\},\{t_{|T|}\},\{s_1\})$ et $\mathcal{T}(\{s,t,\overline{t}\},\{s,t_{|T|}\},\{t\})$.

L'équation (4.56) montre maintenant que les valeurs de α_t pour tout $t \in T\backslash\{t_1,t_2\}$ sont identiques. L'équation (4.55) appliquée à tout $s \in S\backslash\{s_1,s_2\}$ donne : $\alpha_s = \alpha_t$. Soit γ cette valeur.

Si t_3 ou s_3 est plus petit que 4, γ vaut 0 (puisque $\alpha_{\min(s_3,t_3)}$ est nul). Sinon, s_1 et t_1 appartiennent à $\{1,2,3\}$. La transformation $\mathcal{T}(\{\overline{s},\overline{t}\},\{t_{|T|}\},\{\overline{s}\})$ donne $\gamma = \alpha_{\max\{\overline{s},\overline{t}\}}$. Nous déduisons, ainsi, que α_t vaut 0. Puisque $\max\{\overline{s},\overline{t}\}$ et $\min\{\overline{s},\overline{t}\}$ valent tous deux 0, nous concluons que $\alpha_{\overline{s}}$ et $\alpha_{\overline{t}}$ sont nuls.

Enfin, pour prouver que α_{s_1,s_2} et α_{t_1,t_2} sont égaux à $-\beta$, nous utilisons les équations (4.55) et (4.56) avec $t = t_1$ et $s = s_1$.

Cas 3 : $|S| \geq 2$ et $|U| \geq 1$

Pour tout s dans S et tout t,t' distincts dans $T\backslash\{t_1\}$, nous prouvons tout d'abord que $\alpha_{s,t}$, $-\alpha_{t,t'}$ et $-\alpha_{t_1,t_{|T|}}$ sont tous égaux à une constante que nous appelons β.

Soit s' un sommet de $S\backslash\{s\}$. Les transformations représentées dans les figures 4.23 et 4.24 amènent à $\alpha_{s,t} = -\alpha_{t,t'}$ et

$$\alpha_{u_1}\mathbb{1}(t < u_1) + \alpha_{t_1,t} + \beta = \alpha_t\mathbb{1}(t < u_1). \tag{4.58}$$

L'équation (4.58) pour t égal à $t_{|T|}$ et la transformation $\mathcal{T}(\{u_1\}\ \{s',t,t_{|T|}\}\ \{t_{|T|}\})$ nous permettent d'obtenir : $\alpha_{t_1,t_{|T|}} = -\beta$.

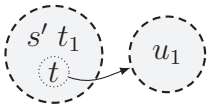


FIGURE *4.23* – $\mathcal{T}(\{s',t_1,t\},\{u_1\},\{t\})$.

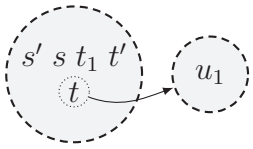


FIGURE *4.24* – $\mathcal{T}(\{s',s,t_1,t,t'\},\{u_1\},\{t\})$.

Nous montrons maintenant que les deux sommets d'indices les plus élevés d'un triplet $(s,t,u) \in S \times T\backslash\{t_1,t_{|T|}\} \times U$ ont des composantes de représentant égales à $\alpha_{t_{|T|}}$.

La transformation $\mathcal{T}(\{s,t\},\{t_{|T|}\},\{s\})$ donne

$$\alpha_{t_{|T|}} = \alpha_{\max\{s,t\}}. \tag{4.59}$$

Nous rappelons que pour une expression booléenne donnée, $\mathbb{1}\{B\}$ correspond à la fonction indicatrice prenant la valeur 1 si B est vraie et 0 sinon. Ainsi, la transformation, $\mathcal{T}(\{s,t,t_{|T|}\},\{u\},\{t\})$ permet d'obtenir $\alpha_t\,\mathbb{1}\{u < t < s\} + \alpha_u ßt < u = \alpha_t\,\mathbb{1}\{s < t < u\} + \alpha_s\,\mathbb{1}\{t < s\}$. , qui peut-être simplifiée en

$$\alpha_{\max\{u,t\}} = \alpha_{\max\{s,t\}}\,, \tag{4.60}$$

en utilisant le fait que $\mathbb{1}\{u < t < s\} - \mathbb{1}\{s < t < u\}$ est égal à $\mathbb{1}\{u < t\} - \mathbb{1}\{s < t\}$.

Si t est plus petit que s ou u, les équations (4.59) et (4.60) prouvent que les deux sommets les plus élevés du triplet (s,t,u) ont des variables de représentants égales à $\alpha_{t_{|T|}}$. Dans le cas contraire, la transformation $\mathcal{T}(\{u\},\{s,t,t_{|T|}\},\{s,t\})$ donne $\alpha_{\max\{u,s\}} = \alpha_{t_{|T|}}$, ce qui amène, en utilisant l'équation (4.59), au même résultat.

L'étape suivante de cette preuve consiste à montrer que les composantes α_i pour tout i de $V\backslash\{t_1\}$ sont nulles. Nous savons que ce résultat est vrai si au moins deux sommets du triplet (s_1,t_2,u_1), appartiennent à $\{1,2,3\}$ (car les variables de représentants des deux plus grands sommets du triplet sont égales à $\alpha_{t_{|T|}}$). Il reste à considérer les cas où seul un des trois sommets de ce triplet appartient à $\{1,2,3\}$. Appelons x ce sommet.

- Si $x = t_2$, t_3 appartient nécessairement à $\{1,2,3\}$. La composante $\alpha_{\min(s_1,u_1)}$ est égale à $\alpha_{t_{|T|}}$ et la transformation $\mathcal{T}(\{s_2,t_2,t_3\},\{s_1,t_{|T|},u_1\},\{t_3\})$ fournit le résultat.
- Si $x = u_1$, u_2 appartient nécessairement à $\{1,2,3\}$ et nous concluons avec $\mathcal{T}(\{u_1,u_2\},\{t_2\},\{u_2\})$.
- Si $x = s_1$, s_2 appartient nécessairement à $\{1,2,3\}$. Si le troisième sommet de cet ensemble est t_1, $\mathcal{T}(\{s_2,t_1\},\{t_{|T|}\},\{s_2\})$ permet d'obtenir le résultat. Dans le cas contraire, S contient au moins trois sommets et $(s_1,s_2,s_3) = (1,2,3)$. Pour conclure, nous utilisons $\mathcal{T}(P \cup \{s_1,s_3,t_1,t_3\}\{t_{|T|}\}\{s_3\})$ avec P successivement égal à $\emptyset$ puis $\{s_2,t_2\}$ et la transformation $\mathcal{T}(\{s_1,s_2,s_3,t_1,t_3,t_{|T|}\},\{t_2\},\{s_1,s_2,t_{|T|}\})$.

Soit $s \in \{s_1,s_2\}$. Si $t_1 \in \{4,\ldots,n\}$, nous montrons que α_{t_1} est nul grâce à $\mathcal{T}(\{s,t_1,u_1\},\{t_{|T|}\},\{t_1,t_{|T|}\})$ et $\mathcal{T}(\{s_1,s_2,t_1,t_2,u_1\},\{t_{|T|}\},\{t_1,t_{|T|}\})$. Finalement, nous prouvons, pour tout $s,s' \in S$ distincts et tout $t \in T\backslash\{t_1\}$, que les expressions α_{s,t_1}, $-\alpha_{s,s'}$ et $-\alpha_{t_1,t}$ sont égales à β via les transformations : $\mathcal{T}(\{s,t_1\},\{t_{|T|}\},\{s\})$, $\mathcal{T}(\{s,s',t_1,t_2\},\{t_{|T|}\},\{s\})$ et $\mathcal{T}(\{s_1,t_1,t\},\{s_2,t_{|T|}\},\{t\})$. □

4.3.2.8 Renforcement d'inégalités triangulaires

Le théorème 4.3.14 indique que les inégalités (4.3) ne définissent pas de facettes si i est plus grand que j et k. Cependant, elles peuvent être renforcées, dans les cas où i est différent de 3, en ajoutant la variable x_i au membre de gauche de l'inégalité :

$$x_{i,j} + x_{i,k} - x_{j,k} + x_i \leq 1. \tag{4.61}$$

Si x_i est nul, l'inégalité revient à une inégalité triangulaire et elle est donc valide. Si x_i vaut 1 dans une solution entière, les composantes $x_{i,j}$ et $x_{i,k}$ sont nécessairement nulles (puisque le sommet i ne peut être représentant que s'il est le sommet le plus petit de sa partie). L'inégalité (4.61) est donc valide pour $P_{n,K}$.

Pour trois sommets distincts i, j et k, soit $F_{i,j,k}$ la face de $P_{n,K}$ définie par (4.61).

Théorème 4.3.23 *Soient i,j et k trois sommets de V tels que $i > j > k$ et $i > 3$. Si $K \in \{3,\ldots,n-2\}$, l'inégalité* (4.61) *définit une facette de $P_{n,K}$ si et seulement si $(j > 3)$ ou $(K \neq n-2)$.*

Démonstration Montrons tout d'abord que (4.61) ne définit pas de facette si $j \leq 3$ et $K = n-2$. Soit π une $K-$partition telle que les sommets 1, 2 et 3 se trouvent dans la même partie. Puisque K vaut $n-2$ les $K-1$ autres parties sont nécessairement réduites à un unique sommet. Ainsi, le membre de gauche de (4.61) est égal à 0 et π ne figure donc pas dans $F_{i,j,k}$. Les sommets 1, 2 et 3 ne pouvant se trouver ensemble, nous vérifions que pour toute K-partition de $F_{i,j,k}$, l'équation suivante est vérifiée : $\sum_{i=4}^{n} x_i - x_{1,2} - x_{1,3} - x_{2,3} = K - 3$.

Soient t et t' tels que $\{t, t'\} = \{j, k\}$. Soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ incluant $F_{i,j,k}$. Dans le cas $j > 3$ ou $K \leq n-3$, nous montrons que : $\alpha^T = \alpha_{s,t}(x_{s,t} + x_{s,t'} - x_{t,t'} + x_s) \ \forall x \in F_{i,j,k}$.

Soit $U = \{u_1, u_2, \ldots, u_{|U|}\}$ l'ensemble de sommets $V \backslash \{i, j, k\}$ tels que $u_1 < u_2 < \ldots < u_{|U|}$. De façon similaire à la preuve du lemme 4.3.22, nous pouvons prouver que $\alpha_{u,v}$ vaut 0 pour tout $u \in U$ et tout $v \in V \backslash \{u\}$.

Nous considérons ensuite les transformations $\mathcal{T}(\{C\}, \{t, u_1\}, \{u_1\})$, avec $C = \{t', u_2\}$ si $K = n-2$ et $C = \{t', u_2, i\}$ sinon. En remarquant que $\min\{t', u_2\}$ est plus petit que 4, nous déduisons que α_t est nul.

Nous prouvons ensuite que α_s est égal à $\alpha_{s,t}$ grâce à $\mathcal{T}(\{t, u_1\}, \{i\}, \{t\})$. La transformation $\mathcal{T}(\{k, u_1, u\}, \{i\}, \{k, u_1\})$ donne : $\alpha_u = 0$, $\forall u \in U \backslash \{u_1\}$. Enfin, nous obtenons via $\mathcal{T}(\{i, t, t'\}, \{u\}, \{t\})$ l'équation : $\alpha_{t,t'} = -\alpha_{i,t}$. □

4.3.2.9 Inégalités de clique généralisées

Les *inégalités de clique* ont été introduites par Chopra et Rao [24]. Elles sont issues de la constatation que pour toute partition π contenant au plus K parties et tout ensemble $Z \subset V$ de taille $K+1$, il existe nécessairement dans π au moins deux sommets de Z se trouvant dans la même partie (*i.e.* : au moins une des arêtes de la clique engendrée par Z est contenue dans π). Ceci peut être représenté par l'*inégalité de clique* définie de la manière suivante :

$$x(Z) \geq 1. \tag{4.62}$$

Chopra et Rao présentent aussi les *inégalités de clique généralisées* qui consistent à ne plus uniquement considérer des ensembles Z de taille $K+1$, mais aussi des ensembles de taille plus grande. En effet, le nombre d'arêtes de la clique engendrée par Z se trouvant nécessairement dans la partition augmente avec la taille de Z. Ainsi, étant donné un ensemble $Z \subset V$ et soient q et r respectivement le quotient et le reste de la division euclidienne de Z par K (*i.e.* : $|Z| = qK + r$ avec $q \in \mathbb{N}$ et $r \in \{0, 1, \ldots, K-1\}$), l'inégalité de clique généralisée engendrée par Z est définie par :

$$x(Z) \geq \binom{q+1}{2} r + \binom{q}{2} (K - r). \tag{4.63}$$

Comme le représente l'exemple en figure 4.25, la borne inférieure de l'inégalité (4.63) correspond au nombre minimal d'arêtes de Z nécessairement contenues dans une partition – ce qui assure la validité de cette inégalité. Elle est obtenue en plaçant $q+1$ sommets de Z dans chacune des r premières parties et q sommets dans chacune des $(K-r)$ parties restantes.

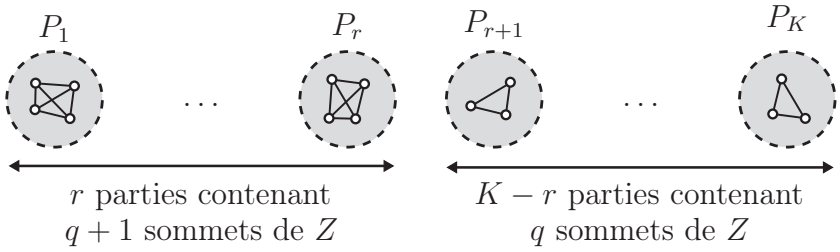


FIGURE *4.25 – Répartitions des sommets de Z dans une K-partition contenue dans la face F_Z (cas où q vaut 3).*

Ces inégalités ont aussi été étudiées par Labbé et Öszoy [81] dans le cas où chaque partie ne peut contenir moins de F_L sommets. Dans ce contexte, l'ensemble Z doit être de taille supérieure ou égale à $\lfloor \frac{n}{F_L} \rfloor$. Ji et Mitchell ce sont, eux aussi, intéressés à ces inégalités qu'ils nomment *inégalités de pigeon* [66].

Dans la suite, les sommets de $Z = \{z_1, \ldots, z_{|Z|}\}$ seront ordonnés de telle sorte que $z_1 < z_2 < \ldots < z_{|Z|}$. Les sommets de $U = V \backslash$, seront nommés et ordonnés de manière similaire. La face de $P_{n,K}$ induite par (4.63) est notée F_Z.

Le théorème suivant présente les conditions nécessaires et suffisantes sous lesquelles les inégalités de clique généralisées définissent des facettes de $P_{n,K}$.

Théorème 4.3.24 *Si $K \in \{3, \ldots, n-2\}$, étant donné un ensemble $Z \subset V$, l'inégalité* (4.63) *définit une facette de $P_{n,K}$ si et seulement si les conditions suivantes sont satisfaites :*

(i) $|U| \geq 1$ *et* $>u_1 \leq 3$ *;*
(ii) $z_{|Z|} = n$ *;*
(iii) $|Z| \in \{K+1, \ldots, 2K-1\}$.

Afin d'assurer que les partitions considérées dans la preuve du théorème 4.3.24 figurent dans F_Z, nous présentons le lemme suivant.

Lemme 4.3.25 *Soient V_1 et V_2 deux sous-ensembles disjoints de V et soit Z un sous-ensemble de V vérifiant les conditions du théorème 4.3.24. Il existe une $K-$partition de F_Z contenant V_1 et V_2 si $\{|V_1 \cap Z|, |V_2 \cap Z|\}$ est égal à $\{1, 2\}$.*

Démonstration Étant données les bornes fixées sur la taille de Z, q est nécessairement égal à un et $r \in \{1, \ldots, K-1\}$. En conséquence, toute K-partition de F_Z contient au moins une partie composée d'un seul sommet de Z et au moins une autre composée de deux.

Soit la K-partition $\pi = \{P_1, \ldots, P_K\}$ suivante :
– $P_1 = V_1$ et $P_2 = V_2$.
– Les parties P_3 à P_{r+1} contiennent chacune $q+1$ sommet de Z.
– Les parties P_{r+2} à P_K contiennent chacune q sommets de Z.
– Les sommets de U qui ne figurent pas dans V_1 ou V_2 sont dans P_K.

Cette construction est toujours possible puisque $|Z|$ est égal à $qK + r$. Il est aisé de vérifier – en calculant $x^\pi(Z)$ – que π est dans F_Z. □

Toute transformation $\mathcal{T}(P_1, P_2, R) \mapsto \{P'_1, P'_2\}$, considérée dans la preuve du théorème 4.3.24, est telle que les couples (P_1, P_2) et (P'_1, P'_2) vérifient les conditions imposées à V_1 et V_2 dans le lemme 4.3.25. La validité des transformations par rapport à F_Z est ainsi assurée.

Démonstration du théorème 4.3.24 Si la première condition du théorème 4.3.24 n'est pas satisfaite, l'ensemble $\{1, 2, 3\}$ est inclus dans Z. Les trois sommets de cet ensemble ne peuvent pas figurer dans une même partie puisque nous avons montré que q était égal à 1 (ce qui implique que chaque partie ne contienne qu'un ou deux sommets de Z). En conséquence, F_Z serait inclus dans l'hyperplan défini par $\sum_{i=4}^{n} x_i - x_{1,2} - x_{1,3} - x_{2,3} = K - 3$. Si (ii) n'est pas vérifiée, chaque sommet u supérieur à $z_{|Z|}$ ne peut être le représentant de sa partie puisque chaque partie contient au moins un sommet de Z. En conséquence, F_Z serait incluse dans les hyperplans induits par : $x_u = 0 \ \forall u > z_{|Z|}$. Enfin, si l'ensemble Z comporte plus de $2K - 1$ sommets, chaque partie contiendra nécessairement au moins deux sommets de Z et il est ainsi impossible que $z_{|Z|}$ soit un représentant. F_Z est, dans ce cas, incluse dans l'hyperplan induit par $x_{z_{|Z|}} = 0$.

Soit $F_\alpha = \{x \in P_{n,K} \mid \alpha^T x = \alpha_0\}$ une facette de $P_{n,K}$ contenant F_Z et soient $z_i < z_j < z_k$ trois sommets de Z. La transformation $\mathcal{T}(\{z_i, z_k\}, \{z_j\}, \{z_k\})$, montre tout d'abord que α_{z_i,z_k} et α_{z_j,z_k} sont égaux. Ainsi, pour un indice k donné et pour tout $j \in \{1, \ldots, k-1\}$, les composantes α_{z_j,z_k} sont égales à une constante que nous appelons β_k.

Pour tout j et k supérieurs à i, $\mathcal{T}(\{z_i, z_k\}, \{z_j\}, \{z_i\})$ donne

$$\beta_k - z_k = \beta_j - z_j. \tag{4.64}$$

Soient z, z' et z'' trois sommets distincts de Z. La transformation $\mathcal{T}(\{u_1, z\}, \{z'\}, \{u_1\})$ permet d'obtenir $\alpha_{z'} + \alpha_{u_1,z} = \alpha_z + \alpha_{u_1,z'}$. Ce résultat et la transformation $\mathcal{T}(\{u_1, z\}, \{z', z''\}, \{u_1\})$ donnent pour tout $h \in \{2, \ldots, |Z|\}$: $\alpha_{u_1,z_h} = 0$ et

$$\alpha_{u_1,z_1} + \alpha_{z_h} = \alpha_{z_1}. \tag{4.65}$$

Des équations (4.64) et (4.65), nous obtenons que pour tout $h \in \{2, \ldots, |Z|\}$ les composantes α_h sont identiques et qu'il en va de même pour les constantes β_h.

Si U contient un unique sommet, la preuve est terminée. En effet, dans ce cas z_2 appartient à $\{1, 2, 3\}$ et ainsi α_{z_2} vaut 0, ce qui donne via l'équation (4.65) $\alpha_{u_1,z_1} = 0$.

Si U contient deux sommets ou plus, nous prouvons que $\alpha_{u,z}$ vaut 0 pour tout $u \in U \backslash \{u_1\}$ et tout $z \in Z$ grâce à $\mathcal{T}(\{u_1, u, z_1\}, \{z\}, \{u_1, u\})$, $\mathcal{T}(\{u_1, u, z_1\}, \{z, z'\}, \{u_1, u\})$ et à l'équation (4.65).

Puisque $\min(u_2, z_1)$ et $\min(u_2, z_2)$ appartiennent nécessairement à $\{1, 2, 3\}$, nous montrons que α_{z_1} est égal à α_{z_2}, grâce à $\mathcal{T}(\{u_2, z_2\}, \{z_1\}, \{u_2\})$ ce qui nous amène, en utilisant l'équation (4.65), à $\alpha_{u_1,z_1} = 0$.

Si U contient deux sommets, α_z vaut 0, et il reste uniquement à prouver que α_{u_1,u_2} est nul, ce qui peut être réalisé par $\mathcal{T}(\{u_1, u_2, z_2\}, \{z_1\}, \{u_2\})$.

Sinon, soient u un sommet de U et U' un sous-ensemble quelconque de $U\backslash\{u\}$ contenant u_1 ou u_2. La transformation $\mathcal{T}(\{u, z_1, U'\}, \{z_{|Z|}\}, \{u\})$ donne

$$\alpha(u, U') + \alpha_z = \alpha_u \ \forall z \in Z. \tag{4.66}$$

Cette équation montre que $\alpha(u, U')$ est égal à une constante pour tout U'. Soient u' et u'' deux sommets distincts de $U\backslash\{u\}$. En choisissant successivement U' égal à $\{u''\}$ et $\{u', u''\}$, nous obtenons : $\alpha_{u,u'} = 0$. En conséquence, l'équation (4.66) donne : $\alpha_z = \alpha_u$ $\forall(u, z) \in U \times Z$. Puisque α_{u_1} vaut 0, il en est de même pour toutes les composantes relatives aux représentants. □

4.4 Conclusion

Dans ce chapitre, nous avons présenté deux formulations du problème de K-partitionnement sous la forme de programmes linéaires en nombres entiers. Les relaxations de ces deux formulations ont été comparés, ce qui a permis de montrer que la formulation (F_2) est meilleure du point de vue de la qualité de la solution de la relaxation linéaire.

La dimension de l'enveloppe convexe $P_{n,K}$ des solutions entières a été caractérisée pour toutes les valeurs de K possibles.

Les conditions sous lesquelles les inégalités présentes dans la formulation (F_1) constituent des facettes de $P_{n,K}$ ont été caractérisées. De plus, trois familles d'inégalités non triviales ont été étudiées (*i.e.* : les inégalité *2-chorded cycles*, les inégalité de 2-partitions et les inégalités de clique généralisées).

Ces résultats théoriques constituent une base prometteuse pour la résolution exacte du problème de K-partitionnement. Nous montrons dans la chapitre suivant comment ils peuvent être utilisés dans cette optique.

5 | Étude numérique

L'étude polyèdrale réalisée dans le chapitre précédent nous a permis de démontrer des conditions sous lesquelles des inégalités provenant de trois familles d'inégalités non triviales définissent des facettes du polyèdre $P_{n,K}$ (*i.e.* : les inégalités 2-*chorded cycles*, les inégalités de 2-partitions et les inégalités de clique généralisées). Dans ce chapitre, nous étudions comment utiliser judicieusement ces résultats théoriques afin d'accélérer la résolution exacte de problèmes de K-partitionnement. Nous évaluons, tout d'abord, en section 5.1 la qualité de chacune des trois familles d'inégalités en fonction du type de graphe complet considéré. Ensuite, nous présentons en section 5.1 un algorithme de plans coupants efficace se basant sur ces observations.

5.1 Évaluation de la qualité des inégalités définissant des facettes non triviales de $P_{n,K}$

La résolution de la relaxation linéaire d'un programme linéaire en nombres entiers (*i.e.* : le programme obtenu lorsque les contraintes d'intégrité des variables sont relaxées), permet d'obtenir une borne sur la solution optimale. Dans l'exemple présenté précédemment en figure 3.4a page 78, l'optimum x^* de la relaxation linéaire (■) fourni une borne supérieure de valeur $7,5$ tandis que la valeur optimale (●) est égale à 7.

La solution x^* d'une relaxation linéaire peut être améliorée en lui ajoutant une contrainte $a^T x \leq b$ (aussi appelée *coupe*) vérifiée par l'ensemble des solutions entières mais violée par x^*. Dans l'exemple, l'inégalité $x_2 \leq 4,5$ est violée par la relaxation $x^* = (x_1^*, x_2^*)$, puisque x_2^* vaut 5. En ajoutant cette contrainte à la formulation de la relaxation linéaire (figure 3.4b), puis en résolvant cette dernière à nouveau, une meilleure borne – égale à $7,29$ – est obtenue.

Afin d'améliorer le plus rapidement possible cette borne, nous souhaitons ajouter à la relaxation linéaire les coupes qui entraînent la plus forte amélioration de x^*. C'est la raison pour laquelle nous évaluons dans cette section l'amélioration apportée à x^* lors de l'ajout de coupes de chacune des trois familles d'inégalités identifiées au chapitre 4.

Pour ce faire, il est tout d'abord nécessaire de définir des algorithmes permettant d'identifier les coupes entraînant une amélioration de la solution de la relaxation linéaire.

5.1.1 Algorithmes de séparations

Le problème consistant à trouver une coupe séparant la relaxation linéaire x^* de l'enveloppe convexe des points entiers est appelé *problème de séparation*.

Afin d'évaluer l'efficacité d'une famille d'inégalités, l'idéal serait d'ajouter à la relaxation linéaire des coupes de ce type jusqu'à ce que plus aucune d'entre elles ne permette de l'améliorer. Ceci nécessite une résolution exacte du problème de séparation qui n'est, cependant, pas toujours envisageable (de part la complexité du problème de séparation ou parce qu'aucun algorithme connu ne le permet).

Dans cette sous-section, nous présentons un algorithme de séparation pour chacune des familles d'inégalités étudiées. Contrairement aux deux autres, l'algorithme utilisé pour les inégalités 2-*chorded cycles* permet de résoudre exactement le problème de séparation.

Séparation des inégalités 2-chorded cycles

En 1996, Müller [97] adapte une approche introduite par Barahona et Mahjoub [14] afin de permettre la séparation en temps polynomial des inégalités dites de type *odd closed walk* dans des graphes orientés. Müller montre, de plus, que le même algorithme peut être appliqué à des graphes non orientés afin de permettre la séparation d'une classe d'inégalités incluant les inégalités 2-*chorded cycles*. Afin d'évaluer au mieux l'impact des inégalités 2-*chorded cycles*, nous souhaitons que l'algorithme sépare une classe d'inégalités aussi proche que possible de cette famille. Ainsi, nous avons adapté cet algorithme pour qu'il sépare uniquement les inégalités de 2-*chorded cycles* associées à des cycles pouvant contenir des répétitions.

Pour ce faire, nous définissons un graphe orienté $H = (V_H, A_H)$ – associé au graphe d'origine $G = (V, E)$ – tel que pour chaque arête $ij \in E$, A_H contienne (voir exemple figure 5.1) :

– huit sommets : u_1^{ij}, u_2^{ij}, v_1^{ij}, v_2^{ij}, u_1^{ji}, u_2^{ji}, v_1^{ji} et v_2^{ji} ;

– quatre arcs : (u_1^{ij}, u_2^{ij}), (v_1^{ij}, v_2^{ij}), (u_1^{ji}, u_2^{ji}), (v_1^{ji}, v_2^{ji}) de poids x_{ij}.

De plus, pour toute paire d'arêtes ij et ik de E comportant un sommet en commun, A_H comporte les quatre arcs suivants : (u_2^{ji}, v_1^{ik}), (v_2^{ji}, u_1^{ik}), (u_2^{ki}, v_1^{ij}) et (v_2^{ki}, u_1^{ij}) de poids $-x_{jk} - \frac{1}{2}$.

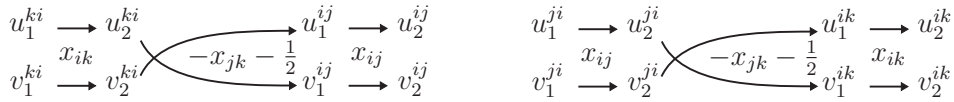


FIGURE *5.1 – Sommets et arcs de H associés aux arêtes (ij) et (ik) in E.*

Soit $C = \{c_1, \ldots, c_{2p+1}\}$ un cycle de taille impaire bde G. Par construction, C induit un chemin dans H allant de $u_1^{c_1,c_2}$ à $v_1^{c_1,c_2}$ (voir exemple figure 5.2), de poids

$$\begin{aligned} x_{c_1,c_2} - \tfrac{1}{2} - x_{c_1,c_3} + \ldots + x_{c_{2p+1},c_1} - \tfrac{1}{2} - x_{c_{2p+1},c_2} &= x(C) - x(\overline{C}) - \tfrac{2p+1}{2} \\ &= x(C) - x(\overline{C}) - \lfloor \tfrac{|C|}{2} \rfloor - \tfrac{1}{2}. \end{aligned}$$

Ainsi, il existe un cycle correspondant à une inégalité 2-*chorded cycle* (*i.e.* : inégalité (4.39) page 105) violée si et seulement si il existe deux sommets c_1 et c_2 de V tels qu'il

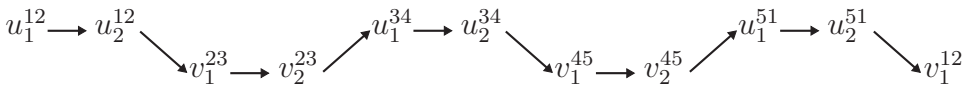


FIGURE *5.2 – Chemin de H correspondant au cycle $C = \{1, 2, 3, 4, 5\}$ de G.*

est possible de trouver un chemin dans H de longueur supérieure à $-\frac{1}{2}$ allant de $u_1^{c_1,c_2}$ à $v_1^{c_1,c_2}$.

L'approche de Müller pour les graphes non orientés ne considère que quatre sommets par arête (u_1^{ij}, u_2^{ij}, v_1^{ij} et v_2^{ij}). En conséquence, un chemin de H allant de u_1^{ij} à v_1^{ij} correspond à une séquence d'arêtes de G telle que tous les couples d'arêtes consécutives possèdent un sommet en commun. Une telle séquence n'est pas nécessairement un cycle (*e.g* : $\{ij, ik, il\}$) et c'est la raison pour laquelle la classe d'inégalités séparée par cet algorithme est bien plus large que les inégalités *2-chorded cycles*. L'ajout de quatre sommets additionnels par arête permet de donner une orientation aux arêtes considérées dans la séquence et assurent que le résultat obtenu est un cycle (dans lequel un sommet est susceptible d'apparaître plusieurs fois).

Après création du graphe H, les inégalités *2-chorded cycles* violées peuvent être identifiées en calculant pour toute arête ij de E le chemin le plus court entre les sommets u_1^{ij} et v_1^{ij}. Nous calculons ces chemins grâce à l'algorithme de Floyd-Warshall [5]. Chaque chemin de poids supérieur à $-\frac{1}{2}$ correspondra à une inégalité violée qui sera ajoutée à la relaxation linéaire. La solution de la relaxation est alors recalculée et le processus est répété jusqu'à ce qu'aucune inégalité violée ne soit trouvée.

Séparation des inégalités de 2-partitions

Nous utilisons un algorithme de type Kernighan-Lin [85, 73] pour séparer les inégalités de 2-partitions. Deux ensembles S et T sont, tout d'abord, générés aléatoirement. Soit U l'ensemble des sommets qui ne sont ni dans S ni dans T (*i.e.* : $U = V \backslash \{S \cup T\}$). Plusieurs types de transformation de ces trois ensembles – représentés figure 5.3 – sont considérés :

- déplacer une sommet d'un ensemble à un autre (deux transformations possibles par sommet) ;
- échanger deux sommets se trouvant dans des ensembles différents (une transformation possible par arête non contenue dans un des ensembles S, T ou U).

Chacune de ces transformations engendre une variation du score de la coupe que nous évaluons. Les ensembles S, T et U sont, ensuite, mis à jour en fonction de la transformation permettant de maximiser ce score. Le score d'une coupe $a^T x \leq b$ est, habituellement, calculé en soustrayant b à $a^T x$. Cependant, nous avons ici choisi d'utiliser le ratio $\frac{a^T x}{b}$. Ce choix a été justifié par une expérience dont les résultats sont présentés en table 5.1. Pour chaque couple (n, K), la valeur représentée correspond à la moyenne sur 5 graphes du pourcentage d'amélioration de la solution de la relaxation linéaire après ajout de 100 coupes lorsque que le score $b - a^T x$ est utilisé pour évaluer la qualité de ces dernières. Le score qui figure entre parenthèses montre la différence obtenue lorsque le ratio $\frac{a^T x}{b}$ est utilisé. Nous pouvons constater qu'à l'exception d'un unique cas, le ratio fournit toujours des résultats significativement meilleurs.

Le processus de choix puis d'application de la meilleure transformation est ensuite répété – en imposant que chaque transformation ne puisse être appliquée qu'une seule fois – jusqu'à ce que toutes les transformations aient été considérées. La meilleure coupe

n	K					
	2	**3**	**4**	**5**	**6**	**7**
20	179.6 (+0.8)	120.7 (+0.5)	108.2 (+1.1)	94.2 (+1.4)	78.7 (+0.5)	63.2 (+0.1)
21	160.4 (+0.5)	105.4 (+0.5)	85.4 (+0.7)	75.0 (+0.8)	68.0 (+0.2)	56.8 (+0.2)
22	183.6 (+0.7)	126.8 (+0.7)	105.4 (+0.8)	90.0 (+0.6)	79.7 (-0.1)	73.2 (+0.4)
23	208.9 (+1.2)	138.0 (+1.7)	112.3 (+1.4)	101.8 (+1.2)	92.6 (+0.6)	82.6 (+0.4)
24	207.1 (+2.5)	135.1 (+1.7)	110.8 (+1.4)	95.1 (+0.9)	82.1 (+1.1)	72.6 (+0.7)
25	230.8 (+3.4)	148.1 (+2.0)	116.8 (+1.5)	100.0 (+1.0)	88.7 (+1.5)	79.0 (+1.7)
26	213.5 (+2.9)	141.8 (+1.8)	120.1 (+1.3)	107.3 (+2.0)	96.4 (+1.8)	87.0 (+1.0)
27	221.5 (+2.6)	143.9 (+2.3)	113.8 (+2.8)	97.4 (+2.6)	86.5 (+2.8)	79.9 (+1.4)
28	221.9 (+5.7)	151.5 (+1.6)	120.7 (+2.8)	106.1 (+2.0)	91.4 (+2.2)	82.2 (+1.3)
29	223.7 (+5.4)	147.0 (+2.0)	120.1 (+2.4)	105.7 (+2.2)	93.8 (+2.3)	85.3 (+2.2)
30	251.5 (+4.1)	161.0 (+1.4)	128.3 (+2.6)	110.5 (+2.4)	97.8 (+2.8)	86.8 (+2.1)

TABLE *5.1 – Moyenne sur* 5 *graphes du pourcentage d'amélioration de la relaxation linéaire après ajout de 100 coupes de type* 2*-partitions obtenues avec l'algorithme de type Kernighan-Lin. Pour chaque couple* (n, K)*, le premier nombre correspond aux résultats obtenus lorsque que le critère d'évaluation d'une coupe* $a^T x \leq b$ *considéré est la différence* $b - a^T x$*. Le nombre entre parenthèse représente le gain en pourcentage lorsque le ratio* $\frac{a^T x}{b}$ *est utilisé par rapport au premier critère.*

rencontrée durant l'ensemble du processus est ensuite sélectionnée. Cet algorithme peut être répété – en utilisant les meilleurs ensembles S et T de l'itération précédente comme point de départ – tant que la coupe est améliorée.

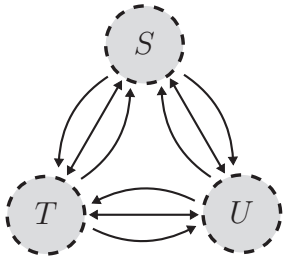


FIGURE *5.3 – Représentation des différents types de transformations considérés dans l'algorithme de type Kernighan-Lin utilisé pour séparer les inégalités de 2-partitions.*

Séparation des inégalités de clique généralisées

L'algorithme utilisé pour séparer les inégalités de clique généralisées est très similaire à celui employé pour les inégalités de 2-partitions. La différence réside dans le fait que seuls deux ensembles sont considérés $Z \subset V$ et $U = V \backslash Z$, ce qui implique un nombre plus réduit de transformations envisageables (voir représentation en figure 5.4)

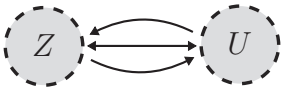


FIGURE *5.4 – Représentation des différents types de transformations considérés dans l'algorithme de type Kernighan-Lin utilisé pour séparer les inégalités de clique généralisées.*

5.1.2 Amélioration de la relaxation par famille d'inégalités

Soit r la valeur de la borne fournie par la résolution de la relaxation linéaire. Afin de calculer l'amélioration que peut apporter une famille d'inégalités à r, nous réalisons successivement les étapes suivantes :

1. calcul de la relaxation linéaire ;
2. ajout de coupes violées de la famille d'inégalités considérée.

Ce procédé est ainsi répété jusqu'à ce qu'aucune coupe violée ne soit identifiée. Une solution améliorée de la relaxation linéaire correspondant à la borne r^* est ainsi obtenue. Pour évaluer l'amélioration apportée, nous calculons le *pourcentage d'amélioration* défini par

$$\frac{|r - r^*|}{r} \times 100. \tag{5.1}$$

Afin de réaliser l'évaluation, nous avons généré pour chaque valeur de n considérée, 100 graphes complets dont le poids des arêtes a été généré aléatoirement entre 0 et 500. Pour chaque couple (n, K), l'amélioration que nous fournissons correspond à la moyenne des pourcentages d'amélioration obtenue sur les 100 graphes correspondants.

Le type de graphes considéré peut avoir une forte influence sur l'amélioration apportée par une famille d'inégalités ou sur la résolution du problème en lui-même. Par exemple, comme nous l'avons mentionné, Goldschmidt *et al.* [48] ont montré que le problème de K-partitionnement était polynomial lorsque le poids des arêtes était négatif, même si ce polynôme dépend fortement de K (complexité pseudo-polynomiale). Ainsi, pour obtenir une meilleure évaluation de la qualité des familles d'inégalités, nous calculons le pourcentage d'amélioration moyen obtenu dans les trois cas suivants :

- le poids des arêtes appartient à $[0, 500]$ (ce sont les graphes décrit ci-dessus) ;
- le poids des arêtes appartient à $[-250, 250]$ (les poids sont décalés de -250) ;
- le poids des arêtes appartient à $[-500, 0]$ (les poids sont décalés de -500).

Graphes dont les arêtes ont des poids positifs

Les graphes dont les arêtes ont des poids positifs sont ceux qui sont susceptibles d'être les plus difficiles car ils sont les plus éloignés du cas polynomial où les arêtes ont des poids négatifs. Les inégalités les plus efficaces seront donc tout particulièrement intéressantes pour la résolution de cas les plus ardus.

La table 5.2 présente les résultats obtenus avec les inégalités *2-chorded cycles*. Nous pouvons premièrement observer que le pourcentage moyen d'amélioration est faible (inférieur à 5%). Ceci indique que les chances pour que ces inégalités soient efficaces sur ce type de graphe est faible. De plus, nous remarquons que l'amélioration diminue lorsque K s'approche de n. Cette observation est valable pour les trois tableaux de résultats liés à ces graphes. Ceci peut, tout d'abord, s'expliquer par le fait que le nombre de solutions

entières réalisables diminue fortement lorsque K est proche de n. La seconde raison est que, comme l'illustre la table 5.3 la relaxation linéaire fournit souvent la solution optimale dans cette partie du tableau, ce qui rend impossible toute amélioration pour ces instances.

n	**K**								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7	0,2	0,2	0,1	0,0	0,0				
8	0,4	0,5	0,4	0,0	0,0	0,0			
9	0,8	0,7	0,8	0,3	0,0	0,0	0,0		
10	1,2	1,5	0,9	0,7	0,1	0,1	0,0	0,0	
11	0,9	1,1	0,9	0,6	0,2	0,0	0,0	0,0	0,0
12	1,7	1,9	1,7	1,4	1,0	0,6	0,2	0,0	0,0
13	1,8	2,0	1,9	1,5	0,6	0,3	0,0	0,0	0,0
14	2,2	2,5	2,5	2,1	1,3	0,7	0,4	0,1	0,0
15	2,1	2,9	3,0	2,5	1,9	1,1	0,5	0,2	0,0
16	2,5	2,8	2,7	2,1	1,7	1,4	0,7	0,4	0,1
17	2,9	3,2	3,0	2,7	2,3	1,7	1,4	0,7	0,3
18	2,9	3,4	3,3	2,9	2,5	1,9	1,3	0,8	0,4
19	3,2	3,6	3,9	3,7	3,2	2,6	2,0	1,2	0,7
20	4,1	4,7	4,8	4,6	4,1	3,4	3,0	2,6	1,6

TABLE *5.2 – Amélioration moyenne de la relaxation linéaire par utilisation d'inégalités 2-chorded cycles sur 100 graphes complets dont le poids des arêtes est positif.*

Les résultats obtenus par les inégalités de 2-partitions sont présentés en table 5.4. Ils montrent une amélioration moyenne plus significative. Enfin, la table 5.5 montre que les inégalités de clique généralisées apportent une amélioration spectaculaire de la solution de la relaxation, allant parfois jusqu'à dépasser les 1000% d'amélioration dans les meilleurs cas. Ces inégalités seront donc à générer en priorité dans le cas du partitionnement de graphes dont le poids des arêtes est positif.

Graphes dont les arêtes ont des poids de signe quelconque

Dans le cas où le poids des arêtes du graphe peut être à la fois positif ou négatif, les inégalités *2-chorded cycles* fournissent des résultats qui restent faibles comme le présente la table 5.7. Nous pouvons remarquer que les valeurs sont plus équitablement réparties en fonction de K, excepté pour les premières lignes du tableau pour lesquelles les premières colonnes ont des pourcentages d'amélioration plus faible. Ceci peut une nouvelle fois s'expliquer par la qualité de la relaxation linéaire pour ces couples de valeurs (n, K), comme l'illustre la table 5.6 qui représente le nombre de problèmes résolus directement par la relaxation linéaire pour ces graphes.

Les valeurs obtenues pour les inégalités de 2-partitions, représentées en table 5.8, sont réparties de manière similaire. Bien que plus faibles que pour les graphes précédents, elles sont globalement deux fois plus élevées que celles des inégalités *2-chorded cycles*.

n	K																	
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
7	0	0	52	91	100													
8	0	0	10	61	92	100												
9	0	0	0	44	79	94	100											
10	0	0	0	12	47	75	95	100										
11	0	0	0	0	15	50	77	94	100									
12	0	0	0	0	8	29	70	91	99	100								
13	0	0	0	0	0	8	37	60	87	99	100							
14	0	0	0	0	0	0	20	45	72	91	98	100						
15	0	0	0	0	0	0	6	28	61	84	94	99	100					
16	0	0	0	0	0	0	3	13	31	66	80	91	98	100				
17	0	0	0	0	0	0	0	3	18	46	69	84	90	96	100			
18	0	0	0	0	0	0	0	0	6	19	51	70	91	97	99	100		
19	0	0	0	0	0	0	0	0	1	13	34	56	76	92	96	99	100	
20	0	0	0	0	0	0	0	0	0	4	12	37	60	78	94	99	100	100

TABLE *5.3 – Pour chaque couple (n, K), nombre de graphes parmi les 100 considérés et dont la solution de la relaxation linéaire est entière.*

n	K								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7	2,9	2,9	2,0	0,1	0,0				
8	4,5	4,9	3,5	1,9	0,2	0,0			
9	6,8	8,0	8,3	5,7	1,3	0,5	0,0		
10	7,5	9,3	8,1	7,1	3,5	1,6	0,5	0,0	
11	9,6	10,9	10,6	8,2	5,3	2,9	1,6	0,8	0,0
12	11,4	13,2	12,6	10,6	7,7	4,2	2,8	1,1	0,1
13	12,8	15,3	15,6	13,5	9,5	6,2	3,4	2,1	0,5
14	14,9	17,0	17,8	15,7	12,8	9,4	5,7	3,1	1,8
15	16,9	19,6	20,5	19,3	17,3	14,3	10,6	6,1	3,5
16	17,8	19,9	20,5	19,8	17,9	14,9	11,3	8,2	5,0
17	19,1	21,5	22,1	21,3	20,2	17,3	14,0	10,6	7,3
18	21,1	23,3	23,8	23,2	21,3	18,7	15,6	12,1	8,4
19	22,5	25,0	26,0	26,2	25,7	23,5	20,1	16,5	12,1
20	25,1	28,6	30,1	29,6	28,4	26,1	22,8	18,9	14,4

TABLE *5.4 – Amélioration moyenne de la relaxation linéaire par utilisation d'inégalités de 2-partitions sur 100 graphes complets dont le poids des arêtes est positif.*

n	K								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7	130,1	46,4	11,9	1,1	0,0				
8	176,3	78,5	24,8	8,0	0,9	0,0			
9	232,1	111,1	52,1	17,8	3,9	1,1	0,0		
10	298,0	153,6	73,7	32,2	15,9	4,3	0,4	0,0	
11	368,7	197,8	104,3	49,6	17,8	7,5	3,8	0,8	0,0
12	434,0	239,2	135,5	73,0	31,2	12,4	5,4	2,3	0,6
13	505,7	288,4	171,5	98,5	49,2	19,9	9,1	5,1	2,0
14	593,6	343,6	206,9	119,2	68,2	33,2	14,7	6,3	3,4
15	673,2	395,4	244,7	152,0	95,5	54,2	25,5	13,6	6,1
16	782,9	465,0	294,0	181,8	112,1	68,1	35,4	17,6	9,8
17	855,6	515,5	328,1	214,8	137,8	84,9	48,2	23,7	13,4
18	964,1	582,2	376,3	247,7	159,8	102,0	63,6	36,9	19,6
19	1 041,6	637,4	421,9	289,1	199,5	134,0	88,2	52,4	26,8
20	1 186,6	739,3	491,1	336,3	228,9	153,1	103,1	63,5	35,9

TABLE *5.5 – Amélioration moyenne de la relaxation linéaire par utilisation d'inégalités de clique généralisées sur 100 graphes complets dont le poids des arêtes est positif.*

n	K																	
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
7	45	59	19	9	4													
8	37	55	30	6	4	0												
9	34	48	21	1	0	0	0											
10	20	53	31	4	3	0	0	0										
11	16	37	16	4	1	1	0	0	0									
12	8	21	16	4	0	0	0	0	0	0								
13	12	17	11	2	1	0	0	0	0	0	0							
14	6	14	9	3	0	0	0	0	0	0	0	0						
15	2	10	9	3	0	0	0	0	0	0	0	0	0					
16	2	5	5	1	0	0	0	0	0	0	0	0	0	0				
17	2	10	4	2	0	0	0	0	0	0	0	0	0	0	0			
18	2	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
19	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE *5.6 – Pour chaque couple (n, K), nombre de graphes parmi les 100 dont le poids des arêtes est entre -250 et 250 et dont la solution de la relaxation linéaire est entière.*

n	K								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7	0,1	0,0	0,2	0,2	0,0				
8	0,2	0,1	0,1	0,2	0,2	0,0			
9	0,3	0,3	0,3	0,5	0,8	0,4	0,0		
10	0,4	0,4	0,4	0,6	0,8	0,8	0,4	0,0	
11	0,4	0,4	0,4	0,6	1,0	1,5	1,2	0,5	0,0
12	0,9	0,7	0,7	0,9	1,3	2,1	2,5	1,7	0,7
13	1,0	1,0	1,0	1,1	1,3	1,9	2,7	2,8	1,7
14	1,2	1,2	1,2	1,3	1,5	1,9	2,9	3,5	2,8
15	2,1	2,1	2,2	2,3	2,5	2,7	3,5	4,2	4,4
16	2,6	2,7	2,7	2,7	2,8	3,1	3,6	4,7	5,2
17	3,1	3,2	3,2	3,3	3,3	3,4	3,7	4,5	5,5
18	4,0	4,0	4,0	4,0	4,1	4,2	4,3	4,8	5,9
19	4,5	4,8	4,9	5,0	5,0	5,0	5,0	5,2	5,9
20	5,5	5,8	5,8	5,8	5,8	5,8	5,9	6,0	6,4

TABLE *5.7 – Amélioration moyenne de la relaxation linéaire par utilisation d'inégalités 2-chorded cycles sur 100 graphes complets dont le poids des arêtes peut être positif ou négatif.*

Enfin, les pourcentages moyen d'amélioration obtenus avec les inégalités de clique généralisées, représentés en table 5.9, sont bien plus faibles que dans le cas des graphes qui possèdent des arêtes de poids positif. Cette détérioration peut s'expliquer par le fait que les inégalités de clique généralisées imposent qu'un certain nombre d'arêtes de la clique engendrée par un ensemble $Z \subset V$ figurent dans la partition. Puisque nous cherchons à minimiser le poids de la K-partition, lorsqu'une arête $ij \in E$ possédant un poids $w_{i,j}$ positif se trouve dans une partition, elle détériore la valeur de la fonction objectif. Au contraire, si $w_{i,j}$ est négatif, la fonction objectif est améliorée (*i.e.* : diminuée) par la présence de ij dans une partition. Ainsi, le nombre d'arêtes dans la partition optimale a tendance à être plus faible dans le cas de graphes ne contenant que des arêtes de poids positifs. Ainsi, les inégalités de clique généralisées qui imposent la présence d'un nombre minimum d'arêtes dans la partition sont plus souvent violées lorsque des graphes de poids positifs sont considérés.

En conclusion, pour ce type de graphes, les inégalités de 2-partitions sont généralement à préférer, excepté lorsque K est très faible (*i.e.* : 2 ou 3) où nous utilisons en priorité les inégalités de clique généralisées.

Graphes dont les arêtes ont des poids négatifs

Pour ces graphes, qu'elle que soit la valeur du couple (n, K), aucune coupe violée n'est identifiée par les trois algorithmes de séparation. Le pourcentage d'amélioration est donc nul. Ainsi, tant qu'aucune famille d'inégalités permettant une amélioration n'est pas identifiée, un algorithme de résolution basé sur la recherche de coupe risque de s'avérer

n	K								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7	1,3	1,3	1,8	3,0	5,8				
8	1,6	1,1	1,1	2,4	5,0	7,6			
9	1,5	1,3	1,6	3,4	5,4	8,3	10,7		
10	1,8	1,4	1,2	2,2	3,8	6,4	10,1	13,5	
11	2,3	1,9	1,8	2,6	4,0	6,9	9,1	11,7	13,2
12	3,0	2,3	2,2	2,9	4,4	6,6	9,5	11,8	13,2
13	3,7	3,4	3,5	3,9	4,8	6,6	9,6	12,5	14,4
14	4,3	4,0	3,9	4,2	5,0	6,5	8,5	11,1	13,1
15	5,3	5,0	5,1	5,3	5,9	6,8	8,8	11,4	14,0
16	5,6	5,5	5,5	5,6	6,1	7,0	8,3	11,0	13,1
17	7,3	7,1	7,3	7,3	7,6	8,1	8,8	10,5	13,1
18	8,0	7,7	7,9	7,9	8,2	8,3	9,0	10,3	12,5
19	9,2	9,5	9,6	9,6	9,9	9,8	10,3	11,1	12,5
20	10,5	10,6	10,6	10,6	10,8	10,7	10,9	11,7	12,7

TABLE *5.8 – Amélioration moyenne de la relaxation linéaire par utilisation d'inégalités de 2-partitions sur 100 graphes complets dont le poids des arêtes peut être positif ou négatif.*

peu efficace lorsqu'il est appliqué au partitionnement de graphes valués négativement, c'est à dire dans la variante pseudo polynomiale du problème de K-partitionnement.

Conclusion

Nous remarquons que la nature des données et le type de coupes ajoutées influencent grandement l'amélioration de la solution de la relaxation linéaire.

Tout comme l'avaient remarqué Grötschel et Wakabayashi dans le cas du partitionnement non contraint, les inégalités *2-chorded cycles* fournissent des résultats significativement moins bons que les deux autres familles d'inégalités. C'est pourquoi, leur utilisation n'est pas privilégiée dans notre algorithme de résolution.

Les inégalités de 2-partitions fournissent de bons résultats en moyenne et elles semblent efficaces quelles que soient les valeurs du couple (n, K) considéré.

Enfin, les inégalités de clique généralisées entraînent les améliorations les plus importantes pour les graphes comportant des poids positifs. Elles sont à privilégier en priorité pour ces derniers.

n	K								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
7	21,0	2,2	0,2	0,0	0,0				
8	22,2	2,1	0,1	0,0	0,0	0,0			
9	11,2	1,5	0,2	0,0	0,0	0,0	0,0		
10	16,7	2,6	0,2	0,0	0,0	0,0	0,0	0,0	
11	16,2	2,7	0,4	0,0	0,0	0,0	0,0	0,0	0,0
12	17,5	3,2	0,5	0,1	0,0	0,0	0,0	0,0	0,0
13	17,2	4,0	0,8	0,1	0,0	0,0	0,0	0,0	0,0
14	17,3	4,3	0,8	0,1	0,0	0,0	0,0	0,0	0,0
15	19,5	5,7	1,2	0,1	0,0	0,0	0,0	0,0	0,0
16	20,2	6,1	1,6	0,2	0,0	0,0	0,0	0,0	0,0
17	22,5	7,7	2,1	0,3	0,0	0,0	0,0	0,0	0,0
18	23,6	8,1	2,5	0,4	0,0	0,0	0,0	0,0	0,0
19	24,8	9,2	2,9	0,5	0,0	0,0	0,0	0,0	0,0
20	24,9	9,8	3,2	0,5	0,0	0,0	0,0	0,0	0,0

TABLE *5.9 – Amélioration moyenne de la relaxation linéaire par utilisation d'inégalités de clique généralisées sur 100 graphes complets dont le poids des arêtes peut être positif ou négatif.*

5.2 Résolution numérique par un algorithme de plans coupants

Nous choisissons de nous orienter vers un algorithme de type plans coupants car un algorithme similaire a prouvé par le passé son efficacité pour la résolution de problèmes de partitionnement dans le cas non contraint [49].

5.2.1 Description générale de l'algorithme

Le principe d'un algorithme de type plans coupants est d'améliorer la solution de la relaxation linéaire jusqu'à ce qu'elle corresponde à une solution entière. Pour ce faire, les deux étapes suivantes sont successivement répétées :

(E_1) calcul de la relaxation linéaire x^* du problème ;

(E_2) obtention d'une contrainte violée par x^* mais valide pour l'ensemble des solutions entières du problème.

L'algorithme est exécuté jusqu'à ce qu'une des trois conditions d'arrêt suivantes soit rencontrée :

(T_1) x^* est entier (dans ce cas la solution optimale est obtenue) ;

(T_2) aucune coupe permettant de séparer x^* n'a été trouvée ;

(T_3) le temps d'exécution maximal accordé à l'algorithme est atteint.

Nous utilisons dans notre algorithme la formulation étendue (F_2) dont la relaxation linéaire est au moins aussi bonne que celle de (F_1). De plus, lorsque la majorité des arêtes du graphe possède un poids positif, nous ajoutons à cette formulation les inégalités de clique généralisées correspondant aux ensembles Z de taille n et $n-1$. Ces $n+1$ inégalités ont empiriquement montré une nette amélioration de la relaxation linéaire comme le montre la table 5.10.

n	**K**								
	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
20	490.3	367.3	226.9	143.6	95.1	68.2	48.7	22.8	6.0
21	451.8	311.2	179.8	114.9	80.4	52.3	42.6	26.9	7.4
22	506.0	351.0	236.8	158.7	113.5	66.7	58.8	42.9	22.6
23	540.0	404.3	277.0	185.2	126.9	88.0	62.3	48.8	30.4
24	596.1	444.3	298.5	218.6	161.9	126.3	84.6	67.9	45.5
25	613.9	464.7	330.3	241.3	165.1	129.5	87.0	77.2	58.5
26	674.3	502.0	349.7	233.0	169.8	133.0	97.9	79.3	66.8
27	693.1	522.2	366.5	256.1	186.8	137.4	100.8	71.4	59.9
28	753.5	537.4	352.8	246.0	181.4	132.8	106.0	70.8	60.7
29	732.3	561.7	373.4	264.3	192.2	135.7	111.7	78.4	62.4
30	716.6	565.2	397.6	282.5	208.8	154.9	123.0	93.9	68.8

TABLE *5.10 – Moyenne sur* 5 *graphes (dont les arêtes ont des poids positifs) du pourcentage d'amélioration de la relaxation linéaire après ajout des inégalités de clique généralisées correspondant aux cliques de taille* n *et* $n-1$.

5.2.2 Obtention d'une solution entière à partir d'une solution fractionnaire

L'inconvénient d'un algorithme de plans coupants classique est qu'aucune solution réalisable n'est obtenue si une des conditions d'arrêt (T_2) ou (T_3) est satisfaite. C'est la raison pour laquelle, afin d'obtenir une solution réalisable à partir d'une solution fractionnaire x^*, nous utilisons un algorithme glouton.

Pour ce faire, nous nous sommes basés sur le fait que les variables de représentants x^*_r pour tout sommet r de V traduisent à quel point le sommet i est susceptible d'être un représentant de la solution optimale.

Tout d'abord, les variables x^*_r sont triées, puis les K plus grandes sont sélectionnées. Ensuite, chaque sommet i de V dont la variable de représentant n'a pas été sélectionnée, est ajouté à la partie du représentant r permettant de maximiser $x^*_{i,r}$.

Afin d'éviter que les utilisations successives de cet algorithme ne détériorent les performances globales, ce dernier n'est pas appelé à chaque itération de (E_1). Il est, tout d'abord, utilisé toutes les 300 itérations, puis de manière plus systématique à mesure que le temps d'exécution restant diminue.

L'avantage de cet algorithme glouton est qu'il permet une obtention rapide d'une solution réalisable approchée du problème.

Afin de résoudre le problème de K-partitionnement en utilisant une méthode de plans coupants, il est également nécessaire de définir des algorithmes de séparation.

5.2.3 Algorithmes de séparation

Les algorithmes de séparation présentés précédemment recherchent de manière approfondie des inégalités violées. Ceci est un avantage lorsque, comme dans la sous-section précédente, nous souhaitons ajouter le plus possible de coupes afin d'évaluer la qualité d'une famille d'inégalités. Cependant, cette recherche détaillée est effectuée au détriment des performances. En effet, la complexité élevée de ces algorithmes ne permet pas d'en envisager une utilisation récurrente dans le cadre d'un algorithme de résolution efficace du problème de K-partitionnement.

Ainsi, nous avons sélectionné des algorithmes de la littérature qui réalisent un bon compromis entre efficacité et complexité.

Séparation des inégalités de 2-partitions

Afin de séparer cette famille d'inégalités, nous utilisons deux heuristiques que nous appelons A_1 et A_2 – développées par Grötschell et Wakabayashi [49] – dans lesquelles sont recherchées des inégalités de 2-partitions dont l'ensemble S associé est réduit à un unique sommet s. Pour tout sommet s de V, les deux algorithmes tentent de construire itérativement un ensemble T tel que l'inégalité de 2-partitions correspondante soit violée. Le détail de l'algorithme A_1 est présenté en figure 5.5. L'ensemble W contient les sommets susceptibles d'être ajoutés à T. Les sommets u de $V\backslash\{s\}$ tels que $x_{s,u}$ est égal à 0 ou 1 ne sont sont pas utiles dans la recherche d'une coupe violée si les inégalités triangulaires sont satisfaites. C'est la raison pour laquelle ils ne figurent pas dans l'ensemble W. L'algorithme A_2 est identique à A_1 à l'exception de la condition sous laquelle un sommet $w \in W$ est ajouté dans T qui devient : $x_{s,w} - x(w,T) > 0$.

Séparation des inégalités de clique généralisées

L'heuristique utilisée pour séparer efficacement les inégalités de clique généralisées est un e adaptation d'un algorithme glouton d'approximation de facteur 2 du problème du k-sous-graphe le moins dense (DalkS) [75]. Étant donné un graphe $G = (V, E)$, soit Z un sous-ensemble de V et $E_Z \subset E$ l'ensemble des arêtes connectant deux sommets de Z. La densité du sous-graphe $G_Z(Z, E_Z)$ est définie par $\frac{|E_Z|}{|Z|}$. Le problème DalkS consiste à trouver le sous-graphe de G le plus dense contenant au moins k sommets. L'algorithme débute en prenant Z égal à V et à chaque itération, le sommet le moins dense est retiré jusqu'à ce qu'il ne reste que k sommets. Le sous-graphe le plus dense obtenu durant ce processus est ensuite retourné.

Afin d'obtenir des inégalités de clique généralisées violées, nous cherchons des ensembles Z tels que $x(Z)$ est minimal. Ainsi, à chaque étape, au lieu de retirer le sommet le moins dense, nous retirons le sommet i de Z qui maximise $x(i, Z\backslash\{i\})$. Nous répétons le processus jusqu'à ce que Z ne contienne que $K + 1$ sommets. Cet algorithme, que

Données : Un graphe complet $G = (V, E)$ et une solution fractionnaire x^*
Résultat : $\{S, T\}$ si une coupe violée est obtenue ; $\emptyset$ sinon
pour *tout sommet* s *de* V **faire**
  $S \leftarrow \{s\}$;
  $W \leftarrow \{w \in V \backslash \{s\},\ 0 < x^*_{s,w} < 1\}$;
  Mélanger W;
  $T \leftarrow W[0]$;
  $W \leftarrow W \backslash W[0]$;
  pour *tout* $w \in W$ **faire**
    si $x^*_{w,t} = 0$ *pour tout* $t \in T$ **alors**
      $T \leftarrow T \cup w$;
  si $x^*(s, T) > 1$ **alors**
    retourner *{S,T}*;
  sinon
    retourner $\emptyset$;

FIGURE *5.5 – Algorithme* A_1 *séparant des inégalités de 2-partitions.*

nous appelons a_3, pourrait être utilisé tel quel, cependant, il ne permettrait de retourner qu'une unique coupe dans le meilleur des cas. Afin d'obtenir des coupes aussi diversifiées que possible, a_3 a été adapté afin de forcer l'ensemble Z retourné à contenir un sommet i donné de V. Nous appelons $a'_3(i)$ le résultat retourné par cette adaptation. Comme le représente la figure 5.6, notre algorithme de séparation – nommé A_3 – consiste à appeler $a'_3(i)$ pour chaque sommet i ne figurant dans aucun ensemble Z actuellement obtenu.

Données : Un graphe complet $G = (V, E)$ et une solution fractionnaire x^*
Résultat : Un ensemble d'ensembles correspondant chacun à une inégalité de clique généralisée violée
$R \leftarrow \emptyset$;
pour *tout* i *dans* V **faire**
  si i *n'apparaît dans aucun ensemble de* R **alors**
    $R \leftarrow R \cup a'_3(i)$;
retourner *R;*

FIGURE *5.6 – Algorithme* A_3 *utilisé pour séparer les inégalités de clique généralisées. La fonction* $a'_3(i)$ *retourne un ensemble* Z *contenant le sommet* i *correspondant à une inégalité violée ou* $\emptyset$ *si aucune coupe de ce type n'est trouvée.*

Séparation des inégalités de la formulation

Afin de permettre un calcul rapide de la relaxation linéaire, nous retirons de la formulation (F_2) toutes les familles de contraintes contenant plus de n inégalités (*i.e. :* les inégalités (4.3), (4.11), (4.12) et (4.13)). Pour chacune de ces familles, nous définissons un algorithme de séparation. Puisque le nombre d'inégalités figurant dans ces familles est polynomial, il est possible de réaliser une énumération exhaustive et de n'ajouter

que celles qui sont les plus violées. Pour garder la formulation aussi légère que possible et limiter le temps d'exécution, seules les 500 inégalités les plus violées, parmi les 3000 premières inégalités violées trouvées, sont ajoutées à chaque appel d'un de ces algorithmes.

Une fois les algorithmes de séparation sélectionnés, il est nécessaire de définir la façon dont ces derniers sont gérés au sein de la phase de séparation (E_2).

5.2.4 Gestion des algorithmes de séparation et des coupes ajoutées

Les algorithmes de séparation présentés ne sont pas tous équivalents. Comme le représente la table 5.11, nous pouvons distinguer trois catégories d'algorithmes. Les algorithmes de Kernighan-Lin ont une complexité qui rend leur utilisation systématique coûteuse en temps de calculs. Ainsi, ces derniers ne seront utilisés que dans l'éventualité où les autres algorithmes ne parviendraient pas trouver de coupe. A chaque fois qu'une nouvelle solution fractionnaire est obtenue, nous utilisons chacun des algorithmes des deux autres catégories et ajoutons à la formulation l'ensemble des coupes identifiées.

Algorithmes	**Nombre maximal de coupes retournées**	**Complexité**
(A_1), (A_2) et (A_3)	$\|V\|$	$O(n^2)$
Kernighan-Lin (une phase)	1	$O(n^4)$
Séparation des inégalités triviales	500	$O(n^2)$ ou $O(n^3)$

TABLE *5.11 – Complexité des algorithmes de séparation considérés. La dernière ligne correspond aux algorithmes séparant des inégalités de la formulation.*

La gestion des coupes ajoutées à la formulation peut, elle aussi, permettre d'améliorer les temps de calcul. Une inégalité $a^T x \leq b$ est dite *serrée* pour une solution x^* si $a^T x^*$ est égal à b. Plus le nombre de coupes ajoutées à la formulation augmente, plus le temps de calcul de la relaxation linéaire est important. Afin de pallier cela, Grötschell et Wakabayashi [49] retirent de la formulation, à chaque itération de leur algorithme de plans coupants, les inégalités qui ne sont pas serrées pour x^*. De ce fait, il est possible que certaines inégalités soient générées plus d'une fois. Cependant, les auteurs ont constaté que ce procédé permettait d'améliorer les performances globales de leur algorithme. Dans des expériences préliminaires, il s'est montré plus efficace pour la résolution de notre problème de ne réaliser cette purge des inégalités non serrées que toutes les 500 itérations de l'algorithme.

5.2.5 Résultats numériques

Afin d'évaluer les performances de notre algorithme de plans coupants, nous étudions ses performances sur des graphes complets correspondant aux instances aléatoires les plus complexes (*i.e.* : celles comportant des poids positifs sur les arêtes). Nous considérons ensuite deux cas d'étude avec des données réelles issues de l'article de Grötschell et Wakabayashi [49]. L'ensemble de nos résultats a été obtenu à l'aide d'un ordinateur Intel Xeon équipé de 12 GByte de RAM.

Graphes aléatoires

Dans le cadre des graphes aléatoires, nous comparons les performances de notre algorithme de plans coupants à celles du logiciel CPLEX version 12.7. Les temps de calcul ont été limités à une heure. Lorsque notre algorithme ne converge pas après 2000 secondes (*i.e.* : dans les cas où il ne semble pas permettre d'obtenir la solution optimale), nous utilisons le *branch-and-cut* de CPLEX sur une formulation à laquelle a été ajoutée l'ensemble des inégalités qui sont serrées pour la meilleure relaxation obtenue à ce point.

Les résultats sont présentés en table 5.12. Le *gap* correspond à l'écart relatif entre la valeur de la meilleure solution entière et de la meilleure relaxation obtenues. Nous pouvons tout d'abord observer que notre approche permet plus souvent d'obtenir une solution optimale du problème (représentée dans la table 5.12 par un gap de 0^*). De plus, lorsque la solution optimale n'est pas atteinte, le gap est plus faible que celui de CPLEX, ce qui montre l'efficacité de notre heuristique de recherche de solution entière.

Deux cas d'étude issus de la littérature

Nous illustrons aussi notre méthode sur deux cas d'étude issus de l'article de Grötschell et Wakabayashi [49]. Le problème considéré par ces auteurs est celui du partitionnement non contraint. Le nombre de parties n'est donc pas imposé. Il est cependant raisonnable de supposer qu'un intervalle relativement serré sur la valeur de K soit connu *a priori*. Pour chaque instance, nous appliquons notre algorithme de plans coupants pour les valeurs de K situées autour de celles fournissant la solution optimale. Les temps de calculs sont indiqués à titre informatif puisque les expériences de Grötschell et Wakabayashi ont été réalisées il y a plus de 20 ans. Cependant, nous pouvons comparer les nombres d'itérations ainsi que les types de coupes ajoutées.

La taille des graphes considérés est bien plus élevée que celle des graphes aléatoires que nous avons jusqu'à présent traités (jusqu'à 158 sommets). Cette différence peut tout d'abord s'expliquer par le fait que les graphes possèdent des poids sur les arêtes qui peuvent être négatifs, ce qui tend à faciliter la résolution. Une seconde justification est que les graphes issus d'exemples réels comportent généralement une cohérence dans leur structure qui facilite leur partitionnement. Par exemple, si le sommet i est très similaire aux sommets j et k, alors il est fortement probable que les sommets j et k seront eux aussi relativement similaires dans un graphe réel, ce qui n'est pas forcément le cas dans un graphe aléatoire.

La première instance correspond à un graphe, que nous nommons G_1, comportant 158 sommets et dont le poids de chaque arête est un entier compris entre -3 et 3.

Les résultats obtenus sont présentés en table 5.13. Il est intéressant de constater que le nombre d'itérations de plans coupants nécessaires à la convergence est similaire dans

		Branch-and-cut			**Plans coupants**			
n	**K**	Temps total (s)	Gap	Nœuds	Temps total (s)	Temps b&c (s)	Gap	Nœuds
30	4	3 602	6,6	23 734	3 057	688	0*	233
30	6	1 779	0*	17 101	2 443	65	0*	9
30	8	160	0*	2 388	83	12	0*	5
30	10	257	0*	11 445	16	5	0*	5
35	4	2 419	0*	3 903	1 302	0	0*	0
35	6	3 604	9,1	11 494	2 165	164	0*	32
35	8	3 603	9,1	17 284	2 111	111	0*	27
35	10	542	0*	3 373	2 044	44	0*	10
40	4	3 601	28,5	1 113	3 600	1 600	3,4	190
40	6	3 602	36,2	4 072	3 600	1 600	5,3	415
40	8	3 604	36,9	7 464	2 886	885	0*	310
40	10	3 603	16,3	10 243	2 223	223	0*	42
45	4	3 600	42,3	202	3 600	1 600	19,0	45
45	6	3 601	42,3	533	3 602	1 600	30,4	115
45	8	3 600	54,5	1 191	3 600	1 600	16,5	286
45	10	3 601	26,4	3 265	3 069	1 069	0*	166
50	4	3 609	43,2	0	3 601	1 600	24,3	9
50	6	3 600	59,2	0	3 600	1 600	34,0	20
50	8	3 601	72,6	313	3 600	1 600	48,2	85
50	10	3 603	58,3	1 225	3 600	1 600	46,7	162

TABLE *5.12 – Comparaison des résultats obtenus avec le branch-and-cut de CPLEX 12.7 et notre algorithme de plans coupants. Le gap correspond à l'écart relatif entre la meilleure solution entière obtenue et la valeur de la meilleure relaxation. Le nombre de nœuds correspond au nombre de sommets parcourus durant le branch-and-cut.*

les deux cas où des solutions comportant 6 parties sont obtenues. Le fait d'ajouter des inégalités autres que les inégalités triangulaires ne semble pas permettre de diminuer significativement le nombre d'itérations nécessaire à la résolution. Lorsque nous fixons K à 5 les résultats sont similaires. En revanche, pour K égal à 7, la résolution est bien plus longue car elle nécessite alors l'ajout d'un nombre bien plus conséquent d'inégalités durant 139 itérations. Cette plus grande difficulté à résoudre le problème dans ce cas, peut s'interpréter par le fait qu'il peut s'avérer plus ardu de trouver sept classes dans un graphe en contenant naturellement six. Tout comme nous l'avons mentionné dans le cas de graphes aléatoires, le fait de rechercher un type de partition dans des données ne comportant pas naturellement une telle structure, tend à complexifier significativement le problème.

Le second graphe G_2 utilisé par Grötschell et Wakabayashi que nous considérons comporte 34 sommets et des poids d'arêtes de valeurs entières entre -13 et 13. Les

	K	Temps	Itérations	Inégalités ajoutées			
				(∇)	(af)	(2p)	(cg)
Partitionnement non contraint	6	852s	6	2400	-	-	-
K-partitionnement	5	20s	7	2862	6	622	258
	6	26s	8	3420	470	11	591
	7	507s	139	10821	16702	9	162

TABLE *5.13 – Résultats obtenus pour le graphe G_1 comportant* 158 *sommets. La première ligne correspond aux résultats de l'article de Grötschell et Wakabayashi [49]. (∇) : inégalités triangulaires, (af) : autres inégalités de la formulation, (2p) : inégalités de* 2*-partitions, (cg) : inégalités de clique généralisées.*

résultats correspondant à ce graphe sont présentés en table 5.14. Une nouvelle fois, le nombre d'itérations pour atteindre la solution optimale est similaire dans les deux cas lorsque la valeur de K est la même.

	K	Temps	Itérations	Inégalités ajoutées			
				(∇)	(af)	(2p)	(cg)
Partitionnement non contraint	3	258s	8	1729	2	-	-
K-partitionnement	2	1s	6	2038	90	12	129
	3	<1s	6	2012	175	6	106
	4	1s	8	2203	340	11	58

TABLE *5.14 – Résultats obtenus pour le graphe G_2 comportant* 34 *sommets. La première ligne correspond aux résultats de l'article de Grötschell et Wakabayashi [49]. (∇) : inégalités triangulaires, (af) : autres inégalités de la formulation, (2p) : inégalités de* 2*-partitions, (cg) : inégalités de clique généralisées.*

5.3 Conclusion

Nous avons étudié dans ce chapitre le pourcentage d'amélioration moyen apporté à la solution de la relaxation linéaire lors de l'ajout de coupes issues de chacune des trois familles d'inégalités non triviales identifiées au chapitre 4. Nous avons constaté que la valeur de ces améliorations ainsi que la difficulté du problème de K-partitionnement sont très dépendantes des données. Notamment, dans le cas où le poids des arêtes du graphe sont tous positifs, les inégalités de clique généralisées fournissent des améliorations spectaculaires.

Des algorithmes de séparations efficaces ont été développés pour chacune des trois familles d'inégalités considérées. L'ensemble de ces contributions nous ont permis de définir un algorithme de plans coupants efficace dont les performances sont significativement meilleures que celles du logiciel CPLEX 12.7.

Troisième partie

Mise en œuvre et conclusions

6 Expérimentations **141**
6.1 Présentation des données 141
6.2 Logiciel VIESA 142
6.3 Évaluation de LPCA-DC et des heuristiques de partitionnement 143
6.3.1 Calcul de la dissimilarité inter-motifs avec LPCA-DC 143
6.3.2 Protocole d'expérimentation 145
6.3.3 Résultats et discussion 148
6.4 Evaluation de SABRE et l'algorithme de plans coupants 153
6.4.1 Calcul de la dissimilarité inter-motifs avec SABRE 153
6.4.2 Protocole d'expérimentation 154
6.4.3 Résultats 157
6.5 Discussion 161

Conclusion et perspectives **164**

6 | Expérimentations

Dans ce chapitre, nous présentons deux expérimentations réalisées afin d'évaluer l'efficacité des méthodes que nous avons sélectionnées ou développées dans le cadre de l'extraction de régularités dans un corpus de tableaux d'annotations. Le corpus de tableaux d'annotations utilisé dans ce cadre est présenté en section 6.1.

Les expérimentations ont été réalisées par Emilie Chanoni, psychologue de l'Université de Rouen, experte du corpus de tableaux d'annotations considéré. Son objectif dans ce cadre est d'identifier les stratégies mises en place par les parents afin de faire comprendre aux enfants que les comportements de personnages dans des histoires peuvent être induits par leur état mental. Afin de faciliter l'évaluation des alignements et des partitions de motifs, un logiciel de visualisation nommé VIESA a été développé. La section 6.2 lui est dédiée.

La première de nos expérimentations est, ensuite, décrite en section 6.3. Dans cette dernière, nous considérons la méthode d'extraction LPCA-DC et les différentes heuristiques de partitionnement. La seconde expérience, présentée en section 6.4 a été réalisée après le développement de SABRE et de l'algorithme de plans coupants. Elle permet, dans un premier temps, de comparer les deux méthodes d'extraction. Puis les motifs obtenus avec SABRE sont ensuite partitionnés par l'algorithme de plans coupants ainsi que par les deux meilleures heuristiques identifiées précédemment et les régularités obtenues sont évaluées par l'expert.

6.1 Présentation des données

Dans le cadre de cette expérimentation, nous considérons un corpus de 113 tableaux d'annotations correspondant chacun à un dialogue entre un parent et son enfant lors de la narration d'une histoire. Comme l'illustre la table 6.1, les lignes d'un tableau d'annotations de ce corpus correspondent à des énoncés d'un dialogue ordonnés chronologiquement. La longueur des tableaux d'annotations varie de 28 à 249 énoncés pour une moyenne de 82. Le schéma de codage utilisé comporte quatre alphabets (*i.e.* : quatre colonnes) :

- La première colonne est dédiée à la référenciation de l'énoncé (P : au personnage, I : à l'interlocuteur, L : au locuteur) ;
- La seconde colonne encode les états mentaux (E : émotion, V : volition, S : surprise, OC et NOC : cognition respectivement observable et non observable) ;
- Les deux dernières colonnes sont consacrées aux justifications (O : par opposition, C : par cause ; CH : pour expliquer l'histoire, PC : pour expliquer une situation

par l'évocation d'un contexte personnel).

Le caractère '-' figure dans chaque alphabet. Sa présence dans une colonne i signifie que l'énoncé correspondant ne véhicule aucune des notions représentées par les autres caractères de l'alphabet Ω_i. Par exemple, le troisième énoncé de la table 6.1 ne fait référence ni au locuteur, ni à l'interlocuteur, ni à un personnage de l'histoire (le mot "Elle" désigne ici la couronne) et en conséquence, le caractère en première colonne de la troisième ligne est '-'.

Locuteur	**Énoncé**	**Annotations**			
P	C'est la couronne,	-	-	-	-
P	mais Babar ne l'a pas vu.	P	OC	O	ES
P	Elle est cachée derrière la porte,	-	OC	-	-
P	mais tu la vois, c'est bien.	I	OC	O	PC
E	Oui je la vois.	L	-	-	-

TABLE *6.1 – Extrait d'un tableau d'annotations figurant dans le corpus de dialogue étudié. A chaque ligne du tableau de caractères est associé l'énoncé du dialogue correspondant et son locuteur (P : parent, E : enfant).*

6.2 Logiciel VIESA

Afin de permettre à l'expert, une évaluation simple et intuitive des alignements et des partitions de motifs, nous avons développé un logiciel nommé VIESA (VIsualisation d'ElementS Annotés) en Java.

Ce logiciel permet à l'utilisateur de définir un corpus en sélectionnant des tableaux d'annotations suivant le même schéma de codage, de sélectionner les colonnes d'annotations prises en compte lors de l'extraction des motifs et de régler les différents paramètres des méthodes (scores des opérations d'édition, score minimum à partir duquel un motif est obtenu, etc.) puis d'exécuter l'extraction et le partitionnement de motifs.

Les résultats obtenus sont visualisables au sein d'un onglet représenté en figure 6.1. Dans ce dernier, il est possible de sélectionner les tableaux d'annotations choisis en entrée (2) ainsi que les différents motifs regroupés par parties (3) afin de les visualiser dans une des deux tables (4) ou dans un graphe (5). La représentation graphique des partitions (5) a été réalisée avec la librairie GraphStream [34] développée au laboratoire LITIS. Dans les deux tables, les motifs sélectionnés sont visualisables avec leur contexte (*e.g.* : dans le cadre de notre expérience le contexte est pour chaque ligne, le texte des dialogues et le locuteur correspondant). Lorsque différentes méthodes de partitionnement ont été utilisées, le passage de l'une à l'autre peut être effectué via une liste déroulante (1). Le logiciel est disponible sur sourceforge à l'adresse suivante : *http ://sourceforge.net/projects/viesa*.

Dans le cadre de nos expérimentations, les résultats sont calculés au préalable et l'expert n'utilise que la partie visualisation du logiciel.

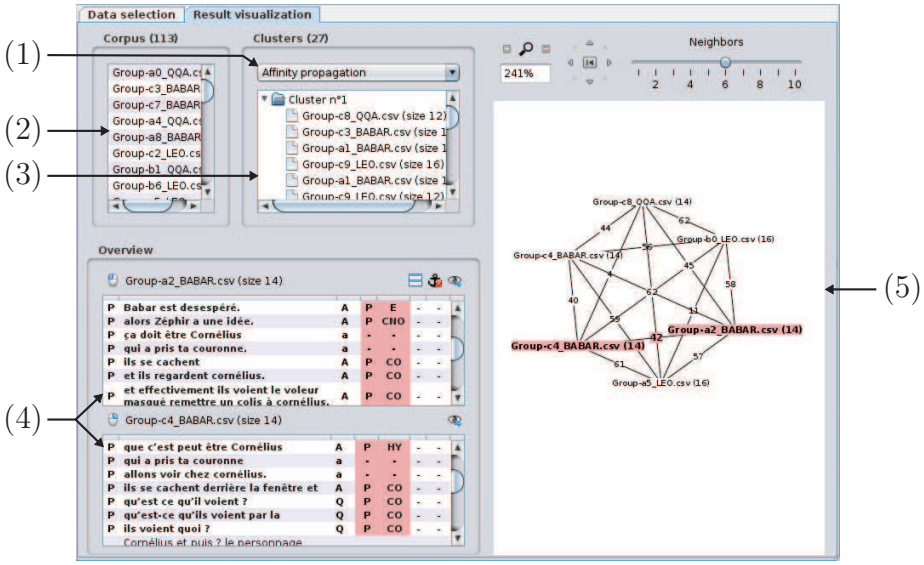


FIGURE *6.1 – Capture d'écran de l'interface de VIESA.* ***(1)*** *: Sélection des méthodes de partitionnement.* ***(2)*** *: Sélection du tableau d'annotations.* ***(3)*** *: Sélection de motifs partitionnés.* ***(4)*** *: Visualisation de motifs.* ***(5)*** *: Visualisation de parties.*

6.3 Évaluation de LPCA-DC et des heuristiques de partitionnement

L'objectif de cette première expérimentation – réalisée préalablement au développement de SABRE et de l'algorithme de plans coupants – est d'évaluer l'efficacité de la méthode LPCA-DC pour l'extraction de motifs récurrents et de comparer les différentes heuristiques de partitionnement. En vue de réaliser le partitionnement il est nécessaire de définir une dissimilarité entre les différents motifs. C'est la raison pour laquelle, nous présentons en premier lieu comment les mécanismes de LPCA-DC peuvent être utilisés afin de calculer la dissimilarité entre deux motifs.

6.3.1 Calcul de la dissimilarité inter-motifs avec LPCA-DC

Une variante de l'algorithme LPCA, proposée dans le même article [83], permet de calculer un alignement global (*i.e.* : une distance d'édition globale) entre deux tableaux de caractères. Il paraît donc naturel de se baser sur cette distance pour définir la dissimilarité entre deux motifs. Tout comme cela a été réalisé pour l'algorithme LPCA, nous tirons parti de la structure des tableaux d'annotations afin de réduire la complexité de cette variante de l'algorithme. C'est cette adaptation que nous présentons ici.

La méthode utilisée considère en entrée deux tableaux de caractères. Il est donc nécessaire de définir une représentation d'un motif m sous la forme d'un tableau de caractères. Soit $t[1..n_1][1..n_2]$ le tableau de caractères dans lequel apparaît un motif m et soit l_m (respectivement l_M) la première (respectivement dernière) ligne de t à laquelle apparaît une annotation de m. Le motif m est représenté par les lignes de t comprises entre l_m et l_M (*i.e.* : $t[l_m..l_M][1..n_2]$) comme l'illustre l'exemple en figure 6.2. L'ensemble des colonnes de t sont conservées car il est nécessaire pour l'algorithme LPCA-DC que

les tableaux comportent le même nombre de colonnes. Enfin, il est nécessaire de faire la distinction entre les annotations de $t[l_m..l_M][1..n_2]$ qui appartiennent à m et les autres. Ainsi, ces dernières sont remplacées par le caractère '*'. Le coût de la substitution de ce caractère par lui même est nul (*i.e.* : $sub(*,*) = 0$), tandis que sa substitution avec tout autre caractère 'a' est égale au coût de suppression de 'a' (*i.e.* : $sub(*,a) = sup(a)$).



(a) Tableau d'annotations t. Les annotations en rouge correspondent à m.

(b) Représentation de m.

FIGURE *6.2 – Exemple de représentation d'un motif m pour le calcul de sa dissimilarité avec d'autres motifs.*

Étant donnés deux tableaux de caractères $t_A[1..m_A][1..n_A]$ et $t_B[1..m_B][1..n_A]$, représentant chacun un motif, l'algorithme va construire une table $T[0..m_A][0..n_A][0..m_B]$ telle que $T[i][j][k]$, pour tout $(i,j,k) \in \{1,\ldots,m_A\} \times \{1,\ldots,n_A\} \times \{1,\ldots,m_B\}$, soit égal à la dissimilarité entre $t_A[1..i][1..j]$ et $t_B[1..k][1..j]$.

De manière similaire à l'algorithme LPCA-DC, six opérations d'édition bidimensionnelles seront considérées (voir en figure 6.3). Cependant, afin d'obtenir la dissimilarité entre les deux motifs, la valeur de ces opérations correspondra maintenant à un coût global et non plus à un score local.

Cet algorithme construit deux tables $R[1..m_A][1..n_A][1..m_B]$ et $C[1..m_A][1..n_A][1..m_B]$ dont chaque valeur correspond au coût d'un alignement global (obtenu grâce à l'algorithme de Needleman-Wunsch présenté en section 1.2.1), de telle sorte que :

- $R[i][j][k]$ est égal au coût de l'alignement global entre $t_A[i][1..j]$ et $t_B[k][1..j]$;
- $C[i][j][k]$ est égal au coût de l'alignement global entre $t_A[1..i][j]$ et $t_B[1..k][j]$.

De plus, lorsqu'une suppression (respectivement une insertion) de ligne est réalisée (opérations (a) et (c) de la figure 6.3), l'ensemble des caractères de la ligne considérée doivent maintenant être supprimés (respectivement insérés). Ceci nécessite le calcul de deux tables supplémentaires $D[1..m_A][1..n_A]$ et $I[1..m_B][1..n_A]$ telles que :

- $D[i][j] = \sum_{p=1}^{j} sup(t_A[i][p])$;
- $I[i][j] = \sum_{p=1}^{j} ins(t_B[i][p])$.

Les bords de la table T sont ensuite initialisés de la manière suivante :

- $T[0][j][k] = \sum_{l=1}^{k} I[l][j]$ (si l'alignement débute à partir de $t_B[k][j]$ et de la première ligne de t_A, l'ensemble des caractères de $t_B[1..k][1..j]$ doivent être insérés) ;
- $T[i][j][0] = \sum_{l=1}^{i} D[l][j]$ (si l'alignement débute à partir de $t_A[i][j]$ et de la première ligne de t_B l'ensemble des caractères de $t_A[1..i][1..j]$ doivent être supprimés) ;
- $T[i][0][k] = 0$ (si l'alignement débute à partir de la première colonne de t_A et de t_B, il n'est pas pénalisé).

Enfin, la valeur de chaque position (i,j,k) de T ne se trouvant pas sur un bord est

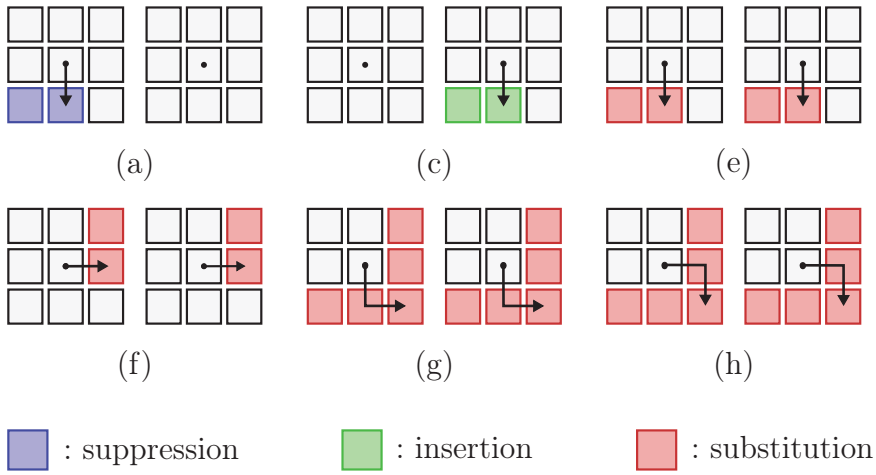


FIGURE *6.3 – Opérations d'édition considérées lors du calcul de la dissimilarité entre deux tableaux d'annotations. (a) Suppression de ligne. (c) Insertion de ligne. (e) Substitution de ligne. (f) Substitution de colonne. (g) et (h) Substitution de ligne et de colonne (dans un ordre différent). Les lettres identifiant les opérations correspondent à celles utilisées dans la figure 1.16.*

obtenue en utilisant la formule de récurrence suivante :

$$T(i,j,k) = \min \begin{cases} T[i-1,j,k,l] + D[i][j] & (a) \\ T[i,j,k-1,l] + I[k][j] & (c) \\ T[i-1,j,k-1,l] + R[i][j][k] & (e) \\ T[i,j-1,k,l-1] + C[i][j][k] & (f) \\ T[i-1,j-1,k-1,l-1] + R[i-1][j][k-1] + C[i][j][k] & (g) \\ T[i-1,j-1,k-1,l-1] + R[i][j][k] + C[i][j-1][k] & (h) \end{cases}$$

Ainsi, la dissimilarité entre les deux motifs correspond à la valeur de $T[m_A][n_A][m_B]$.

6.3.2 Protocole d'expérimentation

L'objectif de cette évaluation est d'évaluer l'efficacité de la méthode d'extraction LPCA-DC et des heuristiques de partitionnement. Pour ce faire, nous demandons à l'expert d'évaluer les régularités (*i.e.* : les regroupements de motifs) obtenues en faisant varier différents arguments des méthodes. Quatre paramètres sont ainsi considérés, deux pour la phase d'extraction – le score des substitutions *(1)* et le score minimum τ *(2)* – et deux pour le partitionnement – le nombre de parties K *(3)* et l'heuristique de partitionnement *(4)*. La variation de ces quatre paramètres permettra, d'une part, d'évaluer leur impact sur les régularités extraites et, d'autre part, de juger de la qualité des méthodes utilisées.

Voici une description détaillée des quatre paramètres considérés et des valeurs qui sont utilisées dans cette expérience.

Paramètre (1) **: Table de substitution de caractères**

La table 6.2, qui avait été réalisée précédemment par un expert du corpus, présente la similarité entre les différents couples d'annotations du schéma de codage utilisé dans

le corpus. Il est nécessaire pour réaliser l'extraction de motifs, que ces scores comportent des valeurs positives et négatives (afin que le score de l'alignement local courant augmente dans les zones similaires des deux tableaux d'annotations et tende vers 0 dans les parties qui diffèrent). Afin d'obtenir une répartition équitable entre valeurs positives et négatives, les valeurs de substitutions que nous avons considérées correspondent à celles de la table 6.2 auxquelles ont été retranché la valeur moyenne 5. La table de substitution de référence obtenue est nommée S_{ref}. Pour les insertions et les suppressions des diverses annotations, il a été décidé avec l'expert, après divers essais sur une sous-partie du corpus, de choisir un score unique de -7.

	AG	AH	P	AR	HR	C	O	Emp	CH	CP	E	CO	CNO	V	EP	HY	S
AG	10																
AH	10	10															
P	0	0	10														
AR	0	0	9	10													
HR	0	0	7	8	10												
C	0	0	2	2	2	10											
O	0	0	2	2	2	9	10										
Emp	0	0	2	8	2	8	8	10									
CH	0	0	7	4	3	1	1	2	10								
CP	0	0	4	7	3	1	1	7	9	10							
E	0	0	5	7	5	3	3	8	2	4	10						
CO	0	0	5	5	5	3	3	3	4	4	8	10					
CNO	0	0	5	5	5	3	3	3	4	4	8	10	10				
V	0	0	5	5	5	3	3	3	2	4	9	7	7	10			
EP	0	0	7	5	5	3	6	3	4	4	8	7	8	7	10		
HY	0	0	5	5	5	4	3	3	4	4	8	7	8	7	9	10	
S	0	0	5	5	5	3	3	6	3	4	9	7	7	7	9	8	10

TABLE *6.2 – Table des scores de substitution.*

Afin d'évaluer l'impact de la table S_{ref} sur la qualité des motifs obtenus, deux tables S_- et S_+ ont été générées de telle sorte que :

- les scores de substitution d'une annotation par elle-même sont conservés (il serait contre-intuitif de mettre des valeurs négatives pour ces opérations) ;
- les autres scores de substitutions sont générés pseudo-aléatoirement de sorte que les moyennes des tables vérifient : $\overline{S}_- \leq \overline{S}_{ref} \leq \overline{S}_+$.

Paramètre *(2)* : Score minimal τ d'un alignement

Le deuxième paramètre de la phase d'extraction est le score minimal à partir duquel une position de la table T de l'algorithme LPCA-DC dénote un alignement. Sa valeur influence directement le nombre d'alignements obtenus. En effet, un score minimum faible sera atteint par un plus grand nombre de positions de T.

Le nombre de motifs obtenus est aussi directement influencé par les scores des opérations d'édition de T et τ doit donc être choisi en conséquence. Ainsi, nous avons sélectionné pour τ trois valeurs (*i.e.* : 36, 38 et 40) permettant d'obtenir avec les tables

S_-, S_{ref} et S_+ des ensembles de motifs récurrents dont la taille est aussi variée que possible (un nombre très modeste de motifs n'aura que peu de chances de permettre l'identification de régularités représentatives du corpus, tandis qu'un nombre trop élevé ne pourrait être raisonnablement évaluer manuellement par un expert). Le nombre de motifs obtenus pour chaque couple de paramètres est présenté en table 6.3.

Table de substitution	**Score minimum τ**		
	36	**38**	**40**
$\mathbf{S_-}$	261	150	75
$\mathbf{S_{ref}}$	413	259	151
$\mathbf{S_+}$	694	442	270

TABLE *6.3 – Nombre de motifs récurrents obtenus en fonction de la table de scores de substitution utilisée et du score minimum τ.*

Paramètre *(3)* : Nombre de parties K

En plus de la méthode LPCA-DC, cette expérience a aussi pour but d'évaluer les différentes heuristiques de partitionnement. Un paramètre essentiel de ces dernières est le nombre de parties K puisqu'il influence directement la façon dont les résultats sont présentés à l'expert. La quantité de travail à fournir par l'expert lors de l'évaluation augmente rapidement avec le nombre de valeurs de K considérées. C'est la raison pour laquelle nous avons uniquement sélectionné trois valeurs : 5, 20 et 120. Ce choix a été réalisé en collaboration avec l'expert et devrait permettre d'obtenir des régularités de granularités différentes (une partie contenant moins d'une dizaine de motifs très similaires correspond à une régularité très spécifique pour le corpus ; tandis qu'une partie contenant des dizaines de motifs relativement similaires dénote une régularité plus générale).

Pour les méthodes de partitionnement spectral, la valeur de K est directement fixée comme paramètre d'entrée. Dans le cas des méthodes hiérarchiques – qui renvoient un ensemble de partitions imbriquées – seules les partitions correspondant aux trois valeurs de K souhaitées seront conservées (lorsqu'elles sont disponibles, ce qui n'est pas toujours le cas pour CHAMELEON). Enfin, la méthode de propagation d'affinité ne permet pas de fixer le nombre de parties puisque ce dernier émerge au cours des itérations. Nous présentons donc à l'expert la solution retournée quelque soit le nombre de parties qu'elle contient.

Paramètre *(4)* : Heuristique de partitionnement

Le dernier des paramètres est l'heuristique utilisée pour le partitionnement. Nous considérons les méthodes sélectionnées en section 3.3 qui utilisent des mécanismes variés pour réaliser le partitionnement :

- Single-link (hiérarchique, basé sur la plus petite distance) ;
- ROCK (hiérarchique, basé sur le voisinage) ;
- CHAMELEON (hiérarchique basé sur l'interconnectivité) ;
- propagation d'affinité (type partition, utilisant l'envoi de messages) ;
- trois méthodes spectrales utilisant des laplaciens de graphe différents :
  - méthode non normalisée (laplacien de graphe non normalisé) ;
  - méthode de Shi et Malik (laplacien de graphe normalisé dit de marche aléatoire) ;

– méthode de Ng, Jordan et Weiss (laplacien de graphe normalisé dit symétrique).

Fiche d'évaluation

Pour chaque combinaison de valeurs des quatre paramètres présentés, une partition de motifs récurrents est obtenue. Il est demandé à l'expert de lui donner une note entre 0 et 4 pour les deux critères suivants : la pertinence du nombre de parties (qui est utilisé pour juger de la qualité du paramètre K) et la pertinence de la solution (qui est utilisé pour évaluer les autres paramètres). Plus la note est élevée pour un critère donné, plus la partition considérée est pertinente. Un exemple de fiche d'évaluation utilisée (pour une valeur de τ et une table de substitution donnés) est présentée en table 6.4.

Méthode	Propagation d'affinité	CHAMELEON		ROCK			...	Single-link		
Nombre de parties	51	5	20	5	20	120	...	5	20	120
Pertinence des parties	☐	☐	☐	☐	☐	☐	...	☐	☐	☐
Pertinence du nombre de parties	☐	☐	☐	☐	☐	☐	...	☐	☐	☐

TABLE *6.4 – Exemple de fiche remplie pendant l'évaluation pour un score minimum τ et une table de substitution donnés. Chaque carré blanc doit être complété par une valeur entre* 0 *et* 4 *(*0 *pour "non pertinent" et* 4 *pour "extrêmement pertinent").*

6.3.3 Résultats et discussion

Nous souhaitons déterminer pour chacun des quatre paramètres considérés, si une de leurs valeurs fournit des résultats significativement meilleurs. Le critère de pertinence du nombre de parties est utilisé pour évaluer K tandis que le second critère est considéré pour les trois autres paramètres.

Étant donné un paramètre p, nous utilisons un test statistique pour comparer les résultats obtenus pour chaque couple de valeurs de p (*e.g.* : dans le cas des scores de substitution les couples de valeurs seront (S_-, S_{ref}), (S_-, S_+) et (S_{ref}, S_+)) et déterminer si une des deux valeurs du couple fourni, en moyenne, des résultats significativement meilleurs. Pour ce faire, étant donné un couple de valeurs (p_1, p_2) d'un paramètre p (*e.g.* : (S_-, S_{ref}) pour la table de substitution), les résultats de l'expérience peuvent être reformulés afin de juxtaposer ceux obtenus avec p_1 et ceux obtenus avec p_2 en fonction des trois autres paramètres. Par exemple, dans le cas où le couple (S_-, S_{ref}) est considéré, la table 6.5 est obtenue. Cette représentation permet d'obtenir deux échantillons appariés. En effet, pour une colonne donnée de la table obtenue (*e.g.* : table 6.5), seule la valeur du paramètre p est modifiée, la valeur des trois autres paramètres reste la même (*e.g.* : dans la table 6.5, pour l'heuristique ROCK, un τ égal à 36 et 20 parties, le critère a une valeur de 0 pour S_- et de 2 pour S_{ref}). Ces échantillons appariés vont nous permettre d'utiliser un test signé des rangs de Wilcoxon (que nous abrégeons SRW) afin de déterminer si un des échantillons possède une moyenne significativement supérieure. Nous utilisons ce test plutôt que l'habituel test de Student car les résultats fournis par l'expert ne peuvent être supposés normalement distribués. L'hypothèse du test SRW est :

H_0 : la différence moyenne entre les deux échantillons est nulle.

Si le test est satisfait, H_0 est rejetée et la valeur du paramètre fournissant en moyenne les meilleurs résultats peut être obtenue.

Heuristique		ROCK									...
Score minimum τ		36			38			40			...
Nombre de parties K		5	20	120	5	20	120	5	20	120	...
Scores de substitution	S_-	0	0	1	0	1	0	0	1	0	...
	S_{ref}	0	2	4	0	1	3	0	1	0	...

TABLE *6.5 – Résultats du critère "pertinence des parties" présentés en fonction des valeurs S_- et S_{ref} du paramètre de table de substitution. Dans cette représentation, les deux dernières lignes de la table correspondent à des échantillons appariés.*

Résultats du paramètre *(1)* : Scores des substitutions de caractères

Trois tables de scores de substitutions ont été considérées dans cette expérience. Une d'entre elles (S_{ref}) avait été définie par un expert et les deux autres (S_- et S_+) ont été générées pseudo-aléatoirement afin d'évaluer l'influence de ce paramètre.

Comme le montre la table 6.6, les tests SRW des couples (S_-, S_{ref}) et (S_{ref}, S_+) sont satisfaits. Dans les deux cas, la valeur moyenne de S_{ref} est plus grande, confirmant ainsi, d'une part, l'idée intuitive que la table S_{ref} définie par l'expert fournit de meilleurs résultats et, d'autre part, que ce paramètre influence significativement la qualité des régularités obtenues.

	Couple de valeurs		
	(S_-, S_{ref})	(S_-, S_+)	(S_{ref}, S_+)
Le test SRW est-il satisfait ?	oui	non	oui
En faveur de quelle valeur ?	S_{ref}	-	S_{ref}

TABLE *6.6 – Résultat du test SRW sur chaque pair de table de substitution.*

L'évaluation des trois autres paramètres a été réalisée en ne conservant que les résultats liés à la table S_{ref}.

Résultats du paramètre *(2)* : Score minimum τ d'un alignement

Le score τ correspond à la valeur en dessous de laquelle il n'est plus pertinent de faire figurer un alignement dans les résultats. Son choix va directement influencer le nombre d'alignements de motifs obtenus. Dans cette expérience, les valeurs 36, 38 et 40 ont été considérées afin de permettre l'obtention de listes d'alignements de taille aussi variée que possible.

Les résultats obtenus sont représentés en table 6.7. Ces derniers indiquent que la qualité des régularités obtenues est significativement meilleur lorsque τ est égal à 38. Nous pensions initialement que plus le nombre de motifs était élevé, plus les régularités obtenues seraient précises et pertinentes. Cette expérience prouve le contraire. En effet, l'expert nous a indiqué qu'un trop grand nombre de motifs tend à complexifier l'évaluation en rendant l'interprétation des regroupements de motifs plus ardue.

	Couple de valeurs		
	$(36, 38)$	$(36, 40)$	$(38, 40)$
Le test SRW est-il satisfait ?	oui	non	oui
En faveur de quelle valeur ?	38	-	38

TABLE *6.7 – Résultat du test SRW sur chaque pair de valeurs du score minimum τ.*

Ces résultats prouvent que le score minimum est un paramètre ayant un fort impact sur la qualité des régularités obtenues et qu'il doit être choisi avec soin en fonction du corpus de tableaux d'annotations et du score des opérations d'édition.

Résultats du paramètre *(3)* : Nombre de parties K

Les résultats obtenus pour le paramètre K sont présentés en table 6.8. D'après les tests de Wilcoxon, tout comme pour le paramètre τ, c'est la valeur intermédiaire (ici 20) qui fournit les meilleurs résultats.

	Couple de valeurs		
	$(5, 20)$	$(5, 120)$	$(20, 120)$
Le test SRW est-il satisfait ?	oui	oui	oui
En faveur de quelle valeur ?	20	5	20

TABLE *6.8 – Résultat du test SRW sur chaque pair de valeurs du nombre de parties.*

Chacune des valeurs de K sélectionnées pour cette expérience correspondait à un niveau de granularité auquel il semblait envisageable pour l'expert d'obtenir des régularités. Cependant, il est apparu que les solutions avec 5 parties tendaient à regrouper des motifs de nature différente tandis qu'avec K égal à 120, de nombreuses parties étaient réduites à un seul et unique motif.

Résultats du paramètre *(4)* : Heuristique de partitionnement

Préalablement à l'étude des résultats de l'expert, nous pouvons comparer les différentes partitions obtenues par le poids de leur K-coupe (*i.e.* : le poids de la somme des dissimilarités reliant deux sommets appartenant à des parties différentes). Ce critère n'est pas aussi proche de la vérité terrain que l'avis d'un expert et n'est donc pas suffisant en lui-même pour évaluer les partitions. Cependant, il donne une estimation de leur qualité puisqu'une partition dont la K-coupe possède un poids élevé traduit des parties homogènes et clairement séparées. La table 6.9 présente les résultats obtenus lorsqu'un score minimum de 38 et la table S_{ref} sont considérés. Des tables similaires sont obtenues pour les autres scores de substitution et les autres valeurs de τ.

Dans cette table, nous pouvons remarquer que les méthodes spectrales normalisées ainsi que la propagation d'affinité et ROCK tendent à fournir des partitions dont le poids de la K-coupe est élevé. En revanche, Single-link ainsi que la méthode spectrale non normalisée ont des résultats plus inégaux.

Tous comme pour les autres paramètres, les résultats sont évalués par des tests de Wilcoxon en considérant tous les couples de valeurs possibles (*i.e.* : les couples d'heuristiques). Les résultats sont présentés en table 6.10. La comparaison entre la méthode

Méthode de partitionnement	**Nombre de parties**			
	5	20	38	120
Méthode spectrale non normalisée	29	100	-	334
Single-link	49	230	-	**358**
ROCK	279	341	-	357
Méthode spectrale de Shi et Malik	309	346	-	357
Méthode spectrale de Ng, Jordan et Weiss	**312**	**351**	-	**358**
Propagation d'affinité	-	-	**355**	-
CHAMELEON	276	-	-	-

TABLE *6.9 – Valeurs arrondies du poids des partitions ($\times 10^{-4}$) de chaque heuristique de partitionnement en fonction du nombre de parties K (solutions correspondant à la table S_{ref} et à un score minimum de* 38*). Le symbole '-' est utilisé lorsqu'une méthode ne produit pas de solution pour une valeur de K donnée ou lorsque la solution correspondante n'est pas considérée dans l'évaluation. Les valeurs en gras dénotent les meilleurs résultats de chaque colonne.*

spectrale de Ng, Jordan et Weiss et celle de Shi et Malik n'a pas pu être effectuée puisque de nombreuses valeurs du critère de pertinence des partitions étaient identiques et que le test SRW nécessite un minimum de valeurs distinctes pour pouvoir être appliqué. La table permet, néanmoins, de conclure que les méthodes ROCK et propagation d'affinité fournissent les meilleurs résultats. Au contraire les méthodes single-link, CHAMELEON et la méthode spectrale de Ng, Jordan et Weiss offrent les moins bonnes performances. Ceci peut s'expliquer par le fait que ces trois dernières méthodes tendent à fournir des partitions contenant une ou deux parties comportant la majorité des motifs et de nombreuses parties réduites à un unique sommet.

Méthodes	PA	RO	MS_{nn}	MS_{sm}	SL	MS_{njw}	CH
Propagation d'affinité (PA)		PA	PA	PA	PA	PA	PA
ROCK (RO)	PA		RO	RO	RO	RO	RO
Méthode spectrale non normalisée (MS_{nn})	PA	RO		MS_{nn}	MS_{nn}	-	SC_u
MS Shi et Malik (MS_{sm})	PA	RO	MS_{nn}		MS_{sm}	MS_{sm}	MS_{sm}
Single-link (SL)	PA	RO	MS_{nn}	MS_{sm}		SL	SL
MS Ng, Jordan et Weiss (MS_{njw})	PA	RO	-	MS_{sm}	SL		$\times$
CHAMELEON (CH)	PA	RO	MS_{nn}	MS_{sm}	SL	$\times$	

TABLE *6.10 – Résultats des tests SRW sur chaque pair d'heuristique de partitionnement. Si le test est satisfait, le nom de la méthode retournant des résultats significativement meilleurs est représenté. Dans le cas contraire, le symbole $\times$ est utilisé. Enfin, le symbole $-$ dénote les couples pour lesquels le test n'a pas pu être appliqué.*

Évaluation des partitions obtenues avec les meilleurs paramètres

Afin de déterminer l'efficacité de notre approche et la qualité des méthodes utilisées, nous avons demandé à l'expert d'étudier plus en détails les partitions obtenues lorsque les paramètres sont fixés aux valeurs ayant été identifiées comme les meilleures (*i.e.* : S_{ref}, τ égal à 38 et 20 parties). Ceci a permis l'identification de régularités correspondant à des stratégies mises en œuvre par les parents et que de précédentes analyses manuelles n'avaient pas permis d'identifier. Par exemple, les trois motifs représentés en figure 6.4

Locuteur	Dialogues	Annotations			
Parent	Il pleure.	P	E	-	-
Parent	Parce qu'il est colère.	P	E	C	CH
Parent	Parce qu'il est pas content.	P	E	C	CH

(a)

Locuteur	Dialogues	Annotations			
Parent	Léo veut la pelle de Timothée.	P	V	-	-
Parent	Mais Timotée ne veut pas lui donner.	P	V	O	CH
Parent	C'est parce qu'il a pas de pelle.	P	-	C	CH

(b)

Locuteur	Dialogues	Annotations			
Parent	Oh ! Léo il n'est pas content.	P	E	-	-
Parent	Parce que son copain il ne veut pas lui donner la pelle !	P	V	C	CH
Parent	Alors il donne un gros coup de pied dans le château de son copain !	P	-	C	CH

(c)

FIGURE *6.4 – Trois motifs regroupés ensemble par la méthode ROCK.*

ont été regroupés ensemble par la méthode ROCK. Selon l'expert, ils correspondent à une stratégie utilisée par les parents afin de fournir une explication mentaliste à l'enfant. En vue de permettre la compréhension des sentiments d'un personnage, l'état mental correspondant est, tout d'abord, exprimé (annotation E ou V). Ce dernier est ensuite expliqué par deux justifications, dont au moins la première contient un état mental proche ou identique (*e.g.* : "il pleure", "parce qu'il est colère").

Le fait que notre outil permette l'identification de régularités jusqu'à présent inconnues d'un corpus ayant été étudié manuellement de nombreuses fois permet de confirmer l'intérêt de la méthodologie en deux étapes que nous avons mise en œuvre.

Cependant, il a été remarqué durant l'expérience que les motifs obtenus par la méthode LPCA-DC possédaient une forme particulière (en escalier). Comme nous l'avons décrit dans le chapitre 2, ceci est dû à l'algorithme lui-même et au choix réalisé sur les opérations bidimensionnelles. Ainsi, le parcours des deux tableaux d'annotations dans l'algorithme de programmation dynamique entraîne nécessairement une contrainte sur les motifs qu'il est possible d'obtenir. Notamment, l'ordre dans lequel figurent les colonnes d'annotations aura un impact significatif sur les résultats.

Concernant la phase de partitionnement, des améliorations sont aussi envisageables. En effet, l'expert a constaté que même les méthodes fournissant les meilleurs résultats comportaient des motifs non cohérents avec la partie dans laquelle ils figuraient.

Ces observations montrent que des progrès sont envisageables tant au niveau de la

phase d'extraction que de celle de partitionnement. Ceci justifie pleinement la création de la méthode d'extraction SABRE et la définition d'un algorithme de plans coupants permettant la résolution exacte du problème de K-partitionnement.

Dans la section suivante, nous présentons une comparaison de ces nouvelles méthodes avec celles considérées durant cette expérience.

6.4 Evaluation de SABRE et l'algorithme de plans coupants

L'objectif de cette seconde expérimentation, est de comparer, d'une part, les résultats fournis par les méthodes d'extraction SABRE et LPCA-DC et, d'autre part, les performances des meilleurs heuristiques identifiées durant la première expérience (*i.e.* : ROCK et propagation d'affinité) avec l'algorithme de plans coupants.

Tout comme LPCA-DC, il est nécessaire, afin de pouvoir réaliser le partitionnement, d'adapter les mécanismes utilisés dans SABRE pour permettre le calcul de la dissimilarité entre deux motifs.

6.4.1 Calcul de la dissimilarité inter-motifs avec SABRE

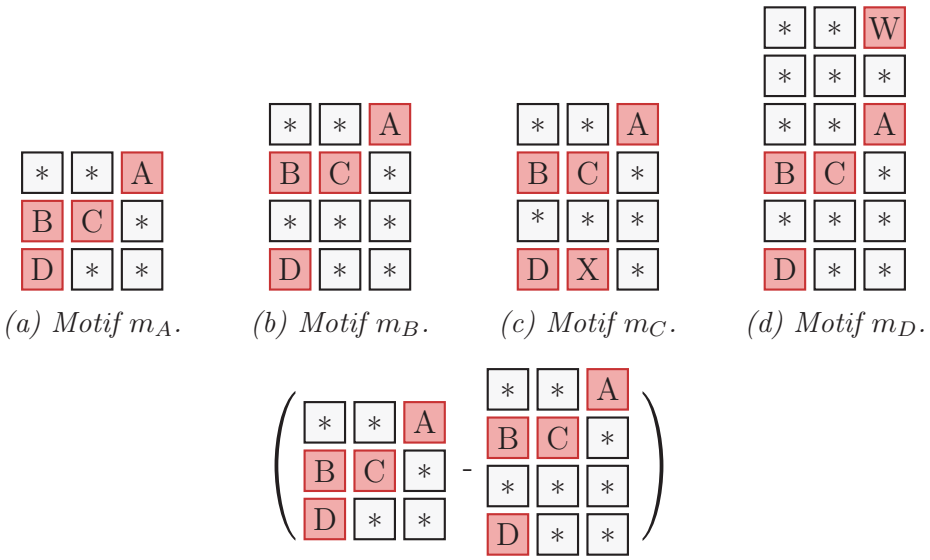
Afin de permettre le calcul de la dissimilarité, il est, de nouveau, nécessaire de représenter un motif m sous la forme d'un tableau d'annotations. Le tableau d'annotations correspondant à un motif m est obtenu de la même manière que pour la méthode LPCA-DC, comme présenté en sous-section 6.3.1.

Soient t et t' les tableaux d'annotations ainsi obtenus pour deux motifs m et m'. Nous souhaitons que plus t et t' sont similaires, plus la dissimilarité entre m et m' (notée $diss(m, m')$) soit faible. Ainsi, la dissimilarité est composée d'un premier terme égal à l'opposé du score d'un alignement local *bien choisi* entre m et m' (nous verrons par la suite comment cet alignement est choisi). Nous notons $\mathcal{A}^*_{m,m'}$ cet alignement et $S(\mathcal{A}^*_{m,m'})$ son score.

La valeur $-S(\mathcal{A}^*_{m,m'})$ n'est cependant pas suffisante pour caractériser la dissimilarité entre deux motifs comme l'illustrent les quatre motifs m_A, m_B, m_C et m_D en figure 6.5. Le meilleur alignement local $\mathcal{A}^*$ (meilleur au sens du score) entre m_A et m_B est représenté en figure 6.5e. L'alignement $\mathcal{A}^*$ correspond aussi au meilleur alignement local entre m_A et m_C ou m_D. Cependant, les motifs m_B, m_C et m_D sont différents et la dissimilarité entre le motif m_A et chacun d'entre eux ne devrait pas être la même. Contrairement à m_B, le motif m_C contient une annotation 'X' qui ne figure pas dans l'alignement $\mathcal{A}^*$. En conséquence, m_C est plus distant de m_A que ne l'est m_B (*i.e.* : $diss(m_A, m_B) < diss(m_A, m_C)$). Ainsi, pour chaque annotation ne figurant pas dans $\mathcal{A}^*$ la dissimilarité entre les deux motifs est augmentée (*i.e.* : pénalisée) d'une valeur $d \in \mathbb{R}^*$.

Un dernier paramètre est à prendre en compte dans le calcul de la dissimilarité. En effet, les motifs m_C et m_D contiennent le même nombre d'annotations, cependant, le motif m_D s'étend sur un plus grand nombre de lignes. En conséquence, la structure de m_D est plus éloignée de m_A que celle de m_C. Ainsi, nous devrions avoir : $diss(m_A, m_C) <$

$diss(m_A, m_D)$. Ceci est pris en compte en augmentant de la valeur d la dissimilarité pour chaque ligne d'un motif non couverte par $\mathcal{A}^*$.



(a) Motif m_A. *(b) Motif m_B.* *(c) Motif m_C.* *(d) Motif m_D.*

(e) Représentation du meilleur alignement local possible $\mathcal{A}^*$ entre m_A et chacun des trois autres motifs.

FIGURE *6.5 – Exemples de motifs, illustrant le fait que le score d'un alignement local ne soit pas suffisant pour déterminer la proximité entre deux motifs.*

Étant donnés deux motifs quelconques m et m', soient
- a_{nc} le nombre d'annotations non couvertes par $\mathcal{A}^*_{m,m'}$;
- l_{nc} le nombre de lignes non couvertes par $\mathcal{A}^*_{m,m'}$.

La dissimilarité entre les motifs m et m' est obtenue par la formule suivante :

$$diss(m, m') = d(a_{nc} + l_{nc}) - S(\mathcal{A}^*_{m,m'}). \tag{6.1}$$

Nous pouvons maintenant préciser comment l'alignement $\mathcal{A}^*_{m,m'}$ est choisi. Ce dernier, correspondra à l'alignement qui permet de minimiser la dissimilarité (*i.e.* : l'équation 6.1). Son obtention est réalisée en utilisant une variante de l'algorithme SABRE. Les deux premières phases de cette adaptation sont identiques (*i.e.* : l'extraction et la fusion de graines), seule la phase d'extension des graines est modifiée. Durant cette phase, la meilleure annotation à ajouter à une graine, ne correspondra plus à celle qui permet de maximiser le score de l'alignement mais à celle qui permet de minimiser la dissimilarité.

6.4.2 Protocole d'expérimentation

L'objectif de cette expérimentation est de comparer l'efficacité des méthodes utilisées durant la première expérimentation (*i.e.* : LPCA-DC et les heuristiques de partitionnement) aux nouvelles méthodes (*i.e.* : SABRE et l'algorithme de plans coupants).

Cette expérimentation est réalisée en quatre étapes :

1. Étude du paramètre c_d de SABRE ;
2. Étude du score minimum τ de SABRE ;

3. Comparaison de SABRE et de LPCA-DC ;
4. Comparaison des heuristiques de partitionnement et de l'algorithme de plans coupants.

L'objectif des deux premières étapes est de déterminer les paramètres de SABRE permettant d'obtenir la meilleure liste d'alignements. Dans la troisième étape, cette liste est comparée à la meilleure des listes obtenues avec LPCA-DC dans l'expérience précédente. Enfin, les méthodes de partitionnement sont ensuite comparées en les appliquant à la meilleure des deux listes de motifs précédentes.

Étape 1 : Étude du paramètre c_d de SABRE

La méthode SABRE comporte plusieurs paramètres :
- une table contenant le score associé à chaque couple d'annotations d'une même colonne(un score est strictement positif si les deux annotations associées sont similaires et nulle sinon) ;
- le score minimum d'une graine g_{min} qui détermine à partir de quand une diagonale de la table C dénote une graine ;
- le coût d'une désynchronisation c_d ;
- le score minimum τ à partir duquel une graine dénote un alignement.

Il convient de déterminer des valeurs pertinentes pour chacun de ces paramètres. Les deux premiers peuvent être directement fixés. En effet, afin de permettre la comparaison avec la méthode LPCA-DC, le score associé à un couple d'annotations (A, B) est égal au score de substitution $sub(A, B)$ utilisé dans l'expérience précédente. Cependant, puisque seuls les scores positifs sont utilisés dans SABRE, les scores négatifs sont fixés à 0.

Le paramètre g_{min} permet de limiter le nombre de graines obtenues dans le cas où la taille des tableaux d'annotations est trop élevée. Dans le corpus considéré dans ce chapitre, nous avons constaté que ce paramètre pouvait être fixé à sa valeur minimum 1 sans que les temps de calculs de SABRE n'en soit significativement impactés. Nous utilisons donc cette valeur minimale qui assure que toutes les graines envisageables ont été prises en compte.

Il reste donc à fixer le coût de désynchronisation c_d et le score minimum τ. Le paramètre c_d aura une influence sur le type de motifs extraits (plus c_d est élevé, plus un motif pourra contenir de pauses et de désynchronisation) tandis que τ n'interviendra qu'en fin d'extraction pour filtrer les alignements dont le score est trop faible. Il convient donc de fixer le paramètre c_d en premier.

Pour ce faire, nous considérons les 20 meilleurs alignements obtenus par SABRE pour différentes valeurs de c_d et demandons à l'expert d'évaluer chaque alignement en répondant aux deux questions suivantes :
- Est-ce que l'alignement est pertinent ? (*i.e.* : est-ce que le fait d'aligner ces deux motifs ensemble a du sens ?). Donner une note entre 0 et 4 (de 0 pour "non pertinent" à 4 pour "pertinent").
- Est-ce que l'alignement est optimal ? Un alignement est optimal si aucune annotation ne devrait être ajoutée ou retirée aux deux motifs qui le composent. Plus le nombre d'annotations à ajouter ou retirer est élevé, moins l'alignement est optimal. Donner une note entre 0 et 4 (de 0 pour "non optimal" à 4 pour "optimal").

La première question a pour objectif d'évaluer si les motifs rapprochés sont sémantiquement similaires. La seconde question permet de déterminer si un alignement pourrait

être améliorer selon l'expert.

Pour chaque valeur de c_d considérée, la moyenne des réponses à ces deux questions sur les 20 meilleurs alignements est calculée et permet d'identifier si une valeur fournit des résultats plus pertinents.

Il reste à choisir les valeurs de c_d qui seront considérées. Le coût d'une désynchronisation doit nécessairement être positif, en conséquence la plus faible valeur considérée est : $c_d = 1$. Étant donnés le corpus et les scores considérés, la valeur maximale de ce paramètre est 9 puisque lorsque cette valeur est utilisé, les 20 meilleurs alignements obtenus ne comportent ni désynchronisations, ni pauses. Ainsi, des valeurs supérieures de c_d fourniraient la même liste de 20 alignements. En conséquences, nous avons sélectionné cinq valeurs pour c_d réparties entre 1 et 9 (à savoir 1, 3, 5, 7 et 9).

Étape 2 : Étude du paramètre τ de SABRE

Une fois le paramètre c_d fixé, une liste d'alignements peut être obtenue. La taille de cette liste dépend du score minimum τ à partir duquel il est considéré qu'une graine dénote un alignement. Plus ce score est faible, plus le nombre d'alignements obtenu est élevé.

Pour fixer ce paramètre, nous prenons une valeur de τ très faible permettant d'obtenir une liste d'alignements aussi complète que possible (plus de 2000 alignements). Les alignements de la liste sont ordonnés en fonction de leur score et l'expert détermine à partir de quel score (*i.e.* : pour quelle valeur de τ) les alignements ne sont plus pertinents.

Étape 3 : Comparaison de SABRE et de LPCA-DC

En fixant les paramètres c_d et τ, la méthode SABRE nous fournit une liste d'alignements. Cette liste est comparée par l'expert à celle obtenue avec LPCA-DC lorsque son paramètre τ est fixé à la meilleure valeur identifiée dans l'expérience précédente (*i.e.* : 38). Dans un premier temps, la pertinence et l'optimalité moyenne des alignements de chacune de ces deux listes est évaluée (de manière similaire à la première étape de l'expérience). L'expert donne aussi une note entre 0 et 4 pour la qualité de la couverture de chacune des listes d'alignements (*i.e.* : quelle proportion de régularités connues du corpus figurent dans ces listes). Ces trois grandeurs permettent de comparer les deux méthodes et de voir si l'une d'entre elle est significativement meilleure.

Étape 4 : Comparaison des deux meilleures heuristiques de partitionnement et de l'algorithme de plans coupants

Dans cette dernière étape, les deux meilleurs heuristiques de partitionnement identifiées durant l'expérience précédente (à savoir ROCK et la propagation d'affinité) sont comparées à l'algorithme de plans coupants. Si nous considérons comme critère le poids de la K-partition, l'algorithme de plans coupants est nécessairement meilleur puisqu'il permet d'obtenir la solution optimale. Cependant, il est peu probable que ce critère reflète à la perfection la façon dont un expert réaliserait le partitionnement de motifs. Ainsi, cette expérience permettra d'évaluer si la solution exacte fournie, représente, aux yeux d'un expert, une amélioration significative par rapport aux solutions heuristiques.

L'inconvénient majeur de la méthode des plans coupants est sont temps d'exécution qui ne permet pas de l'appliquer à une liste de motif conséquente. Comme nous le montrons dans la sous-section suivante, nous avons eu à réaliser un filtrage des motifs

par l'expert avant de pouvoir appliquer les différentes méthodes de partitionnement. Ainsi, seuls les 79 meilleurs motifs ont été conservés. Les valeurs de K considérées dans cette expérience ont été choisies en fonction de ce nombre et sont les suivantes : 5, 20, 40 et 60. Pour chacune des partitions obtenues, l'expert répond aux deux questions suivantes :

- Quelle est, selon vous, la cohérence moyenne des parties ? Une partie est cohérente si elle ne contient que des motifs similaires ; plus le nombre de motifs à retirer de la partie pour qu'elle devienne cohérente est élevé, plus la partie est incohérente (note entre 0 : incohérent et 4 : cohérent).
- Est-ce que, selon vous, certaines parties devraient être fusionnées ? (note entre 0 : aucun et 4 : beaucoup).

La première question a pour but d'évaluer si la méthode de partitionnement renvoie des parties cohérentes. La seconde permet d'évaluer si le nombre de parties est judicieusement choisi.

6.4.3 Résultats

Nous présentons maintenant les résultats obtenus à chacune des étapes de l'expérience.

Résultat étape 1 : Étude du paramètre c_d de SABRE

Pour chacune des 5 valeurs de c_d considérées dans cette expérience, les 20 alignements dont le score est le plus élevé ont été extraits. Ces 100 motifs ont été mélangés aléatoirement puis évalués par l'expert selon leur pertinence et leur optimalité. Les résultats moyens obtenus pour chaque valeur de c_d sont présentés en table 6.11.

Valeur de c_d	1	3	5	7	9
Pertinence moyenne	0,550	1,400	1,550	1,800	2,375
Optimalité moyenne	1,750	2,400	2,400	2,750	3,025

TABLE *6.11 – Résultats moyens obtenus pour les deux critères en fonction de la valeur du coût de désynchronisation c_d.*

Cette table montre clairement que la qualité des alignements – tant du point de vue de la pertinence que de l'optimalité – augmente avec la valeur de c_d. Ceci indique que les alignements compacts ne comportant que peu de pauses ou de désynchronisations sont à privilégier pour ce corpus de tableaux d'annotations.

Ainsi, dans la suite de l'expérience, la valeur de c_d est fixée à la valeur maximale 9.

Résultat étape 2 : Étude du paramètre τ de SABRE

Dans cette étape, nous considérons la liste d'alignements, ordonnés par score décroissant, obtenus lorsque c_d est égal à 9. A partir de cette liste, l'expert a alors déterminé que les alignements dont le score est inférieur à 34 ne sont plus pertinents. Le paramètre τ est donc fixé à 34.

Résultat étape 3 : Comparaison de SABRE et de LPCA-DC

Durant cette étape nous déterminons si, selon l'expert, SABRE permet l'obtention

d'alignements de meilleurs qualité que LPCA-DC. La liste d'alignements obtenue avec SABRE pour les paramètres c_d et τ déterminés dans les deux premières étapes est comparée à la liste d'alignements retournée par LPCA-DC pour sa meilleure valeur de τ (*i.e.* : 38).

Le nombre de motifs obtenus pour ces deux listes ainsi que la répartitions par colonne des annotations figurant dans les motifs sont représentés en table 6.12. Nous pouvons en premier lieu constater dans ce tableau que bien que LPCA-DC retourne un nombre plus faible de motifs, le nombre de lignes du corpus couvertes (*i.e.* : le nombre de lignes des tableaux d'annotations contenant au moins une annotation figurant dans un motif) est plus élevé. Ceci peut être expliqué par le fait que, contrairement à SABRE, LPCA-DC va autoriser des substitutions de score négatif entraînant l'apparition d'annotations dans l'alignement qui ont des significations éloignées. Par exemple, comme l'illustre la figure 6.6, LPCA-DC aura généralement tendance a effectuer une substitution de score négatif entre deux annotations (*e.g.* : substitution de B et Z en figure 6.6a), plutôt que de supprimer l'une et d'insérer l'autre (*e.g.* : insertion de Z et suppression de B en figure 6.6b). Ainsi, les lignes où sont réalisées ce type de substitutions seront susceptibles d'être couvertes par LPCA-DC mais pas par SABRE.

Méthode	**Nombre de motifs**	**Nombre de lignes couvertes**	**Pourcentage d'annotations figurant dans un motif par colonne**			
			1	**2**	**3**	**4**
LPCA-DC	259	408	44	34	11	11
SABRE	371	259	39	29	16	16

TABLE *6.12 – Nombre de motifs contenus dans les deux listes d'alignements fourni par SABRE et LPCA-DC ainsi que le nombre de lignes du corpus couvertes et la répartition des annotations dans les différentes colonnes du schéma de codage.*

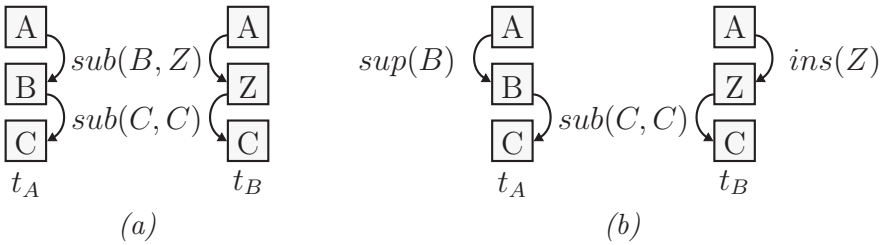


FIGURE *6.6 – Exemple de cas où LPCA-DC peut réaliser une substitution de score négatif. Si le coût de sub(B, Z) est plus faible que celui de sup(B)+ins(Z), les transformation représentées en (a) seront préférées à celle figurant en (b).*

Il est, de plus, possible d'observer en table 6.12, que SABRE permet d'obtenir un pourcentage supérieur d'annotations dans les deux dernières colonnes. Ceci est dû au fait que, contrairement à SABRE, la forme des motifs obtenus par LPCA-DC est contrainte – comme nous l'avons vu dans le chapitre 2 – et que les alignements obtenus auront tendances à principalement couvrir les premières lignes du corpus qui comportent un nombre plus important d'annotations non vides (*i.e.* : l'annotation vide est le caractère '-').

Les résultats fournis par l'expert, représentés en table 6.13, nous permettent de constater que ces deux observations qualitatives semblent avoir un impact significatif sur la qualité des alignements obtenus. Pour chacun des trois critères étudiés, la note attribuée à SABRE est meilleure que celle de LPCA-DC. De plus, le temps d'exécution de LPCA-DC est nettement supérieur. Ceci confirme que SABRE permet une amélioration significative de la qualité des alignements extraits.

Méthode	**Temps d'exécution** (en secondes)	**Pertinence** (note entre 0 et 4)	**Optimalité** (note entre 0 et 4)	**Couverture** (note entre 0 et 4)
LPCA-DC	33	1,5	1,5	1
SABRE	6	2,5	3,5	3

TABLE *6.13 – Résultats de la comparaison des deux listes d'alignements retournées respectivement par SABRE et LPCA-DC. Les valeurs de pertinence et d'optimalité correspondent à une note moyenne, donnée par l'expert, de l'ensemble des motifs d'une liste. La dernière colonne correspond à l'efficacité de la couverture des motifs d'une liste par rapport aux régularités du corpus connues par l'expert.*

Ce sont donc les motifs fournis par SABRE qui seront utilisés pour évaluer la phase de partitionnement durant la dernière étape de cette expérience.

Résultat étape 4 : Comparaison des deux meilleures heuristiques de partitionnement et de l'algorithme de plans coupants

Dans cette dernière étape, les 371 motifs retournés par SABRE vont être partitionnés en utilisant l'algorithme de plans coupants décrit en section 5.2 ainsi que les heuristiques ROCK et propagation d'affinité.

Le graphe correspondant aux 371 motifs retournés par SABRE est, cependant, de taille trop conséquente pour que l'algorithme de plans coupants soit utilisé sur lui. Pour permettre d'obtenir un graphe de taille plus réduite, tout en gardant les motifs les plus pertinents, nous avons demandé à l'expert de donner une note entre 1 et 4 (1 : non pertinent, 4 : très pertinent) à chaque motif. Le nombre de motifs obtenu pour chaque valeur est représenté en table 6.14. Le nombre 79 étant une alternative raisonnable pour l'utilisation de l'algorithme de plans coupants, nous avons conservé l'ensemble des motifs notés "très pertinents".

Note	1	2	3	4
Nombre de motifs	95	63	134	79

TABLE *6.14 – Répartition en quatre catégories (de* 1 *: non pertinent à* 4 *: très pertinent) des motifs faite par l'expert.*

La méthode ROCK a été appliquée sur le graphe correspondant et les solutions pour K égal à 5, 20, 40 et 60 ont été sélectionnées. L'algorithme de plans coupants a été appliqué pour chacune de ces valeurs également et la propagation d'affinité (pour laquelle K n'est pas fixé par l'utilisateur) a retourné une solution comportant 5 parties.

Ces diverses solutions peuvent, tout d'abord, être comparées du point de vue de la fonction objectif de l'algorithme de plans coupants afin d'observer si ROCK et la propagation d'affinité fournissent des partitions dont le poids est très éloigné de l'optimum. La

table 6.15 présente les résultats obtenus et montre que les heuristiques fournissent des partitions de poids nettement plus élevé. Néanmoins, aux yeux d'un expert, la qualité d'une partition ne correspondra pas nécessairement au poids de cette dernière. Ainsi, cette table ne peut prétendre fournir la vérité terrain et c'est pourquoi nous passons par une validation par un expert.

Méthode	**Nombre de parties** K			
	5	20	40	60
ROCK	12763	2369	574	318
Propagation d'affinité	3742	-	-	-
Plans coupants	-4612	-5974	-4846	-2679

TABLE *6.15 – Poids des différentes partitions considérées.*

Les résultats obtenus pour la question portant sur la cohérence moyenne des parties sont présentés en figure 6.16a. Nous pouvons, tout d'abord, remarquer que les partitions comportant 5 et 60 parties ont toutes un score égal à 0. Ceci confirme les résultats de l'expérience précédente dans laquelle les partitions correspondant à des valeurs extrêmes de K fournissaient des résultats peu pertinents (car elles regroupaient ensembles trop ou pas assez de motifs). Nous observons, ensuite, que l'ensemble des partitions correspondant aux heuristiques ont, elles aussi, un score de cohérence moyenne minimal, ce qui n'est pas le cas des solutions obtenues par la méthode des plans coupants pour K égal à 20 et 40. D'après l'expert, les partitions correspondant à l'algorithme de plans coupants ont permis une analyse plus fine que celles effectuées durant la première expérience. Dans cette dernière, l'expert mettait en particulier l'accent sur la forme des motifs plutôt que sur le détail de leur contenu. La qualité des partitions obtenues par les plans coupants a permis une expertise plus sévère des partitions, expliquant ainsi les notes accordées aux deux meilleures heuristiques. Notre algorithme de plans coupants fournit donc de très bonnes partitions. Les notes associées à ces partitions ne sont, néanmoins, pas maximales car dans ces dernières, un certain nombre de parties sont réduites à un unique sommet qui aurait pourtant sa place dans une autre partie, baissant ainsi la cohérence moyenne de la partition.

La seconde note donnée par l'expert durant cette phase quantifie la proportion de parties d'une partition qui gagneraient à être fusionnées avec une autre. Les résultats obtenus sont présentés en table 6.16b. Puisqu'une partie issue d'une fusion entre deux parties incohérentes est nécessairement incohérente, les notes accordées aux partitions relatives au heuristiques ROCK et propagation d'affinité sont égales à 0. La dernière ligne de cette table indique que pour K supérieur ou égal à 20, la méthode des plans coupants gagnerait à fusionner des parties entre elles. Étant donnés les scores de cohérence de la table 6.16a, nous pouvons supposer que l'utilisation de valeurs de K entre 5 et 20 permettrait l'obtention de partitions encore plus pertinentes.

Régularités extraites par SABRE et l'algorithme de plans coupants

Cette expérience a permis de montrer, d'une part, que SABRE fournit de meilleurs motifs que LPCA-DC et, d'autre part, que l'algorithme de plans coupants retourne des partitions de bien meilleure qualité que les algorithmes heuristiques que nous avons considérés. Ainsi, l'utilisation conjointe de ces deux méthodes permet l'obtention de régularités que des extractions manuelles précédentes n'avaient pas permis d'obtenir et

Méthode	**Nombre de parties** K			
	5	20	40	60
ROCK	0	0	0	0
Propagation d'affinité	0	-	-	-
Plans coupants	0	2	1	0

(a) Quelle est la cohérence moyenne des parties ? (0 : incohérent, 4 : cohérent)

Méthode	**Nombre de parties** K			
	5	20	40	60
ROCK	0	0	0	0
Propagation d'affinité	0	-	-	-
Plans coupants	0	2	3	4

(b) Est-ce que certaines parties devraient être fusionnées ? (0 : aucun, 4 : beaucoup)

TABLE *6.16 – Résultat des méthodes de partitionnement pour les deux critères étudiés.*

qui, de plus, sont de bien meilleure qualité que celles obtenues à l'issue de la première expérience.

Nous présentons ici deux regroupements de motifs faisant partie de ceux ayant été jugés les plus pertinents par l'expert. Ils proviennent tous deux de la solution obtenue avec l'algorithme de plans coupants pour une valeur de K égale à 20.

Le premier regroupement comprend les quatre motifs représentés en figure 6.7. Il correspond à une stratégie d'explication mise en œuvre par les parents. Afin de permettre la compréhension du comportement d'un personnage, les parents énoncent, tout d'abord, une justification fondée sur un état mental (*e.g.* : "parce qu'il est un peu triste"). Cependant, cette explication ne semble pas être suffisante pour les enfants de 3 à 5 ans impliqués dans les quatre dialogues concernés et les parents renforcent leur explication en proposant par deux fois des énoncés portant soit sur la même catégorie d'état mental que celle citée dans l'explication (*e.g.* : "il pleure" et "Popi à un peu peur"), soit par des états mentaux complémentaires.

La seconde régularité ayant été interprétée par l'expert est présentée en figure 6.8. Dans les trois motifs qui la composent, les parents tentent de faire comprendre à l'enfant que le comportement du personnage est la conséquence de son état mental. Ainsi, ils prennent le temps de décrire l'état mental du personnage avant d'apporter par la suite une justification de son comportement. Du point de vue applicatif, il est intéressant de remarquer que la forme du motif présenté en figure 6.8a rend peu probable son obtention par l'algorithme LPCA-DC. La régularité correspondante ce serait donc avérée plus ardue à identifier si cet algorithme d'extraction avait été utilisé.

6.5 Discussion

Ces deux expérimentations ont permis de justifier – en entraînant l'obtention de régularités préalablement inconnues – la pertinence de notre approche consistant à fournir

Locuteur	Dialogues	Annotations			
Parent	alors Léo n'est pas content,	P	E	C	CH
Parent	Il pleure.	P	E	-	-
Parent	Pendant ce temps Popi à un peu peur.	P	E	-	

(a)

Locuteur	Dialogues	Annotations			
Parent	parce qu'il est un peu triste.	P	E	C	CH
Parent	D'habitude il est toujours content,	P	E	-	-
Parent	et là il est triste	P	E	-	-

(b)

Locuteur	Dialogues	Annotations			
Parent	mais comme Léo voulait emprunter la pelle à Timothée	P	V	C	CH
Parent	il voulait pas Timothé	P	V	-	-
Parent	Il voulait pas	P	V	-	-

(c)

Locuteur	Dialogues	Annotations			
Parent	et qu'en fin de compte c'était une...	P	S	O	CH
Parent	Ils voulaient lui dire	P	V	-	-
Parent	qu'ils étaient content d'avoir un roi pareil	P	E	-	-

(d)

FIGURE *6.7 – Représentation de motifs appartenant à la première partie de la solution de l'algorithme des plans coupants pour K égal à* 20.

un outil d'aide à la décision pour l'extraction de régularités dans des corpus de tableaux d'annotations. Nous avons, de plus, pu constater l'importance cruciale des méthodes utilisées tant pour la phase d'extraction que pour celle de partitionnement.

Du point de vue de l'extraction, la méthode SABRE permet d'obtenir de bien meilleurs résultats que l'algorithme LPCA-DC et ce, en des temps de calculs plus faibles.

Parmi les heuristiques de partitionnement, ROCK et la propagation d'affinité sont celles qui fournissent les meilleurs partitions. Ces dernières sont néanmoins sans comparaison avec les partitions qu'il est possible d'obtenir avec l'algorithme de plans coupants. De part sa complexité, ce dernier a, néanmoins, pour premier désavantage de ne pouvoir être appliqué sur un graphe comportant un nombre élevé de sommets. Il serait envisageable d'adapter la méthode de partitionnement utilisée (soit approchée, soit exacte), en fonction du nombre de motifs extraits, ou bien de proposer une phase de filtrage des motifs à l'utilisateur afin d'en obtenir un nombre suffisamment petit pour permettre

Locuteur	Dialogues	Annotations			
Parent	On dirait	AR	HY	-	-
Parent	que Léo commence à être moins content	P	E	-	-
Parent	peut-être que Timothée lui a pris sa pelle.	P	HY	-	-
Parent	Alors Popi il se cache.	P	-	C	CH
Parent	Il a pas envie	P	V	-	-

(a)

Locuteur	Dialogues	Annotations			
Parent	Parce que les animaux ils ne savent pas ce que c'est le pot.	P	CNO	-	-
Enfant	Ah oui, oui, il sait pas ce que c'est !	P	CNO	-	-
Parent	Bah non, tous les animaux ne savent pas ce que c'est.	P	CNO	-	-
Parent	Toi tu sais	HR	CNO	-	-
Parent	parce que tu es un petit garçon	HR	-	C	CP

(b)

Locuteur	Dialogues	Annotations			
Parent	Mais comme Léo voulait emprunter la pelle à Timothée.	P	V	C	CH
Parent	Il voulait pas Timothée.	P	V	-	-
Parent	Il voulait pas.	P	V	-	-
Enfant	Pourquoi ?	-	-	-	-
Parent	Bah parce qu'il avait pas envie de prêter sa pelle.	P	V	C	CH
Parent	Alors qu'est-ce qu'il à fait Léo ?	P	-	C	CH

(c)

FIGURE *6.8 – Représentation de motifs appartenant à la troisième partie de la solution de l'algorithme des plans coupants pour K égal à* 20.

l'utilisation de l'algorithme de plans coupants. Une dernière solution envisageable serait d'interrompre l'algorithme de plans coupants avant obtention de la solution optimale et d'utiliser la meilleure solution entière obtenue. Enfin, l'algorithme de plans coupants tend à retourner des parties réduites à un unique sommet qui, selon l'expert, gagneraient à être fusionnées avec des parties de taille plus conséquentes. Ceci confirme que la fonction objectif ainsi que les contraintes choisies, ne correspondent pas de manière totalement parfaite à la façon dont un expert partitionnerait les motifs. Il serait intéressant de chercher à déterminer ces différences et d'étudier comment adapter la formulation du problème en conséquence.

Conclusion et perspectives

Cette thèse s'inscrit dans le cadre de l'extraction de schémas dialogiques récurrents dans des dialogues. Afin de permettre la prise en compte d'interactions aussi bien verbales que non verbales, nous choisissons de considérer un corpus de tableaux d'annotations correspondant à des dialogues encodés par le biais d'un schéma de codage.

Dans cette optique, nous proposons une méthodologie pour le problème d'identification de régularités dans laquelle des motifs dialogiques sont extraits puis partitionnés. Chacune de ces deux étapes nous amène à considérer un problème d'optimisation combinatoire particulièrement difficile que nous étudions de manière approfondie. Ces études aboutissent à la réalisation du logiciel d'aide à la décision VIESA permettant d'extraire et de visualiser des régularités dialogiques. Au sein de ce dernier, l'utilisateur peut orienter sa recherche en fonction de ses besoins en fixant les paramètres importants de la méthodologie (corpus considéré, colonnes du schéma de codage prises en compte, scores de similarité inter-annotations et nombre de régularités obtenues). Actuellement le choix de la méthode de partitionnement n'est pas laissé à l'utilisateur. À terme, il pourrait être envisageable de calculer une estimation de la durée de résolution exacte du problème de partitionnement et de laisser à l'utilisateur le choix de la méthode de partitionnement utilisée si une résolution exacte n'est pas envisageable.

Dans la suite de cette conclusion, nous résumons pour chacune des deux phases de notre méthodologie les contributions apportées, les qualités et les points faibles des approches mises en œuvre ainsi que les perspectives de recherche.

Extraction de motifs dans des tableaux d'annotations

Nous montrons que différentes approches, telles que la fouille de motifs récurrents ou la fouille de séquences de caractères, sont envisageables afin d'extraire des motifs récurrents d'un corpus de tableaux d'annotations. La méthode que nous identifions comme la plus pertinente est un algorithme de programmation dynamique, que nous nommons LPCA, et qui permet d'extraire l'alignement local de poids maximal entre deux tableaux de caractères. Nous définissons une adaptation appelée LPCA-DC permettant de répondre plus efficacement aux besoins de notre problématique. La première adaptation provient du fait que les tableaux d'annotations que nous considérons suivent un schéma de codage bien spécifique. Nous constatons qu'il est possible de tirer parti de cette structure des tableaux d'annotations afin de réduire la complexité de l'algorithme. La seconde adaptation est une modification de la phase de tracé arrière afin qu'elle ne retourne plus uniquement l'alignement de score maximal, mais un ensemble d'alignements de scores supérieurs ou égal à $\tau \in \mathbb{R}^*$.

Nous montrons, lors d'une première expérience sur un corpus de tableaux d'annotations encodant des dialogues, que LPCA-DC permet d'obtenir des régularités pertinentes

et interprétables par un expert. Cependant, l'expérience permet aussi de mettre en avant les faiblesses de cette approche. En effet, de part les opérations d'édition considérées dans LPCA-DC, le parcours des tableaux d'annotations lors de la recherche de motifs est contraint, empêchant ainsi l'obtention de certaines formes de motifs. Notamment, l'ordre dans lequel les colonnes d'annotations sont disposées influence significativement les résultats.

Afin de pallier ces faiblesses, nous présentons une définition plus adéquate du score d'un alignement local, qui est indépendante de l'ordre dans lequel se trouvent les colonnes d'annotations. Ceci nous permet de définir formellement le problème d'extraction d'alignements de motifs dans des tableaux d'annotations, sous la forme d'une recherche d'arborescences de poids maximale dans un graphe. Nous créons ensuite une heuristique, nommée SABRE, qui tient compte de l'indépendance de l'ordre des colonnes au sein du schéma de codage lors de la résolution du problème. Enfin, une expérience a mis en évidence que SABRE fournit des alignements bien plus pertinents que LPCA-DC.

En perspective de ces travaux, il serait important d'étudier la complexité du problème d'extraction d'arborescences que nous avons défini afin de déterminer si ce dernier est $\mathcal{NP}$-complet, comme nous le supposons, et le cas échéant de rechercher un schéma d'approximation polynomiale. L'algorithme SABRE pourrait être amélioré en permettant, par exemple, la fusion de graines incompatibles lors de ses deux dernières phases. Pour cela, il serait nécessaire de déterminer la façon la plus pertinente de gérer les incompatibilités.

Classification non supervisée de motifs

Nous identifions, sélectionnons, puis implantons diverses méthodes heuristiques de partitionnement provenant de la littérature basées sur des mécanismes variés. L'expérience en section 6.3 illustre la variabilité des partitions obtenues en fonction de l'heuristique utilisée. Cela nous amène à supposer qu'une approche exacte du problème de partitionnement pourrait améliorer de manière significative la pertinence des résultats. Nous définissons ainsi deux formulations originales du problème de K-partitionnement utilisant des variables de représentants afin de fixer le nombres de parties à K. Puis, une étude polyèdrale de l'enveloppe convexe des points entiers $\mathcal{P}_{n,K}$ correspondante est réalisée. Trois familles de facettes non triviales sont, en particulier, étudiées – à savoir les inégalités de 2-partition, les inégalités 2-*chorded cycles* et les inégalités de clique généralisées – et des conditions sous lesquelles elles définissent des facettes de $P_{n,K}$ sont définies.

Le pourcentage d'amélioration moyen de la solution de la relaxation linéaire lors de l'ajout de coupes provenant de ces familles d'inégalités est évalué. Il apparaît que le type de données considérées influence grandement, d'une part, cette amélioration et d'autre part, la résolution du problème de K-partitionnement en elle-même. De plus, nous constatons que les coupes susceptibles d'être les plus efficaces sont les inégalités de 2-partition et les inégalités de clique généralisées. Ces résultats sont mis à profit afin de développer un algorithme de type plans coupants performant pour la résolution du problème de K-partitionnement.

Les performances des meilleurs heuristiques de partitionnement identifiées sont comparées avec celles de l'algorithme de plans coupants lors de la seconde expérience présentée en section 6.4. Bien que les temps d'exécution de ce dernier soient plus courts que

ceux du logiciel commercial CPLEX, ils sont, néanmoins, sans comparaison avec ceux des approches heuristiques considérées. Notre algorithme n'est, cependant, pas dépourvu d'avantages puisque l'évaluation de ses partitions a montré qu'elles étaient nettement meilleures que celles obtenues par les heuristiques.

L'expert du corpus de dialogues étudié a remarqué, durant la seconde expérience, que notre algorithme fournit généralement des partitions contenant un grand nombre de parties réduites à un ou deux motifs. Ceci pourrait être évité en ajoutant une borne minimale sur le nombre de sommets qu'une partie doit contenir. Cependant, il est tout à fait envisageable qu'un motif récurrent obtenu dans un alignement n'apparaisse pas plus de deux fois dans le corpus et, dans ce cas, il est souhaitable qu'il se trouve dans une partie de taille inférieure ou égale à deux. Une solution alternative pourrait être de modifier la façon dont les dissimilarités inter-motifs sont calculées. En effet, si une plus grande proportion des arêtes du graphe avait des valeurs positives, les sommets seraient plus équitablement distribués entre les différentes parties.

Nous avons pu constater qu'au delà d'une certaine taille de graphes, notre algorithme de plans coupants ne permettait pas de trouver la solution optimale du problème de K-partitionnement en un temps raisonnable. Diverses pistes sont envisageables afin de repousser cette limite. La plus naturelle d'entre elles serait de poursuivre l'étude polyèdrale en identifiant de nouvelles familles d'inégalités définissant des facettes de $P_{n,K}$ ou en renforçant les inégalités de certaines familles déjà identifiées. Par exemple, il serait intéressant d'étudier si les inégalités de clique généralisées correspondant à des ensembles de taille supérieure à $2K-1$ peuvent être renforcées afin de définir des facettes. L'amélioration des algorithmes de séparation utilisés constitue une seconde piste sérieuse pour l'amélioration des performances. Enfin, il est important de préciser que dans les cas où une résolution exacte ne peut être envisagée, notre approche permet néanmoins – grâce à l'utilisation de l'algorithme glouton qui retourne une solution entière à partir d'une solution fractionnaire – d'obtenir rapidement des solutions réalisables du problème de K-partitionnement. Le développement de techniques similaires plus efficaces constitue une dernière perspective de recherche prometteuse.

Index

A

algorithme de Needleman-Wunsch . . . 36
algorithme de Smith-Waterman . . . 38
algorithme de type Kernighan-Lin . . . 121
algorithme CHAMELEON . . . 70
algorithme de Kernighan-Lin . . . 72
algorithme de plans coupants . . . 129
algorithme LPCA . . . 41
algorithme ROCK . . . 69
algorithmes de séparation . . . 77
alignement . . . 52
alignement local . . . 36
alignement global . . . 35
alphabet . . . 10

B

branch and bound . . . 77
branch-and-bound . . . 75
branch-and-cut . . . 76
branch-and-price . . . 77
branch-and-price-and-cut . . . 77

C

caractère joker . . . 10
coupe . . . 72
couple d'annotations simultanés . . . 53
couple d'annotations désynchronisés . . . 53
couple d'annotations . . . 52
couple d'annotations consécutifs . . . 53
couple d'annotations successifs . . . 53
couples d'annotations compatibles . . . 52
coût de désynchronisation . . . 53
coût d'agencement d'un alignement . . . 55
coût d'agencement d'un couple d'annotations . . . 53
coût de pause . . . 54

D

dimension d'un schéma de codage . . . 9
distance d'édition . . . 34
distance de Levenshtein . . . 35

F

facette . . . 77
facette triviale . . . 99
fonction indicatrice . . . 86

G

génération de colonnes . . . 77
graines . . . 58
graines compatibles . . . 60

H

hiérarchie . . . 68

I

id-list . . . 26
inégalité valide . . . 77
inégalité de clique . . . 115
inégalités de clique généralisées . . . 115
inégalité serrée . . . 133
inégalités 2−chorded cycles . . . 104
inégalités de 2-partitions . . . 107
inégalités triangulaires . . . 78, 87
insertion . . . 34
itemset . . . 19
itemset fréquent . . . 19

K

k-itemset . . . 20
K-coupe . . . 72
k-séquence . . . 23

L

LPCA-DC . . . 47

M

méthode de plans coupants . . . 75
métrique single-link . . . 69
mincut . . . 72
motif récurrent . . . 10

N

nombre de Stirling de seconde espèce . . . 71

O
occurrence de motif 11
opération d'édition 34

P
partition 68
pause 54
polytope 77
positions candidates 49
pourcentage d'amélioration 123
problème de séparation 77, 120
programme linéaire 74
propagation d'affinité 71

R
relaxation linéaire 75, 90
représentant 88

S
schéma de codage 9
score d'un couple d'annotations 52
séquences d'itemsets 23
sous-séquence d'itemsets 23
substitution 34
suppression 35

T
taille d'une désynchronisation 53
transaction 19
transformation 86
transformation valide 86

Références bibliographiques

[1] Mohamed Abouelhoda and Moustafa Ghanem. String mining in bioinformatics. In *Scientific Data Mining and Knowledge Discovery*, pages 207–247. Springer, 2010.

[2] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 3–14. IEEE, 1995.

[5] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[6] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, David J Lipman, et al. Basic local alignment search tool. *Journal of Molecular Biology (J. Mol. Biol.)*, 215(3) :403–410, 1990.

[7] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast : a new generation of protein database search programs. *Nucleic acids research*, 25(17) :3389–3402, 1997.

[8] Amihood Amir and Martin Farach. Efficient 2-dimensional approximate matching of non-rectangular figures. In *SODA*, pages 212–223, 1991.

[9] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics : Ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[10] Alberto Apostolico, Laxmi Parida, and Simona E Rombo. Motif patterns in 2d. *Theoretical Computer Science*, 390(1) :40–55, 2008.

[11] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435. ACM, 2002.

[12] Ricardo Baeza-Yates and Gonzalo Navarro. Fast two-dimensional approximate pattern matching. In *LATIN'98 : Theoretical Informatics*, pages 341–351. Springer, 1998.

[13] Hans-Jürgen Bandelt, Maarten Oosten, Jeroen HGC Rutten, and Frits CR Spieksma. Lifting theorems and facet characterization for a class of clique partitioning inequalities. *Operations research letters*, 24(5) :235–243, 1999.

[14] Francisco Barahona and Ali Ridha Mahjoub. On the cut polytope. *Mathematical programming*, 36(2) :157–173, 1986.

[15] Earl R Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4) :541–550, 1982.

[16] R. Barzilay, M. Collins, J. Hirschberg, and S. Wittaker. The rules behind roles : Identifying speaker role in radio broadcasts. In *National Conference on Artificial Intelligence (NCAI)*, pages 679–684. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999, 2000.

[17] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[18] Pierre Bonami, VietHung Nguyen, Michel Klein, and Michel Minoux. On the solution of a graph partitioning problem under capacity constraints. In A.Ridha Mahjoub, Vangelis Markakis, Ioannis Milis, and VangelisTh. Paschos, editors, *Combinatorial Optimization*, volume 7422 of *Lecture Notes in Computer Science*, pages 285–296. Springer Berlin Heidelberg, 2012.

[19] Lorenzo Brunetta, Michele Conforti, and Giovanni Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78(2) :243–263, 1997.

[20] Harry Bunt. The dit++ taxonomy for functional dialogue markup. In *AAMAS 2009 Workshop, Towards a Standard Markup Language for Embodied Dialogue Acts*, pages 13–24, 2009.

[21] Manoel Campêlo, Victor A Campos, and Ricardo C Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7) :1097–1111, 2008.

[22] D. Catanzaro, E. Gourdin, M. Labbé, and F.A. Özsoy. A branch-and-cut algorithm for the partitioning-hub location-routing problem. *Computers & Operations Research*, 38(2) :539–549, 2011.

[23] Peter Cheeseman and John Stutz. Bayesian classification (autoclass) : Theory and results. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[24] S. Chopra and MR Rao. The partition problem. *Mathematical Programming*, 59(1) :87–115, 1993.

[25] S. Chopra and MR Rao. Facets of the k-partition polytope. *Discrete Applied Mathematics*, 61(1) :27–48, 1995.

[26] A. Chowdhary, S. Kale, P. Saripella, N. Sehgal, and R. Gupta. A general approach for regularity extraction in datapath circuits. In *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, pages 332–339. IEEE, 1998.

[27] M. Conforti, MR Rao, and A. Sassano. The equipartition polytope. i : formulations, dimension and basic facets. *Mathematical Programming*, 49(1) :49–70, 1990.

[28] Michele Conforti, MR Rao, and Antonio Sassano. The equipartition polytope. ii : valid inequalities and facets. *Mathematical Programming*, 49(1-3) :71–90, 1990.

[29] Elias Dahlhaus, David S Johnson, Christos H Papadimitriou, Paul D Seymour, and Mihalis Yannakakis. The complexity of multiway cuts. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 241–251. ACM, 1992.

[30] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[31] M. Deza, M. Grötschel, and M. Laurent. Clique-web facets for multicut polytopes. *Mathematics of Operations Research*, pages 981–1000, 1992.

[32] S. D'Mello, A. Olney, and N. Person. Mining collaborative patterns in tutorial dialogues. *Journal of Educational Data Mining (JEDM)*, 2(1) :1–37, 2010.

[33] William E Donath and Alan J Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5) :420–425, 1973.

[34] Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné. Graphstream : A tool for bridging the gap between complex systems and dynamic graphs. In Aziz Alaoui and Cyrille Bertelle, editors, *Proceedings of Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007), October 4-5. Dresden, Germany.*, pages 63–72, 2007.

[35] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[36] Neng Fan and Panos M Pardalos. Linear and quadratic programming approaches for the general graph partitioning problem. *Journal of Global Optimization*, 48(1) :57–71, 2010.

[37] Simone Faro and Thierry Lecroq. The exact online string matching problem : a review of the most recent results. *ACM Computing Surveys (CSUR)*, 45(2) :13, 2013.

[38] C.E. Ferreira, A. Martin, C.C. de SOUZA, R. Weismantel, and L.A. Wolsey. Formulations and valid inequalities for the node capacitated graph partitioning problem. *Mathematical Programming*, 74(3) :247–266, 1996.

[39] C.E. Ferreira, A. Martin, CC De Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem : a computational study. *Mathematical Programming*, 81(2) :229–256, 1998.

[40] Pierre Feyereisen, Michèle Van de Wiele, and Fabienne Dubois. The meaning of gestures : What can be understood without speech ? *Cahiers de Psychologie Cognitive/Current Psychology of Cognition*, 1988.

[41] Céline Fiot, Anne Laurent, Maguelonne Teisseire, and Bénédicte Laurent. Why fuzzy sequential patterns can help data summarization : An application to the inpi trade mark database. In *Fuzzy Systems, 2006 IEEE International Conference on*, pages 699–706. IEEE, 2006.

[42] K Florek, J Łukaszewicz, J Perkal, Hugo Steinhaus, and S Zubrzycki. Sur la liaison et la division des points d'un ensemble fini. In *Colloquium Mathematicae*,

volume 2, pages 282–285. Institute of Mathematics Polish Academy of Sciences, 1951.

[43] B.J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814) :972–976, 2007.

[44] Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data clustering : theory, algorithms, and applications*, volume 20. Siam, 2007.

[45] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3) :237–267, 1976.

[46] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[47] Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Spirit : Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.

[48] Olivier Goldschmidt and Dorit S Hochbaum. Polynomial algorithm for the k-cut problem. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 444–451. IEEE, 1988.

[49] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1) :59–96, 1989.

[50] M. Grötschel and Y. Wakabayashi. Composition of facets of the clique partitioning polytope. *Topics in Combinatorics and Graph Theory*, pages 271–284, 1990.

[51] M. Grötschel and Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47(1) :367–387, 1990.

[52] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure : an efficient clustering algorithm for large databases. In *ACM SIGMOD Record*, volume 27, pages 73–84. ACM, 1998.

[53] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock : A robust clustering algorithm for categorical attributes. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 512–521. IEEE, 1999.

[54] L. Hagen and A. Kahng. Fast spectral methods for ratio cut partitioning and clustering. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 10–13. IEEE, 1991.

[55] William W Hager, Dzung T Phan, and Hongchao Zhang. An exact algorithm for graph partitioning. *Mathematical Programming*, 137(1-2) :531–556, 2013.

[56] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining : current status and future directions. *Data Mining and Knowledge Discovery*, 15(1) :55–86, 2007.

[57] Jiawei Han and Yongjian Fu. Discovery of multiple-level association rules from large databases. In *VLDB*, volume 95, pages 420–431, 1995.

[58] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan : frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM, 2000.

[59] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[60] P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical programming*, 79(1) :191–215, 1997.

[61] Yi-Chung Hu, Ruey-Shun Chen, Gwo-Hshiung Tzeng, and Jia-Hourng Shieh. A fuzzy data mining algorithm for finding sequential patterns. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 11(02) :173–193, 2003.

[62] Yi-Chung Hu, Gwo-Hshiung Tzeng, and Chin-Mi Chen. Deriving two-stage learning sequences from knowledge in fuzzy sequential pattern mining. *Information Sciences*, 159(1) :69–86, 2004.

[63] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer, 2000.

[64] A.K. Jain. Data clustering : 50 years beyond k-means. *Pattern Recognition Letters*, 31(8) :651–666, 2010.

[65] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[66] X. Ji and JE Mitchell. The clique partitioning problem with minimum size requirement. Technical report, Technical report, Rensselaer Polytechnic Institute, 2005.

[67] X. Ji and J.E. Mitchell. Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. *Discrete Optimization*, 4(1) :87–102, 2007.

[68] Ruoming Jin, Chao Wang, Dmitrii Polshakov, Srinivasan Parthasarathy, and Gagan Agrawal. Discovering frequent topological structures from graph datasets. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 606–611. ACM, 2005.

[69] E.L. Johnson, A. Mehrotra, and G.L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62(1) :133–151, 1993.

[70] Volker Kaibel, Matthias Peinhardt, and Marc E Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4) :595–610, 2011.

[71] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[72] G. Karypis, E.H. Han, and V. Kumar. Chameleon : Hierarchical clustering using dynamic modeling. *Computer*, 32(8) :68–75, 1999.

[73] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2) :291–307, 1970.

[74] Nikhil S Ketkar, Lawrence B Holder, and Diane J Cook. Subdue : Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open source data mining : frequent pattern mining implementations*, pages 71–76. ACM, 2005.

[75] Samir Khuller and Barna Saha. On finding dense subgraphs. In *Automata, Languages and Programming*, pages 597–608. Springer, 2009.

[76] Kamala Krithivasan and R Sitalakshmi. Efficient two-dimensional pattern matching in the presence of errors. *Information Sciences*, 43(3) :169–184, 1987.

[77] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1) :48–50, 1956.

[78] Hye-Chung Kum, Jian Pei, Wei Wang, and Dean Duncan. Approxmap : Approximate mining of consensus sequential patterns. In *SDM*, pages 311–315. SIAM, 2003.

[79] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 313–320. IEEE, 2001.

[80] Michihiro Kuramochi and George Karypis. Grew-a scalable frequent subgraph discovery algorithm. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 439–442. IEEE, 2004.

[81] M. Labbé and F.A. Özsoy. Size-constrained graph partitioning polytopes. *Discrete Mathematics*, 310(24) :3473–3493, 2010.

[82] Philip Laird. Identifying and using patterns in sequential data. In *Algorithmic learning theory*, pages 1–18. Springer, 1993.

[83] T. Lecroq, A. Pauchet, É. Chanoni, and G.A. Solano. Pattern discovery in annotated dialogues using dynamic programming. *International Journal of Information and Decision Sciences (IJIDS)*, 6(6) :603–618, 2012.

[84] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[85] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, pages 498–516, 1973.

[86] David J Lipman and William R Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693) :1435–1441, 1985.

[87] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming*, 95(1) :91–101, 2003.

[88] Nizar R Mabroukeh and Christie I Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys (CSUR)*, 43(1) :3, 2010.

[89] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[90] Florent Masseglia, Fabienne Cathala, and Pascal Poncelet. The psp approach for mining sequential patterns. In *Principles of Data Mining and Knowledge Discovery*, pages 176–184. Springer, 1998.

[91] Albert Mehrabian and Morton Wiener. Decoding of inconsistent communications. *Journal of personality and social psychology*, 6(1) :109, 1967.

[92] A. Mehrotra and M.A. Trick. Cliques and clustering : A combinatorial approach. *Operations Research Letters*, 22(1) :1–12, 1998.

[93] J.E. Mitchell. Branch-and-cut for the k-way equipartition problem. Technical report, Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, 2001.

[94] John E Mitchell. Realignment in the national football league : Did they do it right ? *Naval Research Logistics (NRL)*, 50(7) :683–701, 2003.

[95] Carl H Mooney and John F Roddick. Sequential pattern mining–approaches and algorithms. *ACM Computing Surveys (CSUR)*, 45(2) :19, 2013.

[96] E. Moritz. *Routine Activity Extraction from Local Alignments in Mobile Phone Context Data*. PhD thesis, INSA de Rouen, 2014.

[97] Rudolf Müller. On the partial order polytope of a digraph. *Mathematical Programming*, 73(1) :31–49, 1996.

[98] S.B. Needleman, C.D. Wunsch, et al. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology (J. Mol. Biol.)*, 48(3) :443–453, 1970.

[99] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4) :308–313, 1965.

[100] Siegfried Nijssen and Joost N Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 647–652. ACM, 2004.

[101] M. Oosten, J. Rutten, and F.C.R. Spieksma. The facial structure of the clique partitioning polytope. *Report M*, pages 95–09, 1995.

[102] M. Oosten, J.H.G.C. Rutten, and F.C.R. Spieksma. The clique partitioning problem : facets and patching facets. *Networks*, 38(4) :209–226, 2001.

[103] Salvatore Orlando, Raffaele Perego, and Claudio Silvestri. A new algorithm for gap constrained sequence mining. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 540–547. ACM, 2004.

[104] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001.

[105] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. Multi-dimensional sequential pattern mining. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 81–88. ACM, 2001.

[106] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6) :1389–1401, 1957.

[107] Maurice Queyranne. On optimum size-constrained set partitions. *Proceedings of AUSSOIS*, 1999.

[108] D.S. Rao and F.J. Kurdahi. On clustering for maximal regularity extraction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 12(8) :1198–1208, 1993.

[109] Ellen Riloff, Rosie Jones, et al. Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI/IAAI*, pages 474–479, 1999.

[110] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[111] Lior Rokach. *Data mining with decision trees : theory and applications*, volume 69. World scientific, 2008.

[112] Masakazu Seno and George Karypis. Slpminer : An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *Data Mining,*

2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, pages 418–425. IEEE, 2002.

[113] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8) :888–905, 2000.

[114] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology (J. Mol. Biol.)*, 147(1) :195 – 197, 1981.

[115] M.M. Sørensen. b-tree facets for the simple graph partitioning polytope. *Journal of combinatorial optimization*, 8(2) :151–170, 2004.

[116] M.M. Sørensen. Facet-defining inequalities for the simple graph partitioning polytope. *Discrete Optimization*, 4(2) :221–231, 2007.

[117] Thorvald Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. *Biol. skr.*, 5 :1–34, 1948.

[118] Ramakrishnan Srikant and Rakesh Agrawal. *Mining sequential patterns : Generalizations and performance improvements*. Springer, 1996.

[119] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM (JACM)*, 44(4) :585–591, 1997.

[120] K. Sudo, S. Sekine, and R. Grishman. Automatic pattern acquisition for japanese information extraction. In *Proceedings of the first international conference on Human language technology research*, pages 1–7. Association for Computational Linguistics, 2001.

[121] William R Swartout, Jonathan Gratch, Randall W Hill Jr, Eduard Hovy, Stacy Marsella, Jeff Rickel, David Traum, et al. Toward virtual humans. *AI Magazine*, 27(2) :96, 2006.

[122] A. Vinciarelli. Speakers role recognition in multiparty audio recordings using social network analysis and duration distribution modeling. *Multimedia, IEEE Transactions on*, 9(6) :1215–1226, 2007.

[123] Wei Wang, Jiong Yang, Richard Muntz, et al. Sting : A statistical information grid approach to spatial data mining. In *VLDB*, volume 97, pages 186–195, 1997.

[124] Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *Acm Sigkdd Explorations Newsletter*, 5(1) :59–68, 2003.

[125] Xifeng Yan and Jiawei Han. gspan : Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[126] Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 344–353. ACM, 2004.

[127] Zhenglu Yang and Masaru Kitsuregawa. Lapin-spam : An improved algorithm for mining sequential pattern. In *Data Engineering Workshops, 2005. 21st International Conference on*, pages 1222–1222. IEEE, 2005.

[128] R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. Unsupervised discovery of scenario-level patterns for information extraction. In *Proceedings of the sixth conference on Applied natural language processing*, pages 282–289. Association for Computational Linguistics, 2000.

[129] Kenichi Yoshida, Hiroshi Motoda, and Nitin Indurkhya. Graph-based induction as a unified learning framework. *Applied Intelligence*, 4(3) :297–316, 1994.

[130] Mohammed J Zaki. Sequence mining in categorical domains : incorporating constraints. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 422–429. ACM, 2000.

[131] Mohammed J Zaki. Spade : An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2) :31–60, 2001.

[132] Mohammed J Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2002.

[133] Minghua Zhang, Ben Kao, Chi-Lap Yip, and David Cheung. A gsp-based efficient algorithm for mining frequent sequences. In *Proc. of IC-AI*, pages 497–503, 2001.

[134] Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. Greedy splitting algorithms for approximating multiway partition problems. *Mathematical Programming*, 102(1) :167–183, 2005.

Titre : Extraction et partitionnement pour la recherche de régularités : application à l'analyse de dialogues.

Mots-clefs : optimisation combinatoire, fouille de données, extraction de régularités, K-partitionnement, approche polyèdrale, analyse de dialogues.