Recherche de motifs fréquents dans une base de cartes combinatoires

Stéphane Gosselin

UNIVERSITÉ CLAUDE BERNARD LYON 1

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de
DOCTEUR de U.C.B.L.
Spécialité : **Informatique**
préparée au **Laboratoire d'InfoRmatique en Images et Système d'information**
dans le cadre de l'École Doctorale **Math-Info**

présentée et soutenue publiquement
par

Stéphane Gosselin

le 24-10-2011

Titre:
Recherche de motifs fréquents dans une base de cartes combinatoires

Directeur de thèse: **Christine Solnon, Professeur, INSA Lyon, LIRIS**
Co-directeur de thèse: **Guillaume Damiand, Chargé de Recherches HDR, CNRS, LIRIS**

Jury

M. Bruno Crémilleux, Professeur, Univ. Caen, GREYC	Rapporteur
M. Pascal Lienhardt, Professeur, Univ. Poitier, XLIM-SIC	Rapporteur
M. Florent Dupont, Professeur, Univ. Lyon 1, LIRIS	Examinateur
M. Jean-Christophe Janodet, Professeur, Univ. Evry, IBISC	Examinateur
Mme. Christine Solnon, Professeur, INSA Lyon, LIRIS	Directeur de thèse
M. Guillaume Damiand, Chargé de Recherches HDR, CNRS, LIRIS	Co-Directeur de thèse

Résumé

Une carte combinatoire est un modèle topologique qui permet de représenter les subdivisions de l'espace en cellules et les relations d'adjacences et d'incidences entre ces cellules en n dimensions. Cette structure de données est de plus en plus utilisée en traitement d'images, mais elle manque encore d'outils pour les analyser. Notre but est de définir de nouveaux outils pour les cartes combinatoires nD. Nous nous intéressons plus particulièrement à l'extraction de sous-cartes fréquentes dans une base de cartes.

Nous proposons deux signatures qui sont également des formes canoniques de cartes combinatoires. Ces signatures ont chacune leurs avantages et leurs inconvénients. La première permet de décider de l'isomorphisme entre deux cartes en temps linéaire, en contrepartie le coût de stockage en mémoire est quadratique en la taille de la carte. La seconde signature a un coût de stockage en mémoire linéaire en la taille de la carte, cependant le temps de calcul de l'isomorphisme est quadratique. Elles sont utilisables à la fois pour des cartes connexes, non connexes, valuées ou non valuées. Ces signatures permettent de représenter une base de cartes combinatoires et de rechercher un élément de manière efficace. De plus, le temps de recherche ne dépend pas du nombre de cartes présent dans la base.

Ensuite, nous formalisons le problème de recherche de sous-cartes fréquentes dans une base de cartes combinatoires nD. Nous implémentons deux algorithmes pour résoudre ce problème. Le premier algorithme extrait les sous-cartes fréquentes par une approche en largeur tandis que le second utilise une approche en profondeur. Nous comparons les performances de ces deux algorithmes sur des bases de cartes synthétiques.

Enfin, nous proposons d'utiliser les motifs fréquents dans une application de classification d'images. Chaque image est décrite par une carte qui est transformée en un vecteur représentant le nombre d'occurrences des motifs fréquents. À partir de ces vecteurs, nous utilisons des techniques classiques de classification définies sur les espaces vectoriels. Nous proposons des expérimentations en classification supervisée et non supervisée sur deux bases d'images.

Abstract

A combinatorial map is a topological model that can represent the subdivisions of space into cells and their adjacency relations in n dimensions. This data structure is increasingly used in image processing, but it still lacks tools for analysis. Our goal is to define new tools for combinatorial maps nD. We are particularly interested in the extraction of submaps in a database of maps.

We define two combinatorial map signatures : the first one has a quadratic space complexity and may be used to decide of isomorphism with a new map in linear time whereas the second one has a linear space complexity and may be used to decide of isomorphism in quadratic time. They can be used for connected maps, non connected maps, labbeled maps or non labelled maps. These signatures can be used to efficiently search for a map in a database. Moreover, the search time does not depend on the number of maps in the database.

Then, we formalize the problem of finding frequent submaps in a database of combinatorial nD maps. We implement two algorithms for solving this problem. The first algorithm extracts the submaps with a breadth-first search approach and the second uses a depth-first search approach. We compare the performance of these two algorithms on synthetic database of maps.

Finally, we propose to use the frequent patterns in an image classification application. Each image is described by a map that is transformed into a vector representing the number of occurrences of frequent patterns. From these vectors, we use standard techniques of classification defined on vector spaces. We propose experiments in supervised and unsupervised classification on two images databases.

Table des matières

Résumé iii
Abstract iv

1 Introduction générale **1**

2 État de l'art **5**
2.1 Extraction de motifs fréquents dans des bases de données transactionnelles 6
2.1.1 Définition du problème 6
2.1.2 Algorithmes de recherche d'ensembles d'attributs fréquents 7
2.2 Extraction de motifs fréquents dans des bases de graphes 9
2.2.1 Définition du problème 9
2.2.2 Algorithmes de recherche de sous-graphes fréquents . . . 11
2.3 Introduction aux cartes combinatoires 14
2.3.1 Cartes combinatoires nD 14
2.3.2 Orbites, Cellules et connexité 17
2.3.3 Morphismes de cartes 19

3 Signatures de cartes combinatoires **25**
3.1 Signatures de cartes connexes 26
3.1.1 Étiquetage de cartes 26
3.1.2 Signature sous forme d'un ensemble de mots (S-signature) 30
3.1.3 Signature sous forme d'un mot (W-signature) 33
3.2 Signatures d'une base de cartes connexes 34
3.2.1 S-signature d'une base de cartes (TS-signature) 35
3.2.2 W-signature d'une base de cartes (TW-signature) 37
3.2.3 Nombre de fils d'un nœud d'un arbre de signatures 39
3.3 Signatures de cartes valuées 42
3.4 Signatures de cartes non connexes 42
3.5 Évaluation 44

4 Extraction de motifs fréquents dans des bases de cartes combinatoires 51
4.1 Définition du problème . . . 52
4.2 Méthode en largeur . . . 54
4.2.1 Calcul des sous-cartes fréquentes composées d'une ou de deux faces . . . 55
4.2.2 Génération des candidats . . . 57
4.2.3 Fusion . . . 59
4.2.4 Comptage de la fréquence . . . 59
4.2.5 Complexités des différentes étapes de l'algorithme *MapA-Priori* . . . 65
4.3 Méthode en profondeur . . . 67
4.3.1 Principe . . . 67
4.3.2 Algorithme . . . 67
4.3.3 Complexité . . . 73
4.3.4 Extension aux cartes nD . . . 74
4.4 Expérimentations . . . 74
4.4.1 Comparaison des performances des algorithmes *mSpan* et *MapApriori*. . . . 75
4.4.2 Comparaison entre les cartes et les graphes (mSpan Vs gSpan). 79

5 Expérimentations 85
5.1 Utilisation de motifs fréquents pour la classification . . . 85
5.2 Données synthétiques . . . 89
5.3 Images . . . 90
5.4 Documents manuscrits . . . 95
5.5 Pavages . . . 97

6 Conclusion générale 99

Bibliographie 107

Liste des Algorithmes

1 $APriori(\mathcal{B}, minSupport)$ 8
2 $VerifierIsomorphisme(C, C')$ 20
3 $ConstruireAppariement(C, C', b_0, b'_0)$ 21
4 $VerifierSousIsomorphisme(C, C')$ 23
5 $BFL(C, b)$ 27
6 $W(C, l)$ 29
7 $SS(C)$ 30
8 $T_{SS}(C)$ 31
9 $WS(C)$ 34
10 $W_v(C, l)$ 42
11 $MapAPriori(BC, \sigma, \varphi)$ 56
12 $genererCandidats(F^{k-1})$ 58
13 $Fusion(f_i^{k-1}, f_j^{k-1}, N^{k-2})$ 59
14 $Compter(\gamma^k, BC)$ 64
15 $mSpan(BC, \sigma, \varphi)$ 68
16 $grow(p, F_1)$ 69

Chapitre 1

Introduction générale

CETTE thèse se place dans le cadre du projet *ANR SATTIC* (Strings And Trees for Thumbnail Images Classification) dont le but est de définir de nouveaux outils pour modéliser des images par un ensemble de données structurées dans l'objectif de faire de la classification d'images. Ces structures peuvent être des chaines [5, 41], des arbres, des graphes [50, 51]... Plus la structure de données est complexe, plus le nombre d'informations représentées est grand, en contrepartie, plus il est difficile de manipuler ces structures. Par exemple, il est plus facile de calculer la distance entre deux chaines qu'entre deux graphes, mais les graphes peuvent contenir plus d'informations que les chaines. Il convient donc de trouver le meilleur compromis entre le pouvoir d'expression de la structure et la complexité d'utilisation.

Dans ce cadre où chaque image est modélisée par un ensemble de données structurées, une méthode de classification d'images se décompose en deux étapes indépendantes : dans un premier temps, il faut une méthode pour transformer une image en données structurées ; puis il faut traiter ces données pour faire de la classification d'images ou de la reconnaissance de formes.

Il est possible de représenter une image soit par des descripteurs locaux, tels que les points saillants, soit en représentant globalement l'image pour une segmentation en régions. Dans cette thèse, nous nous intéressons plus particulièrement à cette seconde approche. Il existe différentes structures de données pour modéliser le partitionnement en régions d'une image. Toutes ces structures sont plus ou moins dérivées du graphe d'adjacence des régions [48]. Cependant, le graphe d'adjacence ne permet pas de représenter toute la topologie entre les régions. Pour répondre à ce manque, les cartes combinatoires $2D$ ont été définies [10, 16, 29, 53], elles ont ensuite été étendues en $3D$ puis en nD [8, 32, 33].

Les cartes combinatoires sont utilisées dans différents domaines comme la modélisation géométrique ou le traitement d'images. Dans ces applications,

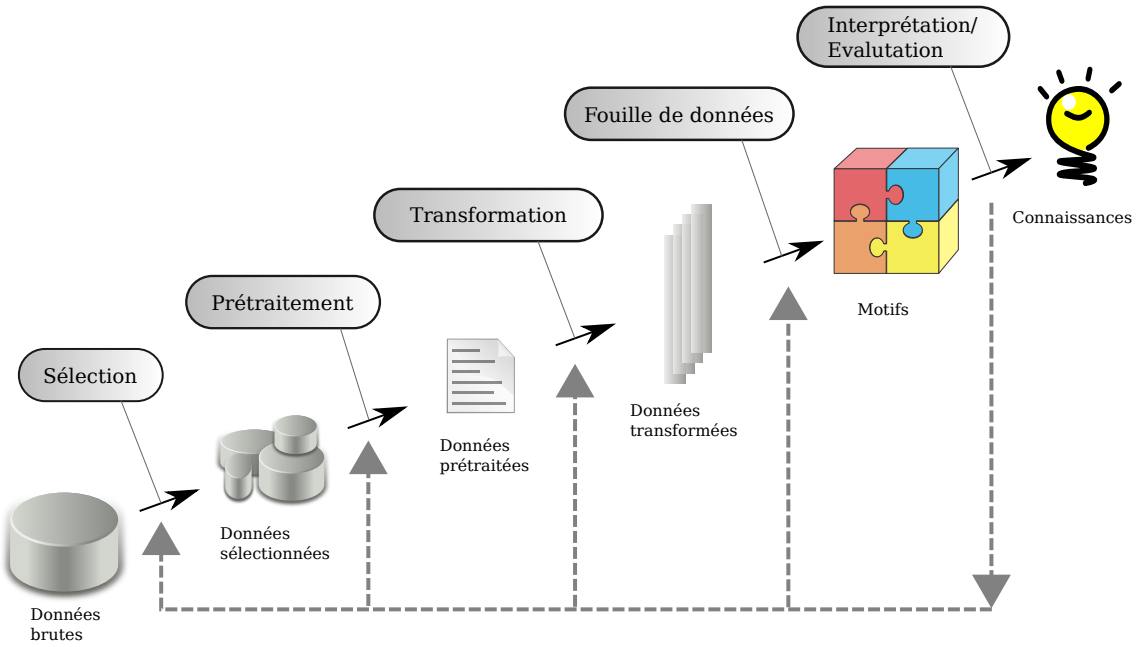


FIGURE 1.1 – Processus d'extraction de connaissances.

différentes opérations existent permettant de retrouver les informations décrites dans les cartes, ou permettant de construire et de modifier les objets manipulés. Mais avant le début de cette thèse, il existait très peu d'outils de comparaison de cartes. Nombre de ces outils dont disposent les graphes peuvent être adaptés de façon plus ou moins complexe sur les cartes $2D$ cependant, le passage en dimension supérieure implique souvent de nouvelles problématiques. Même si l'objectif à terme est la classification d'images, dans un premier temps, notre but est de définir de nouveaux outils sur les cartes combinatoires de la manière le plus générique possible, c.-à-d. en dimension n et sans a priori sur l'application, pour que ceux-ci soient utilisables dans n'importe quel contexte. Dans cette thèse, notre objectif principal est d'extraire des connaissances à partir d'une base de cartes combinatoires.

En 1996, Fayyad [19] décrit le processus d'extraction de connaissances à partir de bases de données comme un processus itératif composé de plusieurs étapes. La figure 1.1 illustre ce processus. Tout d'abord, il faut sélectionner parmi l'ensemble des données à notre disposition le sous-ensemble des données qui peuvent être intéressantes. L'étape de prétraitement des données vise à corriger des données manquantes ou erronées. Ensuite, il faut transformer les données pour qu'elles soient utilisables par l'algorithme de fouille choisi, celui-ci génère un certain nombre de motifs qu'il faut interpréter pour enfin obtenir de nouvelles connaissances sur les données de départ. À la fin, les choix de chaque étape peuvent être remis en cause, il convient alors de réitérer ce processus en faisant des choix différents.

Parmi toutes les étapes du processus, l'étape de fouille de données est la partie la plus complexe du point de vue algorithmique. De nombreuses méthodes existent alliant statistiques, mathématiques et informatique, de la régression

linéaire à l'extraction de motifs fréquents. Toutes ces méthodes ne sont pas applicables ni adaptables au cas des cartes combinatoires. Dans cette thèse, nous définissons le problème de découverte de motifs fréquents dans le cadre d'une base de cartes combinatoires.

Nous avons alors étudié le problème équivalent dans les graphes. Il est ressorti de cette étude qu'il y a plusieurs points clés pour proposer un algorithme efficace. Premièrement, le calcul de la fréquence d'un motif est lié à la notion d'isomorphisme. Le problème principal dans le cas des graphes est la forte complexité du calcul d'isomorphisme de graphes et de sous-graphes, qui sont des problèmes NP-complets. Ce problème ne se pose pas dans le cas des cartes combinatoires, car il existe des algorithmes polynomiaux pour résoudre les problèmes d'isomorphismes de cartes et de sous-cartes. Le second point clé est le choix de la stratégie d'exploration des motifs, les méthodes en largeur sont souvent plus simples à mettre en œuvre, mais les performances sont moins bonnes que pour les méthodes en profondeur. Nous nous intéressons à ces deux types d'approches dans le cas de la recherche de motifs dans une base de cartes combinatoires.

Dans le chapitre 2, nous présentons des travaux existants sur l'extraction de motifs dans des bases de données transactionnelles (section 2.1), puis, dans la section 2.2 nous présentons le problème d'extraction de sous-graphes fréquents ainsi qu'un tour d'horizon des principales méthodes de résolution de ce problème. Enfin, dans la section 2.3 nous définissons les cartes combinatoires nD et les notions d'orbites, de cellules, de connexité, puis nous présentons plusieurs outils existants tels que l'isomorphisme de cartes et de sous-cartes.

Les contributions de cette thèse sont présentées dans les chapitres 3 et 4. Nous définissons deux signatures qui sont aussi des formes canoniques de cartes combinatoires dans la section 3.1. Dans la section 3.2, nous étendons ces signatures pour représenter une base de cartes. Nous présentons des extensions de ces signatures aux cas des cartes non connexes et aux cartes valuées dans les sections 3.3 et 3.4. Puis, nous présentons les résultats d'expériences qui permette de comparer les caractéristiques de ces deux signatures et de ces deux représentations de bases de cartes combinatoires (section 3.5). Dans la section 4.1, nous définissons le problème de recherche de sous-cartes fréquentes. Puis, dans les sections 4.2 et 4.3, nous proposons deux algorithmes qui résolvent ce problème. Le premier algorithme intitulé *MapApriori* utilise une approche en largeur pour découvrir les motifs, tandis que le second, nommé *mSpan* utilise une approche en profondeur.

Dans le chapitre 5, nous donnons un aperçu des applications possibles de sous-cartes fréquentes dans différents domaines tels que la classification d'images, la classification de documents manuscrits ou encore les pavages.

Enfin, dans le chapitre 6, nous présentons le bilan de nos travaux et nous proposons des perspectives pour améliorer et appliquer nos algorithmes.

Chapitre 2

État de l'art

Sommaire

2.1 Extraction de motifs fréquents dans des bases de données transactionnelles 6
2.1.1 Définition du problème 6
2.1.2 Algorithmes de recherche d'ensembles d'attributs fréquents 7
2.2 Extraction de motifs fréquents dans des bases de graphes . . . 9
2.2.1 Définition du problème 9
2.2.2 Algorithmes de recherche de sous-graphes fréquents . . 11
2.3 Introduction aux cartes combinatoires 14
2.3.1 Cartes combinatoires nD 14
2.3.2 Orbites, Cellules et connexité 17
2.3.3 Morphismes de cartes 19

L'OBJECTIF de cette thèse est de définir un certain nombre d'outils sur les cartes combinatoires et plus particulièrement d'extraire des sous-cartes fréquentes dans une base de cartes combinatoires. Les cartes combinatoires sont une structure de données qui peut s'apparenter aux graphes, c'est pourquoi nos algorithmes d'extraction de sous-cartes fréquentes sont adaptés des algorithmes d'extraction de sous-graphes fréquents qui eux-mêmes sont adaptés des algorithmes d'extraction de motifs fréquents dans des bases de données transactionnelles. C'est pourquoi nous présenterons dans un premier temps (en section 2.1) le problème de recherche de motifs fréquents dans une base de données transactionnelles. Puis, dans la section 2.2, nous présenterons les travaux existants sur l'extraction de sous-graphes fréquents. Ces deux sections se décomposent de manière similaire, tout d'abord nous définissons le problème étudié, puis nous présentons les méthodes de résolution selon deux catégories, les méthodes en largeur, qui historiquement ont été les premières à voir le jour, et les méthodes en profondeur, qui sont généralement les plus efficaces.

T \ $\mathcal{I}$	A	B	C	D
t_1	0	0	1	1
t_2	0	1	1	0
t_3	1	1	1	0
t_4	0	1	0	0
t_5	1	1	1	1

TABLE 2.1 – Exemple de base de données transactionnelle.

Enfin, en section 2.3 nous présentons en détail les cartes combinatoires ainsi que des outils existants qui seront utiles pour calculer les sous-cartes fréquentes.

2.1 Extraction de motifs fréquents dans des bases de données transactionnelles

2.1.1 Définition du problème

En 1993, *Agrawal et al.* définissent le problème de recherche de règle d'associations dans une base de données transactionnelle [1]. Il est formalisé de la façon suivante :

Soit $\mathcal{I} = \{i_1, \ldots, i_m\}$ un ensemble de m attributs. Soit T une base de transactions, où chaque transaction $t \in T$ décrit une relation binaire sur $\mathcal{I}$, c.-à-d. , $\forall i \in \mathcal{I}, t[i] = 1$ si l'objet i est présent dans t et $t[i] = 0$ sinon. Un motif M est un ensemble d'attributs de $\mathcal{I}$, c.-à-d. $M \subseteq \mathcal{I}$. M est présent dans une transaction t ssi $\forall i \in M, t[i] = 1$. Le support d'un motif M noté *support(M)* est le nombre de transactions où M est présent. Cette notion de support est aussi appelée fréquence absolue. Notons qu'il existe aussi une notion de fréquence relative qui donne le nombre de transactions où M est présent relativement à la taille de la base de données $|T|$. Pour éviter les confusions, nous notons *support* pour la fréquence absolue et *fréquence* pour la fréquence relative. La fréquence d'un motif M est notée $freq(M) = \frac{support(M)}{|T|}$. Soit un seuil *minSupport* entier (resp. *minFreq* réel compris entre 0 et 1), l'objectif est de trouver tous les motifs qui ont un support (resp. fréquence) supérieur ou égal à *minSupport* (resp. *minFreq*). Un motif M est dit *fréquent* ssi $freq(M) \geq minFreq$ ou $support(M) \geq minSupport$.

Exemple 1. *Le tableau 2.1 montre un exemple de base de données transactionnelle. La base est composée de 5 transactions et de 4 attributs.*

$support(B) = |\{t_2, t_3, t_4, t_5\}| = 4$, $freq(B) = 4/5 = 0,8$. *Pour simplifier la notation, nous notons de manière condensée* ABC *pour le motif* $\{A, B, C\}$.

$support(ABC) = |\{t_3, t_5\}| = 2$, $freq(ABC) = 2/5 = 0,4$.

La fréquence et le support ont pour caractéristique de décroître en fonction de la relation d'inclusion ensembliste $\subseteq$, c.-à-d. pour deux motifs $M_1 \subseteq M_2 \subseteq$

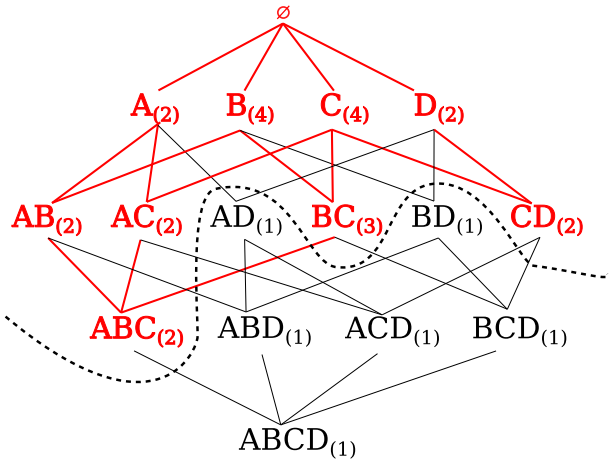


FIGURE 2.1 – Treillis des motifs.

$\mathcal{I}$, $freq(M_1) \geq freq(M_2)$. L'ensemble des motifs possibles pour un ensemble d'attributs $\mathcal{I}$ peut être représenté sous forme d'un treillis. La figure 2.1 illustre l'ensemble de tous les motifs possibles pour l'exemple du tableau 2.1. Les motifs sont ordonnés de haut en bas selon la propriété d'inclusion, tout en haut l'ensemble vide et tout en bas l'ensemble $\mathcal{I}$. Nous notons k-*motif*, un motif du niveau k, c.-à-d. un motif qui contient k attributs. Le support de chaque motif est donné entre parenthèses. La fréquence est une fonction anti-monotone [1], ce qui implique pour la notion de motif fréquent que si un motif M_1 n'est pas fréquent alors pour tout motif M tel que $M_1 \subseteq M \subseteq \mathcal{I}$, M n'est pas fréquent. Nous pouvons vérifier la décroissance du support le long d'un chemin, par exemple $support(\emptyset) = 5$, $support(B) = 4$, $support(BC) = 3$, $support(ABC) = 2$, $support(ABCD) = 1$. Pour cet exemple, si le seuil *minSupport*$= 2$, les motifs fréquents sont en rouge et les motifs non fréquents en noir, nous pouvons remarquer une frontière entre les motifs fréquents et non fréquents dans le treillis. Cette frontière existe toujours pour des contraintes anti-monotones, telle que la contrainte de fréquence.

Notons que dans le pire des cas, le problème de recherche de motifs fréquents est exponentiel par rapport au nombre d'attributs. En effet, s'il y a n attributs, il y a 2^n motifs possibles et dans le pire des cas ils sont tous fréquents. Le nombre de motifs possibles ne dépend pas du nombre de transactions.

2.1.2 Algorithmes de recherche d'ensembles d'attributs fréquents

De nombreux algorithmes ont été développés pour résoudre le problème de recherche d'ensembles d'attributs fréquents [2, 7, 17, 25, 26, 35, 36, 38, 42, 44, 45, 52], nous en détaillerons seulement deux, qui nous semblent les plus importants. Les algorithmes de recherche de motifs fréquents peuvent se séparer en deux catégories, les méthodes qui calculent les motifs du treillis en largeur d'abord et celles qui les calculent en profondeur. Historiquement, les premières méthodes qui sont apparues sont les méthodes en largeur.

1. Une fonction $f : A \rightarrow B$ est anti-monotone si $\forall a \in A, b \in B, a \leq b \implies f(a) \geq f(b)$

Algorithme *APriori* [2] Le premier algorithme à résoudre le problème de recherche de motifs fréquents est l'algorithme *APriori* [2]. Le principe de l'algorithme *APriori* est de calculer les motifs du treillis en largeur d'abord. Les motifs sont calculés niveau par niveau.

Algorithme 1 : $APriori(\mathcal{B}, minSupport)$

Entrées : une base de données transactionnelles $\mathcal{B}$ et un entier $minSupport$.
Sortie : l'ensemble de tous les motifs fréquents.

```
1  L_1 ← attributs fréquents
2  k ← 2
3  tant que L_{k-1} ≠ ∅ faire
4      C_k ← APRIORI-GEN(L_{k-1})
5      pour chaque transaction t ∈ B faire
6          C_t ← subset(C_k, t)
7          pour chaque candidat c ∈ C_t faire
8              c.support ← c.support + 1
9      L_k ← {c ∈ C_k | c.support ≥ minSupport}
10 retourner {L_1, L_2, ..., L_k}
```

La fonction APRIORI-GEN prend pour paramètre L_{k-1}, l'ensemble des motifs fréquents composés de $k-1$ attributs et retourne un ensemble de motifs candidats composés de k éléments. Un motif candidat est un motif dont nous ignorons s'il est fréquent ou non. La génération d'un candidat s'effectue en deux étapes : la première étape est une jointure entre deux motifs de taille $k-1$ qui ont $k-2$ attributs en commun. Par exemple, la jointure des motifs ABC et ABD donne $ABCD$, par contre, la jointure de ABC et CDE ne donne rien, car il n'y a pas $k-2$ attributs en commun. La seconde étape est une étape d'élagage. Pour chaque k-motif M_k généré par l'étape de jointure, il faut tester si tous les $(k-1)$-motifs $M_{k-1} \subseteq M_k$ sont fréquents, c.-à-d. s'ils sont tous présents dans L_{k-1}. Par exemple, prenons $L_3 = \{ABC, ABD, ACD, ACE, BCD\}$, après l'étape de jointure, $C_4 = \{ABCD, ACDE\}$. L'étape d'élagage supprime le motif $ACDE$, car son sous-motif ADE n'est pas présent dans L_3, à la fin de l'exécution de APRIORI-GEN(L_3), $C_4 = \{ABCD\}$.

La fonction $subset$ prend pour paramètres l'ensemble des candidats C_k et une transaction t et elle retourne le sous-ensemble des motifs $c \in C_k$ présents dans t. Cela permet de calculer la fréquence exacte de chaque candidat (lignes 7-8), et ensuite, seul les motifs fréquents sont conservés (ligne 9) pour le prochain pas de la boucle principale.

Algorithme *Eclat* [59, 60] Le premier algorithme proposant une approche en profondeur est l'algorithme *Eclat*. Cette méthode consiste à énumérer les motifs fréquents dans un ordre prédéfini. Par exemple, les motifs $ADEB$ et $ABDE$ sont les mêmes motifs et il existe plusieurs façon de générer ce motif. En choisissant

d'ordonner les motifs par ordre lexicographique, le motif $ABDE$ ne sera généré qu'une seule fois, par son parent ABD, et il sera généré seulement si ABD est fréquent. Ce principe simple améliore grandement les performances de l'algorithme, malheureusement il est difficilement adaptable dans le cas des graphes ou des cartes combinatoires.

L'algorithme *Eclat* a plusieurs avantages, il utilise des classes d'équivalences qui sont indépendantes, il peut donc être parallélisé et après le partitionnement aucune communication n'est nécessaire entre les processus. Les classes d'équivalences sont basées sur la notion de préfixe commun, par exemple, les motifs ABC, ABD et ABE font partie de la même classe d'équivalence, car ils ont le motif AB pour préfixe commun. Les 4-motifs qui peuvent être générés par cette classe d'équivalence sont $ABCD$, $ABCE$ et $ABDE$. Notons que ces motifs ne peuvent pas être générés par une autre classe d'équivalence.

2.2 Extraction de motifs fréquents dans des bases de graphes

2.2.1 Définition du problème

Le problème existant le plus proche de celui que nous traitons dans cette thèse est le problème d'extraction de sous-graphes fréquents. Un graphe est une structure de données très utilisés en mathématique et en informatique qui permet de représenter des relations (arêtes) entre des objets (sommets). De plus, des étiquettes sur ces sommets et ces arêtes permettent de décrire ces objets et ces relations, c'est pourquoi les graphes sont utilisée dans des domaines très variés. La définition 1 donne la définition formelle de graphe étiqueté et la figure 2.2 donne trois exemples de graphes étiquetés. Par la suite, nous noterons graphe pour un graphe étiqueté.

Définition 1 (Graphe étiqueté)**.**
Un graphe est un tuple $G = (S, A, E, e)$ *avec :*
- S *est un ensemble fini de sommets ;*
- $A \subseteq S \times S$ *est un ensemble d'arêtes ;*
- E *est l'ensemble des différentes étiquettes possibles ;*
- $e : S \cup A \rightarrow E$, e *est une fonction qui associe une étiquette à chaque sommet et chaque arête.*

Dans le problème de recherche de motifs fréquents dans une base de graphe, les motifs considérés sont des sous-graphes connexes [2]. Il existe deux types de sous-graphes :
- sous-graphe induit : un sous-ensemble des sommets qui conserve toutes les arêtes entre les sommets sélectionnés ;
- sous-graphe partiel : un sous-ensemble des sommets et un sous-ensemble des arêtes entre les sommets sélectionnés.

2. seul les motifs connexes sont considérés pour réduire le nombre de motifs possibles, cependant, il est facile de retrouver tous les motifs non connexes par un post-traitement.

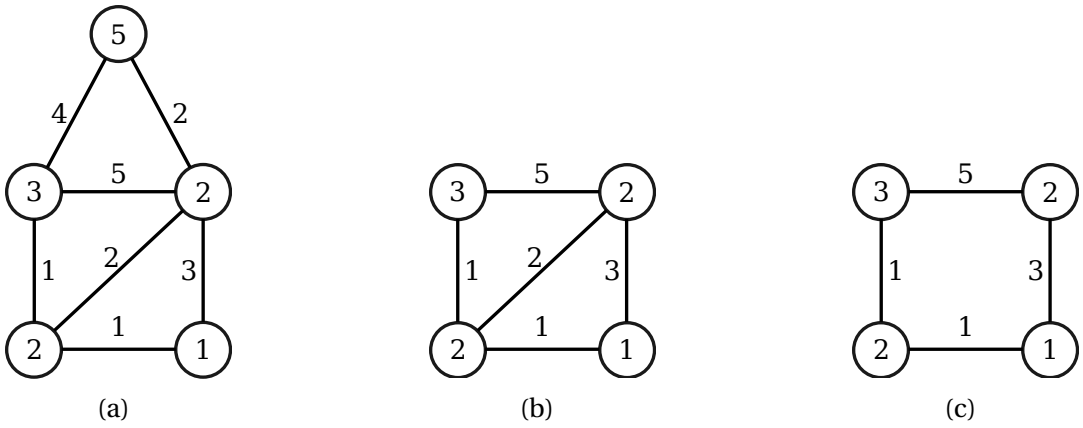


FIGURE 2.2 – Exemple de trois graphes étiquetés. Le graphe (b) est un sous-graphe induit du graphe (a), et le graphe (c) est un sous-graphe partiel du graphe (a).

Dans les deux cas, les étiquettes des sommets et des arêtes sélectionnées sont conservées. La figure 2.2 donne un exemple d'un sous-graphe induit et d'un sous-graphe partiel.

La définition du problème de sous-graphes fréquents est donnée dans la définition 2.

Définition 2 (Extraction de sous-graphes fréquents)**.**
Étant donné une base de graphes $\mathcal{B}$ et un seuil de fréquence minimale σ, l'objectif est de trouver tous les graphes connexes qui sont sous-graphes d'au moins $\sigma \cdot |\mathcal{B}|$ graphes de $\mathcal{B}$.

Le problème d'extraction de sous-graphes fréquents est très proche dans sa formulation du problème d'extraction d'ensembles d'attributs fréquents, cependant plusieurs complications apparaissent. Premièrement, savoir si deux motifs sont identiques est un aspect qui est trivial dans le cas des ensembles d'attributs, mais qui devient complexe avec les graphes. En effet, dans le cas des attributs, pour savoir si ACB et BCA sont identiques, il suffit, de fixer un ordre sur les attributs et de trier les motifs dans l'ordre choisit. Ici, si l'on choisit l'ordre lexicographique, les deux motifs s'écrivent sous la forme ABC et la comparaison est faite en temps linéaire. Dans le cas des graphes, il faut soit calculer une forme canonique [4] des deux motifs et vérifier si elles sont identiques, soit tester s'il existe un isomorphisme entre les deux motifs. Ces deux problèmes sont équivalents et sont NP-difficile. La définition 3 donne la définition d'isomorphisme de graphes. La fonction d'isomorphisme apparie chaque sommet de S à un sommet de S' avec pour contraintes que les étiquettes de deux sommets appariés soient identiques et que s'il existe une arête entre deux sommets, il faut qu'il y ai une arête entre les deux sommets auxquels ils sont appariés et que ces arêtes aient la même étiquette.

Définition 3 (Isomorphisme de graphes)**.**
Soient deux graphes $G = (S, A, E, e)$ et $G' = (S', A', E', e')$. Un isomorphisme

entre G et G' est une fonction bijective $f : S \to S'$ telle que :
- $\forall u \in S, l(u) = l'(f(u))$ *;*
- $\forall (u,v) \in E, (f(u), f(v)) \in A'$ *et* $l(u,v) = l'(f(u), f(v))$*.*

La seconde complication vient cette fois de la difficulté à calculer l'inclusion d'un motif dans un autre. Dans le cas des ensembles d'attributs, cela se fait de façon linéaire, tandis que dans le cas des graphes, cela revient à calculer un isomorphisme de sous-graphes. Ce problème est NP-complet et les meilleurs algorithmes ont une complexité exponentielle. Pourtant ce problème est un point clé de la fouille de graphes, car c'est la recherche d'isomorphisme de sous-graphe (motif) dans un graphe (transaction) qui permet de calculer la fréquence de ce motif.

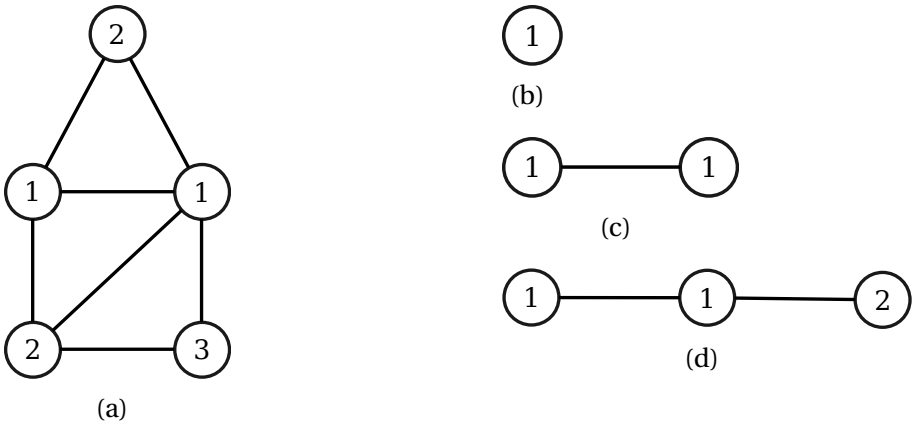
Un troisième problème est la notion de fréquence d'un motif. En effet, contrairement aux ensembles d'attributs, un motif peut être présent plusieurs fois dans une même transaction dans le cas des graphes. Il existe plusieurs possibilités pour calculer la fréquence d'un motif, le choix majoritairement retenu est de compter le nombre de transactions où le motif est présent, qu'il soit présent une fois ou plus. Ce choix vient du fait que la fonction de fréquence n'est pas anti-monotone dans le cas où toutes les occurrences d'un motif sont comptées. De plus, un motif peut posséder des automorphismes, c.-à-d. un isomorphisme avec lui même. Une autre fonction de fréquence est donc définie en comptant toutes les occurrences d'un motif et en multipliant par le nombre d'automorphismes qu'il contient, cependant cette fréquence n'a toujours pas la propriété d'anti-monotonie. La figure 2.3 donne un exemple des différentes fréquences possibles. Le motif (b) est un sous-motif de (c) qui est lui même un sous-motif de (d), nous remarquons que pour la fréquence appelée occ qui compte le nombre d'occurrences d'un motif, $occ(b) > occ(c)$ mais que $occ(b) < occ(d)$, le même problème se pose avec $auto$ qui prend en compte le nombre d'automorphismes d'un motif.

Ces problèmes qui apparaissent dans le cadre de la fouille de graphes sont aussi des points clés dans la fouille de cartes combinatoires.

2.2.2 Algorithmes de recherche de sous-graphes fréquents

Il existe un grand nombre d'algorithmes qui calculent les sous-graphes fréquents dans une base de graphes. Certains résolvent ce problème pour des types de graphes particuliers comme des chaines, ou des arbres. Ces sous problèmes sont plus simples, car il existe des algorithmes plus efficaces pour résoudre l'isomorphisme et le sous-isomorphisme de ces graphes particuliers. Les méthodes que nous allons présenter dans la suite de cette section sont des méthodes qui résolvent le cas général de graphes étiquetés quelconques. Nous séparons ces méthodes en deux types, les méthodes en largeur et les méthodes en profondeur.

Méthodes en largeur Les méthodes en largeur sont basées sur l'algorithme *Apriori*, elles calculent les motifs niveau par niveau, en combinant les motifs du



	(b)	(c)	(d)
freq	1	1	1
occ	2	1	4
auto	2	2	4

FIGURE 2.3 – Exemple des différentes fréquences pour un motif dans un graphe. Le tableau donne les valeurs des différentes fréquences pour les motifs (b), (c) et (d) dans le graphe (a). La ligne *freq* correspond à la fréquence la plus utilisée qui compte 1 si le motif est présent dans un graphe et 0 sinon. La ligne *occ* correspond au nombre d'occurrences du motif dans le graphe et la ligne *auto* correspond au nombre d'occurrences multiplié par le nombre d'automorphismes du motif.

niveau $k-1$ pour calculer les motifs du niveau k. Dans le cas des graphes, un motif de niveau k est un graphe de k arêtes (ou de k sommets pour l'algorithme AGM).

AGM [28] **AGM** est le premier algorithme à répondre à ce problème et il a la particularité d'être le seul à utiliser la notion de sous-graphe induit. Cet algorithme consiste en une adaptation de l'algorithme *Apriori*, les graphes sont représentés par leur matrice d'adjacence et l'algorithme propose une fonction de jointure qui, à partir de deux matrices de niveau $k-1$, produit une matrice de niveau k. Ensuite, il vérifie que tous les sous-motifs de niveau $k-1$ sont bien fréquents et si c'est le cas, il calcule la fréquence du motif candidat. Les deux principaux problèmes d'**AGM** sont l'espace mémoire utilisé qui augmente de manière exponentielle avec le niveau k et la fonction de jointure qui construit un grand nombre de fois des motifs identiques.

FSG [30] L'algorithme **FSG** est aussi une approche en largeur, réalisé après **AGM**. Il corrige donc quelques défauts de la première méthode. Ici, les motifs considérés sont des sous-graphes partiels. Premièrement, deux motifs isomorphes seront forcément considérés comme un même motif. Ensuite, l'opération de jointure permet de diminuer le nombre de candidats qui sont générés de façon redondante, cependant il ne les supprime pas tous. Enfin, il améliore la

phase de calcul de la fréquence en associant à chaque motif la liste des transactions qui contiennent ce motif. Lorsqu'un k-motif candidat est généré, l'algorithme vérifie que tous ses sous motifs de niveau $k-1$ sont fréquents et si c'est le cas, il fait l'intersection des listes d'occurrences de ces (k-1)-motifs. Soit l'intersection des transactions qui contiennent ses sous-motifs ne permet pas de dépasser le seuil de fréquence fixé, soit il faut vérifier l'isomorphisme de ce motif pour les transactions en question. Il y a donc un niveau de plus d'élagage par rapport à **AGM**, ce qui réduit le nombre de calculs d'isomorphisme de sous-graphe qui est très coûteux en temps.

Méthodes en profondeur Comme pour l'extraction d'ensembles d'attributs fréquents, des méthodes explorant l'ensemble des sous-graphes fréquents en profondeur ont vu le jour.

gSpan [55] Partant du constat que la génération des candidats est beaucoup plus compliqué que dans le cas des attributs fréquents et que le calcul d'isomorphisme de sous-graphe est très coûteux en temps, l'algorithme **gSpan** propose une approche totalement différente des deux précédentes. En effet, il utilise une approche en profondeur et utilise une représentation canonique de graphe appelé *DFS code*. Ce code consiste à faire un parcours en profondeur d'un graphe, en codant dans une chaine les arêtes et les sommets ainsi que leurs étiquettes dans l'ordre dans lequel ils sont rencontrés. Un même graphe peut produire différents *DFS code*, mais en fixant un ordre sur les étiquettes il est possible de fixer un ordre sur les codes, et donc il existe un code minimal qui correspond à une forme canonique du graphe. De plus, la construction de ce code est incrémentale. Lorsqu'une arête est ajoutée à un motif, le *DFS code* est mis à jour. Soit ce code est minimal et donc, le motif candidat est pris en compte, soit ce code n'est pas minimal et alors le motif est ignoré. Les auteurs ont montré que cette contrainte sur l'ordre de génération des candidats permettait de construire tous les motifs fréquents.

Grâce à cela, il n'y a pas besoin de stocker en mémoire une liste de candidats, l'algorithme **gSpan** utilise donc très peut de mémoire vive et peut donc extraire un nombre très important de motifs.

FFSM [27] **FFSM** utilise les mêmes idées que l'algorithme **gSpan**, à savoir utiliser une représentation canonique des graphes et explorer les motifs par un parcours en profondeur. La principale différence avec **gSpan** vient de la génération des motifs. **FFSM** utilise une approche hybride entre l'extension utilisée par **gSpan** et la jointure de deux motifs utilisée dans **FSG**.

Gaston [43] L'algorithme **Gaston** est une des principales références dans ce domaine avec **gSpan** et **FFSM**. En partant du constat qu'un certain nombre de motifs extraits par les précédents algorithmes sont des chaines ou des arbres, il calcule dans un premier temps les chaines maximales, puis les arbres maximaux et enfin les graphes (qui ne sont ni des chaines, ni des arbres) fréquents.

Cela permet de limiter l'utilisation de l'isomorphisme de graphes qui est très coûteux aux seuls cas où ce ne sont ni des arbres ni des chaines et d'utiliser des méthodes qui ont une complexité beaucoup plus faible dans les autres cas.

Comparaison de ces méthodes (*Wörlein et al.* 2005 [54]) Dans cet article, une comparaison indépendante a été réalisée en codant les 4 algorithmes (gSpan, FFSM, Gaston et MoFa [6] qui est un algorithme dédié à la recherche de motifs fréquents dans des graphes moléculaires), en utilisant les mêmes structures de données. Des tests ont été réalisés sur des bases de données synthétiques et des bases de données moléculaires. Il ressort de ces expérimentations que les trois algorithmes (gSpan, FFSM et Gaston) sont les plus compétitifs et qu'ils ont des performances très proches.

2.3 Introduction aux cartes combinatoires

Dans la section précédente, nous avons étudié différentes méthodes de fouilles de graphes que nous adapterons au chapitre 4 pour la structure de données que nous utilisons : les *cartes combinatoires*. Dans cette section, nous donnons la définition des cartes combinatoires et nous décrivons plusieurs notions qui nous seront utiles pour faire de la fouille de cartes, notamment les notions de cellules, de sous-cartes et d'isomorphismes.

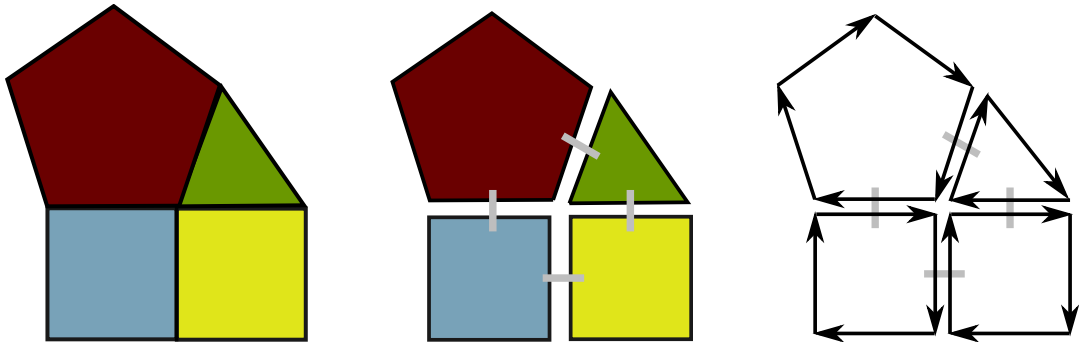
2.3.1 Cartes combinatoires nD

Une carte combinatoire est un modèle topologique qui permet de représenter les subdivisions de l'espace et les relations d'adjacences et d'incidences entre elles. Ce modèle a été défini dans un premier temps en 2D [10,16,29,53] pour représenter les plongements dans un plan de graphes planaires et a ensuite été étendu pour représenter des objets nD orientables et sans bords [8, 32, 33]. Enfin, les cartes combinatoires avec bords ont été définies pour représenter des objets nD orientables et ouverts [46]. Toutes les définitions sur les cartes combinatoires que nous présentons dans cette section peuvent être retrouvées dans [13].

Une carte combinatoire de dimension n est définie par un ensemble de brins et un ensemble de fonctions β_i (avec $1 \leq i \leq n$) où chaque β_i est une relation d'adjacence entre les cellules de dimension i. Par exemple, β_1 est une permutation qui relie deux arêtes (cellules $1D$). β_2 est une involution, qui représente la relation d'adjacence entre les faces (cellules $2D$). β_3 est une involution, qui représente la relation d'adjacence entre les volumes (cellules $3D$) et ainsi de suite pour les n dimensions. Deux brins en relation par une fonction β_i sont dits i-cousus.

La figure 2.4 montre comment un objet $2D$ peut être représenté par une carte combinatoire $2D$.

Avant de définir les cartes combinatoires, nous devons introduire quelques définitions mathématiques utiles pour la suite :



(a) (b) (c)

FIGURE 2.4 – Étapes de transformation d'un objet $2D$ en une carte combinatoire $2D$. (a) L'objet de départ. (b) L'objet est décomposé en faces. (c) Chaque face est décomposée en brins, chaque brin est 1-cousu au brin suivant dans la face (dans le sens des flèches). Notons que, là où il y avait une arête entre deux faces dans l'objet de départ, il y a 2 brins dans la carte combinatoire et ces deux brins sont 2-cousus.

La définition de permutation partielle ainsi que son inverse est donnée dans la définition 4.

Définition 4 (Permutation partielle)**.**

Soit E un ensemble. f une fonction de $E \cup \{\epsilon\} \to E \cup \{\epsilon\}$ est une permutation partielle *sur E si :*

1. $f(\epsilon) = \epsilon$;
2. $\forall e \in E, \forall e' \in E, f(e) = f(e') \neq \epsilon \Rightarrow e = e'$.

L'inverse f^{-1} de cette permutation partielle est également une permutation partielle définie par :

1. $f^{-1}(\epsilon) = \epsilon$;
2. $\forall e \in E$, *si* $\exists e' \in E,\ f(e') = e$, *alors* $f^{-1}(e) = e'$, *sinon* $f^{-1}(e) = \epsilon$.

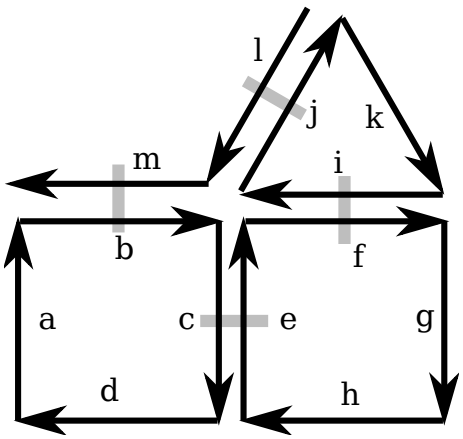
L a définition de permutation partielle ainsi que son inverse est donnée dans la définition 5.

Définition 5 (Involution partielle)**.**

Soit E un ensemble, et f une permutation partielle sur E. f est une involution partielle *si* $\forall e \in E, f(e) \neq \epsilon \Rightarrow f(f(e)) = e$.

La définition 6 donne la définition formelle d'une carte combinatoire nD ouverte.

Définition 6 (Carte combinatoire ouverte)**.**
Une carte combinatoire nD (ou n-carte) ouverte est un $(n+1)$-uplet $C = (B, \beta_1, \ldots, \beta_n)$ tel que



	a	b	c	d	e	f	g	h	i	j	k	l	m
β_0	d	a	b	c	h	e	f	g	k	i	j	ϵ	l
β_1	b	c	d	a	f	g	h	e	j	k	i	m	ϵ
β_2	ϵ	m	e	ϵ	c	i	ϵ	ϵ	f	l	ϵ	j	b

FIGURE 2.5 – Exemple de carte combinatoire $2D$. Les brins sont représentés par des flèches noires. Deux brins 1-cousus se suivent, et deux brins 2-cousus sont joints par un petit segment gris et sont orientés dans des directions opposées.

1. B *est un ensemble fini de brins ;*
2. β_1 *est une* permutation *partielle sur* B *;*
3. $\forall\, 2 \leq i \leq n$, β_i *est une* involution *partielle sur* B *;*
4. $\forall\, 1 \leq i \leq n-2, \forall\, i+2 \leq j \leq n$, $\beta_i \circ \beta_j$ *est une* involution *partielle sur* B.

La fonction β_1 permet à partir d'un brin, de parcourir l'ensemble des brins d'une face dans un ordre donné, la fonction β_1^{-1} correspond au sens inverse.

Pour simplifier les notations, nous noterons par la suite $\beta_0 = \beta_1^{-1}$.

Exemple 2. *La figure 2.5 montre un exemple de* 2*-carte ouverte. Cette carte est composée de 13 brins représentés par des flèches noires. La fonction* β_1 *est donnée par le sens de la flèche. Par exemple* $\beta_1(a) = b$, $\beta_1(k) = i$ *et* $\beta_1(m) = \epsilon$. *La fonction* β_2 *est représentée par un petit segment gris. Par exemple* $\beta_2(b) = m$ *et* $\beta_2(m) = b$. *En cas d'absence de segment gris sur un brin, cela signifie que le brin en question est 2-libre (*$\beta_2(a) = \epsilon$*). Le tableau de la figure 2.5 donne l'intégralité des fonctions* β_0, β_1 *et* β_2.

Dans le cadre des cartes ouvertes, des brins peuvent ne pas être cousus à d'autres brins pour une dimension i, dans ce cas ils sont i-libres. Nous notons ϵ l'élément auquel tous les brins i-libres sont i-cousus et nous posons $\forall i,\ 1 \leq i \leq n$, $\beta_i(\epsilon) = \epsilon$.

Définition 7 (Brin libre)**.**

Soit une n*-carte* $C = (B, \beta_1, \ldots, \beta_n), i \in [0; n] et b \in B$. b *est* i-libre*, si* $\beta_i(b) = \epsilon$.

2.3.2 Orbites, Cellules et connexité

Orbite : Une orbite est l'ensemble des brins rencontrés en partant d'un brin et en utilisant une combinaison de fonctions β_i et leurs inverses.

Nous notons $\langle \beta_{i_1}, \ldots, \beta_{i_k} \rangle(b)$ l'orbite contenant les fonctions $\{\beta_{i_1}, \ldots, \beta_{i_k}\}$. Nous notons également $\langle \widehat{\beta_{i_1}, \ldots, \beta_{i_k}} \rangle(b)$ l'orbite contenant toutes les fonctions $\beta_i \in \{\beta_1, \ldots, \beta_n\}$ sauf $\{\beta_{i_1}, \ldots, \beta_{i_k}\}$, c.-à-d. $\langle \{\beta_1, \ldots, \beta_n\} \setminus \{\beta_{i_1}, \ldots, \beta_{i_k}\} \rangle(b)$.

Cellule : Dans les cartes combinatoires, les sommets (0-cellules), les arêtes (1-cellules), les faces (2-cellules), les volumes (3-cellules), ainsi que les cellules de dimensions supérieures sont représentés de façon implicite comme des ensembles de brins obtenus à l'aide d'orbites spécifiques.

Définition 8 (i-cellule)**.**

Soit $C = (B, \beta_1, \ldots, \beta_n)$ une n-carte, (avec $n > 1$), $b \in B$, et $i \in \{0, \ldots, n\}$. La i-cellule incidente à b, notée $c_i(b)$, est :

- *Si $i = 0$: $\langle \{\beta_i \circ \beta_j | 1 \leq i < j \leq n\} \rangle(b)$.*
- *Sinon : $\langle \widehat{\beta_i} \rangle(b)$.*

Exemple 3. *La figure 2.6 montre les 4 types de cellules (sommet, arête, face, volume) d'un brin dans une 3-carte. Le sommet est obtenu par l'orbite $\langle \beta_1 \circ \beta_2, \beta_1 \circ \beta_3, \beta_2 \circ \beta_3 \rangle$, l'arête par l'orbite $\langle \widehat{\beta_1} \rangle$, la face est obtenue grâce à l'orbite $\langle \widehat{\beta_2} \rangle$ et le volume grâce à l'orbite $\langle \widehat{\beta_3} \rangle$.*

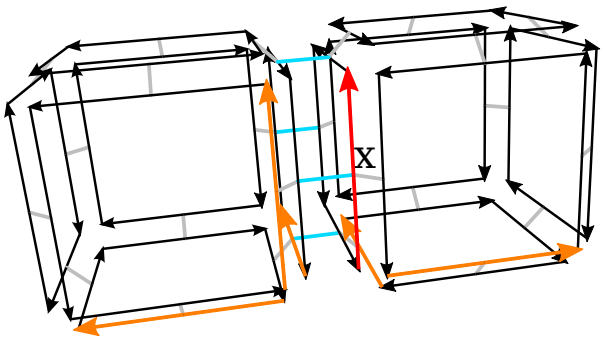
Comme la fonction β_1 est une permutation et que toutes les autres fonctions β_i sont des involutions, il y a deux cas différents pour calculer les cellules. Le premier cas permet de définir les 0-cellules (c.-à-d. les sommets), et l'autre cas est commun à toutes les autres cellules. La i-cellule d'un brin b est l'ensemble des brins qu'il est possible d'atteindre par un parcours d'origine b en utilisant toutes les fonctions ainsi que leurs inverses à l'exception de β_i. En effet, utiliser β_i fait passer à la i-cellule adjacente, donc en enlevant β_i de l'orbite nous assurons de rester sur la même i-cellule tout au long du parcours. Il y a un cas particulier pour les 0-cellules, car il faut parcourir uniquement un brin sur deux, afin de n'atteindre que les brins dont l'origine est le sommet incident à b.

Notons que chaque ensemble de i-cellules est une partition de l'ensemble des brins de la carte. Chaque brin appartient donc exactement à une i-cellule, $\forall i \in \{0, \ldots, n\}$.

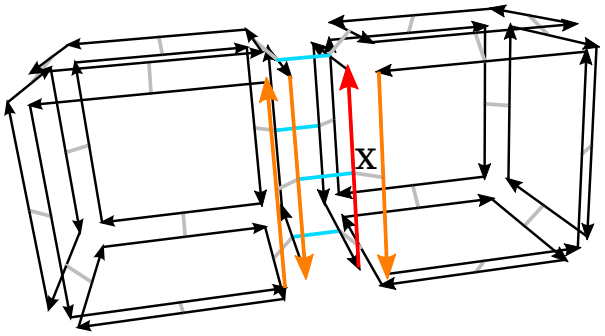
Sur l'exemple de la figure 2.5, le sommet incident au brin c est l'ensemble des brins de l'orbite $\langle \beta_1 \circ \beta_2 \rangle(c) = \{c, f, j, m\}$. L'arête incidente au brin c est l'ensemble des brins de l'orbite $\langle \beta_2 \rangle(c) = \{c, e\}$. Enfin, la face incidente au brin c est l'ensemble des brins de l'orbite $\langle \beta_1 \rangle(c) = \{a, b, c, d\}$.

Connexité Une carte est connexe si tous les brins peuvent être atteints à partir de n'importe quel brin de la carte en utilisant tous les β_i possibles.

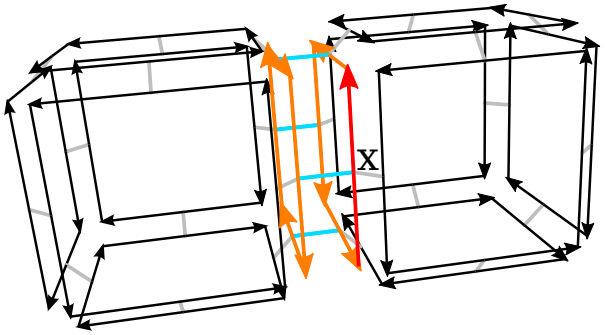
La définition 9 utilise la notion d'orbite pour définir la connexité d'une carte.



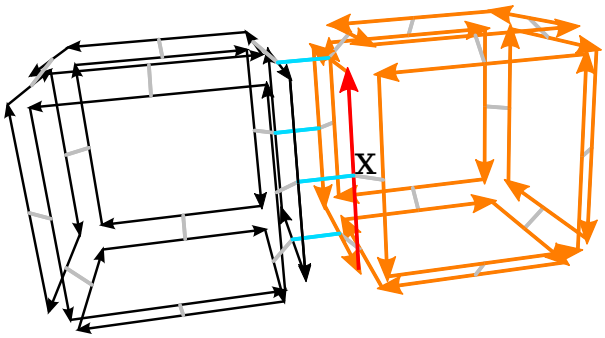
(a)



(b)

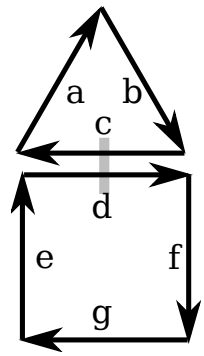


(c)

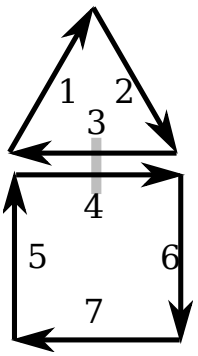


(d)

FIGURE 2.6 – Exemple de cellules. Les brins en orange correspondent aux cellules pour le brin en rouge (brin x). La figure (a) correspond au sommet, la figure (b) correspond à l'arête, la figure (c) correspond à la face et la (d) au volume.



$C = (B, \beta_1, \beta_2)$



$C' = (B', \beta'_1, \beta'_2)$

	a	b	c	d	e	f	g
$f_{(a,1)}$	1	2	3	4	5	6	7
$f_{(a,5)}$	5	4	6	ϵ	ϵ	ϵ	ϵ

FIGURE 2.7 – Exemple de deux cartes isomorphes. Le tableau donne les fonctions d'appariement calculées par l'algorithme 3 pour C, C', a et 1 pour la première ligne et avec les paramètres C, C', a et 5 pour la seconde. La première ligne correspond à la fonction d'isomorphisme entre ces deux cartes.

Définition 9 (Carte connexe)**.**

Soit une carte $C = (B, \beta_1, \ldots, \beta_n)$*. La carte* C *est connexe si*
$$\forall b \in B, \langle \beta_1, \ldots, \beta_n \rangle(b) = B.$$

Notons que si pour un brin $b \in B$ nous avons $\langle \beta_1, \ldots, \beta_n \rangle(b) = B$, alors cette propriété est vrai pour tous les brins de B.

2.3.3 Morphismes de cartes

Dans cette partie nous introduisons la notion de sous-carte et nous présentons différents morphismes de cartes : l'isomorphisme de cartes et de sous-cartes ainsi que l'automorphisme.

Isomorphisme : Deux cartes sont isomorphes si elles sont identiques à un renommage près de leurs brins, c.-à-d. , si elles ont le même nombre de brins et les mêmes fonctions β_i.

Définition 10 (Isomorphisme de cartes [14, 34])**.**
Deux cartes $C = (B, \beta_1, \ldots, \beta_n)$ *et* $C' = (B', \beta'_1, \ldots, \beta'_n)$ *sont isomorphes s'il existe une fonction bijective* $f : B \to B'$*, appelée* fonction d'isomorphisme*, telle que* $f(\epsilon) = \epsilon$ *et* $\forall b \in B, \forall i, 1 \leq i \leq n$*,* $f(\beta_i(b)) = \beta'_i(f(b))$*.*

Exemple 4. *La figure 2.7 montre un exemple de deux cartes isomorphes, la fonction d'isomorphisme est* $f : B \to B'$*,* $f(a) = 1, f(b) = 2, \ldots, f(g) = 7$*.*

Pour calculer cette fonction d'isomorphisme, un algorithme a été proposé dans [14]. Le principe de cet algorithme est de construire une fonction d'appariement en parcourant en parallèle les deux cartes.

Parcours d'une carte Soit $C = (B, \beta_1, \ldots, \beta_n)$ une n-carte connexe. Il est possible de parcourir tous les brins de C en partant d'un brin $b \in B$, en explorant tous les brins cousus à b puis les brins cousus aux voisins de b et ainsi de suite jusqu'à avoir exploré tous les brins de la carte. Pour rendre ce parcours unique, il suffit de fixer l'ordre des β_i pour découvrir de nouveaux brins et une stratégie pour traiter les brins découverts (FIFO, LIFO, ...). La complexité d'un parcours d'une carte est $\mathcal{O}(n \cdot |B|)$.

La méthode décrite dans les algorithmes 2 et 3 choisit d'explorer les fonctions β_i dans l'ordre de β_0 à β_n et d'utiliser une stratégie FIFO pour traiter les brins découverts. L'algorithme fixe un brin de départ pour une des deux cartes, puis va tester tous les brins de départ possible pour la seconde. Ensuite à partir de ce premier couple de brins appariés, il construit la suite de l'appariement lors du parcours. Comme ce parcours est unique, si à la fin d'un parcours la fonction d'appariement n'est pas une fonction d'isomorphisme alors le couple de départ qui avait été choisi ne fait pas partie de la fonction d'isomorphisme si elle existe. Si pour un des couples de départ la fonction d'appariement construite est une fonction d'isomorphisme alors les deux cartes sont isomorphes.

La complexité en temps de cet algorithme est $\mathcal{O}(|B|^2 \cdot n)$.

Algorithme 2 : $VerifierIsomorphisme(C, C')$

Entrées : deux n-cartes connexes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$.
Sortie : retourne vrai ssi C et C' sont isomorphes.

```
1 choisir b_0 ∈ B
2 pour chaque b'_0 ∈ B' faire
3     f ← ConstruireAppariement(C, C', b_0, b'_0)
4     si f est une fonction d'isomorphisme alors
5         retourner VRAI
6 retourner FAUX
```

Sur l'exemple 4, l'algorithme 3 calcul la fonction d'isomorphisme si le couple de brin de départ (b_0, b'_0) est bon, par exemple $(a, 1)$ ou $(c, 3)$. Pour d'autres couples de départ, la fonction calculée ne sera pas une fonction d'isomorphisme. Prenons par exemple le couple $(a, 5)$, la fonction construite est donnée dans le tableau de la figure 2.7 et n'est pas une fonction d'isomorphisme, car $\beta_1(c) = a$ donc $\beta'_1(f(c))$ devrait être égal à $f(a)$, or $\beta'_1(f(c)) = \beta'_1(6) = 7$ et $f(a) = 5$.

Automorphisme : Un automorphisme est une fonction d'isomorphisme d'une carte avec elle-même. Toute carte $C = (B, \beta_1, \ldots, \beta_n)$ possède au moins un automorphisme trivial, la fonction d'isomorphisme étant la fonction identité. $\forall b \in B, f(b) = b$.

Algorithme 3 : $ConstruireAppariement(C, C', b_0, b'_0)$

Entrées : deux n-cartes connexes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$ et un couple de brins de départs $(b_0, b'_0) \in B \times B'$.

Sortie : retourne une injection $f : B \cup \{\epsilon\} \to B' \cup \{\epsilon\}$.

```
1  pour b ∈ B faire f[b] ← ε
2  f[b_0] ← b'_0
3  P est une pile vide
4  insérer b_0 dans P
5  tant que P n'est pas vide faire
6      prendre le premier brin b dans P
7      pour chaque i ∈ {0, ..., n} faire
8          si b n'est pas i-libre et f[β_i(b)] = ε alors
9              f[β_i(b)] ← β'_i(f[b])
10             insérer β_i(b) dans P
11 f[ε] = ε
12 retourner f
```

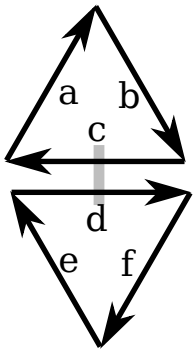


Tableau des automorphismes

	a	b	c	d	e	f
f_1	a	b	c	d	e	f
f_2	f	e	d	c	b	a

FIGURE 2.8 – Exemple d'une 2-carte ayant deux automorphismes. f_1 et f_2 correspondent aux deux fonctions d'automorphismes possibles. f_1 est un automorphisme trivial et f_2 est un automorphisme non-trivial.

Exemple 5. *La figure 2.8 donne un exemple d'une 2-carte possédant deux automorphismes.*

Sous-carte : Une sous-carte est une sous-partie d'une carte, c.-à-d. un sous-ensemble de ses brins et des fonctions β_i réduites aux brins sélectionnés (voir définition 11).

Définition 11 (Sous-carte [14])**.**
Soit une n-carte $C = (B, \beta_1, \ldots, \beta_n)$ et un sous-ensemble de brins $X \subseteq B$, la sous-carte de C induite par X, notée $C_{\downarrow X}$ [3] *$= (X, \beta'_1, \ldots, \beta'_n)$ est telle que :* $\forall b \in$

3. $C_{\downarrow X}$ peut ne pas être une carte valide si les brins sélectionnés dans X ne permettent pas de vérifier les contraintes de validités d'une carte, par la suite nous considérons que des sous-cartes valides.

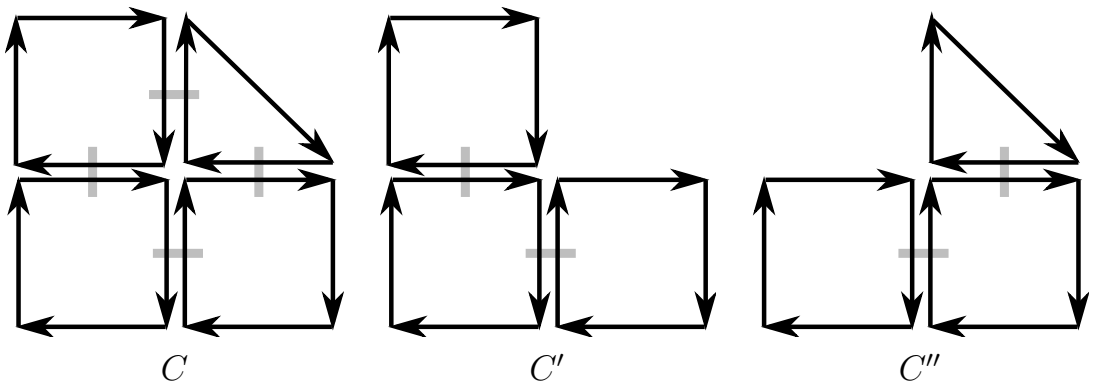


FIGURE 2.9 – Exemple de relation d'ordre entre trois cartes. $C > C'$ et $C > C''$. L'ordre sur les cartes étant partiel, il n'y a pas d'ordre entre C' et C''

$X, \forall i, 1 \leq i \leq n$, *si* $\beta_i(b) \notin X$ *alors* $\beta'_i(b) = \epsilon$ *; sinon* $\beta'_i(b) = \beta_i(b)$

Notons qu'il existe une relation d'ordre partielle entre cartes combinatoires. La figure 2.9 montre trois cartes, C' et C'' sont des sous-cartes de C donc $C' < C$ et $C'' < C$, mais il n'existe pas de relation entre C' et C''.

Isomorphisme de sous-carte :

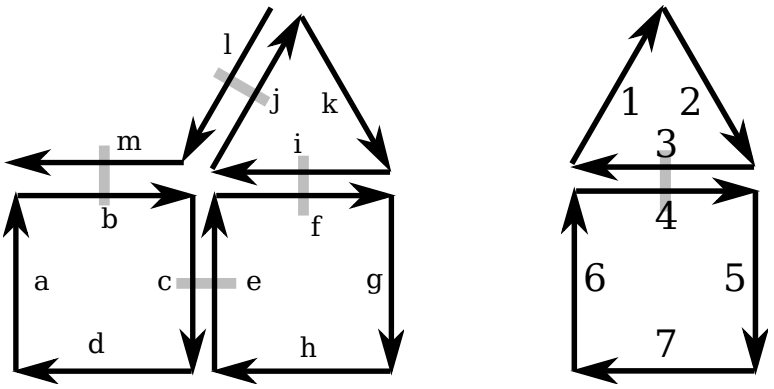
Définition 12 (Isomorphisme de sous-carte)**.**
Soient deux n*-cartes* $C = (B, \beta_1, \ldots, \beta_n)$ *et* $C' = (B', \beta'_1, \ldots, \beta'_n)$*.* C' *est isomorphe à une sous-carte de* C *s'il existe un sous-ensemble de brins* $X \subseteq B$ *tel que* $C_{\downarrow X}$ *soit isomorphe à* C'

Théorème 1. *C est isomorphe à une sous-carte de C' si et seulement si il existe une injection $f : B \cup \{\epsilon\} \to B' \cup \{\epsilon\}$, appelée fonction de sous-isomorphisme telle que :*

1. $f(\epsilon) = \epsilon$ *;*
2. $\forall b \in B, \forall i, 1 \leq i \leq n$ *;*
   - *si b n'est pas i-libre, alors $\beta'_i(f(b)) = f(\beta_i(d))$;*
   - *sinon, $f(b)$ est aussi i-libre, ou $\forall b_k \in B, f(b_k) \neq \beta'_i(f(b))$.*

Exemple 6. *La figure 2.10 donne un exemple d'isomorphisme de sous-cartes.*

Le cas de l'isomorphisme de sous-carte est proche de celui d'isomorphisme de carte, la différence vient du cas particulier des brins i-libres. Si un brin b est i-libre alors son image doit soit être également i-libre, soit si ce n'est pas le cas, que le brin adjacent à b ne soit pas apparié. Cela vient du fait que pour l'isomorphisme de sous-cartes, les deux cartes ne sont pas de même taille, donc la frontière des brins i-libres de la petite carte peut avoir une image dans la grande carte qui ne soit pas i-libre. Sur l'exemple de la figure 2.10, nous pouvons voir que pour le brin 2 qui est apparié au brin k, 2 est 2-libre et $f(2) = k$ est aussi 2-libre. En revanche, le brin 6 est aussi 2-libre, cependant $f(6) = e$ n'est pas 2-libre, car $\beta_2(e) = c$, mais comme c n'a pas d'image dans la petite carte, f est bien



	1	2	3	4	5	6	7
f	j	k	i	f	g	e	h

FIGURE 2.10 – Exemple d'isomorphisme de sous cartes. La fonction d'isomorphisme est donnée dans le tableau.

une fonction d'isomorphisme de sous-carte. La preuve de ce théorème est faite dans [14].

Les algorithmes 2 et 3 peuvent être utilisés presque directement. Il y a deux différences dans l'algorithme 2, premièrement, il faut s'assurer que le premier brin fixé à la ligne 1 fait partie de la petite carte, ensuite, à la ligne 4, il faut vérifier si la fonction f construite par l'algorithme 3 est une fonction de sous-isomorphisme et non pas une fonction d'isomorphisme. La version modifiée de l'algorithme 2 est donnée dans l'algorithme 4.

Algorithme 4 : $VerifierSousIsomorphisme(C, C')$

Entrées : deux n-cartes connexes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$.

Sortie : retourne vrai ssi C est isomorphe à une sous-carte de C'.

```
1 choisir b_0 ∈ B
2 pour chaque b'_0 ∈ B' faire
3     f ← ConstruireAppariement(C, C', b_0, b'_0)
4     si f est une fonction de sous-isomorphisme alors
5         retourner VRAI
6 retourner FAUX
```

Pour vérifier si une carte $C = (B, \beta_1, \ldots, \beta_n)$ est isomorphe à une sous-carte de $C' = (B', \beta'_1, \ldots, \beta'_n)$, il faut parcourir la carte C pour chaque brin de la carte C', la complexité de l'algorithme de calcul de l'isomorphisme de sous-carte est donc $\mathcal{O}(n \cdot |B| \cdot |B'|)$. Notons que la complexité est identique à l'algorithme d'isomorphisme, mais dans le cas de l'isomorphisme, $|B| = |B'|$.

Chapitre 3

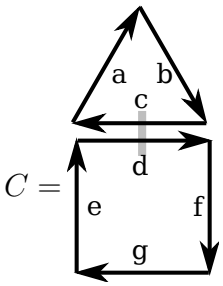
Signatures de cartes combinatoires

Sommaire

3.1 Signatures de cartes connexes 26
3.1.1 Étiquetage de cartes 26
3.1.2 Signature sous forme d'un ensemble de mots (S-signature) 30
3.1.3 Signature sous forme d'un mot (W-signature) 33
3.2 Signatures d'une base de cartes connexes 34
3.2.1 S-signature d'une base de cartes (TS-signature) 35
3.2.2 W-signature d'une base de cartes (TW-signature) 37
3.2.3 Nombre de fils d'un nœud d'un arbre de signatures . . . 39
3.3 Signatures de cartes valuées 42
3.4 Signatures de cartes non connexes 42
3.5 Évaluation 44

Introduction

DANS le chapitre 2, nous avons étudié différents problèmes d'extraction de sous-structures fréquentes, notamment celui de sous-graphes qui est une structure de données assez proche des cartes combinatoires. Nous avons vu qu'il est primordial pour la recherche de sous-structures fréquentes d'avoir une forme canonique de cette structure et de pouvoir décider efficacement de l'isomorphisme entre deux éléments. De plus, nous avons présenté les cartes combinatoires ainsi que des algorithmes efficaces pour décider de l'isomorphisme de cartes et de sous-cartes. Dans ce chapitre, nous définissons en section 3.1 deux signatures de cartes combinatoires qui sont des formes canoniques de n-cartes connexes. Dans la section 3.2, nous étendons ces signatures de



$l = \{\epsilon : 0,\ a : 3,\ b : 1,\ c : 5\ , d : 7,\ e : 2,\ f : 6,\ g : 4\}$
$l' = \{\epsilon : 0,\ a : 5,\ b : 2,\ c : 4\ , d : 1,\ e : 6,\ f : 3,\ g : 7\}$ (a)

$BFL(C, g) = \{\epsilon : 0,\ a : 7,\ b : 6,\ c : 5,\ d : 4,\ e : 3,\ f : 2,\ g : 1\}$
$BFL(C, c) = \{\epsilon : 0,\ a : 3,\ b : 2,\ c : 1,\ d : 4,\ e : 5,\ f : 6,\ g : 7\}$ (b)

$W_{BFL}(C, g) =< 3, 0, 1, 0, 4, 0, 2, 5, 7, 4, 5, 0, 6, 0 >$
$W_{BFL}(C, c) =< 3, 4, 1, 0, 2, 0, 6, 1, 4, 0, 7, 0, 5, 0 >$ (c)

FIGURE 3.1 – Exemples d'étiquetages et de mots d'une carte. (a) l et l' sont deux étiquetages quelconques de la carte C. (b) $BFL(C, g)$ (resp. $BFL(C, c)$) correspond à l'étiquetage en largeur de la carte C en prenant le brin g (resp. c) comme brin de départ. (c) $W_{BFL}(C, g)$ (resp. $W_{BFL}(C, c)$) correspond au mot construit à partir de l'étiquetage retourné par l'algorithme $BFL(C, g)$ (resp. $BFL(C, c)$).

cartes pour définir des signatures de bases de cartes connexes. Nous étudions le cas des cartes valuées et des cartes non connexes dans les sections 3.3 et 3.4. Enfin, nous évaluons les performances des différentes signatures en section 3.5.

3.1 Signatures de cartes connexes

Dans cette section, nous introduisons deux représentations canoniques de cartes, appelées signatures. Nous considérons uniquement le cas des cartes connexes ; une extension aux cartes non connexes est décrite en Section 3.4.

3.1.1 Étiquetage de cartes

Nos signatures sont basées sur un étiquetage des cartes tel que chaque brin est associé à une étiquette différente. Par définition, l'étiquette pour ϵ est 0.

Définition 13 (Étiquetage (labelling))**.**
Soit une carte $C = (B, \beta_1, \ldots, \beta_n)$, un étiquetage de C est une fonction bijective $l : B \cup \{\epsilon\} \to \{0, \ldots, |B|\}$ telle que $l(\epsilon) = 0$.

Exemple 7. *La figure 3.1(a) montre deux exemples d'étiquetages d'une carte.*

Notons que l^{-1} donne pour une étiquette e le brin b tel que $l(b) = e$.

Un étiquetage d'une carte peut se calculer en parcourant celle-ci, et en numérotant les brins au fur et à mesure de leur découverte. Pour une même carte, des étiquetages seront différents si :

1. les brins de départ sont différents ;
2. les stratégies pour gérer l'ordre des brins à traiter sont différentes (par exemple, FIFO ou LIFO) ;
3. les ordres dans lesquels les fonctions β_i sont utilisées pour découvrir de nouveaux brins sont différents.

Algorithme 5 : $BFL(C, b)$

Entrées : Une carte connexe $C = (B, \beta_1, \ldots, \beta_n)$, et un brin $b \in B$.
Sortie : un étiquetage $l : B \cup \{\epsilon\} \to \{0, \ldots, |B|\}$.

```
1  pour tous les brins b' ∈ B faire
2      l(b') ← −1
3  l(ε) ← 0
4  Soit F une file vide
5  ajouter b à la fin de F
6  l(b) ← 1
7  etiquette ← 2
8  tant que F n'est pas vide faire
9      enlever b' de la tête de F
10     pour i ← 0 à n faire
11         si l(β_i(b')) = −1 alors
12             l(β_i(b')) ← etiquette
13             etiquette ← etiquette + 1
14             ajouter β_i(b') à la fin de F
15 retourner l
```

L'étiquetage correspondant à un parcours en largeur de la carte où les fonctions β_i sont utilisées dans l'ordre croissant est défini comme suit.

Définition 14 (Étiquetage en largeur (Breadth First Labelling))**.**
Soit une carte connexe $C = (B, \beta_1, \ldots, \beta_n)$ et un brin $b \in B$. L'étiquetage BFL associé à (C, b) est l'étiquetage retourné par la fonction $BFL(C, b)$ décrite dans l'algorithme 5.

Exemple 8. *La figure 3.1 (b) montre deux exemples d'étiquetages en largeur d'une carte.*

Proposition 1. *L'algorithme 5 retourne un étiquetage.*

Démonstration.
- $l(\epsilon)$ vaut 0 (ligne 2).
- $\forall b, b' \in B, b \neq b' \Rightarrow l(b) \neq l(b')$. En effet, à chaque fois qu'un brin est étiqueté (ligne 6 et 12) le compteur *étiquette* est incrémenté (ligne 13).
- $\forall b \in B, 1 \leq l(b) \leq |B|$. Chaque brin entre exactement une fois dans la file F car (i) la carte est connexe, et (ii) un brin entre dans la file seulement s'il n'a pas encore été étiqueté, enfin il reçoit une étiquette juste avant d'entrer dans la file.

□

Proposition 2. *La complexité en temps de l'algorithme 5 est $\mathcal{O}(n \cdot |B|)$*

Démonstration. La boucle **tant que** (lignes 8-14) est itérée $|B|$ fois, car (i) exactement un brin est retiré de la file à chaque itération ; et (ii) chaque brin $b \in B$ entre une fois dans la file. La boucle **pour** (lignes 10-14) est itérée $n + 1$ fois. □

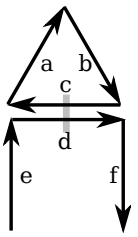


FIGURE 3.2 – Exemple de carte combinatoire avec une face ouverte.

Notons que la boucle (lignes 10-14) itère pour chaque $i \in \{0, \ldots, n\}$, 0 compris. En effet, comme nous considérons des cartes ouvertes, certains brins peuvent ne pas être 1-cousu. Dans ce cas, certains brins peuvent ne pas être atteignables à partir du brin de départ b sans utiliser la fonction β_0.

Exemple 9. *La figure 3.2 illustre l'utilité de la fonction β_0 dans le cas d'une carte avec une face ouverte. Si le brin de départ de BFL est le brin f, alors aucun brin ne sera découvert si seulement β_1 et β_2 sont utilisés pour découvrir de nouveaux brins (car $\beta_1(f) = \beta_2(f) = \epsilon$ dans ce cas). Cependant, en utilisant β_0, β_1, et β_2, tous les brins sont parcourus.*

Soit une carte C et un étiquetage l. C peut être décrite (c.-à-d., ses fonctions β_1 à β_n) par une séquence d'étiquettes de l. L'idée est de lister les n étiquettes des n brins i-cousus au brin étiqueté par 1 (*c.-à-d.*, $l(\beta_1(1)), \ldots, l(\beta_n(1))$), puis par 2 (*c.-à-d.*, $l(\beta_1(2)), \ldots, l(\beta_n(2))$), etc. Plus formellement, le mot associé à une carte et à un étiquetage est défini comme suit.

Définition 15 (Mot (Word))**.**
Soit une carte $C = (B, \beta_1, \ldots, \beta_n)$ et un étiquetage $l : B \cup \{\epsilon\} \to \{0, \ldots, |B|\}$. Le mot associé à (C, l) est la séquence

$$W(C, l) =< w_1, \ldots, w_{n \cdot |B|} >$$

telle que $\forall i \in \{1, \ldots, n\}$, $\forall k \in \{1, \ldots, |B|\}$, $w_{i \cdot k} = l(\beta_i(b_k))$ où b_k est le brin étiqueté k, c.-à-d., $b_k = l^{-1}(k)$.

Algorithme. Soit une carte $C = (B, \beta_1, \ldots, \beta_n)$ et un étiquetage l. Le mot $W(C, l)$ est construit en prenant chaque brin de B dans l'ordre croissant de l'étiquetage et en listant les étiquettes de ses n brins i-cousus. Cela est réalisé par l'algorithme 6 en $\mathcal{O}(n \cdot |B|)$.

Notation. Le mot associé au parcours en largeur (BFL) d'une carte C et en commençant par le brin b est noté $W_{BFL}(C, b)$:

$$W_{BFL}(C, b) = W(C, BFL(C, b))$$

Exemple 10. *La figure 3.1(c) montre deux exemples de mots d'une carte.*

Algorithme 6 : $W(C, l)$

Entrées : Une carte $C = (B, \beta_1, \ldots, \beta_n)$ et un étiquetage l.
Sortie : le mot correspondant à l'étiquetage l de la carte C.

```
1 mot ← ∅
2 pour j ← 0 à |B| faire
3     pour i ← 0 à n faire
4         mot ← mot + l(β_i(l^{-1}(j)))
5 retourner mot
```

Le point principal qui nous permet d'utiliser les mots pour construire une signature est que deux cartes sont isomorphes si et seulement si elles ont au moins un mot en commun construit à partir d'un parcours en largeur, comme l'explique le théorème 2.

Théorème 2. *Deux cartes* $C = (B, \beta_1, \ldots, \beta_n)$ *et* $C' = (B', \beta'_1, \ldots, \beta'_n)$ *sont isomorphes si et seulement si il existe deux brins* $b \in B$ *et* $b' \in B'$ *tels que* $W_{BFL}(C, b) = W_{BFL}(C', b')$

Démonstration. $\Rightarrow$ Soient deux cartes isomorphes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$, montrons qu'il existe deux brins b et b' tels que $W_{BFL}(C, b) = W_{BFL}(C', b')$. Si C et C' sont isomorphes alors il existe une fonction $f : B \to B'$ telle que $\forall b \in B, \forall i \in \{1, \ldots, n\}, f(\beta_i(b)) = \beta'_i(f(b))$ (définition 10). Soit b_1 un brin de B, notons l (resp. l') l'étiquetage retourné par $BFL(C, b_1)$ (resp. $BFL(C', f(b_1))$).

Affirmation 1 : l et l' sont telles que $\forall b_i \in B, l(b_i) = l'(f(b_i))$. Cela est vrai pour le brin de départ, en effet, b_1 et $f(b_1)$ sont étiquetés par 1 au début du parcours. De plus, $\forall i \in \{0, \ldots, n\}, f(\beta_i(b_1)) = \beta'_i(f(b_1))$ (par définition de la fonction d'isomorphisme), donc $l(\beta_i(b_1)) = l'(f(\beta_i(b_1)))$ (conséquence des lignes 10 à 14 de l'algorithme 5). En répétant ce raisonnement récursivement, nous avons bien $\forall b_i \in B, l(b_i) = l'(f(b_i))$.

Affirmation 2 : $\forall k \in \{1, \ldots, |B|\}, f(l^{-1}(k)) = l'^{-1}(k)$. C'est une conséquence directe de l'affirmation 1.

Conclusion : $\forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, |B|\}$, le $i.k^{me}$ élément de $W_{BFL}(C, b_1)$ est égal au $i.k^{me}$ élément de $W'_{BFL}(C', f(b_1))$, c.-à-d., $l(\beta_i(l^{-1}(k))) = l'(\beta'_i(l'^{-1}(k)))$. En effet,

$$\begin{aligned} l(\beta_i(l^{-1}(k))) &= l'(f(\beta_i(l^{-1}(k)))) && \text{(selon l'affirmation 1)} \\ &= l'(\beta'_i(f(l^{-1}(k)))) && \text{(car } f \text{ est une fonction d'isomorphisme)} \\ &= l'(\beta'_i(l'^{-1}(k))) && \text{(selon l'affirmation 2)} \end{aligned}$$

$\Leftarrow$ Soient deux cartes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$ et deux brins b et b' tels que $W_{BFL}(M, d) = W_{BFL}(M', d')$, montrons que C et C' sont isomorphes. Notons l (resp. l') l'étiquetage retourné par $BFL(C, b)$ (resp. $BFL(C', b')$), et définissons la fonction $f : B \to B'$ qui associe les brins qui ont une même

Algorithme 7 : $SS(C)$

Entrées : Une carte connexe $C = (B, \beta_1, \ldots, \beta_n)$.
Sortie : la S-signature de C, $SS(C)$.

```
SS ← ∅
pour tous les brins b ∈ B faire
    SS ← SS ∪ { W_BFL(C, b) }
retourner SS
```

étiquette, c.-à-d. , $\forall b_j \in B, f(b_j) = l'^{-1}(l(b_j))$. Cela implique aussi que $l(b_j) = l'(f(b_j))$.

Affirmation 3 : $\forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, |B|\}, l(\beta_i(l^{-1}(k))) = l'(\beta'_i(l'-1(k)))$. $W_{BFL}(C, b) = W_{BFL}(C', b')$ donc le $i.k^{me}$ élément de $W_{BFL}(C, b_1)$ est égal au $i.k^{me}$ élément de $W'_{BFL}(C', b')$. Or le $i.k^{me}$ élément de $W_{BFL}(C, b)$ correspond à $l(\beta_i(l^{-1}(k)))$ par construction du mot (voir algorithme 6) et le $i.k^{me}$ élément de $W'_{BFL}(C', b')$ vaut $l'(\beta'_i(l^{-1}(k)))$.

Conclusion : $\forall i \in \{1, \ldots, n\}, \forall b_j \in B$,

$$
\begin{aligned}
f(\beta_i(b_j)) &= l'^{-1}(l(\beta_i(b_j))) && \text{(par définition de } f\text{)} \\
&= l'^{-1}(l'(\beta'_i(l'^{-1}(l(b_j))))) && \text{(selon l'affirmation 3 en remplaçant } b_j \text{ par } l^{-1}(k)\text{)} \\
&= \beta'_i(l'^{-1}(l(b_j))) && \text{(par simplification)} \\
&= \beta'_i(l'^{-1}(l'(f(b_j)))) && \text{(par définition de } f, \forall (b, b') \in B \times B' \\
& && \text{si } l(b) = l'(b') \text{ alors } f(b) = b' \text{ et inversement)} \\
&= \beta'_i(f(b_j)) && \text{(par simplification)}
\end{aligned}
$$

Donc, f est une fonction d'isomorphisme et C et C' sont isomorphes. □

3.1.2 Signature sous forme d'un ensemble de mots (S-signature)

Une carte peut être définie par l'ensemble de tous les mots construits par la fonction W_{BFL}, pour tous les brins de cette carte. Cet ensemble constitue notre première signature.

Définition 16 (S-signature)**.**
Soit une carte $C = (B, \beta_1, \ldots, \beta_n)$*. La S-signature de* C *est*
$SS(C) = \{W_{BFL}(C, b) | b \in B\}$.

Algorithme. $SS(M)$ est construit en calculant $W_{BFL}(C, b)$ pour chaque brin $b \in B$ et en conservant tous les mots différents dans $SS(C)$. Donc, la complexité de l'algorithme 7 est $\mathcal{O}(n \cdot |B|^2)$.

Théorème 3. *$SS(M)$ est une signature, c.-à-d., deux cartes C et C' sont isomorphes si et seulement si $SS(C) = SS(C')$.*

Algorithme 8 : $T_{SS}(C)$

Entrées : Une S-signature SS d'une carte connexe $C = (B, \beta_1, \ldots, \beta_n)$.
Sortie : un arbre de S-signature $T_{SS}(C)$.

```
1  Nous considérons SS comme une liste de |B| mots de taille |B| · n
2  T_SS ← ∅
3  pour chaque mot m de SS faire
4      se positionner à la racine de T_SS
5      pour i ← de 1 à |B| · n faire
6          si il existe un fils de T_SS égal à m[i] alors
7              aller sur ce fils
8          sinon
9              créer un fils d'étiquette m[i] fils du nœud courant
10             aller sur ce fils
11 retourner T_SS
```

Démonstration. $\Rightarrow$ Soient $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$ deux cartes isomorphes, montrons que $SS(C) = SS(C')$. D'après le théorème 2, si C et C' sont isomorphes, alors $\exists b \in B$ et $b' \in B'$ tels que $W_{BFL}(C, b) = W_{BFL}(C', b')$ et $f(b) = b'$. De plus, si pour un couple de brins $b \in B$ et $b' \in B'$ $f(b) = b'$, alors que $W_{BFL}(C, b) = W_{BFL}(C', b')$. Donc, étant donnée une fonction d'isomorphisme f entre C et C', on a, pour chaque brin $b \in B$, $W_{BFL}(C, b) = W_{BFL}(C', f(b))$. Donc chaque mot de $SS(C)$, calculé pour chaque brin de B, appartient forcément à $SS(C')$ (et inversement).

$\Leftarrow$ Soient deux cartes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$ telles que $SS(C) = SS(C')$, montrons que C et C' sont isomorphes. Il existe deux mots $W \in SS(M)$ et $W' \in SS(M')$ tels que $W = W'$, donc, selon le théorème 2, C et C' sont isomorphes. □

Une conséquence directe du théorème 2 et du théorème 3 est que deux cartes non isomorphes C et C' sont telles que $SS(C) \cap SS(C') = \emptyset$.

La S-signature d'une carte C peut être représentée par un arbre lexicographique qui regroupe les préfixes communs des mots de l'ensemble. L'algorithme 8 décrit la construction de l'arbre à partir de la S-signature d'une carte.

Définition 17 (S-Signature sous forme d'un arbre de mots)**.**
Soit une carte $C = (B, \beta_1, \ldots, \beta_n)$. L'arbre associé à la S-signature de C est l'arbre $T_{SS}(C)$ tel que :

- *chaque nœud u sauf la racine est étiqueté par $l(u)$ qui est un entier compris entre 0 et $|B|$; notons $w(u)$ le mot obtenu en concaténant toutes les étiquettes rencontrées le long du chemin qui va de la racine de l'arbre au nœud u ;*
- *pour chaque nœud u, tous les fils de u ont une étiquette différente ;*
- *il y a $|SS(M)|$ feuilles et pour chaque feuille u, $w(u) \in SS(M)$.*

Propriété 1. *La complexité en temps pour calculer l'arbre de S-signature $T_{SS}(C)$ d'une carte C est $\mathcal{O}(n \cdot |B|^2)$.*

Démonstration. L'arbre peut être construit de manière incrémentale : en partant d'un arbre qui ne contient que la racine, tous les mots $W_{BFL}(C,b)$ sont ajoutés successivement, pour chaque brin $b \in B$. La complexité en temps pour calculer un mot est $\mathcal{O}(n \cdot |B|)$. La complexité pour ajouter ce mot à l'arbre est aussi $\mathcal{O}(n \cdot |B|)$. En effet, la longueur d'un mot est $n \cdot |B|$. Pour chaque étiquette x du mot, il faut vérifier si le nœud courant de l'arbre à un fils u tel que $l(u) = x$. C'est fait en temps linéaire par rapport au nombre de fils du nœud, car nous utilisons une liste pour mémoriser les fils d'un nœud. Le nombre de feuille de l'arbre est au maximum $|B|$, donc un nœud a au plus $|B|$ fils. Il y a donc au minimum $n \cdot |B|$ nœud entre la racine et une feuille car c'est la longueur d'un mot et au maximum $n \cdot |B| + |B|$. □

Propriété 2. *La complexité spatiale de la S-signature $T_{SS}(C)$ d'une carte C est $\mathcal{O}(n \cdot |B|^2)$.*

Démonstration. L'arbre contient une feuille pour chaque mot de la S-signature, c.-à-d. au plus $|B|$ feuilles, et la taille de chaque chemin de la racine à une feuille est $n \cdot |B|$, soit la longueur d'un mot de la signature. Donc l'arbre a au maximum $\mathcal{O}(n \cdot |B|^2)$ nœuds. □

Notons que pour deux feuilles u et v, $u \neq v \Rightarrow w(u) \neq w(v)$.

Exemple 11. *La figure 3.3 montre un exemple de S-signature d'une carte.*

Exemple 12. *La figure 3.4 illustre l'impact des automorphismes sur la signature. En effet, la carte est composée de six brins, la signature devrait donc être composée de six mots. Cependant, la carte possède deux automorphismes, c.-à-d. , il existe deux fonctions différentes d'isomorphismes avec elle-même, donc il existe des couples de brins $b \in B$ et $b' \in B$ tels que $b \neq b'$ et $W_{BFL}(C,b) = W_{BFL}(C,b')$. Dans cette exemple, les brins a et f produisent le même mot, c'est aussi le cas pour b et e ainsi que pour c et d.*

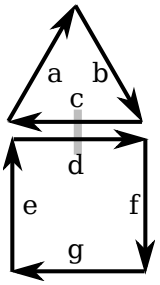
Propriété 3. *Soient une carte $C = (B, \beta_1, \ldots, \beta_n)$ et une S-signature $T_{SS}(C')$ d'une autre carte C', savoir si C et C' sont isomorphes peut être déterminé en $\mathcal{O}(n \cdot |B|)$.*

Démonstration. Pour décider de l'isomorphisme, il faut construire un mot $W_{BFL}(C,b)$ à partir de n'importe quel brin $b \in B$ et vérifier si ce mot correspond à un chemin de l'arbre allant de la racine à une feuille. La construction du mot est faite en $\mathcal{O}(n \cdot |B|)$. Comme le nombre de fils entre la racine et une feuille de la S-signature est borné par $n \cdot |B| + |B|$, la recherche d'un mot dans un arbre est faite aussi en $\mathcal{O}(n \cdot |B|)$. □

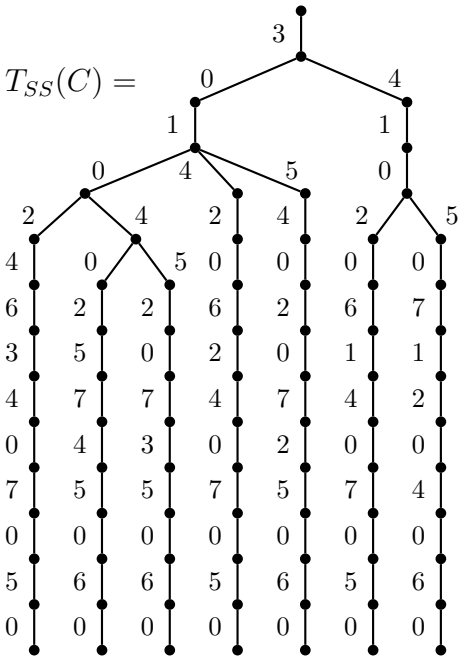
Notons qu'il est possible de vérifier si le mot correspond à un chemin de l'arbre pendant la construction de celui-ci et qu'il est possible de stopper la construction avant la fin si aucune branche ne correspond.

Remarque 1. *Cette méthode est optimale. En effet, pour décider de l'isomorphisme entre deux cartes $C = (B, \beta_1, \ldots, \beta_n)$ et $C' = (B', \beta'_1, \ldots, \beta'_n)$, il faut vérifier si $f(\beta_i(b)) = \beta'_i(f(b))$ pour chaque brin $b \in |B|$ et pour toutes les dimensions $i \in \{1, \ldots, n\}$. Cela ne peut pas être fait en moins de $\mathcal{O}(n \cdot |B|)$.*

$C =$



$T_{SS}(C) =$



$$SS(C) = \{ < 3,0,1,0,4,0,2,5,7,4,5,0,6,0 >,$$
$$< 3,0,1,0,4,5,2,0,7,3,5,0,6,0 >,$$
$$< 3,0,1,4,2,0,6,2,4,0,7,0,5,0 >,$$
$$< 3,4,1,0,2,0,6,1,4,0,7,0,5,0 >,$$
$$< 3,4,1,0,5,0,7,1,2,0,4,0,6,0 >,$$
$$< 3,0,1,0,2,4,6,3,4,0,7,0,5,0 >,$$
$$< 3,0,1,5,4,0,2,0,7,2,5,0,6,0 > \}$$

$$WS(C) = < 3,0,1,0,2,4,6,3,4,0,7,0,5,0 >$$

FIGURE 3.3 – Signatures de carte. $SS(C)$ est la S-signature de la carte C, $T_{SS}(C)$ est la représentation sous forme d'arbre de la S-signature et $WS(C)$ est la W-signature de la carte C.

3.1.3 Signature sous forme d'un mot (W-signature)

Nous avons défini une seconde signature qui ne conserve que le plus petit mot de la S-signature. Cela est possible, car l'ordre lexicographique est un ordre total strict sur les mots et qu'il suffit que deux S-signatures aient un mot en commun pour qu'elles soient égales.

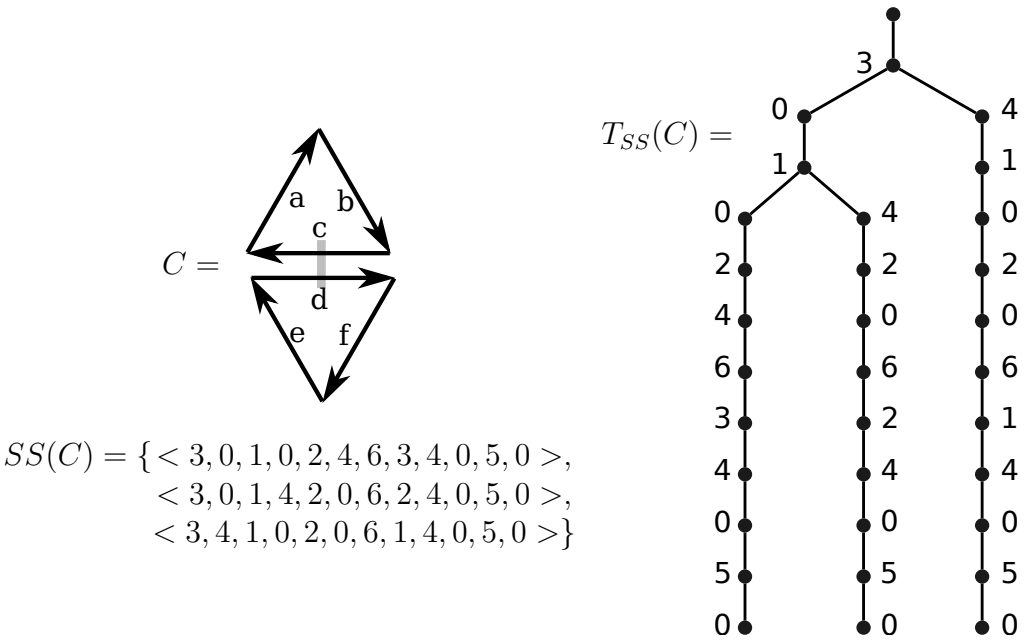
Définition 18 (W-signature)**.**
Soit une carte C donnée, la W-signature de C est, $WS(C) = w \in SS(C)$ tel que $\forall w' \in SS(C), w \leq w'$.

Exemple 13. *La figure 3.3 montre un exemple de W-signature d'une carte.*

Algorithme. La W-signature d'une carte C est construite en appelant la fonction $W_{BFL}(C,b)$ pour chaque brin $b \in B$, et en ne gardant que le plus petit mot, selon l'ordre lexicographique. La complexité en temps de l'algorithme 9 est $\mathcal{O}(n \cdot |B|^2)$. Le processus peut être amélioré (sans changer la complexité dans le pire cas) en comparant le mot en construction avec le plus petit mot actuel et en stoppant la construction dès qu'il est plus grand.

Propriété 4. *La complexité spatiale de la W-signature est $\mathcal{O}(n \cdot |B|)$.*

Propriété 5. *À partir d'une carte $C = (B, \beta_1, \ldots, \beta_n)$ et une W-signature $WS(C')$ d'une autre carte C', déterminer si C et C' sont isomorphes peut être calculé en $\mathcal{O}(n \cdot |B|^2)$.*



$$SS(C) = \{< 3,0,1,0,2,4,6,3,4,0,5,0 >, \\ < 3,0,1,4,2,0,6,2,4,0,5,0 >, \\ < 3,4,1,0,2,0,6,1,4,0,5,0 >\}$$

FIGURE 3.4 – S-signatures d'une carte ayant deux automorphismes.

Algorithme 9 : $WS(C)$

Entrées : Une carte connexe $C = (B, \beta_1, \ldots, \beta_n)$.
Sortie : la W-signature de C, WS.

```
choisir au hasard un brin b_0 ∈ B
WS ← W_BFL(C, b)
pour tous les brins b ∈ B \ b_0 faire
    word ← W_BFL(C, b)
    si word < WS alors
        WS ← word
retourner WS
```

Démonstration. Pour décider de l'isomorphisme, il faut construire tous les mots à partir de tous les brins $b \in B$, et vérifier si $W_{BFL}(C,b) = WS(C')$. Dans le pire des cas, $|B|$ mots sont construits, donc la complexité en temps est en $\mathcal{O}(n \cdot |B|^2)$. □

3.2 Signatures d'une base de cartes connexes

Les signatures de cartes définies précédemment permettent de décider de l'isomorphisme entre deux cartes. Cependant, il est souvent nécessaire de devoir comparer une carte C non pas avec une seule autre carte, mais avec une base entière de cartes pour rechercher toutes les cartes isomorphes à C. Dans cette optique, nous définissons une signature de base de cartes. Cette signature fusionne l'ensemble des signatures des cartes de la base en un unique arbre. Il

est possible d'utiliser aussi bien les S-signatures que les W-signatures.

Dans cette section, nous considérons à nouveau uniquement des cartes connexes ; l'extension aux cartes non connexes est faite en Section 3.4.

3.2.1 S-signature d'une base de cartes (TS-signature)

Il est possible de représenter une base $BC = \{C^1, \ldots, C^k\}$ de k cartes par une liste de k S-signatures indépendantes. Étant donné une nouvelle carte $C = (B, \beta_1, \ldots, \beta_n)$, nous pouvons vérifier si une des cartes de BC est isomorphe à C en $\mathcal{O}(k \cdot n \cdot |B|)$ en recherchant consécutivement C dans chaque arbre de la base. Nous proposons d'améliorer cette complexité en fusionnant les k arbres en un seul.

Si toutes les cartes d'une base ont le même nombre de brins, alors toutes les branches (entre la racine et une feuille) ont la même longueur dans tous les arbres. Dans ce cas, toutes les branches ont aussi la même longueur dans l'arbre fusionné et il suffit de mémoriser pour chaque feuille u, l'ensemble des cartes telles que $w(u)$ appartient à la S-signature de la carte (la base peut contenir plusieurs cartes isomorphes, c'est pour cela qu'une feuille peut correspondre à plusieurs cartes).

Cependant, si toutes les cartes n'ont pas le même nombre de brins, alors nous devons fusionner des arbres dont les branches ont des tailles différentes. Dans ce cas, il peut arriver qu'un mot associé à une feuille dans un arbre soit un préfixe d'un mot associé à la feuille d'un autre arbre. Dans ce cas, quand nous fusionnons deux arbres, un mot peut terminer sur un nœud qui ne soit pas une feuille. Donc, pour chaque nœud u de l'arbre, l'ensemble $m(u)$ des cartes telles que $w(u)$ appartient à la S-signature de ces cartes est stocké.

Définition 19 (TS-signature d'une base de cartes)**.**
Soit $BC = \{C^1, \ldots, C^k\}$ une base de k cartes connexes et t le nombre maximum de brins d'une carte de BC. La TS-signature de BC est l'arbre $T_{SS}(BC)$ tel que

- *Pour chaque carte C de la base, chaque mot de $SS(C)$ correspond à un chemin de la racine à un nœud ;*
- *chaque nœud u sauf la racine à une étiquette $l(u)$ telle que $l(u)$ est un entier compris entre 0 et t ; nous notons $w(u)$ le mot obtenu en concaténant chaque étiquette $l(v)$ de chaque nœud v se trouvant sur le chemin allant de la racine au nœud u ;*
- *pour chaque nœud u, tous les fils de u ont une étiquette différente ;*
- *chaque nœud u sauf la racine contient l'ensemble $m(u)$ défini par*

$$m(u) = \{i \in \{1, \ldots, k\} \mid w(u) \in SS(C^i)\};$$

- $\sum_u |m(u)| = \sum_{i=1}^{k} |SS(C_i)|$.

Exemple 14. *La figure 3.5 donne un exemple de TS-signature d'une base composée de 3 cartes.*

Propriété 6. *La complexité spatiale de la TS-signature $T_{SS}(BC)$ de la base BC est $\mathcal{O}(k \cdot n \cdot t^2)$.*

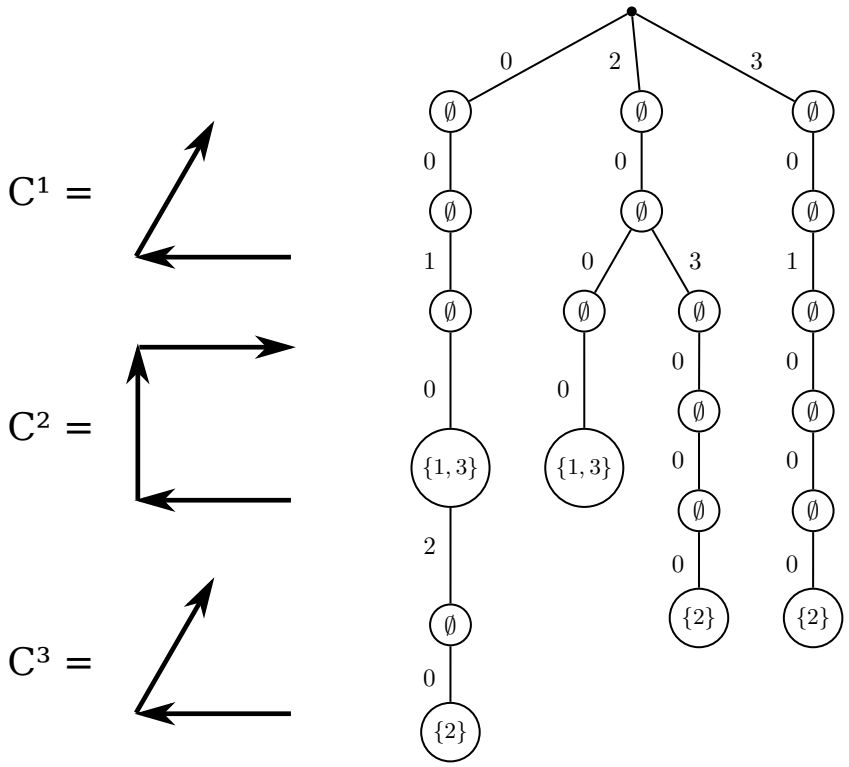


FIGURE 3.5 – À gauche, une base composée de 3 cartes. À droite, la TS-signature de la base. Pour chaque nœud u, l'étiquette $l(u)$ est sur la branche au dessus du nœud, tandis que $m(u)$ est à l'intérieur du nœud.

Démonstration. L'arbre contient au plus $k \cdot t$ feuilles (une pour chaque mot différent de la S-signature de chaque carte), et la taille d'une chemin entre la racine et une feuille est bornée par $n \cdot t$. Donc, le nombre de nœuds de l'arbre est borné par $k \cdot n \cdot t^2$. □

Propriété 7. *La complexité en temps pour calculer la TS-signature $T_{SS}(BC)$ d'une base BC est $\mathcal{O}(k \cdot n \cdot t^3)$.*

Démonstration. Il faut dans un premier temps calculer les S-signatures de toutes les cartes de la base, cela est fait en $\mathcal{O}(k \cdot n \cdot t^2)$. Puis, l'arbre peut être construit de manière incrémentale en ajoutant consécutivement chaque mot de chaque S-signature des cartes qui composent la base. Il y a au maximum $\mathcal{O}(k \cdot t)$ mots à ajouter à l'arbre et la longueur d'un mot est bornée par $\mathcal{O}(n \cdot t)$. Pour chaque étiquette x d'un mot, il faut vérifier si le nœud courant de l'arbre a un fils u tel que $l(u) = x$. C'est fait en temps linéaire par rapport au nombre de fils du nœud, car nous utilisons une liste pour mémoriser les fils d'un nœud, et un nœud a au plus $t + 1$ fils. □

Cette complexité est basée sur le pire des cas : un nœud ne peut pas avoir plus de $t+1$ fils dans la TS-signature. Cependant nous montrons expérimentale-

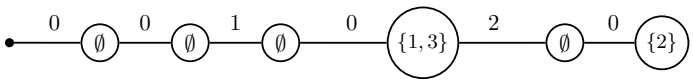


FIGURE 3.6 – TW signature de la base de 3 cartes de la figure 3.5

ment dans la Section 3.2.3 que sur une base de cartes générées aléatoirement, un nœud a très peu de fils.

Propriété 8. *Soient une carte $C = (B, \beta_1, \ldots, \beta_n)$ et une TS-signature $T_{SS}(BC)$ d'une base BC. La complexité pour rechercher toutes les cartes de BC qui sont isomorphes à C est $\mathcal{O}(n \cdot t^2)$, avec t, le nombre maximum de brins des cartes de BC.*

Démonstration. Pour rechercher toutes les cartes qui sont isomorphes à C, nous calculons $W_{BFL}(C, b)$ pour un brin $b \in B$. Cela est fait en $\mathcal{O}(n \cdot t)$. Ensuite, nous cherchons un nœud u de l'arbre tel que $w(u) = W_{BFL}(C, b)$: si un tel nœud existe, alors $m(u)$ donne l'ensemble des cartes de BC qui sont isomorphes à C ; sinon, aucune carte de BC n'est isomorphe à C. Pour chercher un nœud u tel que $w(u) = W_{BFL}(C, b)$, il faut chercher pour chaque w_i de $W_{BFL}(C, b)$ si le nœud courant de l'arbre a un fils étiqueté par w_i. Comme chaque nœud à au plus $t+1$ fils et que $W_{BFL}(C, b)$ a au plus $n \cdot t$ symboles, l'ensemble de la recherche est faite en $\mathcal{O}(n \cdot t^2)$. □

3.2.2 W-signature d'une base de cartes (TW-signature)

Une base de cartes peut aussi être représentée par un arbre contenant les W-signatures de ses cartes.

Définition 20 (TW-signature d'une base de cartes)**.**
Soit $BC = \{C^1, \ldots, C^k\}$ une base de k cartes connexes et t le nombre maximum de brins d'une carte de BC. La TW-signature de BC est l'arbre $T_{WS}(BC)$ tel que

- *Pour chaque carte C de la base, le mot $WS(C)$ correspond à un chemin de la racine à un nœud ;*
- *chaque nœud u sauf la racine à une étiquette $l(u)$ telle que $l(u)$ est un entier compris entre 0 et t ; nous notons $w(u)$ le mot obtenu en concaténant chaque étiquette $l(v)$ de chaque nœud v se trouvant sur le chemin allant de la racine au nœud u ;*
- *pour chaque nœud u, tous les fils de u ont une étiquette différente ;*
- *chaque nœud u sauf la racine contient l'ensemble $m(u)$ défini par*

$$m(u) = \{i \in \{1, \ldots, k\} \mid w(u) \in WS(C^i)\};$$

- $\sum_u |m(u)| = k$.

Exemple 15. *La TW-signature de la base de la figure 3.5 correspond à un arbre d'une seule branche donné figure 3.6.*

La figure 3.7 montre une base de 7 cartes représentée par une TW-signature.

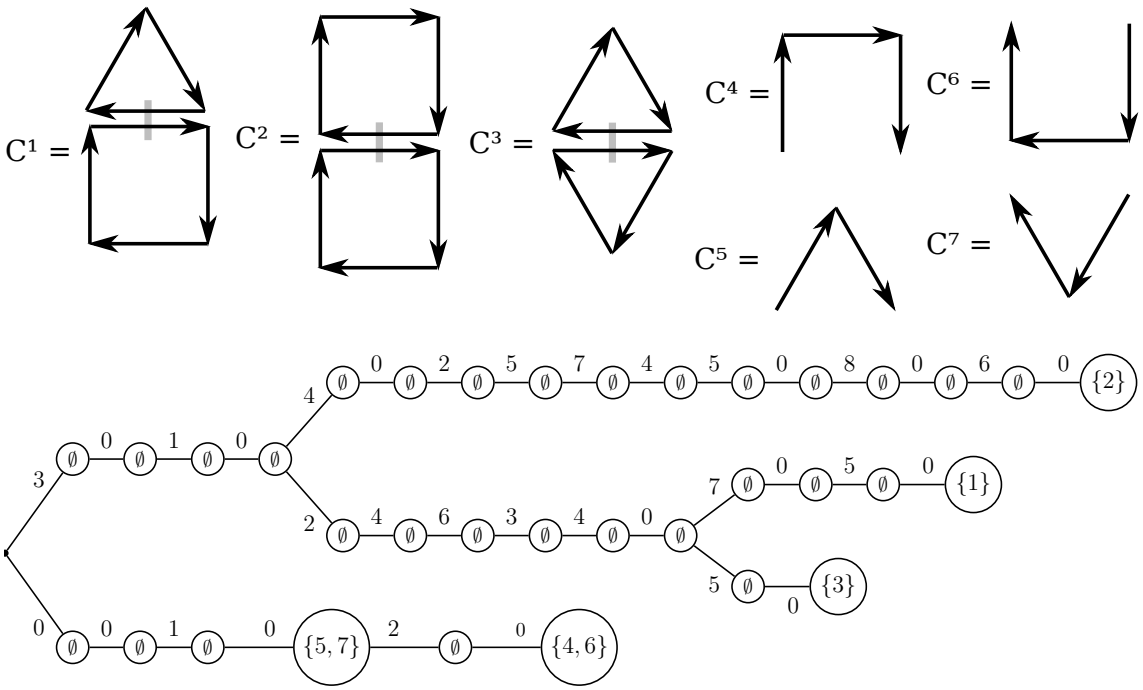


FIGURE 3.7 – Exemple de TW-signature représentant les cartes C^1,C^2,C^3,C^4,C^5,C^6, et C^7.

Propriété 9. *La complexité spatiale et de la TW-signature $T_{WS}(BC)$ d'une base BC est $\mathcal{O}(k \cdot n \cdot t)$, avec t, le nombre de brins de la plus grande carte de BC.*

Démonstration. L'arbre contient au plus k feuilles (une pour chaque carte différente de la base), et la longueur de chaque chemin de la racine à une feuille est bornée par $n \cdot t$. Donc, le nombre de nœuds de l'arbre est borné par $k \cdot n \cdot t$. □

Propriété 10. *La complexité en temps pour calculer la TW-signature $T_{SS}(BC)$ d'une base BC est $\mathcal{O}(k \cdot n \cdot t^2)$.*

Démonstration. Dans un premier temps, il faut calculer les W-signatures de chaque carte de la base, cette opération est faite en $\mathcal{O}(k \cdot n \cdot t^2)$. Puis, l'arbre peut être construit de manière incrémentale en ajoutant consécutivement chaque W-signature des cartes qui composent la base. Il y a au plus k mots à ajouter à l'arbre et la longueur d'un mot est bornée par $\mathcal{O}(n \cdot t^2)$. La longueur maximale d'un mot est $n \cdot t$. Pour chaque étiquette x d'un mot, il faut vérifier si le nœud courant de l'arbre a un fils u tel que $l(u) = x$. C'est fait en temps linéaire par rapport au nombre de fils du nœud, car nous utilisons une liste pour mémoriser les fils d'un nœud, et un nœud a au plus $t + 1$ fils. □

Propriété 11. *Soient une carte $C = (B, \beta_1, \ldots, \beta_n)$ et une TW-signature $T_{WS}(BC)$ d'une base BC. La complexité pour rechercher toutes les cartes de BC isomorphes à C est $\mathcal{O}(n \cdot t^2)$.*

Démonstration. Pour rechercher toutes les cartes qui sont isomorphes à C, nous calculons la W-signature de C. Cela est fait en $\mathcal{O}(n \cdot t^2)$. Ensuite, nous cherchons

un nœud u de l'arbre tel que $w(u) = WS(C)$ comme pour la TS-signature. La complexité totale est donc $\mathcal{O}(n \cdot t^2)$. □

Remarque 2. *Notons que les complexités théoriques de recherche d'un élément dans une base sont identiques que nous utilisions la TW-signature ou la TS-signature. Cependant en pratique, la complexité de la TS-signature n'est pas quadratique, mais plutôt linéaire (cf. section 3.5). Cette différence est due à la complexité de recherche d'un mot dans l'arbre, en effet, cette recherche dépend du nombre de fils que peut avoir un nœud dans l'arbre, or la borne que nous donnons est beaucoup plus grande que dans la pratique. De plus, le temps de recherche d'un mot dans l'arbre est négligeable et la complexité devient identique au calcul d'isomorphisme. Dans la section 3.2.3 nous affinons un peu cette borne en faisant une étude du pire des cas.*

3.2.3 Nombre de fils d'un nœud d'un arbre de signatures

La complexité du calcul de la signature d'une base de cartes et la complexité de recherche d'une carte dans l'arbre dépendent du nombre de fils des nœuds de l'arbre. En effet, le fils d'un nœud est stocké dans une liste, donc la recherche d'un fils ayant une étiquette donnée est faite en temps linéaire par rapport au nombre de fils. Une première borne supérieure du nombre de fils d'un nœud est donnée par le nombre d'étiquettes différentes (c.-à-d. , $t+1$, avec t, le nombre de brins de la plus grande carte), car chaque fils doit avoir une étiquette différente.

Dans cette section, nous montrons que cette borne peut être affinée et qu'il est possible de construire une base de cartes telle que le nombre de fils d'un nœud de la TW-signature est du même ordre de grandeur que cette borne supérieure. Ensuite, nous montrons que pour une base de cartes générées aléatoirement le nombre de fils de chaque nœud est bien inférieur à cette borne.

Étude du pire des cas Pour une base de k cartes de t brins, la TS-signature a $\mathcal{O}(t \cdot k)$ feuilles et la TW-signature a $\mathcal{O}(k)$ feuilles ; dans les deux cas, la longueur des branches de la racine à une feuille est $\mathcal{O}(n \cdot t)$.

Soit u un nœud étiqueté par le brin $l(u)$ et i la profondeur de u dans l'arbre (c.-à-d. , la longueur du chemin entre la racine et u). Si le nombre de fils de u est borné par le nombre d'étiquettes différentes possibles (c.-à-d. , $t+1$), il est aussi borné par le nombre de brins qui ont été découverts quand le brin $l(u)$ a été retiré de la file F (ligne 9 de l'algorithme 5). Ce nombre est égal à $i \cdot n$. En effet, chaque brin $l(v)$ associé au nœud v entre la racine et le nœud u permet de découvrir n nouveaux brins. Donc le nombre de fils de u est borné par $min(t, i \cdot n)$.

Cette borne peut être affinée en prenant en compte le fait que pour chaque dimension $j \in \{1, \ldots, n\}$, β_j est une permutation. De ce fait, chaque brin apparait au plus n fois dans un mot. Plus précisément, si un nœud u est de profondeur i, alors pour chaque dimension, le nombre de brins différents utilisés dans $w(u)$ est i/n, donc le nombre de fils potentiel de u doit être diminué de i/n. Donc, une borne plus précise du nombre de fils d'un nœud de profondeur i est

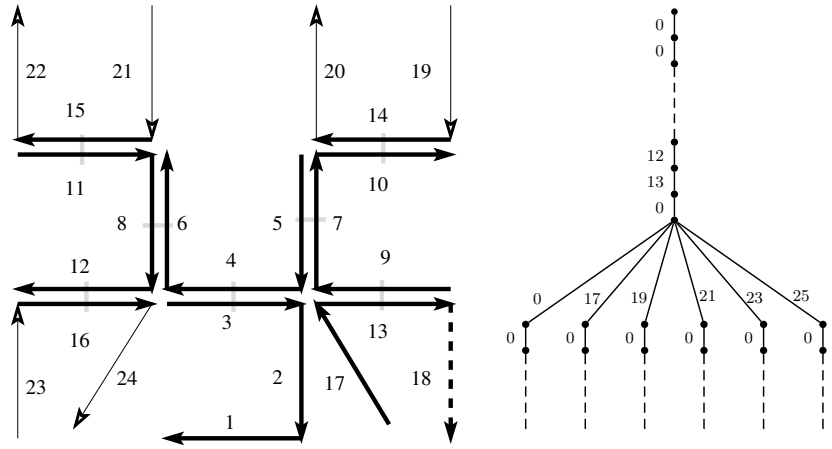


FIGURE 3.8 – Exemple où le nombre de nœuds dans une TW-signature est grand. À gauche : une carte C. À droite : une partie de la TW-signature qui contient six cartes obtenues à partir de C en 1-cousant le brin 18 à ϵ ou aux brins 17, 19, 21, ou 23 ou à un nouveau brin.

$B(i) = min(t, i \cdot n) - i/n$. C'est la borne théorique la plus proche de la réalité que nous ayons pu trouver.

Nous essayons maintenant de créer une base de cartes telle que le nombre de fils d'un nœud de profondeur i dans l'arbre de signature soit le plus grand possible. Pour cela, le nombre de brins visités doit croitre plus vite que le nombre de brins utilisés. Donc, autant que possible, un brin doit être cousu avec le minimum de brins visités et le maximum de nouveaux brins. Pour une profondeur donnée, il est possible de créer autant de branches qu'il y a de brins libres pour une dimension donnée qui sont déjà découverts, mais pas encore utilisés (c.-à-d. , les brins qui ont une étiquette supérieure au brin courant). La figure 3.8 montre un exemple de base de cartes où le nombre de fils d'un nœud est égal à six. Dans cet exemple, nous pouvons 1-coudre le brin 18 à ϵ ou aux brins 17, 19, 21, ou 23 ou encore à un nouveau brin. Chacun de ces cas crée une nouvelle branche à la profondeur $(18 * 2)$. Par contre, nous ne pouvons pas 1-coudre le brin 18 au brin 3, 5, 9, ou 11, car cela changerait l'étiquetage et cela créerait une nouvelle branche à une profondeur inférieure à $(18 * 2)$. Ensuite, pour chacune des huit branches, nous pouvons sur le même principe 2-coudre le brin 18 à ϵ ou au brin 19, 20, 21, 22, 23, ou 24 ou à un nouveau brin. De cette façon, nous pouvons créer un grand nombre de branches où chaque nœud aura un maximum de fils.

Nombre de fils dans une base de cartes aléatoires S'il est possible de construire une base de cartes pour laquelle le nombre de fils de chaque nœud se rapproche de la borne supérieure, il est intéressant d'étudier l'évolution de ce nombre en pratique pour des bases de cartes générées aléatoirement.

Pour générer aléatoirement des n-cartes de t brins, nous partons d'un seul brin, puis, tant que le nombre de brins t n'est pas atteint : nous choisissons une

n	k	TW-signature						TS-signature					
		$i \in [1..10n]$		$i \in]10n..100n]$		$i \in]100n..500n]$		$i \in [1..10n]$		$i \in]10n..100n]$		$i \in]100n..500n]$	
		max	avg	max	avg	max	avg	max	avg	max	avg	max	avg
2	10	3	1.7	1	1	1	1	4	3.3	3	1.11	1	1
	100	4	2.4	2	1.04	1	1	5	3.45	3	1.22	1	1
	1000	4	2.55	3	1.12	1	1	5	3.85	4	1.35	-	-
	10000	4	2.65	3	1.25	1	1	5	3.95	-	-	-	-
3	10	3	1.57	1	1	1	1	3	2.9	2	1.03	1	1
	100	3	2.2	2	1	1	1	4	3.03	3	1.13	1	1
	1000	3	2.6	2	1.06	1	1	4	3.1	3	1.25	-	-
	10000	3	2.63	3	1.15	1	1	4	3.57	-	-	-	-
4	10	3	1.53	1	1	1	1	3	2.78	3	1.13	1	1
	100	3	1.95	2	1.02	1	1	3	2.78	3	1.28	1	1
	1000	3	2.38	3	1.12	1	1	3	2.78	3	1.52	-	-
	10000	3	2.5	3	1.29	1	1	3	2.8	-	-	-	-
8	10	3	1.28	1	1	1	1	3	2.15	2	1.22	1	1
	100	3	1.55	2	1	1	1	3	2.33	3	1.6	2	1.02
	1000	3	2.13	3	1.08	1	1	3	1.95	2	1.98	-	-
	10000	3	2.29	3	1.3	1	1	-	-	-	-	-	-

TABLE 3.1 – Nombre maximum et moyen de fils d'un nœud en fonction de sa profondeur i dans l'arbre, de la dimension n des cartes et du nombre de cartes k différentes dans la base. Les cartes sont générées aléatoirement avec pour paramètre $t = 500$. Pour la TS-signature, le symbole "-" correspond à une base qui ne peut pas être représentée avec 16 Go de RAM.

dimension i au hasard (entre 0 et n). Puis nous choisissons au hasard un brin b i-libre. S'il n'y a pas de brin i-libre, nous choisissons une autre dimension, sinon nous i-cousons b à un nouveau brin et nous propageons les contraintes de validité d'une carte. Cette étape peut créer un grand nombre de brins, donc pour un paramètre t choisit, les cartes générées auront au minimum t brins et au maximum $2 \cdot t$ brins. Il y a un cas particulier pour les 2-cartes car il n'y a pas de contraintes de validité, donc les deux cartes auront exactement t brins.

Le tableau 3.1 donne le nombre maximum et moyen de fils d'un nœud en fonction de :

1. la profondeur i du nœud dans l'arbre ;
2. la dimension n des cartes ;
3. le nombre k de cartes différentes dans la base.

Pour chaque carte, le nombre de brins t est fixé à 500. Ce tableau montre que le nombre de fils d'un nœud est bien plus petit que la borne du pire des cas : dans toutes les bases générées (qui contiennent jusqu'à 10000 cartes de 500 brins), le nombre maximum de fils d'un nœud est de 4 pour la TW-signature et de 5 pour la TS-signature. Notons que le degré dépend de la profondeur du nœud dans l'arbre : quand i est supérieur à $100n$, les nœuds ont presque toujours un seul fils. Le degré dépend aussi du nombre de cartes dans la base ainsi que la dimension des cartes : quand k augmente et/ou n diminue, le nombre de fils augmente un peu. Enfin, le degré est légèrement supérieur dans les TS-signatures que dans les TW-signatures. Cela vient du fait qu'une TS-signature peut contenir jusqu'à t fois plus de branches qu'une TW-signature

Algorithme 10 : $W_v(C, l)$

Entrées : Une carte valuée $C = (B, V, v, \beta_1, \ldots, \beta_n)$ et un étiquetage l.
Sortie : le mot correspondant à l'étiquetage l de la carte C.

```
mot ← ∅
pour j ← 0 à |B| faire
    mot ← mot + v(l^{-1}(j))
    pour i ← 0 à n faire
        mot ← mot + l(β_i(l^{-1}(j)))
retourner mot
```

pour représenter une même base de cartes.

3.3 Signatures de cartes valuées

Il est possible d'associer des informations supplémentaires aux cartes combinatoires en mettant une valeur sur chaque brin, la définition d'une carte valuée est donnée définition 21. Les seules différences avec les n-cartes sont l'ensemble V des valeurs différentes, et la fonction v qui associe une valeur $i \in E$ à chaque brin $b \in B$.

Définition 21 (Carte combinatoire valuée)**.**
Une n-carte valuée est un $(n+3)$-uplet $C = (B, V, v, \beta_1, \ldots, \beta_n)$ tel que

1. B *est un ensemble fini de brins ;*
2. V *est l'ensemble des valeurs possibles ;*
3. v *est une fonction de B sur V ;*
4. β_1 *est une* permutation *partielle sur* B *;*
5. $\forall\, 2 \leq i \leq n$, β_i *est une* involution *partielle sur* B *;*
6. $\forall\, 1 \leq i \leq n-2, \forall\, i+2 \leq j \leq n$, $\beta_i \circ \beta_j$ *est une* involution *partielle sur* B.

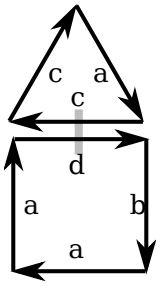
L'étiquetage d'une n-carte valuée est identique à l'étiquetage d'une n-carte, cependant il y a une différence lors de la construction d'un mot à partir d'un étiquetage. En effet, il faut prendre en compte dans ce mot, les valeurs de chaque brin. L'algorithme 10 donne le pseudo-code de la construction d'un mot d'une carte valuée.

Toutes les signatures définies précédemment sont transposables au cas des cartes valuées, il suffit d'utiliser ce nouveau mot W_v à la place de l'ancien. Notons que les complexités données restent inchangées.

La figure 3.9 donne un exemple de carte valuée et de ses signatures.

3.4 Signatures de cartes non connexes

Les signatures introduites précédemment sont définies pour des cartes connexes. Si une carte n'est pas connexe, alors le processus d'étiquetage décrit dans

$C =$ 

$$SS(C) = \{ < a,3,0,b,1,0,a,4,0,d,2,5,c,7,4,a,5,0,c,6,0 >,$$
$$< a,3,0,a,1,0,d,4,5,b,2,0,c,7,3,a,5,0,c,6,0 >,$$
$$< c,3,0,c,1,4,a,2,0,d,6,2,a,4,0,b,7,0,a,5,0 >,$$
$$< c,3,4,a,1,0,c,2,0,d,6,1,a,4,0,b,7,0,a,5,0 >,$$
$$< d,3,4,a,1,0,b,5,0,c,7,1,a,2,0,a,4,0,c,6,0 >,$$
$$< a,3,0,c,1,0,c,2,4,d,6,3,a,4,0,b,7,0,a,5,0 >,$$
$$< b,3,0,d,1,5,a,4,0,a,2,0,c,7,2,a,5,0,c,6,0 > \}$$

$$WS(C) = < a,3,0,a,1,0,d,4,5,b,2,0,c,7,3,a,5,0,c,6,0 >$$

FIGURE 3.9 – Signatures d'une carte valuée. Chaque brin de la carte C a une valeur parmi $\{a,b,c,d\}$. $SS(C)$ est la S-signature de la carte C et $WS(C)$ est la W-signature de la carte C.

l'algorithme 5 ne va pas étiqueter tous les brins, car certains brins ne sont pas atteignables à partir du brin de départ.

Cependant, une carte non connexe peut être décomposée en un ensemble disjoint de cartes connexes en $\mathcal{O}(n \cdot |B|)$ en parcourant successivement les composantes connexes jusqu'à avoir découvert tous les brins de la carte. La signature d'une carte non connexe est alors calculée à partir des signatures de ses composantes connexes de manière similaire à la construction de signatures d'une base de cartes.

Plus précisément, étant donnée une carte non connexe C telle que C est composée de k composantes connexes $C^1,\dots,C^k$. Nous définissons la base $BC_C = \{C^1,\dots,C^k\}$. La S-signature de C est définie par la TS-signature $T_{SS}(BC_C)$ et sa W-signature est définie par la TW-signature $T_{WS}(BC_C)$.

Propriété 12. *Soient une carte $C' = (B', \beta'_1,\dots,\beta'_n)$ et une TS-signature $T_{SS}(BC_C)$, nous pouvons déterminer si C et C' sont isomorphes en $\mathcal{O}(n \cdot |B'|)$.*

Démonstration. En effet, pour déterminer l'isomorphisme, nous procédons de la façon suivante :

1. Nous décomposons C' en un ensemble disjoint de cartes connexes. Soit k' le nombre de composantes connexes. Si $k' \neq k$ alors les deux cartes ne sont pas isomorphes.
   Cette étape est faite en $\mathcal{O}(n \cdot |B'|)$.
2. Si $k' = k$, notons $C'^1,\dots,C'^k$ les différentes composantes connexes de C'. Pour chaque composante connexe C'^i, nous construisons un mot en partant de n'importe quel brin b de C'^i, et nous vérifions si $W_{BFL}(C'^i, b)$ correspond à un chemin dans l'arbre $T_{SS}(BC_C)$ de la racine à un nœud u. Si ce n'est pas le cas, alors les deux cartes ne sont pas isomorphes.
   Cette étape est faite en $\mathcal{O}(n \cdot |B'^i|)$ pour chaque composante connexe i (voir Propriété 8).
3. Il faut vérifier pour chaque composante connexe C'^i, que le nombre de composantes connexes de C' isomorphes à C'^i est égal au nombre de composantes connexes de C isomorphes à C'^i.

Cette étape est faite en $\mathcal{O}(k)$ en utilisant des compteurs. En effet, en associant un compteur c qui prend pour valeur $|m(u)|$, pour tout nœud $u \in T_{SS}(BC_C)$ tel que $m(u) \neq \emptyset$ et pour tout nœud $v \in T_{SS}(BC_C)$ si $m(u) = m(v)$ alors $m(v)$ pointe aussi vers c. À chaque fois qu'un mot $W_{BFL}(C'^i, b)$ termine sur un nœud u, le compteur associé au nœud u est décrémenté de 1, une fois que toutes les composantes connexes ont étés traitées, il suffit de vérifier que tous les compteurs sont nuls.

□

Propriété 13. *Soient une carte $C = (B, \beta_1, \ldots, \beta_n)$ et une TW-signature $T_{WS}(BC_C)$, nous pouvons décider de l'isomorphisme entre C et C' en $\mathcal{O}(n \cdot |B'|^2)$.*

Démonstration. Pour déterminer l'isomorphisme en utilisant la TW-signature, nous procédons de la même façon qu'avec la TS-signature. La seule différence est dans l'étape 2 : il faut d'abord calculer la W-signature de chaque composante connexe C'^i avant de rechercher un nœud u dans l'arbre tel que $w(u)$ est égal à la W-signature. Le calcul de la W-signature d'une composante connexe C'^i est fait en $\mathcal{O}(n \cdot |B'^i|^2)$ qui est également la complexité de l'étape 2 et donc la complexité totale est $\mathcal{O}(n \cdot |B'|^2)$. □

Enfin, il est aussi possible de définir des TS et TW-signatures de bases de cartes non connexes de la même façon : il faut juste stocker pour chaque carte l'ensemble des composantes connexes qui la compose (voir figure 3.10). La TW-signature (resp. TS-signature) d'une base de cartes non connexes contient donc la TW-signature (resp. TS-signature) de l'ensemble des composantes connexes des cartes de la base, plus une liste des composantes connexes qui compose chaque carte non connexe. Pour tester si une carte fait partie de la base, il faut calculer la signature de chacune des composantes connexes de cette carte. Si une des composantes n'est pas dans la signature de la base, alors cette carte ne fait pas parti de la base, sinon une fois que toutes les signatures des composantes connexes sont calculée, il suffit de vérifier si la liste des cartes connexes qui composent la carte non connexe est présente dans la base. Cette dernière étape ce fait en temps linéaire par rapport au nombre de cartes dans la base.

3.5 Évaluation

Dans cette section, nous montrons quelques résultats expérimentaux qui illustrent l'intérêt d'utiliser les signatures de cartes. Nous comparons les deux types de signatures sur des bases de cartes synthétiques et sur des cartes extraites à partir d'images. Tous les résultats ont été obtenus sur une machine Intel Xeon E5520 2.26 GHz avec 16Go de RAM.

Utilisation de la signature de cartes pour la recherche d'images Une carte peut être extraite à partir d'une image segmentée en utilisant l'algorithme décrit dans [11]. Cette méthode qui sera expliquée dans le chapitre 5 est résistante à

1 :	(A,1)	(B,2)	(D,1)	
2 :	**(A,2)**	**(B,1)**	**(C,1)**	
3 :	(A,2)	(C,1)		
4 :	(A,3)	(C,3)	(D,2)	(E,1)
5 :	(B,2)	(C,1)	(E,1)	
6 :	(B,3)			
7 :	(B,5)	(C,2)	(E,1)	
8 :	(C,1)	(D,3)		
9 :	(D,2)			
10 :	(D,3)	(E,1)		

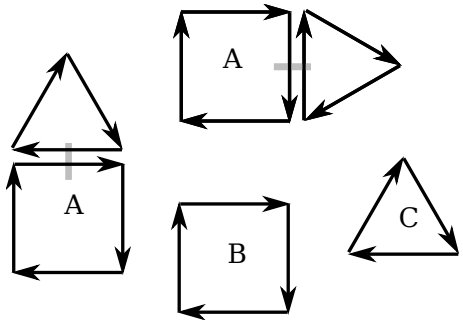


FIGURE 3.10 – Exemple d'une base de cartes non connexes. Dans cet exemple, il y a 10 cartes non connexes et chacune de ces 10 cartes non connexes est une composition de cartes connexes. Il y a 5 cartes connexes différentes, nommées A, B, C, D et E. La cartes non connexes numéro 2 (figure de droite) est composée de 2 fois le motif A, 1 fois le motif B et 1 fois le motif C, chacun de ces motifs n'étant cousu à aucun autre.

plusieurs déformations telles que les rotations (90˚, 180˚ou 270˚) ou les changements de résolution de l'image. La signature de carte peut donc être utilisée pour identifier une image, et est résistante à ces transformations. Le tableau 3.2 donne pour 5 images, le nombre de brins et de faces de la carte extraite de ces images. Il compare le temps CPU utilisé pour calculer les S-signatures et les W-signatures de ces cartes. Nous remarquons que les signatures sont calculées très rapidement pour des cartes ayant jusqu'à 6000 brins.

Image					
Brins	3410	6060	1728	4224	1590
Faces	590	1044	295	716	275
$SS(C)$	$0.83s$	$2.21s$	$0.26s$	$1.14s$	$0.26s$
$WS(C)$	$0.26s$	$0.53s$	$0.15s$	$0.32s$	$0.16s$

TABLE 3.2 – Des images aux cartes : la première ligne montre les images, les deux suivantes donnent le nombre de brins et de faces de la carte correspondant à l'image ; les deux dernières lignes donnent le temps CPU en secondes pour construire la S-signature et la W-signature de ces cartes.

Propriétés de construction des signatures Pour comparer les propriétés des deux signatures, nous avons effectué des expériences avec des 2-cartes de différentes tailles (1000, 2000, 4000 et 8000 brins) générées aléatoirement de la même façon que dans la section 3.2.3. Le tableau 3.3 compare le temps de construction des deux signatures ainsi que le nombre de brins visités durant la construction.

Pour construire la S-signature, nous devons parcourir complètement la carte pour chacun des brins, donc le nombre total de brins visités est toujours égal à $|B|^2$ et la complexité ne dépend pas du brin de départ. Pour construire la W-signature, nous devons aussi parcourir la carte pour chacun des brins, mais ce parcours peut être stoppé dès que le mot en construction est plus grand que le plus petit mot actuel. Donc, si dans le pire des cas la complexité est quadratique, comme pour la S-signature, le tableau 3.3 montre qu'en pratique la complexité est sous-quadratique. En effet, le nombre moyen de brins visités par chaque parcours varie de 19.48 pour des cartes de 1000 brins, à 26.91 pour des cartes de 8000 brins. Le tableau 3.3 montre aussi que même si le nombre de brins visités dépend de l'ordre dans lequel les brins de départ sont choisis, l'écart-type reste assez bas.

	S-signature			W-signature		
	Temps (en s)	$\frac{\lvert Brins\ visités\rvert}{\lvert B\rvert}$		Temps (en s)	$\frac{\lvert Brins\ visités\rvert}{\lvert B\rvert}$	
$\lvert B\rvert$	moy	moy	(écart)	moy	moy	(écart)
1000	0.054	1000	(0)	0.047	19.48	(3.24)
2000	0.228	2000	(0)	0.084	19.27	(3.71)
4000	1.056	4000	(0)	0.262	23.78	(5.31)
8000	4.088	8000	(0)	0.352	26.91	(4.88)

TABLE 3.3 – Comparaison de la complexité en temps pour construire les S-signatures et les W-signatures d'une carte. Chaque ligne donne le nombre de brins $|B|$ d'une carte et pour chacune des signatures, le temps CPU (en secondes) et le ratio entre le nombre de brins visités et $|B|$. La moyenne et l'écart-type sont donnés pour 100 exécutions pour différents brins de départ.

Utilisation des signatures pour décider de l'isomorphisme Nous comparons maintenant les deux signatures avec une méthode d'appariement direct pour vérifier l'isomorphisme entre une carte C dont nous possédons la signature, et une nouvelle carte C'.

La complexité de l'algorithme d'appariement direct décrit dans [12] est en $\mathcal{O}(n \cdot |B|^2)$.

Avec la S-signature $SS(C)$, la complexité dans le pire des cas est en $\mathcal{O}(n \cdot |B|)$, car il suffit de faire un seul parcours de la carte à partir de n'importe quel brin de départ. Le tableau 3.4 montre que, lorsque C et C' sont isomorphes (quand le pourcentage de brins différents est de 0%), l'algorithme visite exactement une fois chaque brin de la carte C'. Si C et C' ne sont pas isomorphes, nous pouvons stopper la construction du mot dès qu'il n'y a plus de branche correspondante dans l'arbre lexicographique. Nous pouvons donc décider que C et C' ne sont pas isomorphes en temps sous-linéaire. Le tableau 3.4 montre que plus C et C' sont différents, plus le nombre de brins visités est petit.

Avec la W-signature $WS(C)$, la complexité dans le pire cas est $\mathcal{O}(n \cdot |B|^2)$, car il faut faire un parcours de la carte C' à partir de chacun de ses brins. Toutefois, le

parcours peut être stoppé dès que le mot en cours de construction est différent de la signature de C. Le tableau 3.4 montre que plus C et C' sont différents, plus le nombre de brins visités est petit. En pratique, chaque brin est visité entre 2 et 4 fois. Notons que ce ratio ne varie presque pas quand la taille des cartes augmente.

Le tableau 3.4 montre que l'algorithme d'appariement direct et la méthode utilisant la W-signature donnent des résultats très proches alors que la méthode utilisant la S-signature permet de décider de l'isomorphisme en un temps bien plus rapide.

		Isomorphisme direct			W-signature			S-signature		
$\|B\|$		Temps (en s)	$\frac{\|Brins\ visités\|}{\|B\|}$		Temps (en s)	$\frac{\|Brins\ visités\|}{\|B\|}$		Temps (en s)	$\frac{\|Brins\ visités\|}{\|B\|}$	
		moy	moy	(écart)	moy	moy	(écart)	moy	moy	(écart)
1000	0%	0.030	2.09	(0.72)	0.035	2.13	(0.64)	0.000099	1.000	(0.000)
	1%	0.058	3.58	(1.51)	0.060	3.71	(1.48)	0.000091	0.298	(0.214)
	10%	0.058	3.42	(1.53)	0.059	3.41	(1.34)	0.000086	0.026	(0.021)
	50%	0.056	1.64	(1.08)	0.056	1.88	(1.19)	0.000072	0.015	(0.006)
	99%	0.055	1.55	(0.93)	0.050	1.59	(0.90)	0.000068	0.011	(0.004)
2000	0%	0.076	2.55	(1.62)	0.084	2.59	(1.47)	0.000215	1.000	(0.000)
	1%	0.102	3.22	(1.64)	0.095	3.08	(1.79)	0.000161	0.069	(0.081)
	10%	0.084	3.01	(1.72)	0.076	2.92	(1.76)	0.000130	0.019	(0.032)
	50%	0.067	1.56	(1.43)	0.073	1.77	(1.40)	0.000098	0.006	(0.005)
	99%	0.066	1.38	(0.93)	0.069	1.38	(0.83)	0.000097	0.006	(0.003)
4000	0%	0.212	2.31	(1.41)	0.262	2.46	(1.30)	0.000341	1.000	(0.000)
	1%	0.451	3.45	(1.58)	0.434	3.09	(1.89)	0.000292	0.015	(0.037)
	10%	0.331	3.02	(1.43)	0.329	2.57	(1.81)	0.000222	0.005	(0.005)
	50%	0.305	2.54	(1.20)	0.286	2.03	(1.41)	0.000178	0.005	(0.006)
	99%	0.265	1.28	(1.09)	0.273	1.43	(0.85)	0.000164	0.005	(0.003)
8000	0%	0.450	2.62	(0.98)	0.352	2.23	(1.04)	0.000697	1.000	(0.000)
	1%	1.397	2.99	(1.62)	1.451	3.11	(1.86)	0.000556	0.032	(0.178)
	10%	1.263	2.89	(1.45)	1.343	3.05	(1.81)	0.000439	0.003	(0.009)
	50%	1.101	2.46	1.31	1.042	2.44	(1.25)	0.000296	0.002	(0.003)
	99%	0.910	1.48	(1.14)	0.993	1.53	(1.02)	0.000353	0.003	(0.003)

TABLE 3.4 – Comparaison entre la W-signature, la S-signature et un algorithme d'appariement direct pour décider de l'isomorphisme entre une carte C et une carte C'. Nous considérons que la signature de la carte C est connue. C et C' ont le même nombre de brins, et sont créées aléatoirement à partir d'un pourcentage de brins communs. Quand la différence est de 0%, C et C' sont isomorphes. Chaque ligne montre : le nombre de brins de C, le pourcentage de brins différents entre C et C' et pour chaque méthode, le temps en secondes et le ratio entre le nombre de brins visités et le nombre de brins de C. La moyenne et l'écart-type sont donnés pour 100 exécutions en mélangeant de manière aléatoire l'ordre des brins de C'.

Utilisation des signatures pour rechercher une carte dans une base Pour comparer les propriétés des TS et TW-signatures pour représenter une base de cartes nous avons généré sept bases de 2-cartes. Chaque base contient 100 cartes non isomorphes deux à deux et toutes les cartes d'une même base ont le même nombre de brins : respectivement 100, 200, 500, 1000, 2000, 4000 et 8000 brins. Le

tableau 3.5 montre l'intérêt de fusionner les signatures des différentes cartes en un seul arbre (colonne *Arbre*) par rapport au stockage indépendant de chaque signature (colonne *Indpt*). La fusion des signatures permet aussi d'économiser de la place en mémoire, mais le gain est minime (moins de 10%). L'intérêt principal est le gain de vitesse dans le processus de recherche d'une carte dans une base : le temps CPU est de 50 à 100 fois inférieur quand nous utilisons les arbres de signatures. En effet, le temps pour rechercher une carte dans un arbre de signatures dépend de la taille de la carte, mais pas du nombre de cartes dans la base. En comparaison, si chaque signature est stockée indépendamment, pour rechercher une carte dans la base, il faut comparer la carte avec chaque signature séparément, donc plus il y a de cartes dans la base, plus il y a de comparaisons à faire et plus la recherche est longue.

	Nombre de nœuds				Temps (en s)			
	TS-signature		TW-signature		TS-signature		TW-signature	
$\|B\|$	Indpt	Arbre	Indpt	Arbre	Indpt	Arbre	Indpt	Arbre
100	1 810 207	1 666 663	20 000	16 832	0.000207	0.000004	0.032033	0.000097
200	7 558 050	7 221 234	40 000	36 551	0.000244	0.000007	0.020617	0.000225
500	48 675 443	47 735 334	100 000	95 929	0.000312	0.000015	0.751917	0.000956
1 000	197 031 371	195 086 897	200 000	195 771	0.000451	0.000030	0.303131	0.003803
2 000	–	–	400 000	394 745	–	–	1.337401	0.014879
4 000	–	–	800 000	794 896	–	–	4.608052	0.053965
8 000	–	–	1 600 000	1 594 375	–	–	15.93316	0.216157

TABLE 3.5 – Comparaison entre la TW-signature et la TS-signature pour rechercher un élément dans une base de cartes. Pour chaque signature, nous comparons le comportement de la recherche d'une carte dans la base si nous stockons la signature de chaque carte séparément (colonne *Indpt*) ou si nous regroupons toute la base en un seul arbre (colonne *Arbre*). Chaque base contient 100 cartes différentes générées aléatoirement. Chaque ligne montre : le nombre de brins de chaque carte de la base, et pour chaque signature, la mémoire utilisée pour stocker la base (en nombre de nœuds) et le temps en secondes pour rechercher un élément dans la base. Le temps de recherche est une moyenne sur 200 cartes recherchées dont 100 appartiennent à la base et les 100 autres n'y appartiennent pas.

Le tableau 3.5 montre que la recherche d'une carte dans une TS-signature est bien plus rapide que la recherche dans une TW-signature. Cependant, la complexité spatiale de la TS-signature est aussi bien plus grande : avec 16 Go de RAM, nous ne pouvons pas représenter la TS-signature d'une base de 100 cartes de 2000 brins et plus, alors que cela ne pose pas de problème avec la TW-signature. Donc, la TS-signature doit être utilisée seulement pour des petites bases de cartes et lorsque le besoin se pose de temps CPU très rapide. Dans les autres cas, nous préférerons la TW-signature. Notons que le temps de recherche d'une carte dans une TS-signature est bien plus court quand la carte recherchée n'est pas présente dans la base. En effet, nous vérifions que le mot que nous construisons en parcourant la carte recherchée est présent dans l'arbre. Au cours de la construction du mot, si aucune branche ne correspond, il

est possible de stopper la recherche et conclure que la carte n'est pas présente dans la base. Avec la TW-signature, les temps sont sensiblement identiques que la carte recherchée appartienne ou non à la base. En effet, dans les deux cas, il faut rechercher le plus petit mot (en appelant W_{BFL} pour chaque brin de la carte recherchée) avant de chercher ce mot dans l'arbre.

Enfin, le tableau 3.6 permet d'étudier les propriétés de la TW-signature quand le nombre k de cartes dans la base augmente (nous ne faisons pas cette étude pour la TS-signature car la capacité mémoire est vite saturée). Cela montre que la taille de l'arbre augmente linéairement en fonction du nombre de cartes et que le temps de recherche d'une carte reste constant et ne dépend pas du nombre de cartes dans la base. Par exemple, la TW-signature permet de trouver une carte de 1000 brins dans une base de 100000 cartes de 1000 brins en 0.003 seconde.

Nombre de Cartes	Nombre de nœuds	Temps (en s)
100	195 771	0.003046
1 000	1 933 461	0.003053
10 000	19 311 582	0.003053
100 000	192 171 529	0.003046

TABLE 3.6 – Influence du nombre de cartes pour la recherche au sein d'une base représentée par une TW-signature. Chaque ligne montre : le nombre de cartes de 1000 brins qui composent la base, l'utilisation mémoire (nombre de nœuds de l'arbre) et le temps de recherche d'un élément dans la base (temps moyen sur 200 recherches).

Conclusion

Dans ce chapitre, nous avons défini deux signatures de cartes combinatoires qui correspondent à des représentations canoniques de n-cartes. La complexité en mémoire de la première signature est quadratique, mais permet de décider de l'isomorphisme en temps linéaire. La complexité en mémoire de la seconde signature est linéaire, et permet de décider de l'isomorphisme en temps quadratique dans le pire cas, mais linéaire en pratique. Nous avons étendu ces deux signatures pour représenter une base de cartes sous forme d'arbre afin d'y rechercher efficacement un élément. En pratique, nous avons observé que la complexité de recherche d'un élément ne dépend pas du nombre de cartes dans la base. La TS-signature est très rapide mais est limité par les ressources mémoire nécessaire. La TW-signature résout ce problème de mémoire, mais la recherche d'un élément dans la base est un peu plus longue.

Dans le chapitre suivant, nous utilisons la W-signature et la TW-signature dans les deux algorithmes d'extraction de sous-cartes fréquentes que nous avons développés, ceci afin de détecter rapidement les isomorphismes.

Chapitre 4

Extraction de motifs fréquents dans des bases de cartes combinatoires

Sommaire

4.1 Définition du problème 52
4.2 Méthode en largeur 54
4.2.1 Calcul des sous-cartes fréquentes composées d'une ou de deux faces 55
4.2.2 Génération des candidats 57
4.2.3 Fusion 59
4.2.4 Comptage de la fréquence 59
4.2.5 Complexités des différentes étapes de l'algorithme *MapAPriori* 65
4.3 Méthode en profondeur 67
4.3.1 Principe 67
4.3.2 Algorithme 67
4.3.3 Complexité 73
4.3.4 Extension aux cartes nD 74
4.4 Expérimentations 74
4.4.1 Comparaison des performances des algorithmes *mSpan* et *MapApriori*. 75
4.4.2 Comparaison entre les cartes et les graphes (mSpan Vs gSpan). 79

L'OBJECTIF de ce chapitre est triple, tout d'abord, dans la section 4.1 nous posons formellement le problème d'extraction de sous-cartes fréquentes

dans une base de cartes combinatoires. Ensuite, nous proposons deux algorithmes basés sur deux types de méthodes : les méthodes en largeur pour le premier, décrit dans la section 4.2, et les méthodes en profondeur pour le second, décrit dans la section 4.3. Les méthodes en profondeur sont généralement plus efficaces, car elles évitent l'étape de génération des candidats qui peut être coûteuse, et simplifie le calcul de la fréquence des motifs. Cela permet un gain très important, notamment dans la recherche de sous-graphes fréquents, car le calcul d'isomorphisme de sous-graphes est un problème NP-complet. Cependant, comme nous l'avons vu dans la section 2.3, le calcul d'isomorphisme de sous-cartes peut se faire en temps polynomial, donc nous pouvons penser que l'apport des méthodes en profondeur sera moins important que pour les graphes. Pour confirmer ou infirmer cette impression, nous comparons et évaluons ces deux méthodes expérimentalement sur des bases de données synthétiques dans la section 4.4.

4.1 Définition du problème

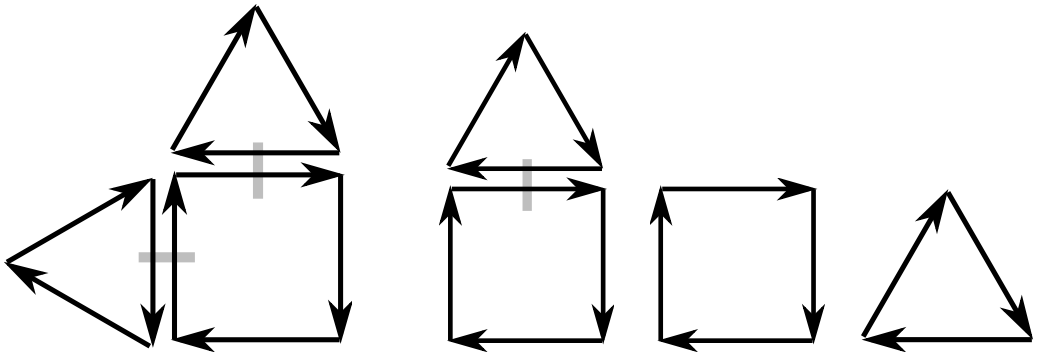
Dans le cadre de recherche de motifs fréquents dans une base de cartes combinatoires nD, un motif est une n-carte.

Les cartes combinatoires nD représentent les subdivisions de l'espace en cellules de dimensions 0 à n. De manière générale, les cellules de plus hautes dimensions contiennent le plus d'information c'est pourquoi, nous choisissons de construire les motifs faces par face en $2D$, volume par volume en $3D$ et n-cellule par n-cellule en nD. Ce choix nous semble le meilleur compromis entre le nombre de motifs qui seront extraits et la pertinence d'un motif. Mais selon l'application et la base de cartes, il pourrait être intéressant de construire les motifs de façon différente. Par exemple, nous pourrions considérer pour une base de cartes nD des motifs constitués d'un ensemble de i-cellules avec i fixé entre 0 et n (par exemple un ensemble de faces dans une base de cartes $3D$),

Nous choisissons de ne considérer que des motifs connexes pour réduire le nombre de motifs possibles. Cependant, il est possible de calculer trivialement l'ensemble des motifs non connexes à partir de l'ensemble des motifs connexes par un post-traitement.

Pour être considéré fréquent, un motif doit satisfaire deux contraintes de fréquence. La première contrainte est la même que celle définie dans [30] pour le problème de sous-graphes fréquents. Étant donnés une base de n-cartes BC et un nombre réel σ tel que $0 < \sigma \leq 1$, un motif respecte cette contrainte si et seulement si $freq_\sigma(C, BC) \geq \sigma \cdot |BC|$, où $freq_\sigma(C, BC)$ correspond à la fréquence de la carte C dans la base de carte BC, c.-à-d. le nombre de cartes $C' \in BC$ telles que C soit sous-carte de C'. Cette première contrainte de fréquence respecte la propriété d'anti-monotonie. En effet, Si une carte C' est présente dans une carte C alors toutes les sous-cartes de C' sont aussi présentes dans C, et inversement, si une des sous-cartes de C' n'est pas présente dans C alors C' est forcément non présente dans C.

La seconde contrainte de fréquence est une contrainte sur le nombre d'oc-



C	C'	C'_1	C'_2
$freq_\varphi$ avec automorphisme	2	4	6
$freq_\varphi$ sans automorphisme	2	1	2

FIGURE 4.1 – Exemple de fréquence des motifs C', C'_1 et C'_2 pour la base $BC = \{C\}$

currences d'un motif dans une même carte. Pour un seuil $\varphi \in [1;\infty[$ donné, un motif respecte cette contrainte pour une carte $C \in BC$ si et seulement si il est présent au moins φ fois dans C. Pour être considéré fréquent, un motif doit respecter ces deux contraintes, donc, pour une base de n-cartes BC et des seuils σ et φ, un motif m est un motif fréquent si et seulement si m est un ensemble de de n-cellules connexes et que $freq_\sigma(m, BC, \varphi) \geq \sigma \cdot |BC|$ avec $freq_\sigma(m, BC, \varphi) = |\{c \in BC \mid freq_\varphi(m, c) \geq \varphi\}|$ où $freq_\varphi(m, c)$ est égal au nombre d'occurrences du motif m dans la carte c.

Cette seconde contrainte possède aussi la propriété d'anti-monotonie. En effet, si un motif C' est présent x fois dans une carte C alors toutes les sous-cartes de C' sont présentes au moins x fois. Notons que pour conserver cette propriété d'anti-monotonie, il est indispensable de compter tous les automorphismes des motifs. La figure 4.1 montre un exemple où le motif C' est présent deux fois dans la carte C. Si les automorphismes ne sont pas pris en compte, nous nous apercevons que l'anti-monotonie n'est pas préservée, en effet, la fréquence de C'_1 est de 1 alors que celle de C' est de 2. Si les automorphismes sont comptés alors la propriété est conservée. La fréquence de C' est de 2 et la fréquence de ses sous-cartes C'_1 est de 4 et C'_2 est de 6. Nous avons vu que pour les graphes, compter les automorphismes ne suffit pas à rendre la fréquence décroissante, en effet, un sommet ne sera compté qu'une seule fois alors qu'il peut faire partie d'un grand nombre de motifs de deux sommets (autant qu'il y a d'arêtes). Dans le cas des 2-cartes, une face fermée de k brins possède k automorphismes et elle peut faire partie au maximum de k motifs de 2 faces, donc la fréquence est bien décroissante. La définition formelle d'un motif fréquent est donnée dans la définition 22

Définition 22 (motif fréquent)**.**
Soient une base de n-cartes BC, deux seuils de fréquences σ et φ et un motif m. m est un motif fréquent si et seulement si m est isomorphe à une sous-carte

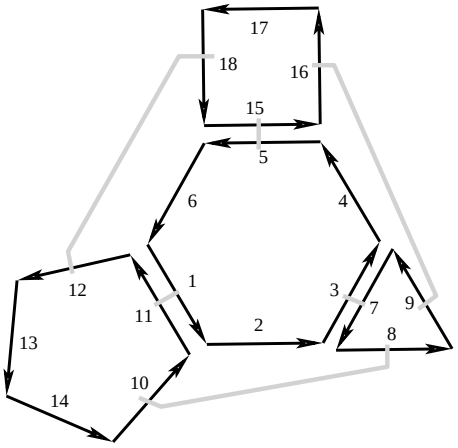


FIGURE 4.2 – Exemple d'une 2-carte de quatre faces où chaque face est 2-cousue à toutes les autres faces de la carte et chaque face est différente des autres faces de la carte. Il y a 16 sous-cartes différentes (la carte vide comprise).

$C_{\downarrow X} = (X, \beta'_1, \ldots, \beta'_n)$ ***de*** $C = (B, \beta_1, \ldots, \beta_n) \in BC$ ***telle que*** $\forall b \in X$, $\widehat{\langle \beta_n \rangle}(b) \in X$. ***Et*** $freq_\sigma(m, BC, \varphi) \geq \sigma \cdot |BC|$ ***avec*** $freq_\sigma(m, BC, \varphi) = |\{c \in BC \mid freq_\varphi(m, c) \geq \varphi\}|$.

Le nombre de sous-cartes fréquentes peut croître de façon exponentielle par rapport au nombre de faces de la 2-carte. En effet, dans le pire des cas, chaque face d'une 2-carte est différente de toutes les autres et elle est 2-cousue à toutes les autres faces de la carte. Dans ce cas, le nombre de sous-cartes différentes est de 2^n avec n, le nombre de faces d'une carte. La figure 4.2 donne un exemple d'une telle carte.

Les deux algorithmes que nous proposons et que nous détaillerons dans les sections suivantes répondent au cas le plus général qui nous permet de traiter le cas de recherche de motifs fréquents dans une base de cartes, mais aussi de rechercher les motifs fréquents dans une seule carte.

4.2 Méthode en largeur

Pour résoudre ce problème de recherche de sous-cartes fréquentes, nous proposons un algorithme appelé *MapAPriori* qui est une adaptation de l'algorithme *Apriori* [2] initialement introduit pour trouver des ensembles d'attributs fréquents dans une base de données relationnelle. Cet algorithme a ensuite été modifié dans [30] pour s'appliquer à la recherche de sous-graphes fréquents.

Pour faciliter la compréhension, nous présentons ici l'algorithme pour le cas de cartes combinatoires $2D$, cependant, toutes les opérations que nous utilisons sont définies en nD. De ce fait, l'extension de cet algorithme au cas nD est directe en remplaçant *face* par n*-cellule* dans les algorithmes de cette section.

Comme nous avons vu dans la section 2.1, l'algorithme *APriori* construit les motifs fréquents niveau par niveau. Chaque motif d'un niveau k donné est de taille k (par exemple, un ensemble d'attributs fréquents de niveau k contient k

attributs, un sous-graphe fréquent de niveau k contient k arêtes, ...). Dans notre cas, une sous-carte fréquente de niveau k contiendra k faces. Enfin, chaque motif de niveau $k+1$ est construit à partir de plusieurs motifs du niveau k. Les motifs de niveau k qui ont permis de construire des motifs de niveau $k+1$ sont appelés des générateurs.

Définition 23 (Générateur)**.**
Soient deux cartes C et C', C est un générateur de C' si et seulement si C est isomorphe à une sous-carte de C' et C a une face de moins que C'.

Nous avons identifié deux points critiques dans l'algorithme *APriori* . Le premier problème est la génération des candidats, c.-à-d. trouver une façon efficace de combiner deux motifs de k faces pour former un motif de $k+1$ faces susceptible d'être un motif fréquent. Le second problème est dans le calcul de la fréquence d'un motif candidat. C'est en effet une phase critique car il faut accéder à la base de données, ce qui est le plus coûteux en temps dans les algorithmes de fouille de données. C'est pour cela qu'il est important de réduire au maximum le nombre d'accès à la base et d'avoir un algorithme efficace pour décider si un motif est présent dans une carte ou non.

L'algorithme 11 construit les sous-cartes fréquentes niveau par niveau. Chaque niveau est découvert à partir du niveau précédent. Dans un premier temps, des candidats potentiellement fréquents sont générés à partir des sous-cartes fréquentes du niveau précédent. Ensuite, nous vérifions pour chaque candidat si son nombre d'occurrences dans BC est supérieur aux seuils fixés, c.-à-d. s'il apparaît au moins φ fois dans au moins $\sigma \cdot |BC|$ cartes.

Cet algorithme utilise les structures de données suivantes :
- BC : l'ensemble des cartes de la base ;
- F^k : l'ensemble des motifs fréquents de niveau k ;
- Γ^k : TW-signature de l'ensemble des candidats du niveau k ;
- Pour chaque motif candidat m de niveau k :
  - $N^{k-1}(m)$: l'ensemble des générateurs de m de taille $k-1$;
  - $Signature(m)$: la W-signature de m ;
  - $Occurences(m,c)$: l'ensemble des occurrences de m dans la carte $c \in BC$;
  - $Freq_{\varphi}(m,c)$: le nombre d'occurrences de m dans la carte $c \in BC$.

L'algorithme 11 comporte trois points clés. Le calcul des fréquents de niveau 1 et 2, la fonction de génération des candidats, (qui va permettre de générer tous les candidats possibles pour un niveau donné à partir du niveau précédent) et la fonction qui permet de calculer la fréquence de chaque candidat et ainsi de vérifier si le candidat est un motif fréquent ou non. Ces trois points clés sont détaillés dans la suite.

4.2.1 Calcul des sous-cartes fréquentes composées d'une ou de deux faces

Pour trouver les sous-cartes fréquentes composées d'une face, nous parcourons toutes les cartes $C \in BC$ face par face. Pour chaque face, la W-signature

Algorithme 11 : $MapAPriori(BC, \sigma, \varphi)$

Entrées : Une base de cartes BC, un réel $\sigma \in]0;1]$, un entier $\varphi \in [1;\infty[$.
Sortie : L'ensemble de tous les motifs fréquents.

$F^1 \leftarrow \emptyset$
$\Gamma^1 \leftarrow \emptyset$
pour *chaque carte* $C \in BC$ **faire**
  pour *chaque face* $f \in C$ **faire**
    Soit γ le motif composé de f
    si $Signature(\gamma) \in \Gamma^1$ **alors**
      $MAJ\_FREQ(\gamma, C)$
    sinon
      $INIT\_FREQ(\gamma, C, BC)$
      $\Gamma^1 \leftarrow \Gamma^1 \cup \gamma$
$F^1 \leftarrow \{\gamma \in \Gamma^1 \mid Freq_\sigma(\gamma, BC, \varphi) \geq \sigma \cdot |BC|\}$
$F^2 \leftarrow \emptyset$
$\Gamma^2 \leftarrow \emptyset$
pour *chaque carte* $C \in BC$ **faire**
  pour *chaque couple de faces adjacentes* $(f_1, f_2) \in C$ **faire**
    Soit γ le motif composé de f_1 et f_2
    si $Signature(\gamma) \in \Gamma^2$ **alors**
      $MAJ\_FREQ(\gamma, C)$
    sinon
      $INIT\_FREQ(\gamma, C, BC)$
      $\Gamma^2 \leftarrow \Gamma^2 \cup \gamma$
$F^2 \leftarrow \{\gamma \in \Gamma^2 \mid Freq_\sigma(\gamma, BC, \varphi) \geq \sigma \cdot |BC|\}$
$k \leftarrow 3$
tant que $F^{k-1} \neq \emptyset$ **faire**
  $\Gamma^k \leftarrow genererCandidats(F^{k-1})$
  pour tous les *candidats* $\gamma^k \in \Gamma^k$ **faire**
    $\gamma^k.freq \leftarrow Compter(\gamma^k, BC)$
  $F^k \leftarrow \{\gamma^k \in \Gamma^k \mid Freq_\sigma(\gamma, BC, \varphi) \geq \sigma \cdot |BC|\}$
  $k \leftarrow k + 1$
retourner $F^1, F^2, \ldots, F^{k-2}$

est calculée. Pendant le calcul de la signature, nous comptons le nombre d'automorphismes a. Posons γ le candidat correspondant à la W-signature. Si $\gamma \in \Gamma^1$, alors, la fréquence de γ pour la carte C est incrémenté de a ($MAJ\_FREQ(\gamma, C)$ ligne 8). Sinon, γ est ajouté à Γ^1 et sa fréquence est initialisée à a pour la carte C et à 0 pour toutes les autres cartes de BC ($INIT\_FREQ(\gamma, C, BC)$ ligne 10). Quand toutes les faces de toutes les cartes ont été parcourues, nous ne gardons dans F^1 que les candidats dont la fréquence est supérieur à φ pour au moins $\sigma \cdot |BC|$ cartes de BC : ce sont les motifs fréquents.

Les sous-cartes fréquentes composées de deux faces sont calculées de la même manière, seulement cette fois il faut parcourir tous les couples de faces adjacentes.

4.2.2 Génération des candidats

L'algorithme 12 permet de générer tous les candidats d'un niveau F^k à partir des sous-cartes fréquentes du niveau précédent F^{k-1}. Pour chaque couple de motifs fréquents ayant un générateur commun nous fusionnons les deux cartes de niveau $k-1$ pour faire un ensemble E^k de candidat γ^k. Pour chaque candidat, s'il n'a pas déjà été généré, alors nous identifions tous ses générateurs. Si un seul de ses générateurs n'est pas fréquent alors ce candidat ne sera pas fréquent non plus. En effet, si une sous-carte fréquente c' est présente dans une carte $C \in BC$ (c.-à-d. , c' est une sous-carte de C) alors tous les générateurs de c' sont aussi des sous-cartes de C. Donc la fréquence de c' est inférieure ou égale à la fréquence minimum de ses générateurs (comme l'illustrent les figures 4.3 et 4.5).

Nous considérons tous les couples de sous-cartes fréquentes qui ont un générateur commun (ligne 2 de l'algorithme 12). En effet, pour construire une carte de k faces à partir de deux cartes de $k-1$ faces, il faut qu'elles aient une partie commune de $k-2$ faces. Notons qu'il est possible de fusionner une carte avec elle même, mais pour qu'il y ait des fusions possibles, il faut obligatoirement qu'il y ait des automorphismes parmi les générateurs. Si une sous-carte a plus de deux générateurs alors elle sera générée plusieurs fois, c'est pourquoi nous vérifions ligne 5, que le motif n'a pas déjà été généré. La mise à jour des occurrences potentielles de γ est effectuée à la ligne 10. Lors de cette étape, pour diminuer au maximum le nombre d'occurrences à vérifier, nous prenons pour chaque carte $C \in BC$, les occurrences du générateur g' de γ telles que $Freq_\varphi(g', C)$ est minimale parmi tous les générateur de γ.

Définition 24 (Opération $\ominus$ (n-suppression))**.**
Soient une carte $C = (B, \beta_1, \ldots, \beta_n)$ et une n-cellule $c \in C$. Nous notons BV l'ensemble des brins de B n-cousus à c et n'appartenant pas à c. La carte obtenue en supprimant c de C est $C' = (B', \beta'_1, \ldots, \beta'_n)$ définie par :

- $B' = B \setminus c$;
- $\forall i \in [0; n] : \beta'_i = \beta_i|_{B'}$;[1]
- $\forall b' \in BV : \beta_n(b) = \epsilon$.

1. β'_i est égal à β_i restreint à B', c.-à-d. $\forall b \in B' : \beta'_i(b) = \beta_i(b)$.

Algorithme 12 : $genererCandidats(F^{k-1})$

Entrée : Ensemble des sous-cartes fréquentes F^{k-1}.
Sortie : Ensemble des candidats de taille k.

```
1  Γ^k ← ∅
2  pour tous les couples (f_i^{k-1}, f_j^{k-1}) ∈ F^{k-1} × F^{k-1} ayant un générateur
   commun g ∈ F^{k-2} faire
3      E^k ← fusion(f_i^{k-1}, f_j^{k-1}, g)
4      pour tous les γ^k ∈ E^k faire
5          si γ^k ∉ Γ^k alors
6              cand ← Vrai
7              pour toutes les faces f de γ^k faire
8                  γ^{k-1} ← γ^k ⊖ f
9                  si γ^{k-1} est connexe et γ^{k-1} ∈ F^{k-1} alors
10                     MAJ_OCCURRENCES(γ^k)
11                 sinon
12                     cand ← Faux
13             si cand = Vrai alors
14                 Γ^k ← Γ^k ∪ γ^k
15 retourner Γ^k
```

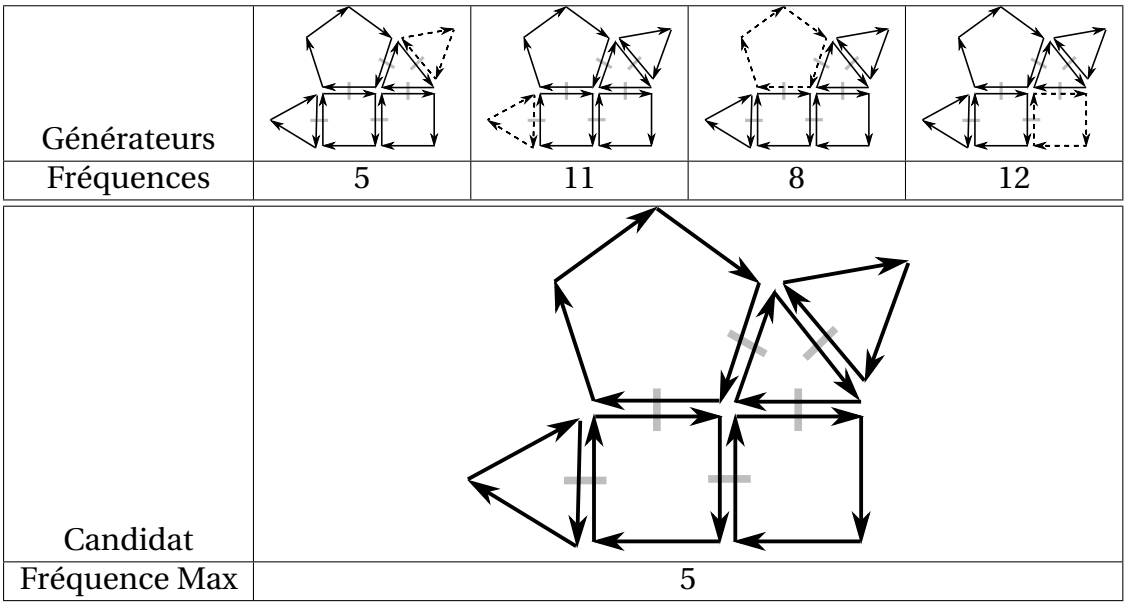


FIGURE 4.3 – Exemple de fréquence d'un candidat. Les générateurs sont en traits pleins et la face en pointillés correspond à la face à ajouter au générateur pour former le candidat. Pour une carte et un candidat γ donnée, la fréquence minimale des générateurs de γ est 5, donc la fréquence de γ est inférieure ou égale à 5.

L'opérateur $\ominus$ utilisé ligne 8, permet de supprimer une face d'une 2-carte. Par exemple, sur la figure 4.3 l'opérateur $\ominus$ appliqué sur le candidat et une des faces en pointillés a pour résultat le générateur en trait plein.

4.2.3 Fusion

Un point clé dans la génération des candidats consiste à fusionner deux sous-cartes fréquentes. Nous fusionnons deux sous-cartes fréquentes de taille $k-1$ qui ont un générateur commun de taille $k-2$ pour former entre 0 et n candidats de taille k, n étant le nombre d'automorphismes du générateur. L'algorithme 13 montre le fonctionnement de la fusion. Nous commençons par identifier le générateur dans la première carte, puis dans la seconde. Il peut y avoir plusieurs façons d'apparier le générateur, lorsqu'il possède des automorphismes. Pour chaque appariement possible nous cherchons la face de f_i^{k-1} et la face de f_j^{k-1} qui ne sont pas dans le générateur. Ces deux faces sont ajoutées au générateur g pour former le candidat γ^k (ligne 8). Il se peut qu'il soit impossible d'ajouter les deux faces au générateur si les deux faces sont liées aux mêmes brins du générateur. Les tableaux 4.1,4.2,4.3 illustrent les différents cas de fusions possibles.

Notons que la face à ajouter peut être cousue à plusieurs brins du générateur, dans ce cas il faut qu'aucun de ces brins ne soit cousu avec la face à ajouter de l'autre carte. Le tableau 4.4 illustre cette possibilité.

Algorithme 13 : $Fusion(f_i^{k-1}, f_j^{k-1}, N^{k-2})$

Entrée : Sous-cartes fréquentes f_i^{k-1}, f_j^{k-1}, g.
Sortie : L'ensemble des fusions possibles entre f_i^{k-1} et f_j^{k-1} selon le générateur g.

1 $E \leftarrow \emptyset$
2 identifier g dans f_i^{k-1}
3 $B \leftarrow \{b \in f_i^{k-1} \mid b \in g$ et $\beta_2(b) \notin g$ et $\beta_2(b) \neq \epsilon\}$
4 **pour tous les** *automorphismes* a *de* g **faire**
5 identifier g dans f_j^{k-1} selon l'automorphisme a
6 $B' \leftarrow \{b' \in f_j^{k-1} \mid b' \in g$ et $\beta_2(b') \notin g$ et $\beta_2(b') \neq \epsilon\}$
7 **si** $B \cap B' = \emptyset$ **alors**
8 $e \leftarrow f_j^{k-1} + f$ avec f une face de f_i^{k-1} tel que $f \notin g$
9 $E \leftarrow E \cup e$
10 **retourner** E

4.2.4 Comptage de la fréquence

Le dernier point critique de l'algorithme *MapAPriori* est le calcul de la fréquence de chaque candidat. Ce qui est très coûteux en temps dans les algorithmes de

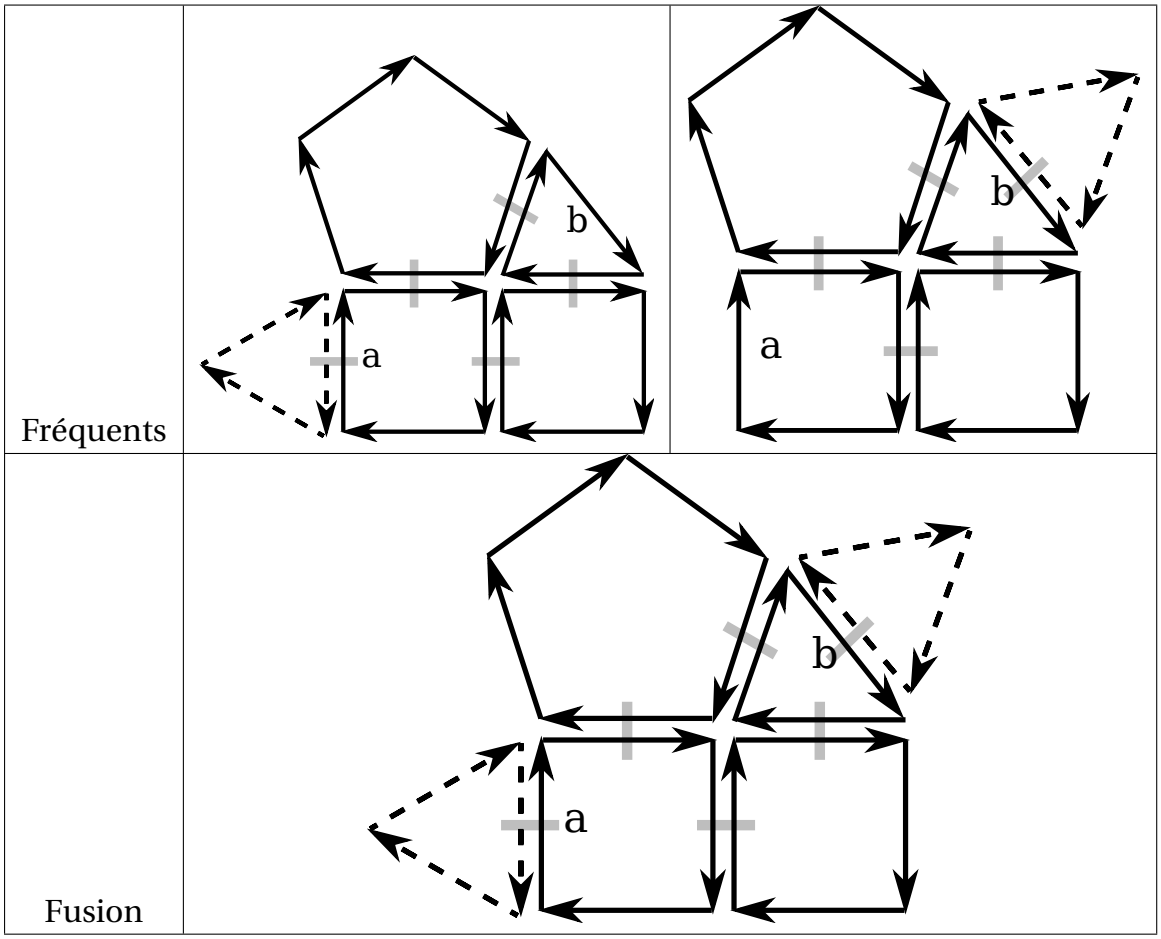


TABLE 4.1 – Exemple de fusion entre deux cartes de cinq faces ayant un générateur commun de quatre faces. Les brins dessinés en traits pleins représentent le générateur (identique dans les deux cartes) et les brins en pointillés indiquent les faces qu'il faut ajouter au générateur pour former un candidat composé de six faces. Comme le générateur n'a pas d'automorphisme, il n'y a qu'une seule façon possible d'apparier le générateur dans chacune des deux cartes, et donc il y aura au maximum un candidat issu de cette fusion. Enfin, les brins du générateur qui relient les faces à ajouter sont différents, la fusion est donc possible.

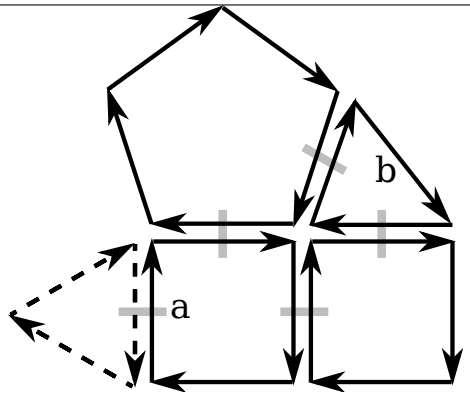
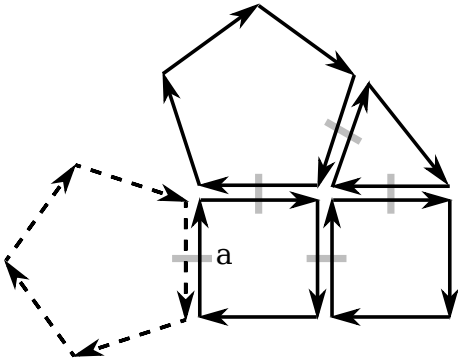
Fréquents		
Fusion	Impossible	

TABLE 4.2 – Exemple où il n'existe pas de fusion possible pour deux cartes composées de cinq faces qui ont un générateur commun de quatre faces. En effet, le générateur commun n'a pas d'automorphisme, il y a donc une seule façon possible d'apparier le générateur dans les deux cartes, donc au maximum une seule fusion possible. Mais les deux faces à ajouter doivent être ajoutées sur le même brin du générateur, donc la fusion est impossible.

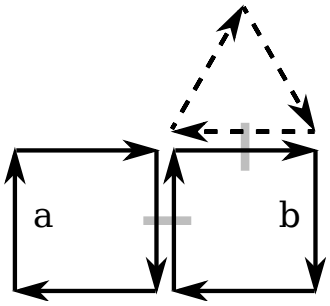
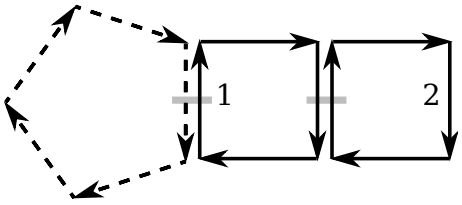
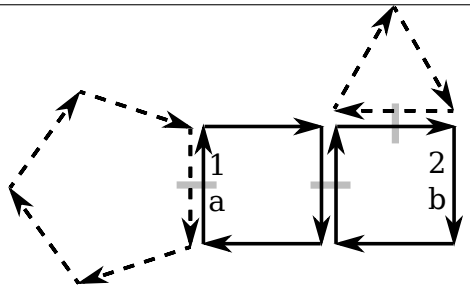
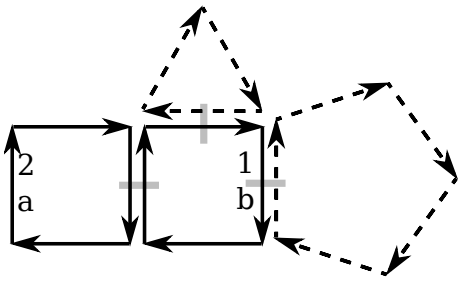
Fréquents		
Fusions		

TABLE 4.3 – Exemple de fusions multiples. Les deux cartes à fusionner sont composées de trois faces et ont un générateur commun de deux faces. Le générateur a deux automorphismes. En effet, pour apparier le générateur de la carte de droite sur celui de la carte de gauche, le brin *1* peut être apparié soit avec le brin *a*, ce qui donne la première fusion, soit avec le brin *b*, ce qui donne la seconde fusion.

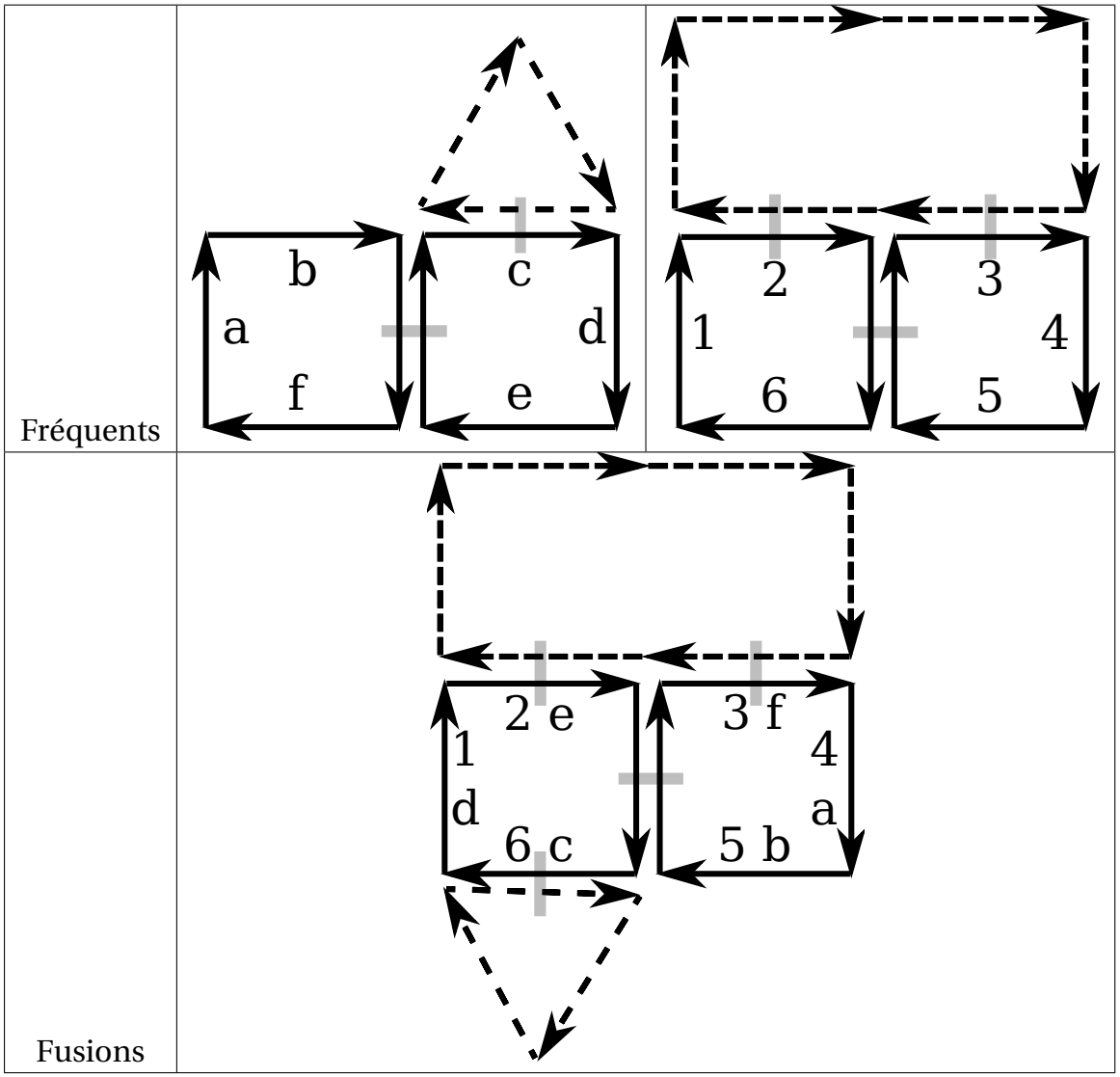


TABLE 4.4 – Exemple de fusion où le générateur a deux automorphismes mais une seule fusion possible. Nous souhaitons fusionner deux motifs composés de trois faces qui ont un générateur commun de deux faces. La face à ajouter sur la carte de droite est cousue aux brins *2* et *3*. Si le brin *4* de la carte de droite est apparié avec le brin *a* de la carte de gauche alors la fusion est possible. Par contre si le brin *4* de la carte de droite est apparié avec le brin *d* de la carte de gauche alors cette fusion est impossible car le brin *3* et le brin *c* sont en conflit.

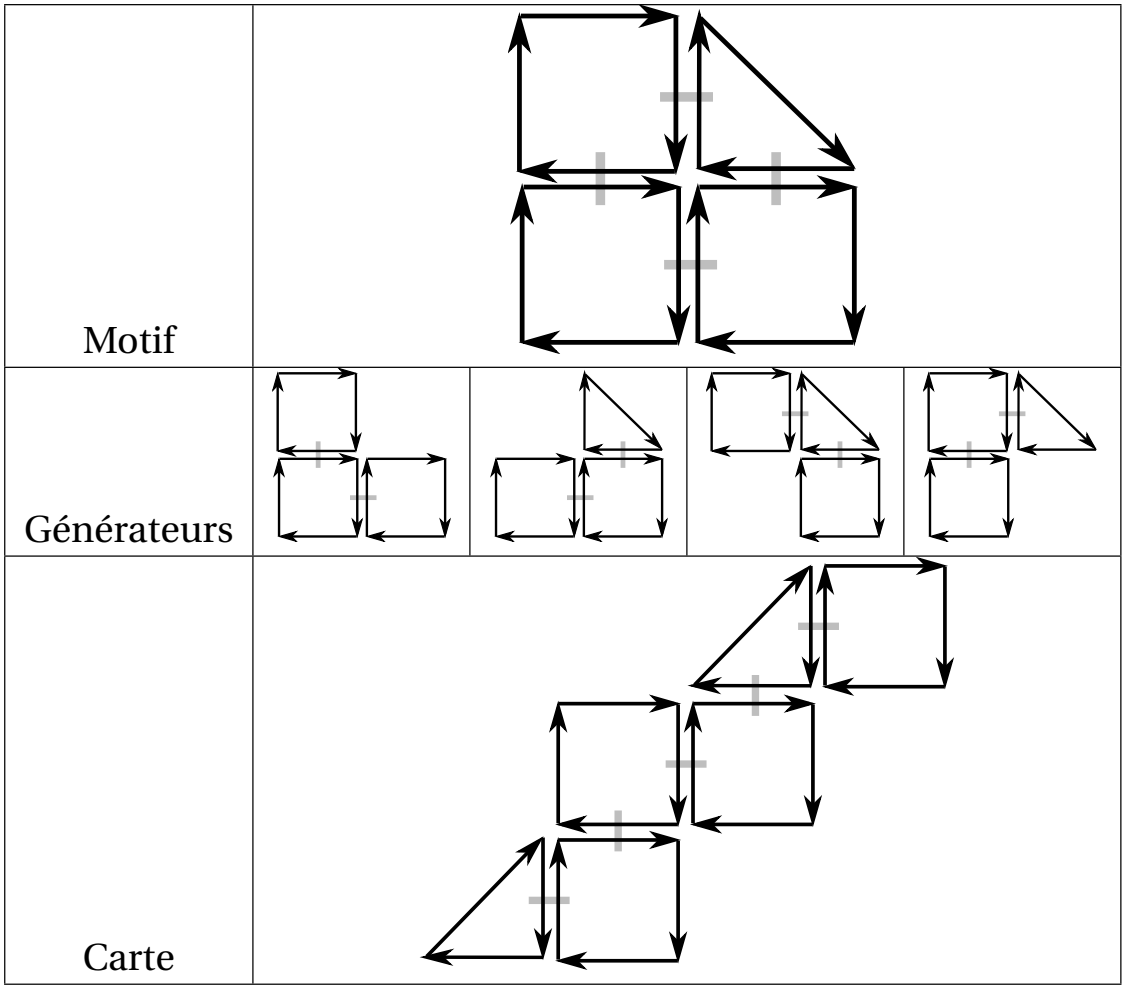


FIGURE 4.4 – Exemple de carte où un motif n'est pas présent alors que tous ses générateurs le sont.

fouille de données, ce sont les accès à la base de données. Dans le cas des ensembles d'attributs fréquents, une amélioration de l'algorithme *APriori* permet de calculer la fréquence de chaque candidat sans aucun accès à la base de données, à condition d'avoir scanné une fois pour toute la base pour construire les motifs de niveau 1 et d'avoir pour chaque fréquent une liste qui donne la liste des transactions dans lesquelles apparaît le motif. Ensuite la liste d'un motif est l'intersection des listes de ses générateurs. Malheureusement, cela n'est pas suffisant pour calculer la fréquence d'un motif dans le cadre de la recherche de sous-cartes fréquentes. En effet, une carte peut contenir tous les générateurs d'un motif mais ne pas contenir le motif lui même.

Exemple 16. *La figure 4.4 montre un motif et tous ses générateurs et une carte dans laquelle sont présents tous les générateurs du motif mais où le motif lui-même n'est pas présent.*

Pour calculer la fréquence d'un candidat, il est obligatoire d'accéder à la base de données et d'utiliser un algorithme d'isomorphisme de sous-cartes. Cependant, nous utilisons la connaissance que nous avons sur les générateurs d'un

candidat pour améliorer la complexité de l'algorithme d'isomorphisme et pour réduire le nombre d'appels à cet algorithme.

Une fois tous les candidats générés, nous devons vérifier si leur fréquence dépasse le seuil fixé. Pour cela nous maintenons à jour une liste qui contient un brin pour chacune des occurrences d'un motif fréquent dans BC (algorithme 14). Pour un candidat γ^k, il faut tester pour chacun des brins par lesquels passent ses générateurs si γ^k passe aussi par ces brins. Nous savons que si un motif est présent, il passe forcément par les brins de ses générateurs. Nous avons donc adapté l'algorithme de recherche de sous-isomorphisme en fixant un brin de la grande carte au lieu d'un brin de la petite carte, la complexité est $\mathcal{O}(n^2)$ avec n le nombre de brin de γ^k au lieu de $\mathcal{O}(n \cdot m)$ avec m le nombre de brin de la carte dans laquelle nous recherchons γ. Potentiellement m peut être très grand par rapport à n, notamment lors de la construction des premiers niveaux, où des motifs de quelques faces auront une dizaine de brins, alors que la carte de départ peut faire plusieurs milliers de brins.

Algorithme 14 : $Compter(\gamma^k, BC)$

Entrée : Candidat γ^k ;
Une base de cartes BC.
Sortie : Nombre d'occurrences de γ^k dans BC.

```
1  i ← 0
2  pour toutes les cartes C ∈ BC faire
3      Freq_φ(γ^k, C) ← 0
4      pour tous les brins b ∈ Occurences(γ^k, C) faire
5          si SousIso(γ^k, C, b)) alors
6              Freq_φ(γ^k, C) ← Freq_φ(γ^k, C) + 1
7          sinon
8              Occurences(γ^k, C) ← Occurences(γ^k, C) \ {b}
9      i ← i + 1
10 retourner Freq_φ(γ^k, C)
```

Soit $C = (B, \beta_1, \ldots, \beta_n)$ une carte de la base et $C' = (B', \beta'_1, \ldots, \beta'_n)$ un motif candidat. Nous voulons rechercher toutes les occurrences de C' dans C. Notons que C est bien plus grand que C'. Dans l'algorithme classique d'isomorphisme de sous-cartes, nous n'avons aucune connaissance a priori sur l'emplacement de C' dans C, donc pour un brin fixé de la carte C', il faut tester s'il existe une correspondance pour chacun des brins de C. La complexité est donc de $\mathcal{O}(|B| \cdot |B'|)$. Un grand nombre de brins de C sont visités pour rien. Dans le cadre général d'isomorphisme de sous-cartes il est impossible de prédire quels sont les brins qu'il n'est pas nécessaire de visiter ou au contraire quels sont les brins qui ont le plus de chances de faire partie de l'isomorphisme. Dans notre cas de recherche de sous-cartes fréquentes, nous connaissons pour chaque motif fréquent son nombre d'occurrences, et l'endroit où elles sont situées dans les cartes de la base. Chaque candidat possède au moins un générateur (qui est un

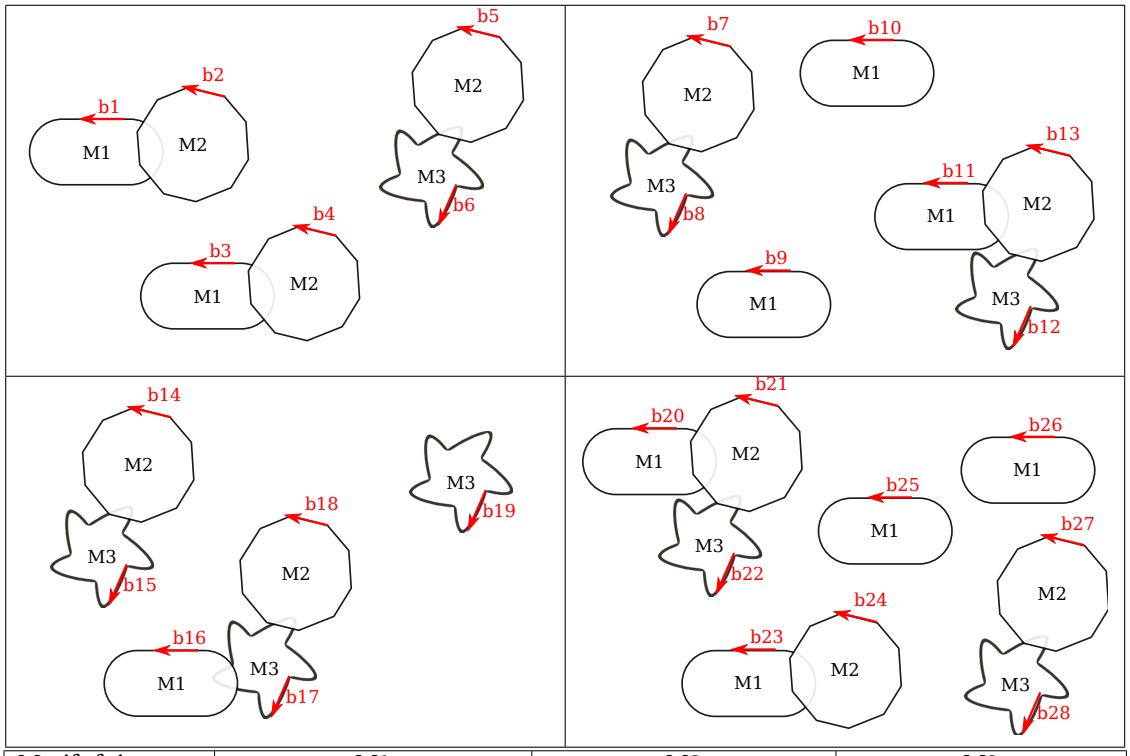
motif fréquent) dont nous connaissons toutes les occurrences. Nous savons que la fréquence d'un candidat est inférieure ou égale à la fréquence de ses générateurs et nous savons aussi que pour chaque occurrence du candidat, une occurrence de ses générateurs est aussi présente. Nous nous servons de cette information pour ne tester qu'une sous partie de l'ensemble des brins de C. Nous stockons en mémoire un brin par lequel passe chaque occurrence de chaque motif. Ensuite pour calculer la fréquence d'un candidat, connaissant la liste des brins par lesquels passent ses générateurs, il suffit de tester l'isomorphisme de sous-carte pour tous ces brins et non pour l'ensemble des brins de la base. Cela nous donne une complexité de $\mathcal{O}(|B'|^2)$, sachant que $B' \ll B$, la complexité est grandement améliorée par rapport à l'algorithme classique.

Pour un candidat donné, il n'est pas nécessaire de tester toutes les occurrences de ses générateurs mais seulement de ceux qui ont la plus faible fréquence dans chacune des cartes (ligne 10 de l'algorithme 12). En effet la fréquence d'un motif sera toujours inférieure ou égale à la fréquence de ses générateurs. Il est donc possible dans certains cas de trouver qu'un candidat n'est pas un fréquent sans même avoir à vérifier sa présence dans la base.

Exemple 17. *La figure 4.5 montre une base composée de 4 cartes, pour un niveau donnée, il y a 3 motifs fréquents (M1, M2 et M3) et 3 candidats sont générés par l'algorithme 12 (M1M2, M1M3 et M2M3) pour* $\sigma = 0.75$ *et* $\varphi = 2$*. Pour les 3 motifs fréquents nous connaissons leur fréquence dans chaque carte ainsi qu'un brin pour chacune de ses occurrences. Le motif M1 est présent 2 fois dans la carte 1 et passe par les brins b1 et b3, il est présent 3 fois dans la carte 2 et passe par les brins b9, b10 et b11, ... Le candidat M1M2 a pour générateur les motifs M1 et M2, donc la fréquence maximale qu'il pourra atteindre dans la carte 2 est de 2 car le motif M1 est présent 3 fois mais le motif M2 n'y est présent que 2 fois. Grâce à cette propriété, nous pouvons savoir sans accéder à la base de données que le motif M1M2 sera forcement non fréquent dans la carte 3, et que le motif M1M3 ne sera pas fréquent pour les cartes 1 et 3 et donc ne dépassera pas le seuil* σ*. Il est donc inutile d'utiliser la fonction d'isomorphisme de sous-carte pour calculer la fréquence exacte de ce motif. Dans cet exemple, nous voyons que pour le motif candidat M1M3 il n'est pas nécessaire d'utiliser la fonction d'isomorphisme pour décider que ce motif n'est pas fréquent. Le motif M1M2 n'est pas fréquent non plus mais pour le savoir nous sommes obligés de tester l'isomorphisme de sous-cartes pour les occurrences à vérifier. Enfin le motif M2M3 est fréquent et il est obligatoire de tester l'isomorphisme de sous-cartes pour connaître les occurrences exactes.*

4.2.5 Complexités des différentes étapes de l'algorithme *MapA-Priori*

Dans l'algorithme 11 en ligne 5, nous construisons au maximum $\frac{n^2}{2} * k * a$ candidats de niveau k à partir des fréquents du niveau $k-1$ (avec n, le nombre de fréquents de niveau k et a, le nombre d'automorphismes des motifs fréquents). Pour chacun de ces candidats C, nous comptons le nombre d'occurrences de C dans l'ensemble des cartes de la base BC (ligne 7) grâce à un algorithme d'iso-



Motifs fréquents	M1		M2		M3	
Fréquence	2	3	3	2	1	2
	1	4	2	3	3	2
Occurrences	b1, b3	b9, b10, b11	b2, b4, b5	b7, b13	b6	b8, b12
	b16	b20, b23, b25, b26	b14, b18	b21, b24, b27	b15, b17, b19	b22, b27

Candidats	M1M2		M1M3		M2M3	
Fréquence Max	2	2	1	2	1	2
	1	3	1	2	2	2
Occurrences	b1, b3	b7, b13				b8, b12
à vérifier		b21, b24, b27			b14, b18	b22, b27
Fréquence réelle	2	1				2
		2			2	2

FIGURE 4.5 – Fréquence des candidats par rapport à leurs générateurs. La fréquence dans chacune des quatre cartes est donnée. Les seuils sont fixés à $\sigma = 0.75$ et $\varphi = 2$ (pour être fréquent, un motif doit être présent 2 fois dans au moins 3 cartes différentes). Le motif M1M3 n'est pas fréquent et nous le savons avant même de vérifier les occurrences. Le motif M1M2 n'est pas fréquent mais il faut d'abord vérifier les occurrences pour le savoir. Enfin, le motif M1M2 est fréquent.

morphisme de sous-cartes pour un sous-ensemble minimale des occurrences des générateurs de C. La complexité pour calculer la fréquence d'un candidat est donc $\mathcal{O}(x \cdot |C|^2)$ avec x, le nombre d'occurrences sélectionnées des générateurs de C.

À la ligne 6 de l'algorithme 12 nous vérifions si un candidat a déjà été généré et à la ligne 9 de l'algorithme 12 nous vérifions si tous les générateurs du candidats sont bien des motifs fréquents. Pour calculer cela de façon efficace, nous utilisons deux TW-signatures. La première nommée Γ^k contient l'ensemble des candidats du niveau en cours de construction et elle est vidée à chaque appel de la fonction *genererCandidats* (ligne 1 de l'algorithme 12). La deuxième TW-signature nommée F^{k-1} contient l'ensemble des sous-cartes fréquentes du niveau $k-1$ lors de la génération des candidats de niveau k.

Dans l'algorithme 13 aux lignes 2 et 5, nous devons identifier un générateur g de taille $k-2$ dans une carte C de taille $k-1$. Pour cela nous utilisons l'algorithme classique d'isomorphisme de sous-carte. Cette opération a une complexité de $\mathcal{O}(|g| \cdot |C|)$.

4.3 Méthode en profondeur

Dans cette section nous présentons un deuxième algorithme pour résoudre le problème d'extraction de sous-cartes fréquentes basé sur une deuxième famille de méthodes qui sont les méthodes qui explorent les motifs en profondeur.

4.3.1 Principe

Le but des algorithmes en profondeur est d'éviter l'étape de génération des candidats des algorithmes de type *APriori* qui est souvent très coûteuse car elle génère de nombreux motifs qui ne sont pas présents dans la base. Pour cela ces méthodes utilisent souvent une liste d'occurrences et à partir de ces occurrences, elles font grandir le motif d'un élément. Cela rend le calcul de la fréquence très simple (elle découle directement de la taille de la liste d'occurrences) et tous les motifs générés sont présents au moins une fois dans la base. Dans le cas de l'extraction de sous-cartes fréquentes, nous utiliserons ce principe pour faire grandir les motifs fréquents face par face et ainsi calculer itérativement tous les motifs fréquents.

4.3.2 Algorithme

L'algorithme 15 décrit notre second algorithme d'extraction de sous-cartes fréquentes appelé mSpan (pour Map-based Substructure Pattern mining). mSpan est une adaptation de gSpan [55], un algorithme d'extraction de sous-graphes fréquents. Comme nous l'avons vu dans la section 2.2, cet algorithme extrait les motifs en profondeur et utilise la contrainte de fréquence pour élaguer l'espace de recherche.

Algorithme 15 : $mSpan(BC, \sigma, \varphi)$

Entrées : Une base de cartes BC ;
un nombre réel $\sigma \in]0; 1]$.
Sortie : l'ensemble F de tous les motifs fréquents.

1 $F_1 \leftarrow$ tous les motifs fréquents composés d'une seule face
2 $F \leftarrow F_1$
3 **tant que** $F_1 \neq \emptyset$ **faire**
4 choisir un motif f dans F_1
5 $Cand \leftarrow \{f\}$
6 **tant que** $Cand \neq \emptyset$ **faire**
7 prendre un motif p dans $Cand$
8 $F_p \leftarrow grow(p, F_1)$
9 $Cand \leftarrow Cand \cup F_p$
10 $F \leftarrow F \cup F_p$
 `/* Tous les motifs contenant la face` f `sont dans` F `*/`
11 enlever f de F_1
12 retourner F

Tout d'abord, nous calculons l'ensemble F_1 de tous les motifs fréquents composés d'une seule face. Puis l'ensemble F qui contiendra tous les motifs fréquents est initialisé avec F_1. Ensuite, pour chaque face f de F_1, nous construisons tous les motifs fréquents contenant f plus des faces de F_1 et nous les ajoutons à F (lignes 4 à 10). Enfin, nous retirons f de F_1 (ligne 11) pour éviter de reconstruire des motifs qui contiennent f à la prochaine itération de la boucle des lignes 3 à 11.

L'ensemble de tous les motifs fréquents qui contiennent f plus des faces de F_1 est construit itérativement en utilisant un ensemble $Cand$ de motifs fréquents qui sont étendues en leur cousant une face de F_1. À chaque itération (lignes 6 à 10), nous retirons un motif p de $Cand$ (ligne 7) et la fonction *grow* calcule tous les motifs fréquents qu'il est possible de construire en cousant une face de F_1 à chacune des occurrences de p (ligne 8). Ces motifs fréquents sont ajoutés à l'ensemble $Cand$ (ligne 9) pour qu'ils puissent ensuite être à leur tour étendue d'une face, et ce, jusqu'à ce qu'il n'y ai plus de nouveaux motifs fréquents découverts.

Algorithme *grow* (ligne 8 Algo. 15). La fonction *grow* est décrite dans l'algorithme 16. Étant donnés un motif fréquent p et un ensemble de face F_1, il retourne toutes les sous-cartes fréquentes obtenues en cousant une face de F_1 à p. Cela est fait en parcourant la frontière de chaque occurrence o du motif p dans une carte C de BC. Pour chaque brin b de la frontière, si la face 2-cousue à d appartient à F_1 alors le motif p_f obtenu en 2-cousant cette face au brin b est candidat pour être un motif fréquent et est ajouté à F_p s'il n'y était pas déjà présent (ligne 9), sinon la fréquence de ce motif dans la carte C est incrémenté (ligne 10). Quand toutes les occurrences de p ont été traitées, nous retournons

tous les motifs de F_p qui sont fréquents (ligne 11).

Algorithme 16 : $grow(p, F_1)$

Entrées : un motif fréquent p ;
un ensemble de motifs fréquents à une face F_1.
Sortie : tous les motifs fréquents composés de p plus une face de F_1.

```
1  F_p ← ∅
2  pour chaque occurrence o du motif p dans une carte C de BC faire
3      pour chaque brin b appartenant à la frontière de l'occurrence o de p
       faire
4          si β2(b) ≠ ε /* donc il existe une face 2-cousu à b          */
5          alors
6              Soit f la face 2-cousue à b
7              si f ∈ F_1 alors
8                  Soit p_f le motif obtenu en 2-cousant la face f au brin d de o
9                  si p_f ∉ F_p alors ajouter p_f à F_p et initialiser freq(p_f)
10                 sinon mettre à jour freq(p_f)
11 retourner {p_i ∈ F_p | Freq_σ(p_i, BC, φ) ≥ σ · |BC|}
```

Identifier une occurrence (ligne 2 Algo. 16). Pour identifier une occurrence d'un motif p dans une carte C, nous n'utilisons pas l'algorithme d'isomorphisme de sous-carte de p dans C car cela serait trop couteux en temps. À chaque fois qu'une nouvelle occurrence d'un motif est trouvée nous mémorisons le brin qui a permis de construire la signature. Nous stockons pour chaque motif sa signature ainsi qu'un brin pour chacune de ses occurrences. Grâce à cela, nous pouvons positionner une occurrence d'un motif dans une carte de façon linéaire par rapport au nombre de brins du motif.

Exemple 18. *La figure 4.6 montre un exemple où le motif C est présent 5 fois dans la carte C'. Un brin suffit pour différencier ces 5 occurrences. Par exemple le brin q correspond à l'occurrence en pointillées.*

Parcourir la frontière d'une occurrence (ligne 3 Algo. 16). Lors de l'identification d'une occurrence d'un motif p dans une 2-carte $C = (B, \beta_1, \beta_2)$, nous mémorisons les brins $b \in B$ tels que $\beta_2(f(b)) = \epsilon$. Cela peut être fait en même temps que l'identification de l'occurrence sans augmenter la complexité.

Ajouter une face à une occurrence (ligne 8 Algo. 16). Nous avons identifié un motif p dans une carte C, et nous avons trouvé un brin b tel que b appartient à une occurrence de p et que la face f adjacente à b appartient à F_1. Nous ajoutons alors la face f au motif p pour former un nouveau motif p_f.

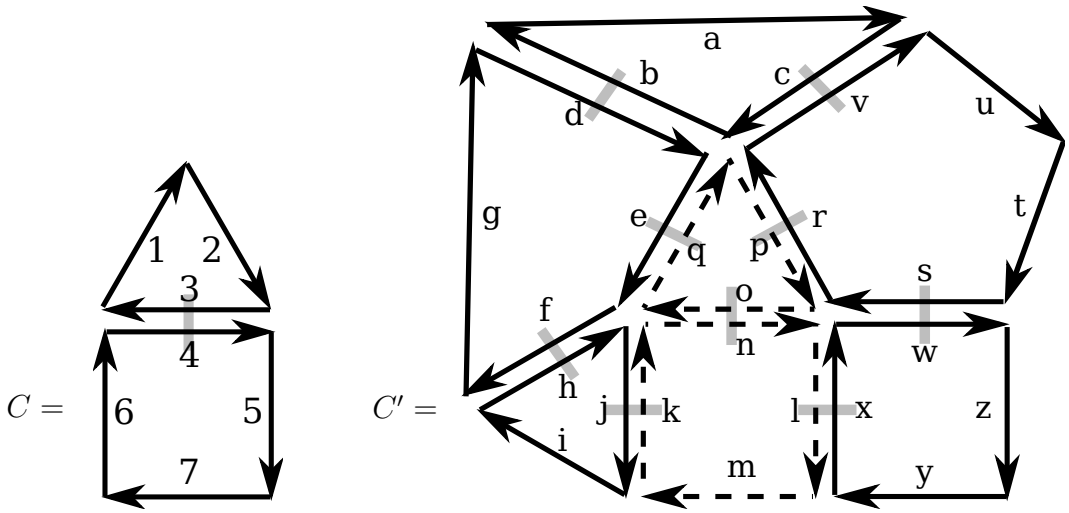


FIGURE 4.6 – Exemple de liste d'occurrences pour identifier un motif. Le motif C est présent 5 fois dans la carte C'. Pour chaque occurrence, nous mémorisons le brin de C' qui correspond au brin 1 du motif C. Donc, la liste d'occurrences associée au motif C dans la carte C' est $Occurrences(C', C) =< q, a, j, p, i >$.

Exemple 19. *La figure 4.7 montre une carte C, un motif p ainsi que les deux motifs construit en ajoutant une face au motif p. En supposant que les faces 1 et 2 appartiennent à F_1, le plongement du motif p dans la carte C possède 3 brins de sa frontière dont la face adjacente appartient à F_1 (les brins a, b et c). Si la face 1 qui est adjacente au brin a est ajouté au motif p, nous obtenons le motif p_1, le brin a est supprimé de la liste des brins à explorer. Si la face 2 qui est adjacente au brin b est ajouté au motif p, nous obtenons le motif p_2, ensuite le brin b est supprimé de la liste des brins à explorer. Notons que le motif p_2 peut être obtenue en ajoutant la face 2 soit à partir du brin b soit à partir du brin c. Il est donc important de supprimer les brins b et c de la liste lorsque le motif p_2 est créé.*

Vérifier la présence d'un motif dans F_p (ligne 9 Algo. 16). À chaque fois qu'un motif est généré (ligne 8), nous calculons sa signature, puis nous recherchons si elle est présente dans l'arbre de signatures F_p. Si c'est le cas, alors le motif a déjà été généré, il faut donc mettre à jour sa fréquence. Sinon, c'est un nouveau motif, donc nous ajoutons sa signature à l'arbre des signatures et sa fréquence est initialisée. Comme nous l'avons vu dans le chapitre 3, le calcul de la signature se fait en $\mathcal{O}(k^2)$ (avec k le nombre de brins du motif). Vérifier si cette signature appartient à l'arbre de signature F_p se fait en $\mathcal{O}(k)$ et ne dépendant pas du nombre de signatures déjà présente dans l'arbre. L'ajout, d'une signature dans l'arbre est aussi en $\mathcal{O}(k)$ et peut être fait pendant la recherche.

Initialiser et mettre à jour la fréquence d'un motif (lignes 9-10 Algo. 16). Nous avons vu que pour préserver la propriété d'anti-monotonie de la fréquence, il faut compter tous les automorphismes d'un motif. Nous avons aussi vu que le calcul de la signature d'une carte donne le nombre d'automorphismes de cette

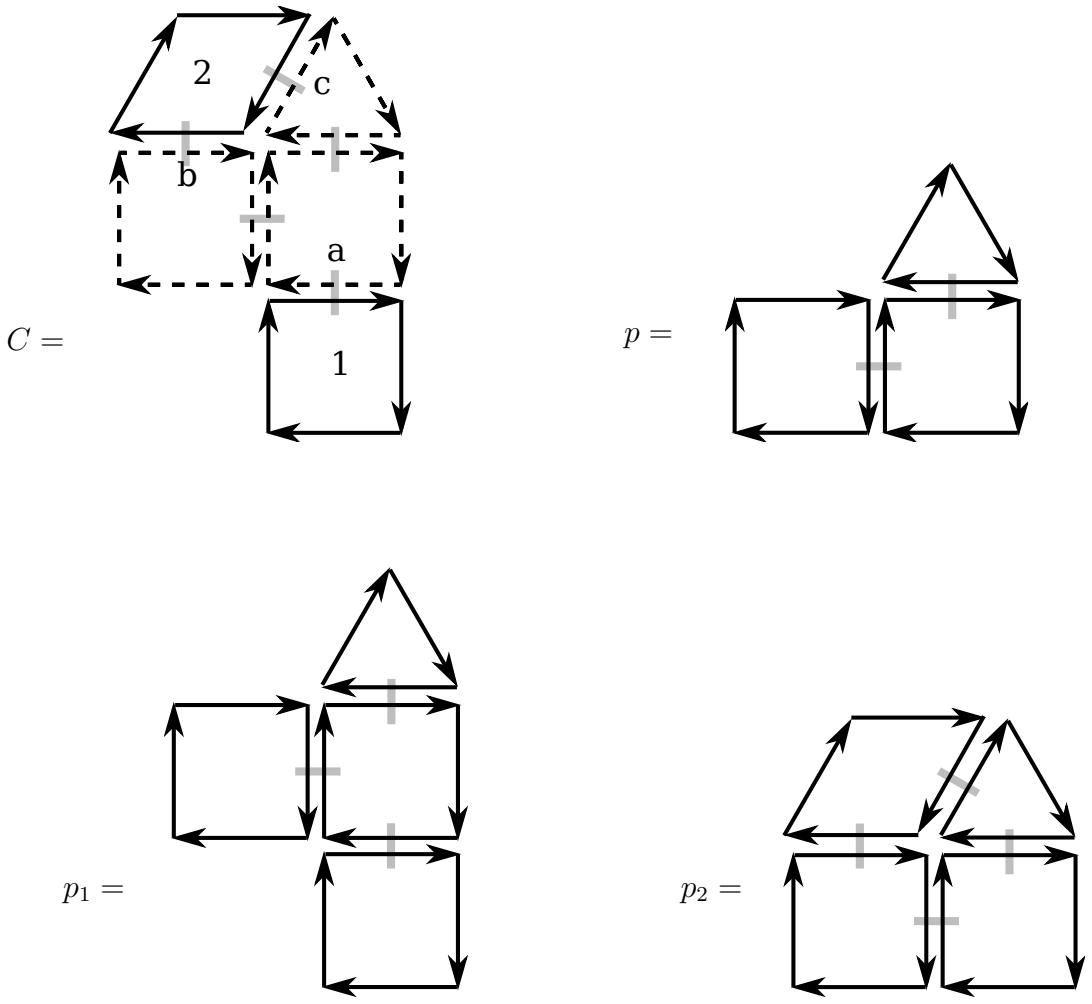


FIGURE 4.7 – Exemple de motifs construit en ajoutant une face à un motif fréquent.

carte. Lorsqu'un motif m est découvert pour la première fois, il faut initialiser sa fréquence (ligne 9). $\forall c' \in BCFreq_\varphi(m, c') \leftarrow 0$ et $Freq_\varphi(m, C) \leftarrow a$ (avec a le nombre d'automorphismes de m et C la carte où m est présent). Pour mettre à jour la fréquence d'un motif il suffit d'ajouter le nombre d'automorphismes du motif m à sa fréquence pour la carte C où il a été découvert : $Freq_\varphi(m, C) \leftarrow Freq_\varphi(m, C) + a$.

Choisir un motif (ligne 4 Algo. 15). L'ordre dans lequel les faces de F_1 sont choisies est important. En effet, en les choisissant dans l'ordre de fréquence croissante, c.-à-d. du moins fréquent au plus fréquent, l'extraction de tous les motifs fréquents est plus rapide. En effet, en commençant par la face $f_1 \in F_1$ qui a le moins d'occurrences, nous allons calculer dans la boucle (lignes 6 à 10) tous les motifs fréquents qui contiennent f_1 en vérifiant un nombre minimum d'occurrences. Ensuite, ligne 11, nous enlevons la face f_1 de l'ensemble F_1, donc tous les motifs qui seront construit par la suite ne contiendront pas cette face. Lors de la deuxième itération de la boucle *tant que* (lignes 3 à 11) nous choisissons la face f_2 qui a le moins d'occurrences parmi les faces qui restent dans F_1. Nous allons calculer tous les motifs fréquents contenant f_2 et ne contenant pas f_1 en parcourant le moins d'occurrences possibles. Au fur et à mesure des itérations de la boucle principale (lignes 3 à 11) nous explorons de plus en plus d'occurrences mais l'ensemble des faces possibles à ajouter aux motifs F_1 diminue, donc le nombre de candidats généré est plus faible, donc le nombre de calculs de signatures est plus faible, et donc l'algorithme est plus rapide.

Exemple 20. *La figure 4.8 illustre un exemple d'exécution de l'algorithme* mSpan. *Pour une base BC composées des trois cartes données, avec $\sigma = 2/3$ et $\varphi = 1$, les motifs en rouge (traits pleins) sont les motifs fréquents et les motifs en noir (pointillés) sont aussi générés par* mSpan*, mais ne sont pas fréquents. Les motifs d'une seule face sont tous générés lors d'un parcours complet des cartes de la base (ligne 1 Algo. 15). Le motif 3 composé de 6 brins est présent dans une seule des trois cartes, donc il n'est pas fréquent et ne fait pas parti de l'ensemble F_1, la même chose est vraie pour le motif 4. F_1 est donc composé des motifs 1 et 2. Ensuite, il faut choisir un motif parmi F_1 (ligne 4 Algo. 15) dans cette exemple c'est le motif 1 qui est choisi. L'algorithme* grow *permet de construire les motifs 4 et 5 en ajoutant une face appartenant à F_1 au motif 1. Le motif 5 est présent que dans une seule des trois cartes de BC et n'est donc pas fréquent, en revanche le motif 6 est fréquent et est ajouté à l'ensemble $Cand$ (ligne 9 Algo. 15). La boucle* tant que *(lignes 6 à 10 Algo. 15) est répété avec le motif 6, les candidats 7, 8, 9 et 10 sont générés et les motifs fréquents 9 et 10 sont ajoutés à l'ensemble $Cand$. La boucle* tant que *est répétée encore 3 fois avec les motifs 9, 10 et 14 et les motifs 11 à 18 sont générés. L'ensemble $Cand$ devient alors vide et le motif 1 est supprimé de F_1. Tous les motifs fréquents contenant la face 1 ont été calculés, lors de la deuxième boucle* tant que *(lignes 3 à 11 Algo. 15) tous les motifs contenant la face 2 seront calculés (les motifs 19 et 20). Notons que le motif 20 est une sous-carte du motif 14, mais, le fait de retirer le motif 1 de F_1 évite de le recalculer inutilement. Cependant, il peut tout de même arriver qu'un même motif soit calculé plusieurs fois,*

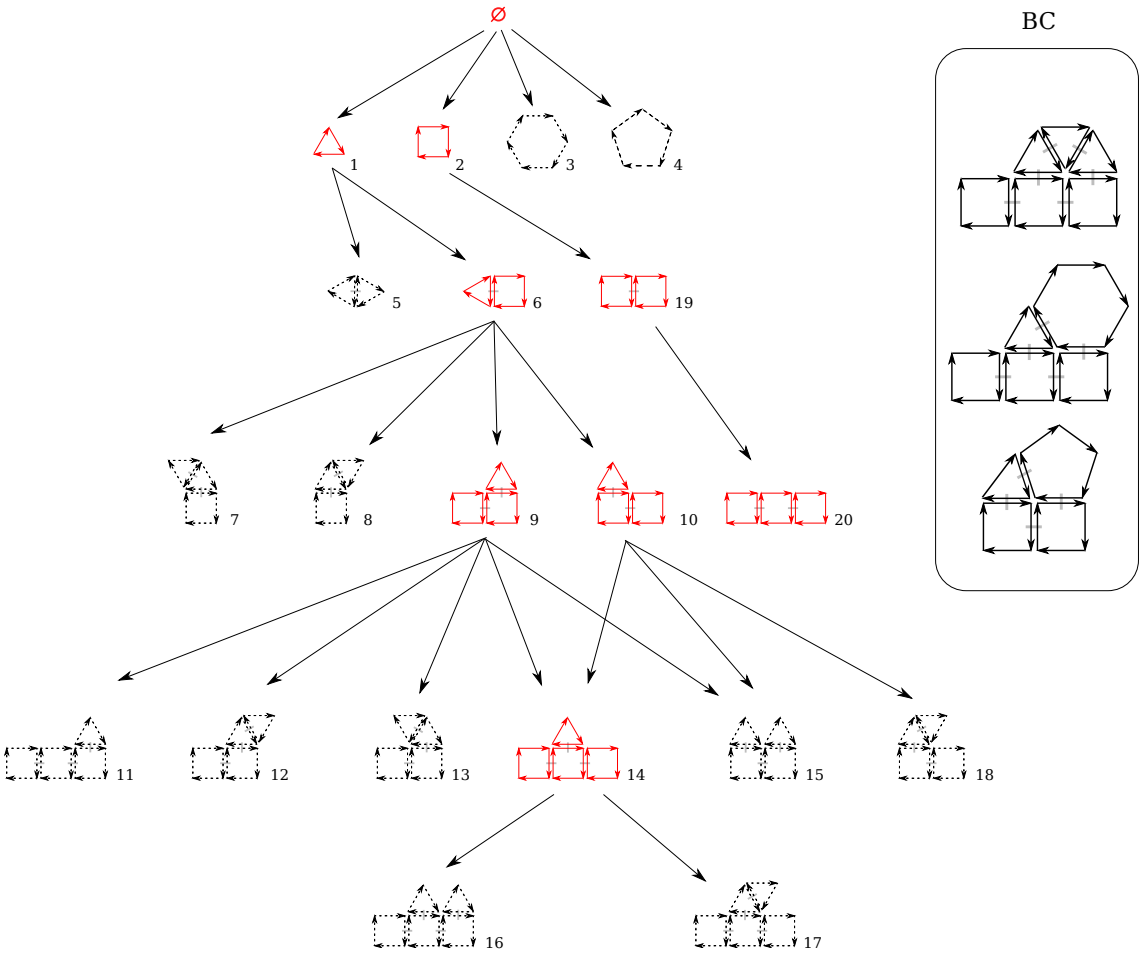


FIGURE 4.8 – Exemple de sous-cartes générées par *mSpan*. En rouge (trais pleins) les motifs fréquents pour $\sigma = 2/3$ et $\varphi = 1$ et pour la base de données BC composé de trois cartes. En noir (pointillés) les motifs non fréquents générés par *mSpan*. Il y a une flèche entre un motif a et un motif b ssi b a été généré par *mSpan* en ajoutant une face à a. Le numéro à coté d'un motif correspond à l'ordre dans lequel il a été découvert.

par exemple, les motifs 14 et 15 sont générés par l'algorithme grow *en ajoutant une face au motif 9, mais aussi à partir du motif 10.*

4.3.3 Complexité

Le calcul des motifs fréquents composés d'une seule face (ligne 1 de *mSpan*) se fait en $\mathcal{O}(\sum_{C \in BC} |C|)$, avec $|C|$ le nombre de brins de C, car il faut parcourir tous les brins de toutes les cartes de la base.

Dans l'algorithme *grow*, la boucle « pour » (ligne 2) est itéré au maximum $|C|$ fois, car nous stockons un brin différent pour chaque occurrence d'un motif, donc au maximum il y a autant d'occurrences que de brins dans la carte. La seconde boucle « pour » (ligne 3) est itéré au maximum $|m|$ fois (avec $|m|$ le nombre de brins du motif). Ensuite, pour chaque pas de boucle, il faut calculer

la signature du nouveau motif, ce qui est fait en $\mathcal{O}(|m|^2)$.

La complexité totale de l'algorithme *grow* est donc $\mathcal{O}(|m|^3 \cdot \sum_{C \in BC} |C|)$. L'algorithme *grow* est appelé pour chaque motif fréquent, il peut donc être appelé un nombre de fois exponentiel par *mSpan*. L'algorithme *mSpan* a donc une complexité polynomiale incrémentale.

4.3.4 Extension aux cartes nD

Pour simplifier la compréhension, nous avons décrit notre algorithme pour la cas d'extraction de sous-cartes fréquentes dans une base de cartes combinatoires $2D$. Cependant, comme pour l'algorithme *MapAPriori*, l'algorithme *mSpan* s'étend très simplement au cas des cartes nD. En n dimension, les motifs fréquents sont un ensemble connexe de n-cellules (c.-à-d. , un ensemble connexe de faces en $2D$, de volumes en $3D$, ...). Dans le cas d'extraction de sous-cartes fréquentes nD, il suffit de remplacer *faces* par *n-cellules* dans les algorithmes 15 et 16. Nous recherchons dans un premier temps tous les motifs fréquents composés d'une seule n-cellule (ligne 1 Algo. 15), puis nous construisons itérativement tous les motifs fréquents composés de ces n-cellules. La fonction *grow* construit de nouveaux motifs en ajoutant une n-cellule à un motif fréquent.

Nous avons présenté deux algorithmes pour résoudre le problème de recherche de motifs fréquents dans une base de cartes combinatoires. Nous ne savons pas donner les complexités exactes de nos algorithmes, cependant, *mSpan* semble avoir une meilleur complexité que *MapAPriori*. Dans la section suivante, nous vérifierons cela expérimentalement.

4.4 Expérimentations

Dans les sections précédentes nous avons défini deux algorithmes d'extraction de sous-cartes fréquentes, dans cette section nous allons étudier leur comportement expérimentalement.

Générateur de cartes. Pour évaluer les performances de nos algorithmes, nous avons créé un générateur de bases de cartes combinatoires $2D$. Ce générateur a 6 paramètres :

- $|BC|$: le nombre total de cartes ;
- T : la taille moyenne des cartes (en nombre de faces) ;
- I : la taille moyenne des sous-cartes potentiellement fréquentes (en nombre de faces) ;
- F : la taille moyenne des faces (en nombre de brins) ;
- $|P|$: le nombre de sous-cartes potentiellement fréquentes ;
- ρ : pourcentage de brins 2-libres.

L'idée principale de ce générateur est de créer un ensemble P de motifs potentiellement fréquents et de combiner ces motifs entre eux pour former une

base de cartes BC. Pour générer chaque élément de l'ensemble P de motifs potentiellement fréquents, nous choisissons le nombre de faces qui le composent selon une distribution de Poisson avec une moyenne de I. Nous choisissons le nombre de brins de chaque face selon une distribution de Poisson de moyenne F. Toutes les faces sont cousues entre elles pour former une carte connexe dont le pourcentage de brins 2-libres est proche de ρ.

Ensuite, pour générer chacune des $|BC|$ cartes de la base, nous choisissons aléatoirement des cartes dans P jusqu'à obtenir T faces. Puis nous cousons ces cartes entre elles jusqu'à obtenir une carte connexe ayant un pourcentage de brins 2-libres proche de ρ.

Ce générateur, peut être étendu pour générer des bases de 3-cartes en ajoutant un paramètre qui indique le nombre de volumes souhaité.

4.4.1 Comparaison des performances des algorithmes *mSpan* et *MapApriori.*

Pour une même base de cartes et les mêmes seuils, les deux algorithmes calculent les mêmes motifs fréquents, donc les seules choses à comparer sont les temps d'exécution et l'utilisation mémoire. Nous avons lancé les deux algorithmes sur différentes bases de cartes construites par notre générateur.

Étude de l'influence du nombre de cartes dans la base. Pour étudier l'influence du nombre de cartes sur le comportement de nos deux algorithmes, nous avons généré 5 bases de données en faisant varier le nombre de cartes et en fixant les autres paramètres. Les valeurs de $|BC|$ choisies sont 100, 500, 2500, 12500 et 62500. La figure 4.9 montre les temps d'exécution des algorithmes *mSpan* et *MapAPriori* sur ces 5 bases de données pour différentes valeurs de σ et pour $\varphi = 1$. Nous voyons très clairement que l'algorithme *MapAPriori* est bien moins performant que l'algorithme *mSpan*. Nous remarquons que plus le seuil σ est faible plus le temps d'exécution est long pour les deux algorithmes. Cela s'explique simplement par le fait que plus le seuil est petit, plus le nombre de motifs fréquents est grand. Nous remarquons aussi que plus il y a de cartes dans la base, plus le temps d'extraction est long. Les deux algorithmes parcourent toutes les cartes de la base dans un premier temps pour calculer les motifs fréquents composés d'une seule face (*mSpan* et *MapAPriori*) et de deux faces (*MapAPriori*). La complexité de ces opérations dépend donc du nombre de cartes dans la base. Plus il y a de cartes, plus il y a d'occurrences d'un même motif. Cela implique pour *MapAPriori* d'exécuter plus de fois la fonction d'isomorphisme de sous-cartes pour calculer la fréquence d'un motif, et pour *mSpan*, il y a plus d'occurrences à vérifier et à faire grandir d'une face, donc plus de calculs de signature à faire. Cela explique l'augmentation du temps de calcul pour les deux méthodes. Enfin, notons que l'algorithme *MapAPriori* n'est pas capable d'extraire les motifs fréquents quand $|BC|$ est supérieur à 12500, car l'espace mémoire nécessaire est trop grand.

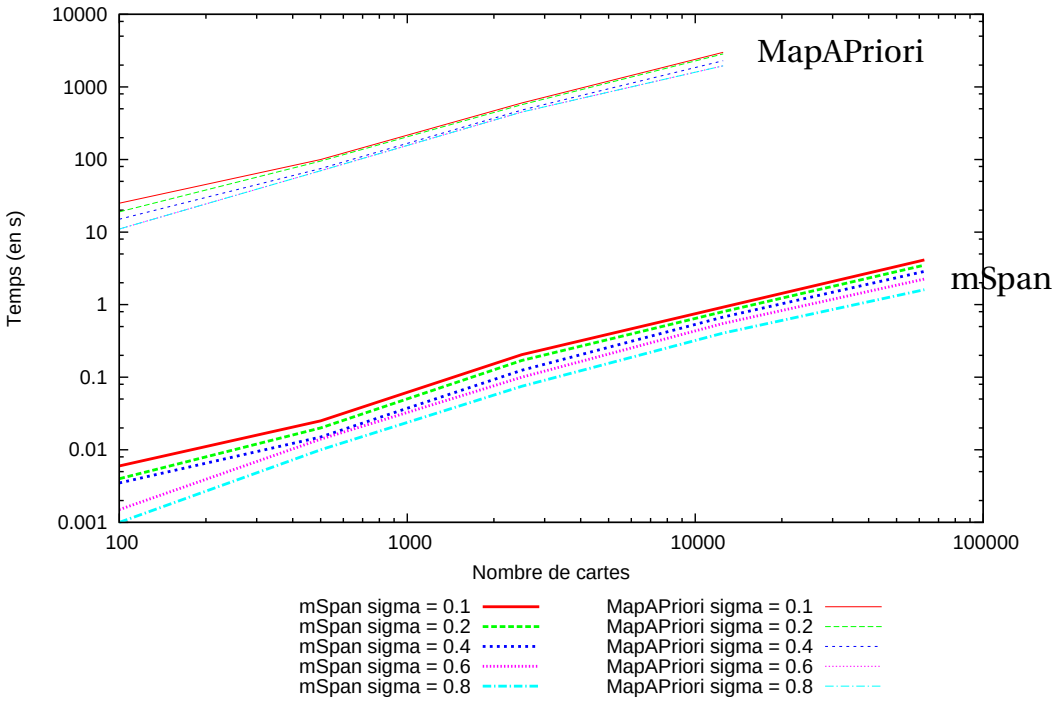
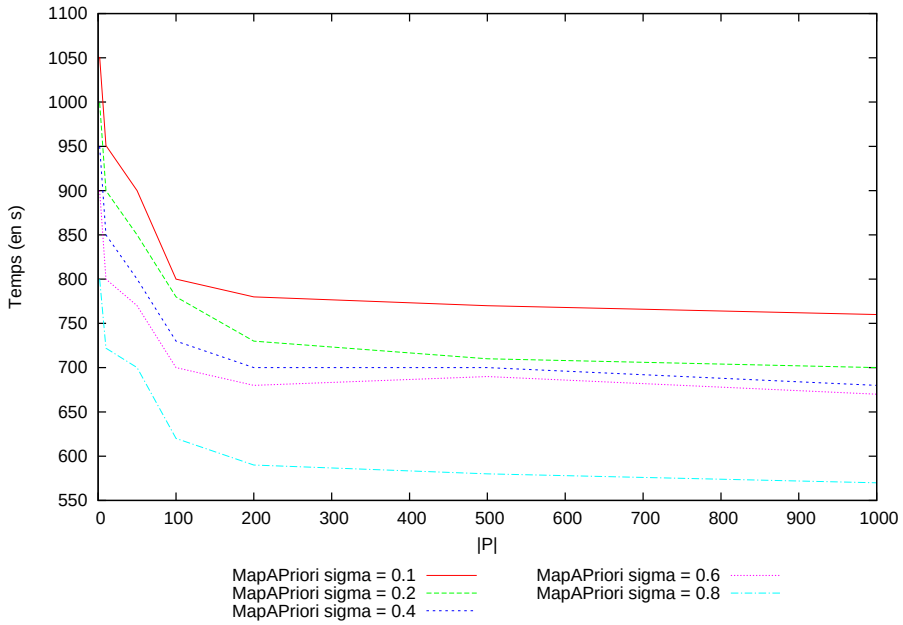


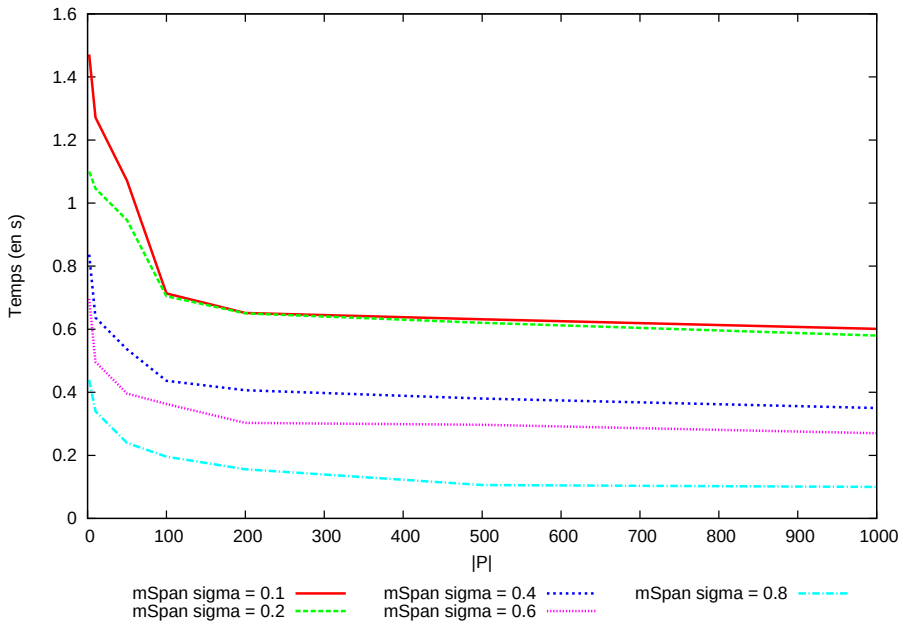
FIGURE 4.9 – Comparaison entre *mSpan* et *MapAPriori*. Étude de l'influence du nombre de cartes dans la base de données.

Étude de l'influence de l'homogénéité des cartes. Notre générateur permet de générer des bases de cartes plus ou moins homogènes en faisant varier le paramètre $|P|$. Plus cette valeur est faible, plus les cartes générées seront semblables, et plus $|P|$ est élevé, plus les cartes de la base seront disparates. Nous avons donc généré 5 bases de données en faisant varier le paramètre $|P|$ de 2 à 1000 et en fixant les autres paramètres. La figure 4.10 montre les temps d'exécution des algorithmes *mSpan* et *MapAPriori* sur ces bases de cartes pour différentes valeurs de σ et φ fixé à 1. *mSpan* étant bien plus rapide que *MapAPriori*, nous présentons les résultats sur deux graphiques avec des échelles différentes. Nous remarquons que les deux algorithmes ont le même comportement global : plus les cartes sont semblables, plus il y a de motifs fréquents et donc plus le temps d'extraction de ces motifs est important.

Étude de l'influence de la densité des cartes. Nous considérons que plus une carte possède de brins libres, moins cette carte est dense et inversement. Pour montrer l'influence de la densité des cartes dans l'extraction de sous-cartes fréquentes, nous avons généré plusieurs base de cartes en faisant varier le paramètre ρ entre $0,2$ et 1. Une densité de 0 signifie qu'aucun brin n'est 2-cousu, or nous ne générons que des cartes connexes, c'est pourquoi nous choisissons comme densité minimale $0,2$. Dans ce cas, en moyenne, nous générons des cartes connexes, où chaque face est 2-cousue avec une seule autre face. Une densité de 1 signifie que tous les brins d'une carte sont 2-cousus à un autre. La figure 4.11 montre le nombre de motifs fréquents en fonction de la densité pour différentes valeurs



(a)



(b)

FIGURE 4.10 – Comparaison entre *mSpan* et *MapAPriori*. Étude de l'influence de l'homogénéité des cartes dans la base de données. Le graphique (a) présente les temps d'exécution de l'algorithme *mSpan* et le (b) ceux de l'algorithme *MapAPriori*.

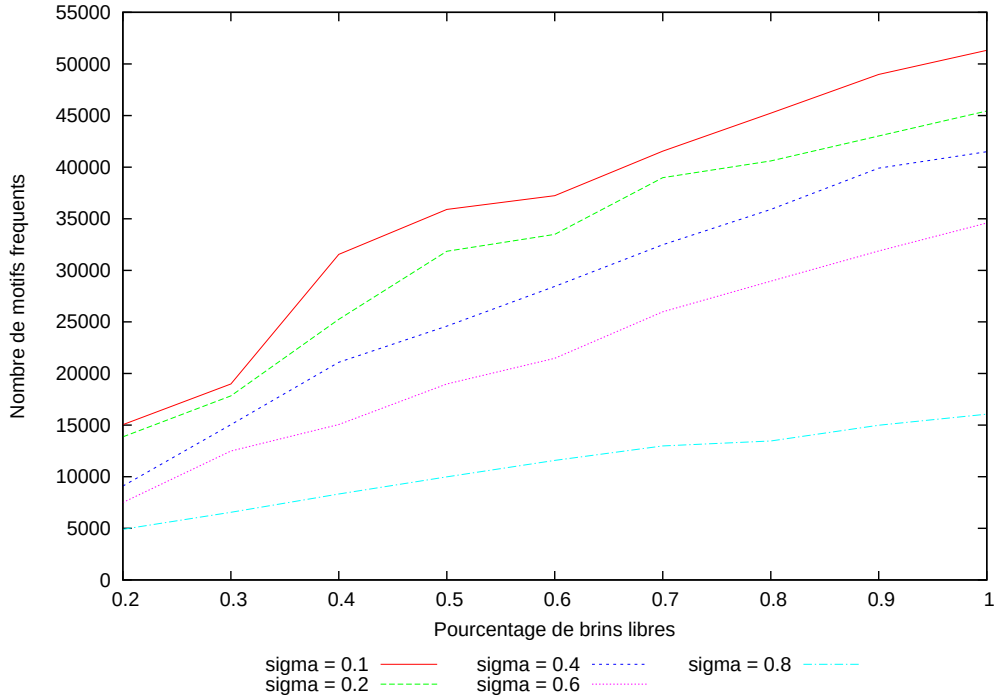


FIGURE 4.11 – Nombre de motifs fréquents en fonction de la densité des cartes dans la base de données.

de σ et pour $\varphi = 1$. Nous remarquons que plus la densité des cartes est grande, plus le nombre de motifs fréquents est élevé. Notons que comme pour les expérimentations précédentes, l'algorithme *mSpan* est beaucoup plus rapide que l'algorithme *MapAPriori* et que les temps d'exécutions des deux algorithmes varient de la même façon que le nombre de motifs fréquents.

Explication des performances de *mSpan* et *MapAPriori*. Pour expliquer cette grande différence de performance entre *mSpan* et *MapAPriori*, nous avons effectué une série d'expériences sur 20 bases de cartes générées avec un échantillon de paramètres variés. Nous avons noté qu'en moyenne, 80% du temps d'exécution de *MapAPriori* est passé dans le calcul d'isomorphisme de sous-cartes. De plus nous avons remarqué que l'algorithme génère beaucoup de candidats qui ne sont présent dans aucune des cartes de la base (c.-à-d. leur fréquence est nulle), ce qui augmente inutilement le nombre d'appels à la fonction d'isomorphisme. Environ 70% des appels de la fonction d'isomorphisme sont fait pour des motifs qui ne sont pas présent une seule fois dans la base. Ce problème n'est pas présent dans l'algorithme *mSpan*. En effet, *mSpan* ne génère que des motifs qui sont présent au moins une fois dans la base, cependant nous avons noté que l'algorithme pouvait générer jusqu'à 120% des motifs en double dans le cas de cartes très dense ($\rho > 0,9$) et jusqu'à 50% dans le cas de densité plus faible.

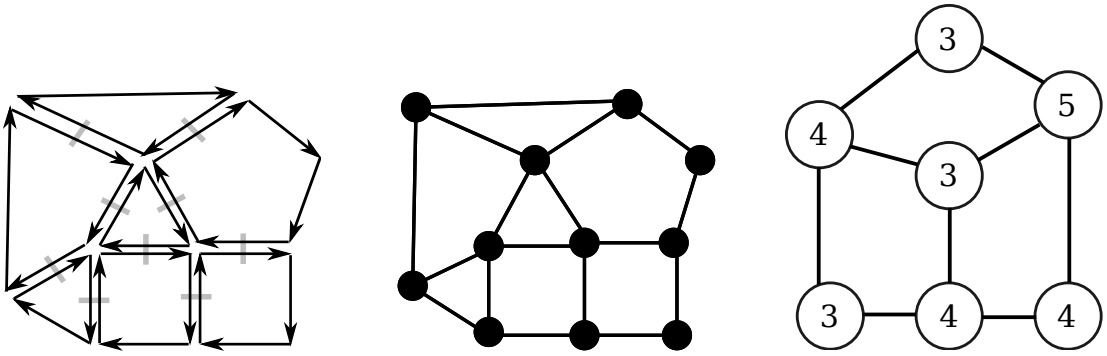


FIGURE 4.12 – Exemple de graphes (primal et dual étiqueté) associés à une carte.

4.4.2 Comparaison entre les cartes et les graphes (mSpan Vs gSpan).

Le problème d'extraction de sous-cartes fréquentes étant un nouveau problème, nous ne pouvons pas comparer notre méthode avec une autre méthode existante. Cependant, le problème d'extraction de sous-graphes fréquents est très proche du notre. Il est donc intéressant de comparer notre algorithme avec l'algorithme *gSpan*.

Nous comparons *mSpan* avec *gSpan*[2], qui est parmis les meilleurs algorithmes sur des données synthétiques pour extraire des sous-graphes fréquents dans une base de graphes [55]. Notons que *mSpan* et *gSpan* résolvent des problèmes différents avec des problématiques de complexité différentes : l'isomorphisme de sous-cartes a une complexité polynomiale tandis que l'isomorphisme de sous-graphe est un problème NP-complet. Étant donné une 2-carte, nous pouvons définir de manière directe le graphe primal correspondant : chaque 0-cellule de la carte correspond à un sommet du graphe et il existe une arête entre deux sommets dans le graphe s'il existe une 1-cellule dans la carte entre les deux 0-cellules correspondantes (voir Fig. 4.12). Cependant, calculer les motifs fréquents dans le graphe primal n'a pas vraiment de sens pour comparer *gSpan* avec notre algorithme car les motifs extraits ne sont pas comparable avec les motifs extraits par notre algorithme. En effet, les motifs extraits par *mSpan* sont des ensembles connexes de faces tandis que ceux extraits par *gSpan* sont des sous-graphes connexes qui peuvent ne pas correspondre à un ensemble de faces. Pour une comparaison plus juste, nous considérons le graphe dual qui associe un sommet pour chaque face de la 2-carte, et il existe une arête entre deux sommets si les deux faces correspondantes de la carte sont adjacentes. Nous ajoutons une étiquette à chaque sommet du graphe qui est l'identifiant de la face correspondante. Deux faces ont deux étiquettes identiques si et seulement si elles sont isomorphes. Les motifs extraits par *gSpan* sur le graphe dual sont des sous-graphes, ce qui correspond bien à un ensemble de faces dans la 2-carte. Les étiquettes sur les sommets permettent à *gSpan* de différencier les

2. implémentation des auteurs sur http ://www.cs.ucsb.edu/∼xyan/software/gSpan.htm

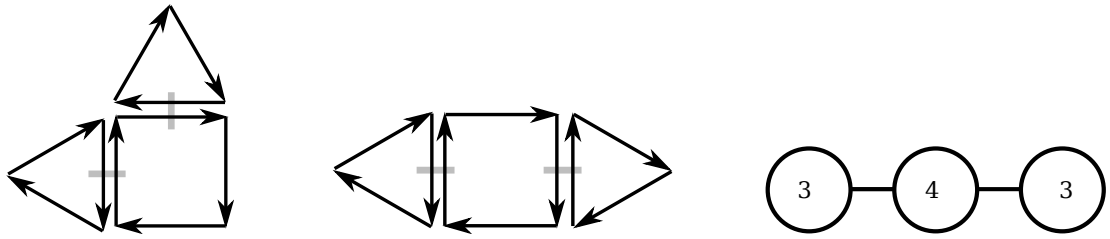


FIGURE 4.13 – Exemple de deux cartes différentes qui ont le même graphe dual.

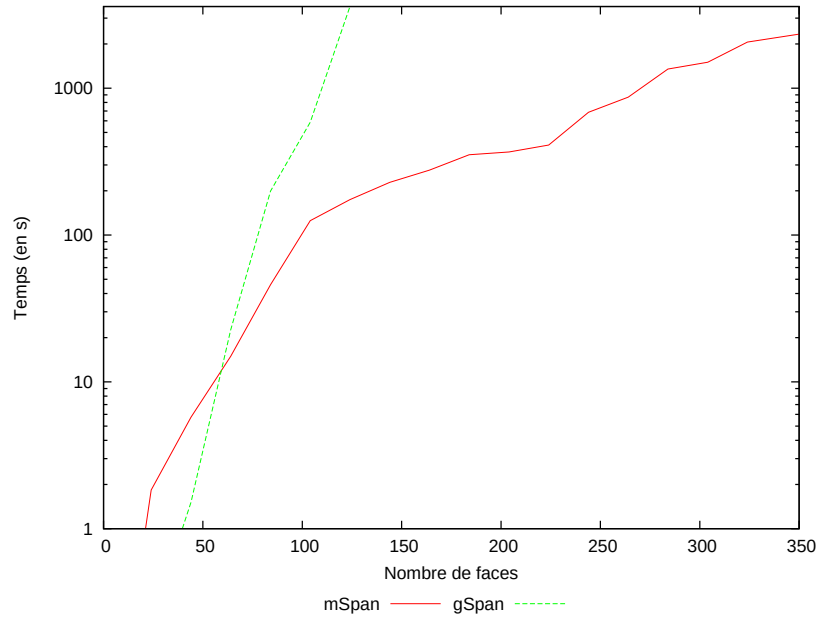
faces différentes et augmente ses performances. Cependant, *gSpan* ne considère pas la topologie des graphes, c.-à-d. l'ordre dans lequel les faces sont positionnées autour d'une autre face. Pour cette raison, deux cartes différentes peuvent correspondre à un même graphe dual, comme l'illustre la figure 4.13. Donc, *mSpan* et *gSpan* ne calculent pas les même motifs fréquents.

Pour des bases de 3-cartes, nous générons aussi les graphes duaux associés. Chaque sommet du graphe correspond à un volume de la carte et il existe une arête entre deux sommets ssi les deux volumes sont adjacents. L'étiquette associé à chaque sommet correspond à l'identifiant du volume (c.-à-d. , deux volumes ont le même identifiant si et seulement s'ils sont isomorphes). De cette manière, un sous-graphe connexe correspond à un ensemble connexe de volumes dans la 3-carte. Cependant, comme en $2D$, plusieurs 3-cartes différentes peuvent produire le même graphe, donc les motifs extraits sont différents.

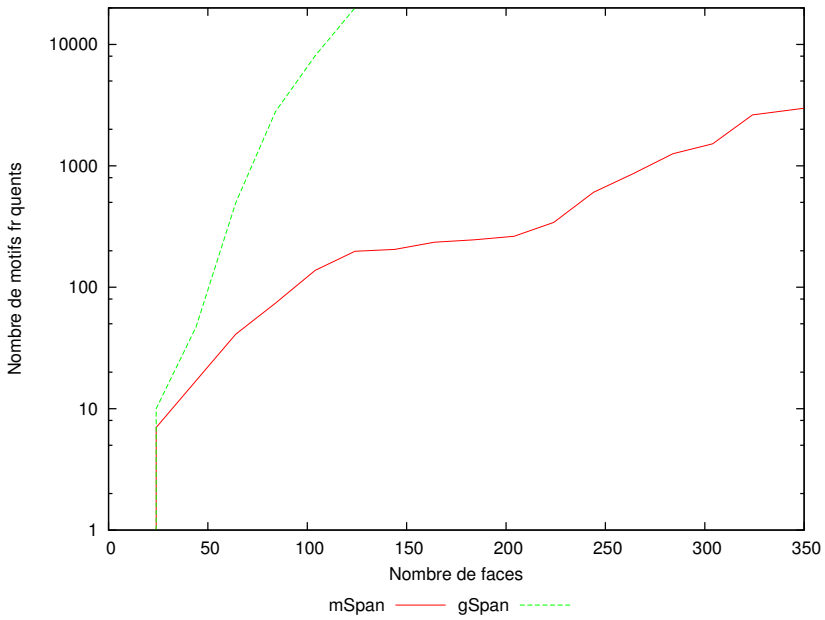
Notons que les graphes sont bien plus petit que les cartes correspondantes. En moyenne dans les cartes que nous avons générés, une 2-carte (resp. 3-carte) composée de 350 faces (resp. 80 volumes) a 1800 (resp. 1200) brins, tandis que le graphe dual associé a 800 (resp. 160) arêtes et 350 (resp. 80) sommets.

Comparaison entre *mSpan* et *gSpan* lorsque la taille des cartes augmente. Les courbes des figures 4.14 et 4.15 montre les résultats de *mSpan* et *gSpan* sur des bases $2D$ et $3D$ pour $\sigma = 0,9$ lorsque l'on augmente le nombre de faces de 4 à 350 en $2D$ et le nombre de volumes de 2 à 80 en $3D$. Chaque exécution est limité à 3600 secondes. *mSpan* extrait tous les motifs fréquents avant la limite de temps dans tous les cas. *gSpan* est plus rapide que *mSpan* pour un nombre de faces inférieur à 50 en $2D$ et un nombre de volumes inférieur à 30 en $3D$. Cependant, pour des cartes plus grandes il devient plus lent et il n'est pas capable de calculer tous les motifs fréquents avant le temps limite de 3600 secondes pour des cartes de plus de 120 faces en $2D$ et plus de 50 volumes en $3D$. En fait, *gSpan* extrait beaucoup plus de motifs que *mSpan* et plus les cartes sont grandes, plus cette différence est importante. Cela vient du fait que les graphes ne représentent pas la topologie, donc différentes sous-cartes qui ne sont pas fréquentes correspondent à un même sous-graphe qui peut alors devenir fréquent.

Comparaison entre *mSpan* et *gSpan* lorsque le seuil de fréquence minimale σ varie. Les courbres de la figure 4.16 montrent les résultats de *mSpan* et *gSpan* sur une base de cartes $2D$ lorsque le seuil σ varie de $0,1$ à 1. Nous voyons que

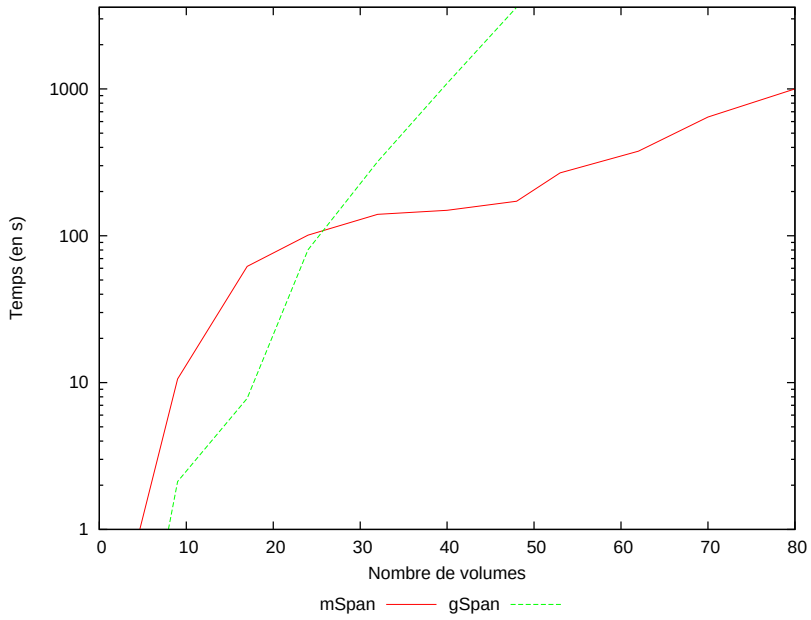


(a)

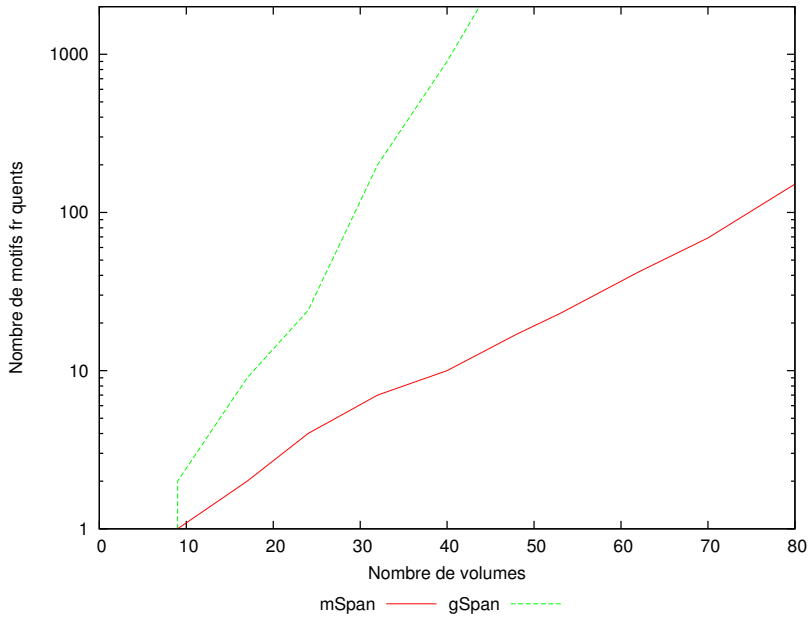


(b)

FIGURE 4.14 – Comparaison de *mSpan* et *gSpan* sur une base de données $2D$: les courbes (a) (resp. (b)) donnent l'évolution du temps d'exécution (resp. du nombre de motifs extraits) en fonction du nombre de faces des cartes de la base.

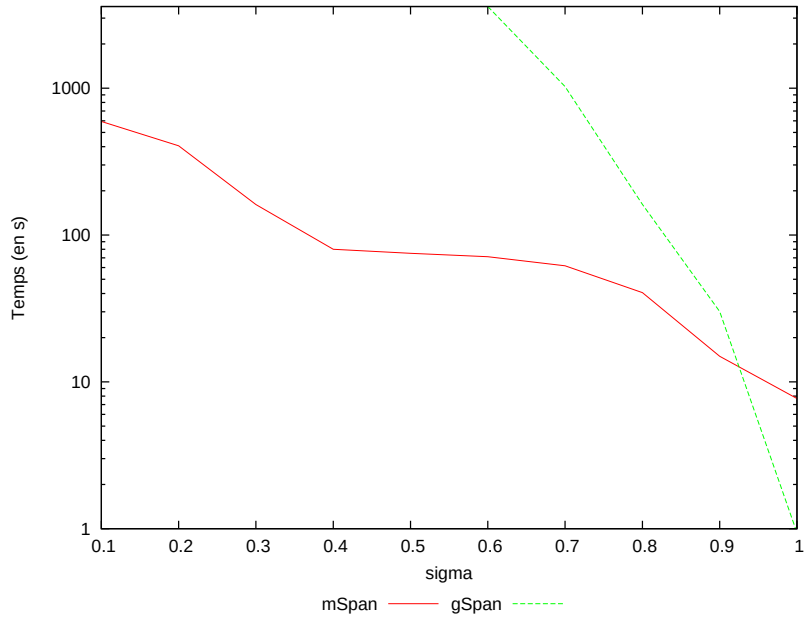


(a)

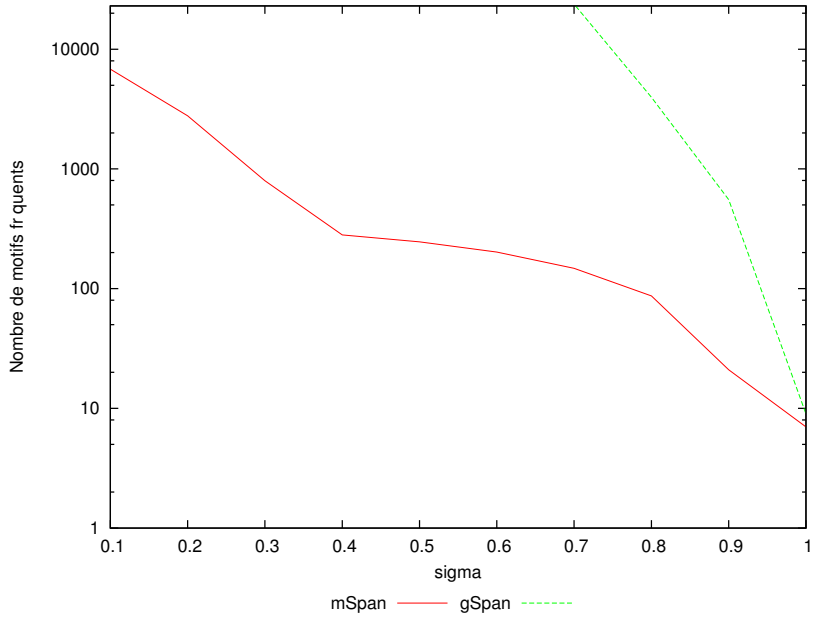


(b)

FIGURE 4.15 – Comparaison de *mSpan* et *gSpan* sur une base de données $3D$: les courbes (a) (resp. (b)) donnent l'évolution du temps d'exécution (resp. du nombre de motifs extraits) en fonction du nombre de volumes des cartes de la base.



(a)



(b)

FIGURE 4.16 – Comparaison de *mSpan* et *gSpan* sur une base de données $2D$: les courbes (a) (resp. (b)) donnent l'évolution du temps d'exécution (resp. du nombre de motifs extraits) en fonction du seuil σ.

gSpan est plus rapide que *mSpan* lorsque σ est supérieur à $0,9$, mais il est plus lent pour des valeurs plus petites et il n'est plus capable de calculer tous les motifs fréquents dans la limite de 3600 secondes pour des valeurs de σ inférieur à $0,7$. Le nombre de motifs fréquents augmente beaucoup plus rapidement pour *gSpan* que pour *mSpan* lorsque le seuil de fréquence σ diminue.

Dans ce chapitre, nous avons défini le problème de recherche de motifs fréquents dans une base de cartes combinatoires et nous avons proposé deux algorithmes pour résoudre ce problème. Nous avons montré que l'algorithme *mSpan* est plus efficace que l'algorithme *MapAPriori* et qu'il a une complexité en temps polynomiale incrémentale. Dans le chapitre suivant, nous utilisons l'algorithme *mSpan* pour calculer les motifs fréquents dans des bases de cartes combinatoires dans le but de faire de la classification.

Chapitre 5

Expérimentations

Sommaire

5.1 Utilisation de motifs fréquents pour la classification 85
5.2 Données synthétiques 89
5.3 Images 90
5.4 Documents manuscrits 95
5.5 Pavages 97

DANS le chapitre précédent, nous avons présenté deux algorithmes d'extraction de sous-cartes fréquentes et nous avons remarqué que l'algorithme *mSpan* est plus efficace que l'algorithme *MapApriori*. Dans ce chapitre, nous présentons quelques applications potentielles de cet algorithme. Ce chapitre ne présente pas une réelle contribution, mais plutôt des perspectives avancées. Nous présentons principalement une piste qui est la classification à partir d'un ensemble de sous-cartes fréquentes. Dans la section 5.1, nous présentons notre classifieur basé sur les sous-cartes fréquentes que nous allons utiliser dans les sections 5.2, 5.3 et 5.4 sur, respectivement, des données synthétiques, des images et des documents manuscrits. Enfin dans la section 5.5 nous montrons que les cartes combinatoires peuvent modéliser des pavages et que l'extraction de motifs fréquents peut permettre de les caractériser.

5.1 Utilisation de motifs fréquents pour la classification

Plusieurs méthodes de classification utilisent des motifs fréquents, que ce soient des ensembles d'attributs [3,9,20,31,37] ou des sous-graphes [15,56–58].

Le principe de ces méthodes se décompose en trois points :

1. extraction de motifs ;
2. sélection d'un sous-ensemble des motifs extraits ;
3. construction d'un classifieur à partir de ce sous-ensemble de motifs.

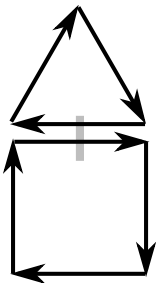
Nous allons reprendre ce principe et l'appliquer à deux bases d'images pour faire de la classification supervisée et non supervisée. L'étape d'extraction de motifs (étape 1) est réalisée à l'aide de l'algorithme *mSpan* sur l'ensemble de la base de cartes. Nous n'avons pas mis au point de méthode de filtrage des motifs, donc dans l'étape 2, nous gardons tous les motifs fréquents extraits. Lors de l'étape 3, nous transformons chaque carte de la base en un vecteur. Soit x le nombre de motifs fréquents extraits. Chaque vecteur est de dimension x et chaque composante i du vecteur V_C correspond au nombre d'occurrences du motif i dans la carte C. La figure 5.1 donne un exemple de transformation d'une carte en vecteur.

Ensuite, à partir de cet ensemble de vecteurs, nous appliquons des méthodes standards de classification telles que *C4.5* [49] pour la classification supervisée et *kmeans* [39] pour la classification non supervisée.

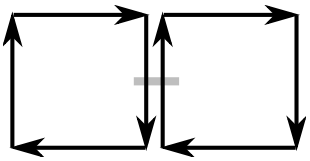
C4.5 L'algorithme C4.5 construit un arbre de décision à partir d'une matrice $Objets \times Attributs$. Le but est de trouver à chaque embranchement les valeurs d'attributs qui différencient au mieux une classe des autres. Sur l'exemple de la figure 5.2, il y a 6 objets répartis en 3 classes et chaque objet possède 3 attributs. Les objets de la classe C_1 sont définis par une valeur de V_1 inférieure à 15, les objets de la classe C_2 sont définis par une valeur de V_1 supérieure à 15 et une valeur de V_2 inférieure à 12, enfin, les objets de la classe C_3 sont définis par une valeur de V_1 supérieure à 15 et une valeur de V_2 supérieure à 12. Soit un objet o qui a pour valeurs $V_1 = 16$,$V_2 = 13$ et $V_1 = 11$, en utilisant ce classifieur, cet objet sera prédit comme étant de la classe C_3.

k-means Le but de l'algorithme *k-means* est de diviser les données réparties sur un espace vectoriel en k partitions (clusters) où chaque objet appartient au cluster dont le barycentre est le plus proche. La figure 5.3 montre un exemple de données réparties sur le plan, et donne les clusters obtenus par *k-means* pour $k = 3$. Le centre d'un cluster est représenté par une croix.

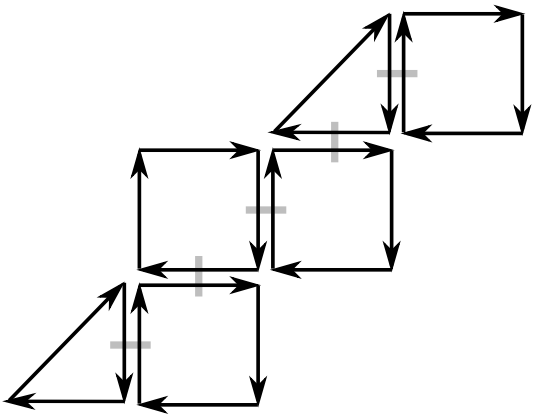
Un classifieur ne prédit pas toujours la bonne classe d'un objet, c'est pourquoi il est indispensable de construire un classifieur à partir d'un jeu de données appelé *données d'apprentissages* et de vérifier pour un autre jeu de données dont nous connaissons la classe si la classe prédite par le classifieur est bien la classe réelle de l'objet. Ces objets sont appelés *données de test*. Il existe différentes méthodes de validation d'un classifieur. Nous avons choisi d'utiliser la validation croisée. Pour un jeu de données connues, nous enlevons 10% des données pour les tests et nous utilisons les 90% restants comme données d'apprentissage. Nous recommençons le même processus 10 fois, mais à chaque itération, les objets qui font partie du jeu de test sont différents. Le classifieur retenu sera celui qui produit le moins d'erreurs parmi les dix.



(a) M^1



(b) M^2



(c) C

$$V_C =< 3, 4 >$$

FIGURE 5.1 – Exemple de transformation d'une carte en vecteur. Soit C une carte qui appartient à une base de cartes, M^1 et M^2 sont les deux motifs fréquents extraits pour cette base. Le vecteur correspondant à C est de dimension 2 car il y a deux motifs fréquents, la première composante vaut 3, soit le nombre d'occurrence du motif M^1 dans C et la seconde vaut 4 car le motif M^2 est présent 4 fois dans C.

	V_1	V_2	V_3	Classe
A	15	18	5	C_1
B	12	8	14	C_1
C	25	5	3	C_2
D	32	12	25	C_2
E	21	15	8	C_3
F	33	18	13	C_3

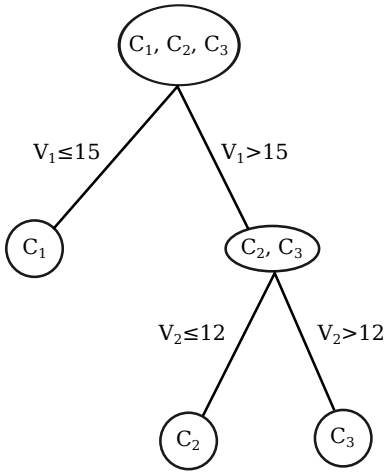


FIGURE 5.2 – Exemple d'arbre de décision.

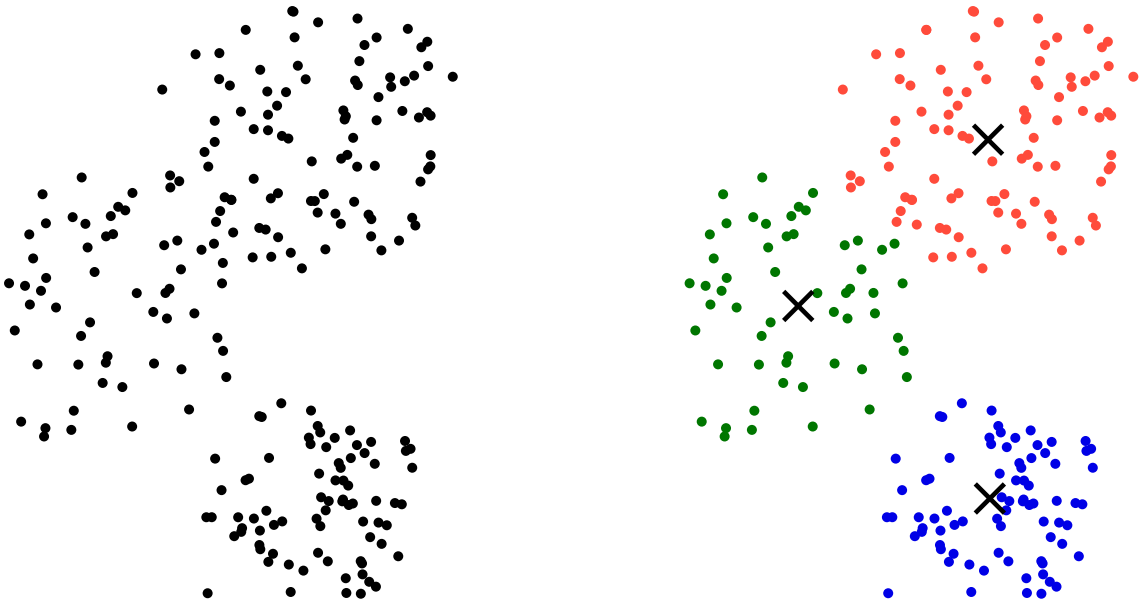


FIGURE 5.3 – Exemple de résultats de l'algorithme *k-means*. (a) Les données d'origine. (b) Les 3 clusters calculés par *k-means*.

Comme nous l'avons vu sur le schéma de processus d'extraction de connaissances (figure 1.1), les étapes d'extraction de motifs et d'interprétation de ces motifs (ici, fabriquer un classifieur), ne sont que des étapes d'un processus beaucoup plus complexe. Dans le cas de la classification d'images, il y a une étape importante qui consiste à transformer une image en carte combinatoire (nous décrivons succinctement cette étape dans la section 5.3). Si cette étape n'est pas robuste, c.-à-d. si elle ne permet pas d'obtenir deux cartes combinatoires « proches » si les deux images sont « proches », alors l'opération de classification n'a aucune chance d'être correcte. C'est pour cela que nous allons dans un premier temps valider les étapes d'extraction de motifs et de construction du classifieur sur des données synthétiques. Nous pourrons alors facilement faire varier le nombre de classes, l'homogénéité intra-classes (c.-à-d. la similarité entre cartes d'une même classe) et inter-classes (c.-à-d. la similarité entre cartes de classes différentes).

5.2 Données synthétiques

À l'aide du générateur de cartes que nous avons décrit dans la section 4.4, nous pouvons générer des bases de cartes qui contiennent différentes classes. Pour rappel, le générateur que nous avons proposé génère de façon aléatoire un ensemble P de cartes. Ensuite, nous combinons entre elles des cartes de P pour former les cartes de la base BC. Pour créer artificiellement des classes, nous allons restreindre l'ensemble P des motifs utilisés pour créer les cartes d'une classe. Par exemple, pour construire une base de n classes, nous construisons un ensemble de cartes aléatoire P. Nous décomposons P en n sous-ensembles $P_1, \cdots, P_n \subseteq P$. Pour construire les cartes d'une classe $i \in [1; n]$, nous combinons des cartes de l'ensemble P_i. Nous pouvons donc modifier les propriétés de la base en faisant varier la taille de P et la taille des P_i. Pour augmenter la diversité inter-classes il suffit d'augmenter la taille de P sans changer la taille des P_i. Pour augmenter la diversité intra-classes il suffit d'augmenter la taille du sous-ensemble P_i. En effet, plus la taille de P_i est petite, plus les cartes de la classe i seront composée des mêmes motifs.

Nous avons effectué des tests sur différentes bases de cartes, et nous présentons ici seulement un exemple représentatif du fonctionnement de notre classifieur. Nous avons généré un ensemble de 25 motifs de 10 faces, puis nous avons généré 10 classes de cartes de 100 faces. La combinaison des motifs utilisés pour chacune des classes est donnée dans le tableau 5.1. Pour construire une carte de 100 faces, il faut combiner 10 motifs de 10 faces et chacun des 10 est choisi au hasard parmi le sous-ensemble des 5 motifs choisi pour une classe. Si un motif est présent une fois dans le tableau, cela signifie que la probabilité de choisir ce motif est de 1/5, s'il est présent deux fois, il a une probabilité de 2/5.

Pour cette base de cartes, le tableau 5.2 donne la matrice de confusion de notre classifieur. Pour des seuils $\varphi = 1$ et $\sigma = 0.1$, *mSpan* a extrait 416 motifs fréquents. Le taux de classification sur l'ensemble de la base est de 93% et les deux classes qui posent le plus de problèmes sont les classes 6 et 9. Nous remar-

	Numéro de motif				
classe 0	5	5	17	19	24
classe 1	7	7	14	19	24
classe 2	0	5	8	19	23
classe 3	2	3	6	13	13
classe 4	4	6	16	17	18
classe 5	7	12	13	20	24
classe 6	0	2	3	11	13
classe 7	2	7	8	8	11
classe 8	5	8	9	18	24
classe 9	0	2	11	13	16

TABLE 5.1 – Définition des classes. Chaque classes est composé de 5 motifs parmi 25.

classés en / classes réélles	0	1	2	3	4	5	6	7	8	9
0	95	0	5	0	0	0	0	0	0	0
1	0	91	2	0	0	6	0	1	0	0
2	9	0	91	0	0	0	0	0	0	0
3	0	0	0	100	0	0	0	0	0	0
4	0	0	0	0	100	0	0	0	0	0
5	0	3	0	0	0	95	0	2	0	0
6	0	0	0	2	0	0	86	0	0	12
7	0	1	0	0	0	0	0	99	0	0
8	2	0	0	0	0	0	0	0	97	0
9	0	0	0	0	0	0	12	3	0	85

TABLE 5.2 – Résultat de la classification. Les valeurs sur la diagonale correspondent au taux de bonne classification pour chaque classe.

quons que ces deux classes partagent 4 motifs sur les 5 utilisés pour construire les cartes de ces classes (voir tableau 5.1), ce qui explique que notre classifieur confonde ces deux classes.

Comme nous le montrent ces résultats, dans le cas idéal où toute l'information voulue est contenue dans les cartes, l'utilisation de motifs fréquents est une piste très intéressante pour la classification. En effet, même pour un nombre de classes assez élevé, le taux de classification est supérieur à 90%. Avec des bases de cartes plus « simples » (avec moins de classes ou avec une plus grande diversité inter-classes), le taux de classification peut atteindre 100%.

5.3 Images

Dans cette section, nous allons appliquer notre classifieur sur des bases d'images. Une carte peut être extraite à partir d'une image segmentée en utilisant

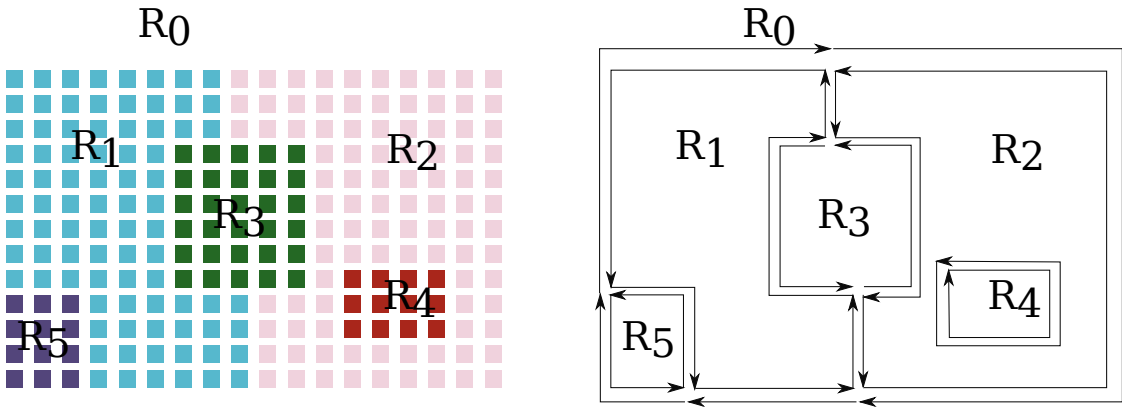


FIGURE 5.4 – Exemple de transformation d'une image segmentée en carte combinatoire $2D$. (a) L'image segmentée. (b) la 2-carte correspondant à l'image.

l'algorithme décrit dans [11]. Le principe de cet algorithme est de créer une face pour chaque région de l'image. Pour ce faire, chaque pixel de l'image est transformé en une face carrée. Ensuite, deux faces sont fusionnées si elles sont adjacentes et qu'elles sont de couleur proche. La figure 5.4 illustre la transformation d'une image (ensemble de pixels) en une 2-carte. Chaque région de l'image est représentée par une face externe. Une région peut aussi contenir des faces internes si elle contient d'autres régions. Sur l'exemple, la région R_2 est composée d'une face interne car la région R_4 est entièrement incluse dans R_2. Toutes les autres régions ne sont représentées que par une face externe. Enfin, chaque brin correspond a une frontière entre deux régions.

Base PASCAL La base *PASCAL*[1] est une base d'images qui a fait l'objet de challenges en classification d'images [18]. La base d'images que nous utilisons est composée de 326 images réparties en 3 classes différentes, le tableau 5.3 montre un extrait de cette base.

Le taux de classification de notre méthode pour cette base d'images est de 80%. Ce résultat ne permet pas de rivaliser avec les méthodes actuelles de classification d'images. Cependant, il est intéressant de noter que la seule information que nous prenons en compte est l'information topologique de l'image, les informations de couleurs et de géométries étant totalement ignorées. Cela signifie qu'il y a beaucoup d'information juste dans la topologie, mais que cette information n'est pas suffisante, comme l'information de couleur seule n'est pas suffisante, ni la géométrie seule. Nous pouvons donc présager qu'une approche hybride intégrant ces trois types d'informations (topologie, couleur, géométrie) serait plus efficace encore. Ce point est une de nos perspectives de recherche.

À partir des mêmes vecteurs construits avec les motifs fréquents, il est possible d'utiliser les techniques de classification non-supervisée. Le tableau 5.5

1. disponible sur http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2005/index.html

TABLE 5.3 – Échantillon de la base d'images *Pascal*. Elle est composée de 326 images réparties en 3 classes.

classe réelle \ classé	Voiture	Vache	Moto
Voiture	86	5	9
Vache	8	85	7
Moto	18	11	71

TABLE 5.4 – Matrice de confusion pour la classification supervisée, en utilisant l'algorithme C4.5.

classe réelle \ cluster	1	2	3
Voiture	70	18	12
Vache	22	75	14
Moto	27	37	51

TABLE 5.5 – Matrice de confusion pour la classification non-supervisée, en utilisant l'algorithme k-means.

donne la matrice de confusion du clustering de la base. L'algorithme utilisé est l'algorithme *k-means*. Le taux global de classification chute à 60% alors que pour la classification supervisée le taux est de 80%. Cette chute du taux de classification pour la classification non supervisée s'explique simplement par le fait que l'information de classe n'est pas utilisée dans le jeu de données d'apprentissage alors qu'elle est prise en compte lors de la classification supervisée. Nous remarquons que c'est la classe *Moto* qui pose le plus de problèmes pour les deux méthodes, car de nombreux motifs sont communs à la fois avec la classe *Voiture* et avec la classe *Vache*.

Flatland Nous avons réalisé des tests sur une seconde base d'images, où toutes les images sont extraites du film d'animation « Flatland : The Movie [2] ». Nous avons extrait une image toutes les 0,5 seconde du film, puis nous avons gardé un sous-ensemble de ces images que nous avons décomposé en 127 classes. Chaque classe correspond à une scène et comporte de 15 à 300 images, sachant qu'il y a 8349 images au total. Nous considérons qu'il y a un changement de scène lorsqu'il y a un changement de décor. Le tableau 5.6 montre quelques images de 5 scènes du film.

Nous avons créé un classifieur dont le taux de bonne classification est de 65%. En étudiant la matrice de confusion, nous nous sommes aperçu que certaines classes étaient souvent confondues. Ces classes correspondent à des scènes éloignées en temps dans le film, mais qui représentent les mêmes décors et les mêmes personnages. En fusionnant de telles classes, le taux de bonne classification monte à 76% pour 108 classes, ce qui est un bon résultat par rapport à un classifieur aléatoire qui ferait moins de 1%.

Cette base d'images est plus simple que la base d'images *PASCAL* car les images sont très proches entre elles, et que la segmentation est plus efficace sur les dessins que sur des photos, car les régions sont plus contrastées. Nous avons appliqué l'algorithme *k-means* sur le sous-ensemble de 5 scènes du tableau 5.6. Le tableau 5.7 donne la matrice de confusion pour ces images. Le taux de classification est de 91.4% pour les 5 classes, ce qui est un résultat bien meilleur que pour la base *PASCAL*.

Ces résultats montre que notre méthode est plus ou moins efficace selon la base d'images. Nous pouvons conclure de ces résultats que notre méthode fonctionne mieux quand les images intra-classes sont proches.

2. disponible sur www.flatlandthemovie.com

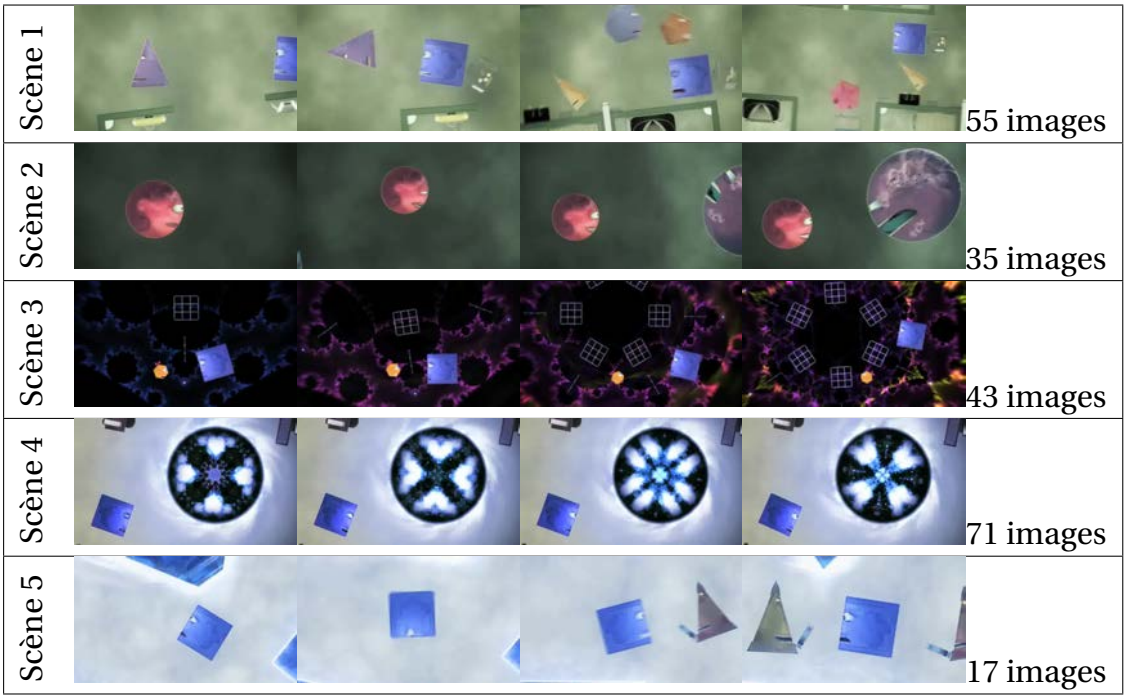


TABLE 5.6 – Échantillon de la base d'images *Flatland*.

classe réelle \ cluster	1	2	3	4	5
scène 1	51	0	4	0	0
scène 2	0	21	14	0	0
scène 3	0	1	42	0	0
scène 4	0	0	0	71	0
scène 5	0	0	0	0	17

TABLE 5.7 – Matrice de confusion pour la classification non-supervisée pour 5 scènes de la base d'images *Flatland*, en utilisant l'algorithme k-means.

5.4 Documents manuscrits

Dans le cadre de cette thèse, nous avons entamé une collaboration avec *V. Malleron*. L'idée est d'utiliser les cartes combinatoires pour représenter la topologie d'un document manuscrit dans le but de faire de la classification. L'algorithme proposé par *V. Malleron* [40] extrait un graphe d'adjacence basé sur des informations issues de l'extraction de la structure physique et logique d'un document manuscrit. La figure 5.5 illustre l'extraction de certaines de ces informations et la figure 5.6 montre un exemple de graphe obtenu à partir de ces informations. Il y a un nœud dans le graphe pour chaque composante extraite. Deux nœuds sont reliés par une arête s'ils sont proches dans une des quatre directions suivante : gauche, droite, haut et bas. Un nœud peut donc avoir jusqu'à quatre arêtes.

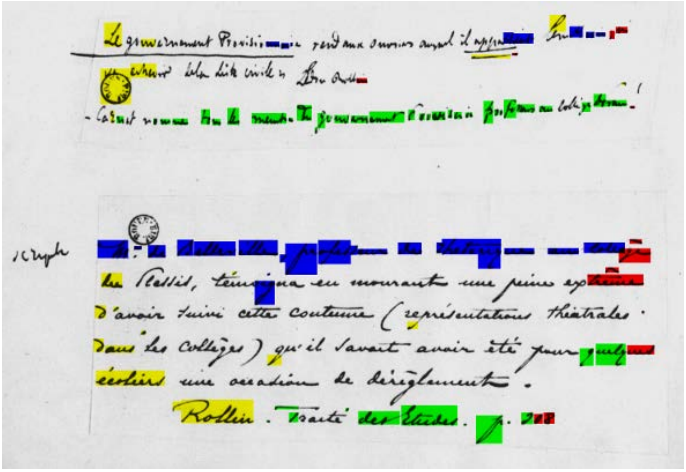


FIGURE 5.5 – Exemple des différentes composantes extraites pour cette image.

Ensuite, le graphe d'adjacence est simplifié : l'ensemble des arêtes de direction gauche ou basse est supprimé et les nœuds isolés sont également exclus. La carte combinatoire est alors extraite, en utilisant les informations obtenues lors de l'extraction des composantes. Chaque brin de la carte correspond à une arête du graphe d'adjacence, deux brins sont 1-cousus s'ils sont voisins horizontalement dans le document et deux brins sont 2-cousus s'ils sont voisins verticalement dans le document. Chaque ligne du document est donc représenté par une face et chaque face est adjacente à au plus deux autres faces.

Premiers résultats et Perspectives L'évaluation de notre approche est réalisée sur une base de 130 documents appartenant à 5 classes de documents manuscrits.

Le tableau 5.8 présente les résultats de la classification effectuée grâce à la recherche des motifs fréquents dans les cartes combinatoires associées aux images du corpus. D'après les experts, les résultats obtenus ne sont pas satisfaisants. Les taux de précision pour les classes « importantes » du corpus sont trop faibles pour pouvoir être utilisées.

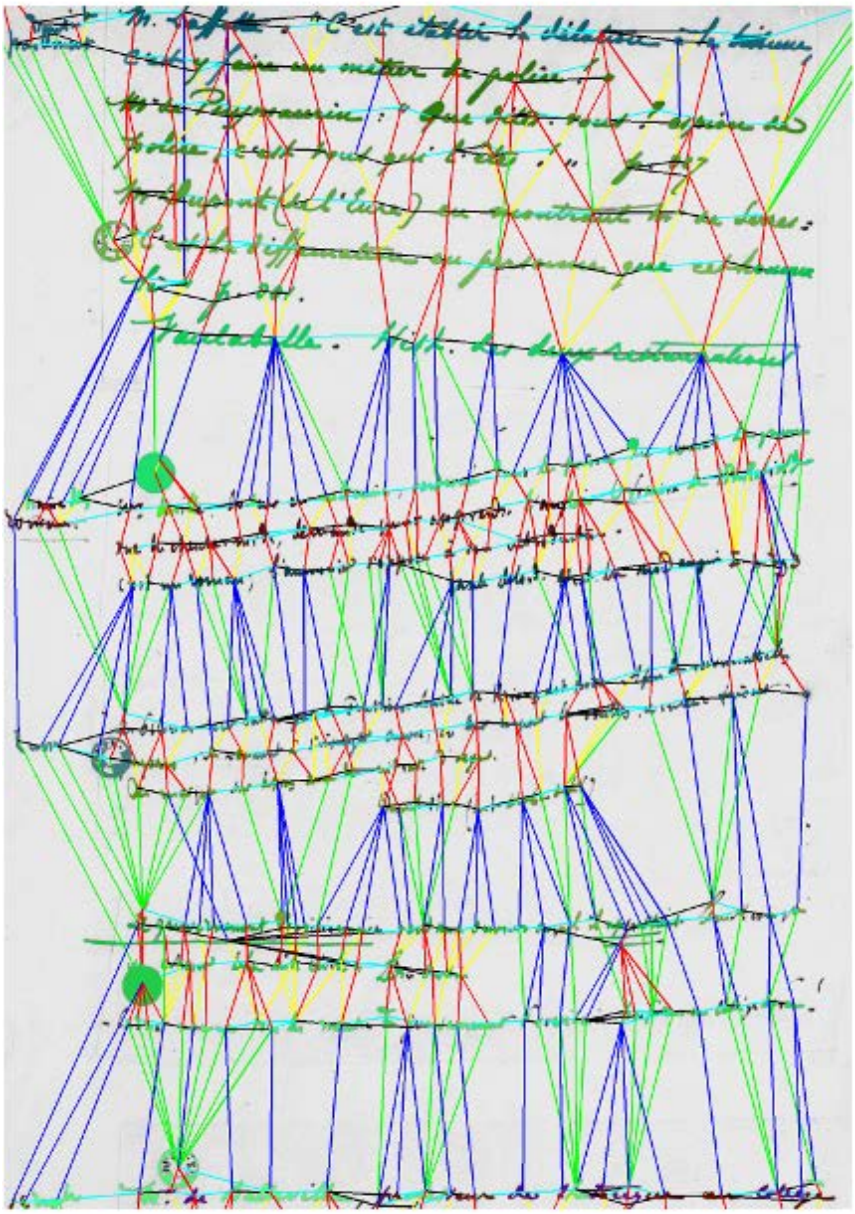


FIGURE 5.6 – Exemple de graphe d'adjacence.

Néanmoins, ces premiers résultats sont prometteurs et un certain nombre de pistes sont envisagées afin d'améliorer les résultats de classification. Ainsi, les informations topologiques contenues dans la carte extraite à partir d'une image de document ne semblent pas assez nombreuses : la carte se résume à un empilement de faces, ce qui limite l'intérêt de l'extraction de motifs fréquents. La simplification de la carte, en changeant de niveau structurel de description et en se positionnant par exemple directement au niveau de la ligne pourrait permettre d'améliorer ce point.

Par ailleurs, l'utilisation d'un modèle multi-échelle a été envisagée afin de disposer des informations extraites de la page à plusieurs niveaux structurels

Classe	Images	Rappel	Précision
Letters	36	0.71	0.64
Gathered Inf.	21	0.6	0.44
Printed Text	13	0.85	0.81
Notes	44	0.54	0.53
Dictionary	16	0.75	0.82
All	130	0.65	0.61

TABLE 5.8 – Table de Rappel/Précision de la classification. Le rappel correspond au nombre de documents bien classés par rapport au nombre de documents de la classe. La précision correspond au nombre de documents bien classés par rapport au nombre total de documents.

(page, paragraphe, ligne, mot, ...).

5.5 Pavages

Les cartes combinatoires peuvent représenter un pavage. Un pavage est un partitionnement de l'espace par un ensemble fini de tuiles. La figure 5.7 donne un exemple de pavage, où chaque tuile est représentée par une zone de même couleur. De la même façon que pour les images, il est possible de transformer un pavage en une carte combinatoire, en représentant une tuile par une face de la carte, et en cousant les faces si les deux tuiles qu'elles représentent sont voisines. De plus, nous ajoutons une information sur les brins pour identifier à quel type de tuile il appartient.

Il existe différents types de pavages : les pavages périodiques, qui peuvent être construits en répétant à l'infini le même motif (ensemble des tuiles connexes) de tuiles ; les pavages quasi-périodiques, qui ne sont pas périodiques, mais qui possèdent des motifs récurrents (fréquents) ; les pavages apériodiques, qui ne sont pas quasi-périodiques.

Notre idée est de calculer les motifs fréquents pour distinguer les pavages apériodiques des pavages quasi-périodiques. Nous pouvons grâce à l'algorithme *mSpan* faire varier les différents seuils, afin de soit trouver des motifs qui seraient présents dans plusieurs pavages différents, soit trouver des motifs qui se répètent à l'intérieur d'un même pavage en faisant varier le seuil φ.

Nous avons testé sur un pavage non-périodique de 162 tuiles, nous avons fixé le seuil φ à 20 et nous avons trouvé 15 motifs qui étaient présents au moins 20 fois dans ce pavage. Le motif le plus grand parmi les 15 est composé de 5 tuiles. Si nous baissons le seuil φ à 10, nous trouvons 40 motifs fréquents et le plus grand est de taille 8. Nous pouvons donc dire que même si ce pavage n'est pas périodique, il possède des motifs qui sont répétés, donc ce pavage peut être considéré comme étant quasi-périodique.

mSpan permet donc de calculer les motifs fréquents à l'intérieur d'un seul objet, et dans le cas des pavages, cela permet d'identifier si un pavage est apériodique ou quasi-périodique.

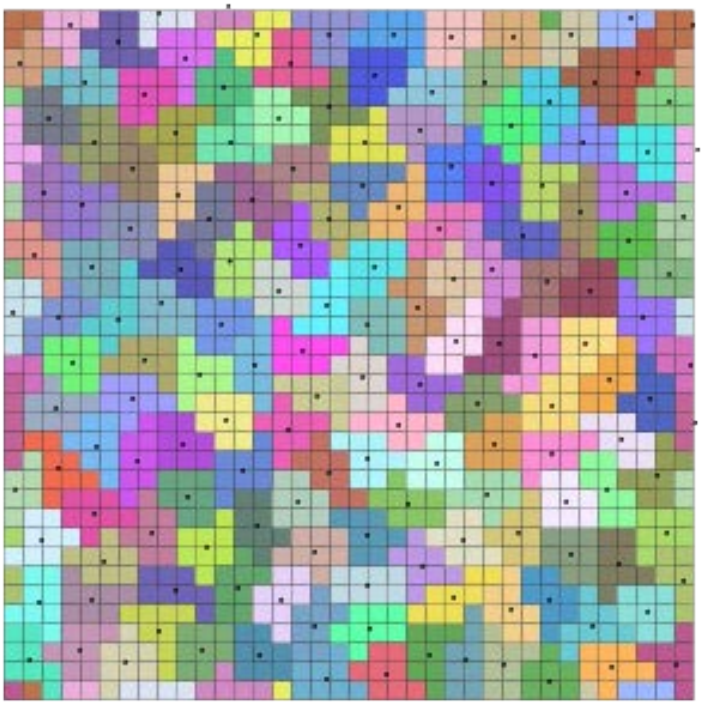


FIGURE 5.7 – Exemple de pavage. Chaque zone de même couleur correspond à une tuile dans le pavage et à une face dans la carte.

Dans ce chapitre, nous avons fait un tour d'horizon de plusieurs applications possibles : la classification supervisée et non supervisée et la détection de pavages quasi-périodiques. Les premiers résultats sont encourageants, mais ne sont pas suffisants pour rivaliser avec les méthodes de l'état de l'art aussi bien dans la classification d'images que dans la classification de documents manuscrits. Plusieurs pistes sont envisageables pour améliorer et elles seront détaillées dans le chapitre suivant.

Chapitre 6

Conclusion générale

L'OBJECTIF de cette thèse était de définir de nouveaux outils pour les cartes combinatoires. Dans ce cadre, nos contributions ont porté sur deux axes : la définition de signatures de cartes combinatoires, et la recherche de motifs fréquents dans une base de cartes combinatoires.

Contributions

Signatures de cartes combinatoires Nous avons proposé deux signatures [21] de cartes combinatoires qui ont chacune leurs avantages et leurs inconvénients. La S-signature permet de décider de l'isomorphisme entre deux cartes en temps linéaire, mais en contrepartie le coût de stockage en mémoire est quadratique en la taille de la carte. La W-signature a elle un coût de stockage en mémoire linéaire en la taille de la carte, cependant le temps de calcul de l'isomorphisme est quadratique. Ces deux signatures sont utilisables à la fois pour des cartes connexes, non connexes, valuées ou non valuées.

Ensuite, nous avons étendu ces signatures pour représenter efficacement une base de cartes combinatoires [22, 23]. Cette représentation d'une base de cartes ne permet pas de compresser les données et n'apporte donc pas de gain sur la mémoire utilisée, mais elle permet de rechercher un élément de manière très efficace. De plus, le temps de recherche n'est pas affecté par le nombre de cartes présent dans la base.

Extraction de sous-cartes fréquentes Nous avons défini le problème de recherche de motifs fréquents dans une base de cartes combinatoires. Dans ce problème, chaque motif doit satisfaire deux contraintes de fréquences pour être considéré fréquent. La première contrainte porte sur le nombre d'occurrences du motif à l'intérieur d'une carte de la base. La seconde contrainte porte sur le nombre de

cartes qui satisfont la première contrainte. Une fois ce cadre posé, nous avons proposé deux algorithmes pour résoudre ce problème.

L'algorithme *MapAPriori* est une adaptation de l'algorithme *APriori* qui construit les motifs fréquent niveau par niveau. Chaque niveau est construit à partir du niveau précédent en combinant les motifs deux à deux pour générer un ensemble de motifs candidats. Il faut vérifier si chaque motif candidat satisfait les contraintes de fréquence. Cet algorithme comporte plusieurs inconvénients :

1. la génération des candidats est complexe ;
2. le nombre de motifs candidats est très grand et beaucoup de ces motifs ne sont pas présents une seule fois dans la base ;
3. le calcul de la fréquence est très coûteux.

Nous avons proposé un deuxième algorithme intitulé *mSpan* [24]. Cet algorithme découvre les motifs par un parcours en profondeur. Chaque motif est construit par ajouts successifs de n-cellules. Les motifs sont construits directement à partir de la base de données, ce qui garantit que chaque motif considéré est présent au moins une fois. L'algorithme *mSpan* possède plusieurs avantages sur *MapAPriori* :

1. utilisation moindre de la mémoire ;
2. temps d'exécution plus court.

mSpan possède aussi des inconvénients. Premièrement, à chaque ajout d'une n-cellule sur un motif il faut recalculer entièrement la signature du nouveau motif. Deuxièmement, des motifs peuvent être générés plusieurs fois. Pour éviter d'avoir deux motifs différents qui seraient isomorphes, il est nécessaire de garder en mémoire la signature de l'ensemble des motifs fréquents.

Nous avons enfin proposé un classifieur basé sur les motifs fréquents. Le classifieur se construit en 3 étapes :

1. calculer tous les motifs fréquents ;
2. transformer chaque carte de la base en un vecteur ;
3. construire un arbre de décision à partir de ces vecteurs.

Les premiers résultats sur des bases d'images ne sont pas assez bons pour rivaliser avec les méthodes de l'état de l'art, mais ces résultats semblent prometteurs sur l'intérêt d'utiliser les informations topologiques dans la classification d'images.

Perspectives

Les perspectives se décomposent en deux points, l'amélioration de l'algorithme *mSpan* et l'application des sous-cartes fréquentes pour la classification d'images.

Signature de carte incrémentale Pour améliorer les performances de *mSpan*, il est nécessaire d'avoir une signature qui puisse être construite de manière incrémentale. Pour un motif donné dont nous possédons la signature, si nous ajoutons une n-cellule à ce motif, alors il faut que la signature puisse être construite sans parcourir de nouveau l'ensemble du motif, mais seulement la n-cellule ajoutée. Avec une telle signature, il serait alors possible de diminuer, voir supprimer la génération de motifs redondants. Il ne serait alors plus nécessaire de garder en mémoire la TW-signature de l'ensemble des motifs fréquents. L'algorithme pourrait alors générer des milliers de motifs sans souci de mémoire. Ce problème de calcul de signature incrémentale reste encore ouvert, malgré plusieurs tentatives infructueuses.

Un algorithme d'extraction de sous-graphes fréquents dans une base de graphes plans [47] à été réalisé en parallèle de notre travail. Cet algorithme intitulé *PLAGRAM* est similaire à *mSpan*, il calcul les sous-graphes par extension récursif de faces. Les complexités en temps sont identiques, cependant *PLAGRAM* à l'avantage de posséder une signature qui se calcul de façon incrémentale, donc il utilise moins de mémoire que *mSpan* car il n'a pas besoin de stocker les motifs déjà traités. Par contre *PLAGRAM* ne traite qu'une seule contrainte de fréquence, il ne peut donc pas calculer les motifs fréquents dans un seul graphe. De plus, il ne traite que les graphes plans, ce qui correspond à un cas particulier de l'algorithme *mSpan* pour des bases de 2-cartes. *mSpan* résout donc un problème plus général que *PLAGRAM*, mais il serait intéressant d'adapter sa signature incrémentale dans le cas des n-cartes.

Classification d'images Les premiers résultats en classification d'images sont encourageants, car nous obtenons de bons résultats en utilisant seulement très peu de l'information contenue dans les images. En effet, notre méthode utilise seulement l'information topologique des images. Or, les informations géométriques et colorimétriques sont des informations très importantes. Nous pensons donc qu'une approche hybride alliant à la fois les informations topologiques, géométriques et colorimétriques d'une image pourrait améliorer les performances des méthodes actuels qui utilisent très peu l'information topologique des images.

Bibliographie

[1] Rakesh Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. pages 487–499, 1994.

[3] M.L. Antonie, O.R. Zaiane, and Alexandru Coman. Application of data mining techniques for medical image classification. *MDM/KDD*, 2001 :94–101, 2001.

[4] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. ACM.

[5] Cécile Barat, Christophe Ducottet, Elisa Fromont, Anne-Claire Legrand, and Marc Sebban. Weighted symbols-based edit distance for string-structured image classification. In José Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 6321 of *Lecture Notes in Computer Science*, pages 72–86. Springer Berlin / Heidelberg, 2010.

[6] Christian Borgelt and Michael R. Berthold. Mining molecular fragments : Finding relevant substructures of molecules. *Data Mining, IEEE International Conference on*, 0 :51, 2002.

[7] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. *Proceedings of the 1997 ACM SIGMOD international conference on Management of data - SIGMOD '97*, pages 255–264, 1997.

[8] E. Brisson. Representing geometric structures in d dimensions : topology and order. In *SCG*, pages 218–227, Saarbrücken, Germany, 1989.

[9] Hong Cheng, Xifeng Yan, Jiawei Han, and C.W. Hsu. Discriminative frequent pattern analysis for effective classification. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 716–725. IEEE, 2007.

[10] R. Cori. Un code pour les graphes planaires et ses applications. In *Astérisque*, volume 27. Soc. Math. de France, Paris, France, 1975.

[11] G. Damiand, Y. Bertrand, and C. Fiorio. Topological model for two-dimensional image representation : definition and optimal extraction algorithm. *CVIU*, 93(2) :111–154, February 2004.

[12] G. Damiand, C. De La Higuera, J.-C. Janodet, E. Samuel, and C. Solnon. Polynomial Algorithm for Submap Isomorphism : Application to searching patterns in images. In *(GbR)*, LNCS, pages 102–112. Springer, May 2009.

[13] Guillaume Damiand. *Contributions aux Cartes Combinatoires et Cartes Généralisées : Simplification, Modèles, Invariants Topologiques et Applications*. Habilitation à diriger des recherches, INSA de Lyon/Université Lyon1, September 2010.

[14] Guillaume Damiand, C. Solnon, C. de la Higuera, J.C. Janodet, and É. Samuel. Polynomial Algorithms for Subisomorphism of nD Open Combinatorial Maps. *Computer Vision and Image Understanding*, 32(9) :1374–1383, July 2011.

[15] M. Deshp, Michihiro Kuramochi, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *Computing*, pages 1–15, 2003.

[16] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[17] ELG Escovar and Mauro Biajiz. SSDM : A Semantically Similar Data Mining Algorithm. *20th Brazilian Symposium of*, 2005.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2) :303–338, June 2010.

[19] U. Fayyad, G.P. Shapiro, and P. Smyth. From data mining to knowledge discovery : an overview. *AI Magazine*, pages 37–54, 1996.

[20] Dominique Gay. *Calcul de motifs sous contraintes pour la classification supervisée*. Thèse de doctorat en informatique, 2009.

[21] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Signatures of combinatorial maps. In *13th International Workshop on Combinatorial Image Analysis (IWCIA)*, LNCS, pages 370–382. Springer, November 2009.

[22] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Recherche efficace dans une base de cartes combinatoires. In *RFIA*, January 2010.

[23] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Efficient Search of Combinatorial Maps using Signatures. *Theoretical Computer Science*, 412(15) :1392–1405, March 2011.

[24] Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Frequent Submap Discovery . In *22nd Annual Symposium on Combinatorial Pattern Matching (CPM2011)*, June 2011.

[25] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2) :1–12, June 2000.

[26] M. Houtsma. Set-oriented mining for association rules in relational databases. *Data Engineering, 1995. Proceedings*, pages 25–33, 1995.

[27] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 549–552. IEEE, 2003.

[28] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. pages 13–23, 2000.

[29] A. Jacques. Constellations et graphes topologiques. In *Combinatorial Theory and Applications*, volume 2, pages 657–673, 1970.

[30] M. Kuramochi and G. Karypis. Frequent subgraph discovery. *Data Mining, IEEE International Conference on*, 0 :313, 2001.

[31] W. Li, J. Han, and J. Pei. CMAR : Accurate and efficient classification based on multiple class-association rules. In *icdm*, page 369. Published by the IEEE Computer Society, 2001.

[32] P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In *SCG*, pages 228–236, Saarbrücken, Germany, 1989.

[33] P. Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1) :59–82, 1991.

[34] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *IJCGA*, 4(3) :275–324, 1994.

[35] Dao-i Lin. Pincer-Search : A New Algorithm for Discovering the Maximum Frequent Set 2 Association Rule Mining. *New York*, pages 1–17, 1997.

[36] JL Lin. Mining association rules : Anti-skew algorithms. *Data Engineering, 1998. Proceedings.,*, pages 486–493, 1998.

[37] Bing Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. *Knowledge discovery and data mining*, pages 80–86, 1998.

[38] Claudio Lucchese, Salvatore Orlando, Paolo Palmerini, and Raffaele Perego. kDCI : A multi-strategy algorithm for mining frequent sets. *Proc. of ICDM Conf*, 2003.

[39] J. MacQueen and Others. 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, 1967.

[40] Vincent Malleron and Véronique Eglin. A mixed approach for handwritten documents structural analysis . In *International Conference on Document Analysis and Recognition*, September 2011.

[41] Christophe Moulin, C. Barat, and Christophe Ducottet. Fusion of tf. idf weighted bag of visual features for image classification. In *Content-Based Multimedia Indexing (CBMI), 2010 International Workshop on*, pages 1–6. IEEE, 2010.

[42] Andreas Mueller. Fast Sequential and Parallel Algorithms for Association Rule Mining : Table of Contents. *Knowledge Creation Diffusion Utilization*, 0057(August) :1–76, 1995.

[43] S Nijssen and J Kok. The Gaston Tool for Frequent Subgraph Mining. *Electronic Notes in Theoretical Computer Science*, 127(1) :77–87, March 2005.

[44] Edward Omiecinski. An Efficient Algorithm for Mining Databases Association Rules in Large. *Technology*, pages 432–444.

[45] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. An effective hash-based algorithm for mining association rules. *SIGMOD Rec.*, 24 :175–186, May 1995.

[46] M. Poudret, A. Arnould, Y. Bertrand, and P. Lienhardt. Cartes combinatoires ouvertes. Research Notes 2007-1, Laboratoire SIC E.A. 4103, October 2007.

[47] Adriana Prado, Baptiste Jeudy, Elisa Fromont, and Fabien Diot. PLAGRAM : un algorithme de fouille de graphes plans efficace. In *CAp*, pages 343–359, 2011.

[48] A. Rosenfeld. Adjacency in digital pictures. *Information and Control*, 26(1) :24–33, 1974.

[49] Steven L. Salzberg. C4.5 : Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16 :235–240, 1994. 10.1007/BF00993309.

[50] Émilie Samuel, Colin de la Higuera, and Jean-Christophe Janodet. Extracting plane graphs from images. In Edwin Hancock, Richard Wilson, Terry Windeatt, Ilkay Ulusoy, and Francisco Escolano, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 6218 of *Lecture Notes in Computer Science*, pages 233–243. Springer Berlin / Heidelberg, 2010.

[51] Anh-Phuong Ta, Christian Wolf, Guillaume Lavoue, and Atilla Baskurt. Recognizing and localizing individual activities through graph matching. *Advanced Video and Signal Based Surveillance, IEEE Conference on*, 0 :196–203, 2010.

[52] Hannu Toivonen. Sampling large databases for association rules. *of the International Conference on Very Large Data*, pages 134–145, 1996.

[53] W.T. Tutte. A census of planar maps. *Canad. J. Math.*, 15 :249–271, 1963.

[54] M. Wörlein, Thorsten Meinl, Ingrid Fischer, and Michael Philippsen. A quantitative comparison of the subgraph miners MoFa, gSpan, FFSM, and Gaston. *Knowledge Discovery in Databases : PKDD 2005*, pages 392–403, 2005.

[55] X. Yan and J. Han. gspan : Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 721–, Washington, DC, USA, 2002. IEEE Computer Society.

[56] Xifeng Yan, P.S. Yu, and Jiawei Han. Graph indexing : A frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, page 346. ACM, 2004.

[57] Xifeng Yan, P.S. Yu, and Jiawei Han. Graph indexing based on discriminative frequent structure analysis. *ACM Transactions on Database Systems (TODS)*, 30(4) :960–993, December 2005.

[58] Xifeng Yan, P.S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, number 1, pages 766–777. ACM, 2005.

[59] M.J. Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering, IEEE Transactions on*, 12(3) :372–390, 2000.

[60] M.J. Zaki, S Parthasarathy, M Ogihara, W Li, and Others. New algorithms for fast discovery of association rules. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, volume 20, pages 283–286, 1997.

TITLE

Frequent pattern discovery in combinatorial maps databases

ABSTRACT

A combinatorial map is a topological model that can represent the subdivisions of space into cells and their adjacency relations in n dimensions. This data structure is increasingly used in image processing, but it still lacks tools for analysis. Our goal is to define new tools for combinatorial maps nD. We are particularly interested in the extraction of submaps in a database of maps.

We define two combinatorial map signatures : the first one has a quadratic space complexity and may be used to decide of isomorphism with a new map in linear time whereas the second one has a linear space complexity and may be used to decide of isomorphism in quadratic time. They can be used for connected maps, non connected maps, labbeled maps or non labelled maps. These signatures can be used to efficiently search for a map in a database. Moreover, the search time does not depend on the number of maps in the database.

Then, we formalize the problem of finding frequent submaps in a database of combinatorial nD maps. We implement two algorithms for solving this problem. The first algorithm extracts the submaps with a breadth-first search approach and the second uses a depth-first search approach. We compare the performance of these two algorithms on synthetic database of maps.

Finally, we propose to use the frequent patterns in an image classification application. Each image is described by a map that is transformed into a vector representing the number of occurrences of frequent patterns. From these vectors, we use standard techniques of classification defined on vector spaces. We propose experiments in supervised and unsupervised classification on two images databases.

KEYWORDS

combinatorial maps, data-mining, classification, images

ADRR : Laboratoire d'InfoRmatique en Image et Systèmes d'information

LIRIS - UMR 5205
Université Claude Bernard Lyon 1
Bâtiment Nautibus
43, bd du 11 novembre 1918
69622 Villeurbanne cedex
ISBN :