Spatio-temporal grid mining applied to image classification and cellular automata analysis.

Romain Deville

N° d'ordre : 2018LYSEI046

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de l'INSA de Lyon

École doctorale n° 512
InfoMaths

Spécialité de doctorat :
Informatique

Soutenue publiquement le 30/05/2018, par

Romain DEVILLE

Spatio-temporal grid mining applied to image classification and cellular automata analysis

---

Devant le jury composé de :

M. Bruno CRÉMILLEUX	Professeur des Universités	Université de Caen Normandie	Rapporteur
M. Jean-Yves RAMEL	Professeur des Universités	Polytech Tours	Rapporteur
M. Jean-Christophe JANODET	Professeur des Universités	Université d'Evry	Examintateur
M^{me} Céline ROUVEIROL	Professeure des Universités	Université Paris XIII	Examinatrice
M^{me} Christine SOLNON	Professeure des Universités	INSA Lyon	Directrice de thèse
M^{me} Élisa FROMONT	Professeure des Universités	Université de Rennes 1	Co-directrice de thèse
M. Baptiste JEUDY	Maitre de Conférence	Université de Saint Étienne	Co-directeur de thèse

This thesis is available at https://github.com/RDeville/GriMA-Grid-Mining-Algorithm

Département FEDORA – INSA Lyon – Écoles doctorales
Quinquennal 2016–2020

SIGLE	ÉCOLE DOCTORALE	NOM ET COORD. RESPONSABLE
CHIMIE	**CHIMIE DE LYON** http://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage **secretariat@edchimie-lyon.fr** INSA : R. GOURDON	M. Stéphane DANIELE Institut de recherches sur la catalyse et l'environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 Avenue Albert EINSTEIN 69 626 Villeurbanne CEDEX **directeur@edchimie-lyon.fr**
E.E.A	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** http://edeea.ec-lyon.fr Sec. : M.C. HAVGOUDOUKIAN **ecole-doctorale.eea@ec-lyon.fr**	M. Gérard SCORLETTI École Centrale de Lyon 36 Avenue Guy DE COLLONGUE 69 134 Écully ☎ 04.72.18.60.97 🖷 04.78.43.37.17 **gerard.scorletti@ec-lyon.fr**
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** http://e2m2.universite-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 ☎ 04.72.44.83.62 INSA : H. CHARLES **secretariat.e2m2@univ-lyon1.fr**	M. Fabrice CORDEY CNRS UMR 5276 Lab. de géologie de Lyon Université Claude Bernard Lyon 1 Bât. Géode 2 Rue Raphaël DUBOIS 69 622 Villeurbanne CEDEX ☎ 06.07.53.89.13 **cordey@univ-lyon1.fr**
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** http://www.ediss-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 ☎ 04.72.44.83.62 INSA : M. LAGARDE **secretariat.ediss@univ-lyon1.fr**	Mme Emmanuelle CANET-SOULAS INSERM U1060, CarMeN lab, Univ. Lyon 1 Bâtiment IMBL 11 Avenue Jean CAPELLE INSA de Lyon 69 621 Villeurbanne ☎ 04.72.68.49.09 🖷 04.72.68.49.16 **emmanuelle.canet@univ-lyon1.fr**
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage ☎ 04.72.43.80.46 🖷 04.72.43.16.87 **infomaths@univ-lyon1.fr**	M. Luca ZAMBONI Bât. Braconnier 43 Boulevard du 11 novembre 1918 69 622 Villeurbanne CEDEX ☎ 04.26.23.45.52 **zamboni@maths.univ-lyon1.fr**
MATÉRIAUX	**MATÉRIAUX DE LYON** http://ed34.universite-lyon.fr Sec. : Marion COMBE ☎ 04.72.43.71.70 🖷 04.72.43.87.12 Bât. Direction **ed.materiaux@insa-lyon.fr**	M. Jean-Yves BUFFIÈRE INSA de Lyon MATEIS Bât. Saint-Exupéry 7 Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX ☎ 04.72.43.71.70 🖷 04.72.43.85.28 **ed.materiaux@insa-lyon.fr**
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Marion COMBE ☎ 04.72.43.71.70 🖷 04.72.43.87.12 Bât. Direction **mega@insa-lyon.fr**	M. Philippe BOISSE INSA de Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX ☎ 04.72.43.71.70 🖷 04.72.43.72.37 **philippe.boisse@insa-lyon.fr**
SCSO	**SCSO** * http://ed483.univ-lyon2.fr Sec. : Viviane POLSINELLI Brigitte DUBOIS INSA : J.Y. TOUSSAINT ☎ 04.78.69.72.76 **viviane.polsinelli@univ-lyon2.fr**	M. Christian MONTES Université Lyon 2 86 Rue Pasteur 69 365 Lyon CEDEX 07 **christian.montes@univ-lyon2.fr**

* ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie.

Remerciements

Je tiens à remercier :

– Baptiste Jeudy, Elisa Fromont et Christine Solnon de m'avoir donné l'occasion de réaliser cette thèse et de m'avoir encadré durant ces quatre années. Je les remercie également pour leur conseil, leur disponibilité, leur compréhension et leur soutien lors de certains moments délicats.

– Messieurs Bruno Crémilleux et Jean-Yves Ramel d'avoir accepté la responsabilité et d'avoir pris le temps de rapporter cette thèse, ainsi que le reste des membres de mon jury, M. Jean-Christophe Janodet et M^{me} Céline Rouverol pour leur présence et l'ensemble de leur retour lors de ma soutenance.

– L'ensemble de l'équipe pédagogique du Département Informatique de l'INSA de Lyon pour leurs enseignements et le savoir dont ils m'ont fait profiter m'ayant permis de réaliser cette thèse et plus particulièrement ceux que j'ai continué de côtoyer au sein du bâtiment Blaise Pascal à la suite de ma formation initiale.

– L'ensemble de l'équipe administrative du LIRIS et du Département IF. Plus particulièrement Yannick P. et Gilles B. pour les discussions que j'ai pu avoir concernant des problèmes informatiques qui n'étaient pas forcément liés à ma thèse et l'ensemble des secrétaires, Catherine L., Sylvie O., Faty B. Caroline F., Dominique B., Marie R. et Renée E-M, pour leur sympathie, leur accueil, leur accompagnement lors des démarches administratives et pour certain.e.s d'entre eux les pauses cigarettes partagées sur le balcon.

– Mes collègues doctorants de bureau, Julien S., Maël M., Loïc B., Maxime C., Vanessa L., Mickaël S-G., Lucas F., Yann C., Lucas G. (et j'espère ne pas en oublier) pour avoir partagé tout ou partie de ces quatre années, pour les pauses thé/café et/ou gouter, pour les moments d'échange qui n'étaient pas forcément en rapport avec nos thèses respectives mais humainement intéressant et aussi pour les moments parfois un plus difficiles.

– L'ensemble de mes collègues doctorants, incluant mes collègues de bureau, dans le désordre Sullivan D, Diana N., Aimene B., Adnene B., Romain M, Mohamed M., Mohamed A. H., Tarek A., Marie L. G., Sébastien D., Vincent P., Vincent B., Manel C., Yvan L, Rémi C., Léopold G., Maëlle M, et tout ceux que j'ai pu oublier pour les repas/débats que nous avons partagés et de manière générale le temps passé en leur compagnie.

– Mes amis, que je ne vois que trop peu pour la plupart, mais qui ont été présent avant et pendant cette thèse et m'ont soutenu dans les hauts comme dans les bas, à savoir Crina C., Virgil R., Ophélie A., Rémi C., Maréva D., John T., Stan H., Olivier R., Jérémy P-M. Sloane E. et Julien L.

– Les membres des diverses associations dans lesquelles je suis membre, Graines d'Images, Illyse et la Maison pour Tous, surtout Sébastien E., Sandrine N. et Laura K., de m'avoir permis de m'épanouir en dans ce cadre que j'affectionne particulièrement qu'est le bénévolat, la compréhension dont ils ont fait preuve lors de mes périodes d'indisponibilités, mais aussi lors de moment plus réjouissant autour d'un barbecue ou d'une raclette/CA.

– Et tout ceux que j'aurai oublié et/ou qui de près ou de loin ont participé et m'ont suivi dans cet épopée que furent ces quatre années.

Enfin, je souhaite remercier tout particulièrement :

– Damien F. et Bérengère M., amis doctorants, qui m'ont permis de passer un moment mémorable à Clermont-Ferrand suivi d'une année tout aussi inoubliable, pour le partage de doutes, mais aussi de réjouissance (surtout lorsqu'un de nos papiers soumis a été accepté).

– Sloane E., pour son point de vue sur le monde, les nombreuses discussions, parfois tard dans la nuit, mais aussi pour sa présence, sa bonne humeur et son imagination débordante.

– Jeannine R., secrétaire du département IF de l'INSA de Lyon, alias "La maman du département" (surnom plus que mérité), pour ce qu'elle est, son soutien, sa bonne humeur, les pauses cafés en sa compagnie qui se sont parfois un peu trop éternisées et ces huit années (ou presque).

– Ma famille, mon père, ma mère, ma soeur et mon frère notamment, pour les sacrifices qu'ils ont dû faire pour me permettre d'en arriver là. Merci aussi à eux pour leur soutien inébranlable, surtout dans les moments de doute et de fatigue tout au long de ces nombreuses années d'études, cela n'a pas été toujours facile. Je tiens aussi à les remercier de me permettre de garder les pieds sur terre et de m'avoir permis d'être tel que je suis. Merci aussi d'avoir été (et d'être) une oreille attentive et de m'avoir permis de profiter, en quelque sorte, d'un havre de paix et de solitude, loin des grandes villes, et ainsi profiter différemment des choses simples et du temps qui passe avec eux.

– Enfin, merci à ces quatre années. Grâce à elles, ce que représente le travail d'une thèse, et grâce à toutes les personnes précédemment mentionnées, j'ai énormément appris sur moi-même et sur ce que j'attends du Monde qui m'entoure. Elles m'ont permis de mieux envisager le futur, de revoir certaines priorités et mieux envisager la suite.

Bref, merci à tou.te.s pour ces quatre années (et beaucoup plus pour certain.e.s).

Contents

Résumé en français

Dans de nombreuses applications, les graphes sont utilisés pour modéliser des objets structurés tels que des molécules, des documents, des réseaux sociaux ou des images. Dans ce contexte, des algorithmes de fouille de graphes (tels que gSpan [Jiang et al. 2013], par exemple) peuvent être utilisés pour caractériser ces objets en termes de motifs fréquents, *i.e.*, des sous-graphes qui apparaissent dans un grand nombre de graphes. Cependant, ces algorithmes génériques de fouille de graphes passent difficilement à l'échelle pour des applications réelles. La difficulté vient d'une part du fait que, dans le cas général, il n'existe pas d'algorithme efficace pour tester l'isomorphisme de graphes, et d'autre part du grand nombre de combinaisons permettant d'étendre un graphe durant la fouille [Cook et al. 2006]. Des algorithmes de fouille efficaces ont été proposés pour certains cas particuliers de graphes, pour lesquels il existe des algorithmes de complexité polynomiale pour résoudre le problème d'isomorphisme tels que, par exemple, les graphes plans [Prado et al. 2013], et les graphes géométriques [Arimura et al. 2007].

Dans cette thèse, nous abordons le problème de la fouille exhaustive de motifs pour un cas particulier de graphes : les grilles. Ces grilles peuvent être utilisées pour modéliser des objets ayant une structure régulière, *i.e.*, des objets dont les composants sont régulièrement répartis dans l'espace ou le temps. Ces structures de grille sont naturellement présentes dans de nombreux jeux de plateaux (les dames, les échecs, ou le go, par exemple) ou encore dans les modélisations d'écosystèmes utilisant des automates cellulaires. On les retrouve également à un plus bas niveau dans les images, qui sont des grilles $2\mathcal{D}$ de pixels, les vidéos, qui sont des grilles spatio-temporelles $2\mathcal{D} + t$ de pixels, ou encore les images tomographiques de volumes, qui sont des grilles $3\mathcal{D}$ de voxels. Des motifs fréquents dans ces grilles bas niveau peuvent capturer une information structurelle de plus haut niveau, et constituer une alternative aux approches structurelles modélisant des images sous forme de graphes de plus haut niveau (tels que, par exemple, les graphes d'adjacence de régions ou les triangulations de points d'intérêts).

Contributions et organisation de la thèse. Dans le chapitre 2, nous rappelons des définitions concernant les graphes, en mettant l'accent sur les

problèmes d'isomorphisme de graphes (utilisés pour décider si deux graphes sont équivalents) et d'isomorphisme de sous-graphes (utilisés pour décider si un graphe motif apparait dans un graphe cible). Nous introduisons aussi des graphes particuliers pour lesquels il existe des algorithmes de fouille efficaces, *i.e.*, les graphes géométriques et les graphes plans.

Dans le chapitre 3, nous décrivons les algorithmes de fouille de graphes existants. Nous nous concentrons plus particulièrement sur les algorithmes utilisant un code canonique pour représenter les graphes et effectuant une exploration en profondeur d'abord de l'espace des codes canoniques. Ces algorithmes sont décrits dans un contexte unifié, sous la forme d'instantiations d'un algorithme générique, afin de mettre en évidence leurs points communs et leurs différences.

Dans le chapitre 4, nous définissons les grilles qui sont un cas particulier des graphes géométriques. Pour les grilles $2\mathcal{D}$, les sommets ont des coordonnées $2\mathcal{D}$. Pour les grilles $2\mathcal{D} + t$, une troisième coordonnée, correspondant à la dimension temporelle, est associée à chaque sommet. Dans les deux cas, les arêtes ne peuvent relier que des sommets ayant des coordonnées voisines (*i.e.*, dont la distance est égale à un). Nous définissons l'isomorphisme de grilles, qui permet de déterminer si deux motifs sont équivalents, ainsi que l'isomorphisme de sous-grilles, qui permet de déterminer si un motif apparait dans une grille. Ces définitions d'isomorphisme sont tolérantes aux rotations spatiales ainsi qu'aux translations spatio-temporelles, de sorte que deux motifs ayant une même structure mais apparaissant avec des orientations différentes ou à des endroits différents de la grille sont considérés comme équivalents.

Dans le chapitre 5, nous décrivons la principale contribution de cette thèse, qui est un nouvel algorithme de fouille de grilles appelé GriMA. Etant donnés une base D de n grilles et un seuil $\sigma \in [1, n]$, GriMA recherche tous les motifs qui apparaissent dans au moins σ grilles de D. GriMA est présenté comme une instantiation de l'algorithme générique décrit au chapitre 3, ce qui nous permet de nous concentrer sur ses spécificités : le code canonique utilisé pour réprésenter de façon unique un motif, et la stratégie d'extension utilisée pour récursivement étendre les motifs. Nous prouvons que GriMA est correct et complet. Nous montrons également que la complexité en temps pour extraire un motif est polynomiale, et que cette complexité est inférieure à celles des algorithmes de l'état de l'art pour la fouille de graphes géométriques et de graphes plans. Enfin, nous introduisons un algorithme efficace pour dénombrer les occurrences d'un motif donné dans une grille.

Dans le chapitre 6, nous évaluons expérimentalement l'efficacité ainsi que l'intérêt de GriMA sur une tâche de classification d'images. Nous proposons

de modéliser les images à l'aide de grilles de mots visuels et d'extraire les motifs fréquents de ces grilles afin de pouvoir caractériser les images par des sacs de grilles (*Bag-of-Grids*, BoG). Dans un premier temps, nous étudions les propriétés de passage à l'échelle de GriMA pour la fouille de grilles $2\mathcal{D}$, et nous le comparons avec des algorithmes de fouille de l'état de l'art. Ensuite, nous comparons notre représentation par BoG avec une représentation simple et non structurée sous la forme de sacs de mots (*Bag-of-Words*, BoW). Nous montrons sur trois jeux d'images que la structuration en grilles BoG peut améliorer la classification par rapport à l'approche non structurée BoW.

Dans le chapitre 7, nous nous intéressons à une seconde application de GriMA, à savoir l'analyse d'automates cellulaires. Là encore, l'objectif est double : il s'agit d'évaluer d'une part l'efficacité en pratique de GriMA, et d'autre part son intérêt pour une tâche de classification. Les automates cellulaires sont des grilles régulières de cellules dont l'état évolue dans le temps, en utilisant pour cela des règles locales et simples. Typiquement, l'état futur d'une cellule est calculé à partir de l'état présent de ses cellules voisines. Les automates cellulaires sont utilisés pour modéliser des phénomènes spatio-temporels tels que, par exemple, la propagation d'espèces invasives ou l'équilibre de différentes espèces dans un éco-système. Ces systèmes sont complexes, et il est généralement difficile de prévoir leur état à long terme (par exemple, leur convergence vers un état stable ou non) à partir de l'état initial. Nous nous intéressons plus particulièrement au jeu de la vie de Conway, et nous montrons que cet automate cellulaire peut être tout naturellement modélisé sous la forme d'une grille $2\mathcal{D}+t$. Nous étudions les propriétés de passage à l'échelle de GriMA pour la fouille de ces grilles $2\mathcal{D} + t$. Nous montrons ensuite que les motifs fréquents extraits par GriMA peuvent être utilisés pour prévoir l'état à long terme d'une instance du jeu de la vie, étant donnés ses k premiers états.

Enfin, nous discutons au chapitre 8 de quelques prolongations possibles pour nos travaux de recherche.

Chapter 1

Introduction

In many applications, graphs are used to model structured objects such as, for example, molecules, documents, social networks, or images. In this context, subgraph mining algorithms [Jiang et al. 2013] may be used to characterize structured objects by means of frequent patterns, *i.e.*, subgraphs that frequently occur. However, general-purpose subgraph mining algorithms are seldom used in real-world applications due to the high complexity of the mining process mostly based on isomorphism tests and countless expansion possibilities during the search [Cook et al. 2006]. Efficient algorithms have been proposed for special cases for which there exist polynomial-time algorithms for graph isomorphism such as, for example, plane graphs [Prado et al. 2013], geometric graphs [Arimura et al. 2007], and outerplanar graphs under the block-and-bridge preserving constraint [Horváth et al. 2010].

In this thesis we tackle the problem of exhaustive graph mining for a special case of graphs called grids. These grids may be used to model objects that have a regular structure, *i.e.*, their components are spatially embedded on a regular grid structure. This grid structure is naturally present in many boardgames (Checkers, Chess, Go, etc) or to model ecosystems using cellular automata [Hogeweg 1988], for example. In addition, this grid structure may be useful to capture low-level topological relationships whenever a high-level graph structure is not obvious to design. In computer vision in particular, it is now widely acknowledged that high-level graph-based image representations (such as region adjacency graphs or interest point triangulations, for example) are sensitive to noise so that slightly different images may result in very different graphs [Samuel et al. 2010, Prado et al. 2013]. However, at a low-level, images basically are grids: $2\mathcal{D}$ grids of pixels for images, and $3\mathcal{D}$ grids of voxels for tomographic or MRI images modelling $3\mathcal{D}$ objects. When considering videos, we may add a temporal dimension to obtain spatio-temporal grids.

Contributions and outline of the thesis. In Chapter 2, we recall definitions on graphs, with a particular emphasis on graph isomorphism, which is

used to decide whether two graphs are equivalent, and subgraph isomorphism which is used to decide whether a copy of a pattern graph occurs in a target graph. We also introduce special kinds of graphs for which there exist efficient graph mining algorithms, *i.e.*, geometric graphs and plane graphs.

In Chapter 3, we describe existing graph mining algorithms. We more particularly focus on algorithms that use canonical codes to represent graphs and perform a recursive depth-first exploration of the search space of all canonical codes. These algorithms are described within a unifying framework that will allow us to clearly position our new algorithm with respect to other related graph mining algorithms.

In Chapter 4, we define grids which are particular cases of labeled graphs. For $2\mathcal{D}$ grids, vertices have $2\mathcal{D}$ coordinates, and edges only link vertices which have neighbor coordinates. For $2\mathcal{D}+t$ grids, we associate a third coordinate to each vertex, corresponding to the temporal dimension. We define grid isomorphism, which is used to decide whether two patterns are equivalent or not, and subgrid isomorphism, which is used to decide whether a pattern occurs in grid or not.

In Chapter 5, we describe the main contribution of this thesis, which is a new algorithm called GriMA. GriMA mines frequent patterns in a database of grids, and it follows the unifying framework introduced in Chapter 3. We study the theoretical time complexity of GriMA, and prove that it is correct and complete.

In Chapter 6, we experimentally evaluate GriMA on an image classification task. To assess the relevance of using a grid structure for this task, we propose to model images by means of grids of visual words and to extract frequent patterns in these grids. We first study scale-up properties of GriMA for mining frequent patterns in $2\mathcal{D}$ grids, and compare it with state-of-the-art graph mining algorithms. Then, we show how to use frequent patterns to obtain Bags-of-Grids (BoGs). We compare these BoGs with a standard classification method which uses simple unstructured Bags-of-Visual-Words (BoWs) in order to compare an unstructured set of descriptors and a set of descriptors structured by a grid topology.

In Chapter 7, we show how GriMA may be used to mine spatio-temporal patterns in Cellular Automata (CA). CA are regular grids of cells which evolve through time, giving rise to the emergence of spatio-temporal patterns which are characteristics of different ecosystem outcomes. We first study scale-up properties of GriMA for mining frequent patterns in $2\mathcal{D}+t$ grids. We also report experimental results on a classification task that aims at forecasting the outcome of CA.

Publications related to the thesis. A first version of our grid mining algorithm dedicated to $2\mathcal{D}$ grids, and its application to image classification, has been described in the following paper:

> Romain Deville, Élisa Fromont, Baptiste Jeudy, Christine Solnon: GriMA*: A Grid Mining Algorithm for Bag-of-Grid-Based Classification*, in Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop (S+SSPR) 2016. Lecture Notes in Computer Science 10029, pages 132-142

The extension of GriMA to mine $2\mathcal{D} + t$ grids, and its application to the analysis of cellular automata, has been described in the following paper:

> Romain Deville, Élisa Fromont, Baptiste Jeudy, Christine Solnon: GriMA2D+t*: Mining Frequent Patterns in* $2\mathcal{D} + t$ *Grid Graphs for Cellular Automata Analysis*, in Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop (GbRPR) 2017. Lecture Notes in Computer Science 10310, pages 177-186

An extended version of these papers is currently submitted to the special issue of Pattern Recognition Letters dedicated to GbRPR 2017.

Chapter 2

Background on Graphs

Contents

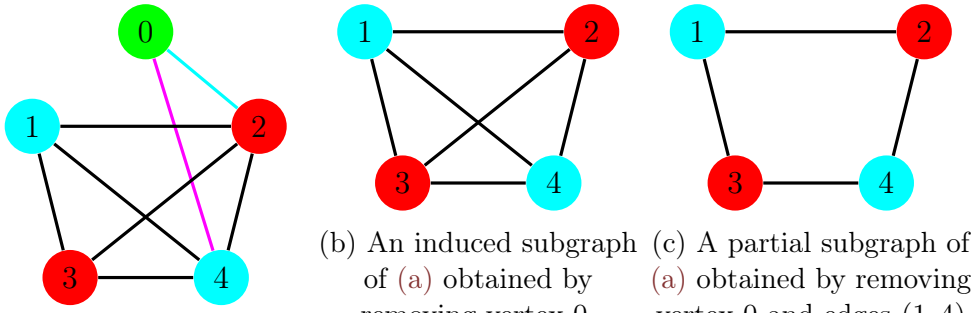
In this chapter, we first recall basic definitions on graphs, and then define two problems that are at the core of the graph mining problem: graph isomorphism, which is used to decide whether two graphs are equivalent, and subgraph isomorphism, which is used to decide whether a copy of a pattern graph occurs in a target graph. Then, we describe two classes of graphs which are used to describe objects embedded in spaces, and for which there exist efficient algorithms for solving the subgraph isomorphism problem: plane graphs and geometric graphs. Finally, we describe graph traversal algorithms, which are used to define canonical codes in graph mining algorithms.

2.1 Basic definitions

Graphs are used to model objects by means of a set of components (called vertices) and a binary relation between these components (called edges). Labels may be associated with vertices or edges.

Definition 2.1 Labeled graph
A labeled graph is defined by a triple $G = \langle V, E, L \rangle$ such that:

- V is the set of vertices,
- $E \subseteq V \times V$ is the set of edges,



(a) A graph .

(b) An induced subgraph of (a) obtained by removing vertex 0.

(c) A partial subgraph of (a) obtained by removing vertex 0 and edges $(1, 4)$ and $(2, 3)$.

Figure 2.1 – Examples of graph, induced and partial subgraphs. Vertex and edge labels are represented by colors.

- $L : V \cup E \to \mathbb{N}$ is a function that associates a label $L(c)$ with every component (vertex or edge) $c \in V \cup E$.

When the relation defined by E is symmetric, *i.e.*, $\forall (v_i, v_j) \in V \times V, (v_i, v_j) \in E \Leftrightarrow (v_j, v_i) \in E$ and $L(v_i, v_j) = L(v_j, v_i)$, the graph is *undirected*. When the relation defined by E is not symmetric, the graph is *directed*. In this thesis, we only consider undirected labeled graphs, and call them graphs for short.

Figure 2.1(a) displays an example of graph with five vertices that have three different labels (labels are represented by colours).

Given a vertex v, the vertices that are linked with v by an edge are called *neighbors* of v. For example, vertex 0 in the graph of Figure 2.1(a) has two neighbors: 2 and 4.

Definition 2.2 Path and cycle

Let $G = \langle V, E, L \rangle$ be a graph. A *path* is a sequence of vertices $(v_1, \ldots, v_n)$ such that $(v_i, v_{i+1}) \in E$, for all $i \in [1, n-1]$. The *length* of a path $(v_1, \ldots, v_n)$ is the number of its edges, *i.e.*, $n - 1$. A *cycle* is a path such that $v_1 = v_n$. A path or a cycle is *elementary* if all its vertices are different (except the first and last vertices for a cycle).

For example, $(0, 2, 3, 1)$ is an elementary path, and $(1, 2, 4, 3, 1)$ is an elementary cycle of the graph displayed in Figure 2.1(a).

Definition 2.3 Distance

The distance between two vertices in a graph is the length of the shortest path between these two vertices.

For example, the distance between vertices 0 and 1 is 2 in the graph displayed in Figure 2.1(a).

Definition 2.4 Connected graph

A graph $G = \langle V, E, L \rangle$ is connected if for every pair of vertices $(v_i, v_j) \in V^2$, there is a path between v_i and v_j.

For example, the graph displayed in Figure 2.1(a) is connected.

Definition 2.5 k-connected graph
A graph $G = \langle V, E, L \rangle$ is k-connected if for every subset of $k-1$ vertices, the induced subgraph of G obtained by removing these $k-1$ vertices is connected.

For example, the graph displayed in Figure 2.1(a) is 2-connected because the subgraph obtained by removing any of its vertices is connected. However, it is not 3-connected because the subgraph obtained when removing vertices 2 and 4 is no longer connected.

Definition 2.6 Induced subgraph
A graph $G' = \langle V', E', L' \rangle$ is an induced subgraph of a graph $G = \langle V, E, L \rangle$ if:

- $V' \subseteq V$,
- $E' = E \cap V' \times V'$,
- L' is the restriction of L to $V' \cup E'$.

In other words, an induced subgraph is obtained by deleting some vertices, and every edge incident to a deleted vertex.

Definition 2.7 Partial subgraph
A graph $G' = \langle V', E', L' \rangle$ is a partial subgraph of a graph $G = \langle V, E, L \rangle$ if:

- $V' \subseteq V$,
- $E' \subseteq E \cap V' \times V$,
- L' is the restriction of L to $V' \cup E'$.

In other words, a partial subgraph is obtained by removing both vertices and edges. Figure 2.1 displays examples of induced and partial subgraphs.

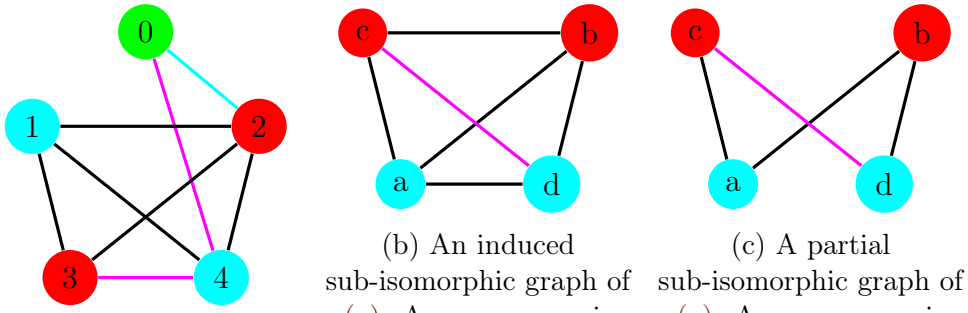
2.2 Graph and subgraph isomorphism

Two graphs G and G' are isomorphic if they are equivalent up to a renaming of their vertices, *i.e.*, if there exists a bijective function that maps each vertex of G to a vertex of G' and that preserves edges and labels.

Definition 2.8 Graph isomorphism
Two graphs $G' = \langle V', E', L' \rangle$ and $G = \langle V, E, L \rangle$ are isomorphic if there exists a bijective function $f : V \to V'$ such that:

- $\forall i, j \in V, (v_i, v_j) \in E \Leftrightarrow (f(v_i), f(v_j)) \in E'$,
- $\forall i \in V, L(v_i) = L'(f(v_i))$,
- $\forall (v_i, v_j) \in E, L(v_i, v_j) = L'(f(v_i), f(v_j))$.

This bijection f is called an *isomorphism function*.



(a) A graph.

(b) An induced sub-isomorphic graph of (a). An occurrence is $a \to 1, b \to 2, c \to 3, d \to 4$.

(c) A partial sub-isomorphic graph of (a). An occurrence is $a \to 1, b \to 2, c \to 3, d \to 4$.

Figure 2.2 – Examples of induced and partial subgraph isomorphisms.

A graph $G = \langle V, E, L \rangle$ is *automorphic* if it is isomorphic with itself, *i.e.*, there exists a bijective function from V to V (different from the identity) that preserves edges and labels. In this case, the bijective function is called an *automorphism*. For instance, the graph of Figure 2.1(b) is automorphic because the bijective function $1 \to 4, 2 \to 3, 3 \to 2, 4 \to 1$ is an automorphism.

Definition 2.9 Subgraph isomorphism
A graph G is sub-isomorphic to a graph G', denoted $G \subseteq G'$, if there exists a subgraph G'' of G' such that G is isomorphic to G''.

Depending on the subgraph relation considered (induced or partial), we obtain two different definitions of subgraph isomorphism, respectively called induced subgraph isomorphism (denoted $\subseteq^i$) and partial subgraph isomorphism (denoted $\subseteq^p$).

When a graph G_1 is sub-isomorphic to another graph G_2, the isomorphism function between the vertices of G_1 and the vertices of the subgraph of G_2 is called an *occurrence* of G_1 in G_2. Figure 2.2 displays examples of subgraph isomorphisms and occurrences.

Computational complexity and algorithms. The complexity of graph isomorphism is still an open question. If it clearly belongs to $\mathcal{NP}$, no polynomial-time algorithm has been found for this problem, and it has not been shown to be $\mathcal{NP}$-complete. There exist many dedicated algorithms for solving the graph isomorphism such as, *e.g.*, [Ullmann 1976, McKay et al. 1981, Cordella et al. 2001, Sorlin et al. 2008]. These algorithms are often very efficient, even though their worst case time complexities are exponential. In 2016, Babai described an algorithm in quasipolynomial $(\exp((\log n)^{\mathcal{O}(1)}))$ time [Babai 2016].

Subgraph isomorphism is more general than graph isomorphism (in the sense that graph isomorphism may be reduced to subgraph isomorphism) and it has

been shown to be $\mathcal{NP}$-complete [Garey et al. 1979]. Subgraph isomorphism problems may be solved by a systematic exploration of the search space consisting of all possible injective matchings from the vertices of the pattern graph to the vertices of the target graph: starting from an empty matching, one incrementally extends a partial matching by matching a non-matched pattern vertex to a non-matched target vertex until either some edges are not matched by the current matching (so the search must backtrack to a previous choice point and go on with another extension), or all pattern vertices have been matched (a solution has been found). To reduce the search space, this exhaustive exploration is combined with filtering techniques that aim at removing candidate pairs of non-matched pattern-target vertices. Different filtering techniques may be considered; some are stronger than others (they remove more candidate pairs), but also have higher time complexities.

The simplest form of filtering is to propagate difference constraints (which ensure that the matching is injective) and edge constraints (which ensure that the matching preserves pattern edges). This simple filtering (called *Forward-Checking*) is very fast to achieve and is used, for example, in McGregor's algorithm [McGregor 1979] and in VF2 [Cordella et al. 2004]. Régin [Régin 1994] introduced a stronger filtering for difference constraints, which ensures that all pattern vertices can be matched with different target vertices, all together. For edge constraints, Ullman [Ullmann 1976] introduced a filtering (called *Arc Consistency*) that removes more candidate pairs than Forward-Checking, but that is also more time consuming. Stronger filtering may be obtained by propagating edge constraints in a more global way, as proposed in [Larrosa et al. 2002, Solnon 2010, Audemard et al. 2014, McCreesh et al. 2015]. These different algorithms have been compared in [Kotthoff et al. 2016] on a large benchmark set of 5725 instances, grouped in 12 different classes. This comparison has shown that the algorithm of [McCreesh et al. 2015] is able to solve more instances than all others when considering large CPU time limits, and that some of these algorithms have complementary performance.

Finally, let us note that if subgraph isomorphism is $\mathcal{NP}$-complete in the general case, there exist subclasses of graphs for which this problem becomes polynomial such as, for example, trees [Matula 1978], 2-connected outerplanar graphs [Syslo 1982] and outerplanar graphs under the block-and-bridge preserving constraint [Horváth et al. 2010]. Also, there exist polynomial-time algorithms for the two classes of graphs introduced in the next two sections, *i.e.*, geometric graphs and plane graphs.

2.3 Geometric graphs

In some applications, vertices correspond to components which have spatial coordinates. In this case, subgraph isomorphism may be solved in polynomial time by exploiting geometrical information when matching pattern vertices to target vertices. As a consequence, efficient pattern mining algorithms may be designed for these graphs, as proposed in [Kuramochi et al. 2002, 2007]. These algorithms are described in the next chapter. In this section, we describe geometric graphs and define geometric (sub)graph isomorphism.

Definition 2.10 Geometric graph
A geometric graph is defined by a tuple $G = \langle V, E, L, C\rangle$ such that:

- $\langle V, E, L\rangle$ is a graph,
- $C : V \to \mathbb{R}^2$ is a function that associates $2\mathcal{D}$ coordinates with each vertex $v \in V$.

The extension of subgraph definitions to geometric graphs is straightforward.

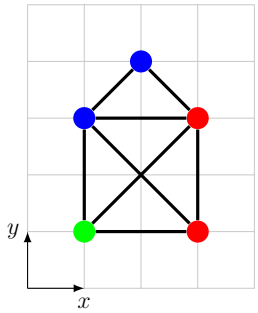
Definition 2.11 Geometric subgraph
Let $G' = \langle V', E', L', C'\rangle$ and $G = \langle V, E, L, C\rangle$ be two geometric graphs. G' is a geometric subgraph of G if :

- $\langle V', E', L'\rangle$ is a subgraph of $\langle V, E, L\rangle$,
- C' is the restriction of C to N'

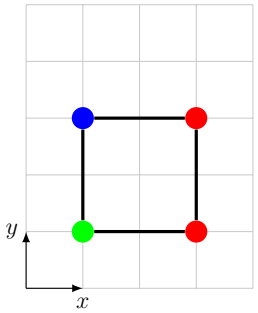
Figures 2.3(b) and 2.3(c) are, respectively, examples of a partial geometric subgraph and an induced geometric subgraph of the geometric graph of Figure 2.3(a).

If the extension of the subgraph definition is straightforward, the definition of isomorphism for geometric graphs deserves a discussion. We may consider a straightforward extension of Definition 2.8, that simply checks that the isomorphism function preserves vertex coordinates. However, since vertex coordinates depend on the particular reference coordinate axes, it is more natural to define geometric isomorphism that allows homogeneous transforms on coordinates, prior to establishing a match, as proposed in [Kuramochi et al. 2002, Arimura et al. 2007]. More precisely, they define the class $\mathcal{T}_{rgeo}$ of rigid transformations as follows.

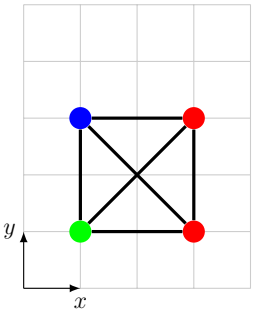
Definition 2.12 Class $\mathcal{T}_{rgeo}$ of rigid transformations
$\mathcal{T}_{rgeo}$ is the class of all $2\mathcal{D}$ affine transformations $T : x \to Ax + t$ where $A = \lambda R$ is the product of a scaling factor λ and a 2×2 rotation matrix R and t is a $2-$vector.



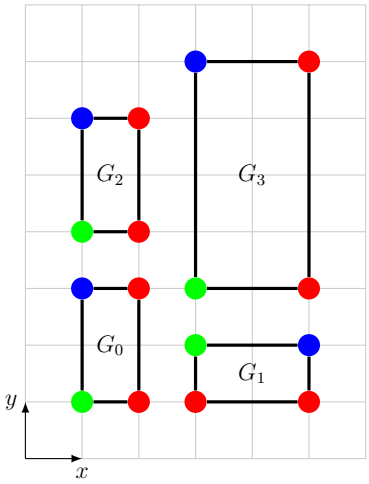
(a) A geometric graph.



(b) A partial geometric subgraph of (a).



(c) An induced geometric subgraph of (a).



(d) Examples of geometrically isomorphic graphs: for G_0 and G_2, the rigid transformation is a translation of 3 units along the y axis, *i.e.*, $A = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ and $t = (0, 3)$; for G_0 and G_1, the rigid transformation is composed of a rotation with $A = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ and a translation with $t = (2, 3)$; for G_0 and G_3, the rigid transformation is composed of a scaling with $A = \left(\begin{smallmatrix} 2 & 0 \\ 0 & 2 \end{smallmatrix}\right)$ and a translation with $t = (1, 1)$.

Figure 2.3 – Examples of geometric graphs, geometric subgraphs, and geometric graph isomorphism.

These rigid transformations preserve the angles between lines and can be determined by a set of three non-collinear points and their images (see [Arimura et al. 2007] for more details). These rigid transformations allow us to define geometric graph isomorphism.

Definition 2.13 Isomorphism of geometric graphs

Let $G = \langle V, E, L, C \rangle$ and $G' = \langle V', E', L', C' \rangle$ be two geometric graphs. G and G' are *geometrically isomorphic* if:

- There exists an isomorphism function f between the graphs $\langle V, E, L \rangle$ and $\langle V', E', L' \rangle$.
- There exists a rigid transformation $T \in \mathcal{T}_{rgeo}$ that preserves coordinates of matched vertices, *i.e.* $T(C(v)) = C'(f(v)), \forall v \in V$.

Given this definition of geometric graph isomorphism, the definition of geometric subgraph isomorphism is straightforward.

Definition 2.14 Geometric subgraph isomorphism
Let $G = \langle V, E, L, C \rangle$ and $G' = \langle V', E', L', C' \rangle$ be two geometric graphs. G is a geometric subisomorphic graph of G', denoted $G \subseteq_{geo} G'$, if there exists a geometric subgraph G'' of G' such that G is geometrically isomorphic to G''.

Depending on whether G'' is an induced subgraph or a partial subgraph of G', we obtain two different definitions of geometric subgraph isomorphism, respectively denoted $\subseteq^i_{geo}$ and $\subseteq^p_{geo}$.

2.4 Plane graphs

Plane graphs are special cases of geometric graphs, such that embedded edges do not intersect except at their endpoints.

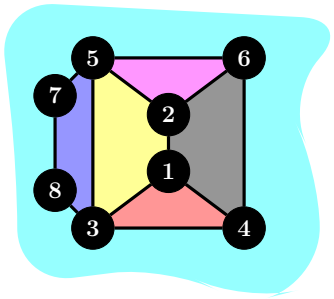
Definition 2.15 Plane graph
A plane graph G is a geometric graph $G = \langle V, E, L, C \rangle$ such that for any pair of different edges $\{(v_1, v_2), (v_3, v_4)\} \subseteq E$, the two open line segments $]C(v_1), C(v_2)[$ and $]C(v_3), C(v_4)[$ have an empty intersection.

A planar graph is a graph $G = \langle V, E, L \rangle$ for which there exists a $2\mathcal{D}$ embedding $C : V \to \mathbb{R}^2$ of its vertices such that $\langle V, E, L, C \rangle$ is a plane graph. Note that a planar graph has an infinite number of $2\mathcal{D}$ embeddings, and that not all these embeddings correspond to plane graphs.

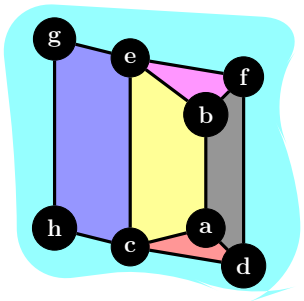
Line segments associated with edges of a plane graph split the $2\mathcal{D}$ plane into regions that are called *faces*. One of these faces is unbounded and called the *external face*; all other faces are bounded by a cycle and are called *internal faces*.

Figure 2.4(a) displays an example of a plane graph composed of five internal faces (respectively bounded by the cycles $(3, 8, 7, 5, 3)$, $(2, 5, 6, 2)$, $(2, 6, 4, 1, 2)$, $(1, 4, 3, 1)$, and $(1, 3, 5, 2, 1)$) and one external face (corresponding to the unbounded region outside the cycle $(5, 6, 4, 3, 8, 7, 5)$).

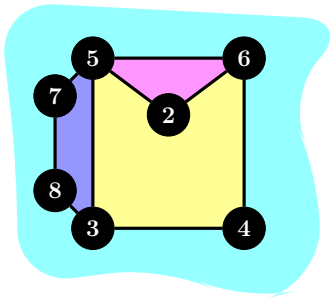
As a plane graph is a special case of geometric graph, the geometric subgraph definition still holds for plane graphs. However, removing a vertex may change internal faces. Let us consider for example the plane graph displayed in Figure 2.4(a): when removing vertex 1 and its three incident edges, we merge three internal faces into a single face bounded by $(2, 6, 4, 3, 5, 2)$ as displayed in Figure 2.4(c). In [Higuera et al. 2013], a constraint is added to ensure that each internal face of the original graph is also an internal face of the subgraph. This is called face-induced plane subgraph.



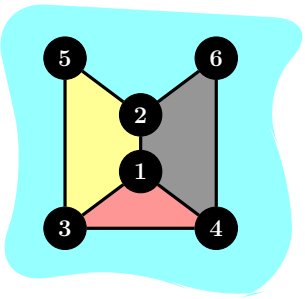
(a) A plane graph.



(b) A plane graph isomorphic to (a).



(c) A subgraph of (a) which is not face-induced.



(d) A face-induced plane subgraph (a).

Figure 2.4 – Example of plane graphs. The external face is represented by the blue area around graphs.

Definition 2.16 Face-induced plane subgraph

Let $G' = \langle V', E', L', C' \rangle$ and $G = \langle V, E, L, C \rangle$ be two plane graphs. G' is a face-induced plane subgraph of G if:

- G' is a geometric subgraph of G,
- each internal face of G' is also an internal face of G.

For instance, the plane graph displayed in Figure 2.4(d) is a face-induced plane subgraph of the plane graph displayed in Figure 2.4(a) as all its faces are preserved. The plane graph in Figure 2.4(c) is not a face-induced plane subgraph because the yellow face, bounded by $(2, 6, 4, 3, 5)$, is indeed an internal face but does not correspond to any internal face of the graph in Figure 2.4(a). The extension of the definition of graph isomorphism to plane graphs also deserves a discussion as we may consider different extensions, whether they preserve the embedding in $\mathbb{R}^2$, the geometry (as defined for geometric graphs) or the faces. The extension that preserved faces was proposed by [Higuera et al. 2013] and is defined below.

Definition 2.17 Plane graph isomorphism

Let $G = \langle V, E, L, C \rangle$ and $G' = \langle V', E', L', C' \rangle$ be two plane graphs. G' and G are *plane graph isomorphic* if there exists an isomorphism function $f : V \to V'$ that preserves all internal faces, *i.e.*, for each internal face bounded by vertices

$(v_1, \ldots, v_n)$ in G there is an internal face bounded by vertices $(f(v_1), \ldots, f(v_n))$ in G'.

For instance, the two graphs of Figures 2.4(b) and 2.4(a) are plane isomorphic. Indeed, the isomorphism function $1 \to a, 2 \to b, 3 \to c, 4 \to d, 5 \to e, 6 \to f, 7 \to g, 8 \to h$ preserves all internal faces. For example, the internal face $(3, 8, 7, 5, 3)$ is matched by the isomorphism function to the internal face (c, h, g, e, c). Note that these two graphs are not geometrically isomorphic as there does not exist a rigid transformation between them.

Finally, we can define face-induced subgraph isomorphism as an isomorphism between a plane graph and a face-induced subgraph of another plane graph.

Definition 2.18 Face-induced plane subgraph isomorphism
Let G and G' be two plane graphs. G is face-induced plane subgraph isomorphic to G', denoted $G \subseteq_{face} G'$, if there exists a face-induced plane subgraph G'' of G' such that G and G'' are plane isomorphic.

For example, the graph of figure 2.4(d) is face-induced plane subgraph isomorphic to the graph of figure 2.4(b).

Computational complexity and algorithms. If subgraph isomorphism is an $\mathcal{NP}$-complete problem in the general case, [Higuera et al. 2013] introduces a polynomial-time algorithm to decide whether a pattern graph is face-induced plane sub-isomorphic to a target graph, provided that the pattern graph is 2-connected. It also shows us that the problem becomes $\mathcal{NP}$-complete when the pattern graph is not 2-connected. [Solnon et al. 2015] gives an FPT algorithm to decide whether a pattern graph is face-induced plane sub-isomorphic to a target graph when the pattern graph is not 2-connected: The time-complexity of this algorithm is exponential only in the number of 2-connected components in the pattern graph, and polynomial in the sizes of the two graphs (this algorithm is described in the context of generalized maps which are data structures to model plane graphs).

2.5 Depth-first search of a graph

Graph traversals are used to visit all vertices that may be reached from an initial vertex v_0. They use an exploration strategy to systematically explore the graph starting from v_0 and using edges to visit new vertices. Two main strategies are possible as described in [Cormen et al. 2009]:

- Breadth-first search (BFS), which expands the frontier between visited and unvisited vertices uniformly across the breadth of the frontier. It visits all vertices at distance k from v_0 before visiting any vertex at distance $k+1$.

- Depth-first search (DFS), which explores the graph from the most recently visited vertex v_i that still has unvisited neighbors. Once all neighbors of v_i have been visited, the search "backtracks" to visit neighbors of the vertex from which v_i was visited. This process continues until it has visited all the vertices that are reachable from v_0.

In this thesis, we focus on DFS, which is used to define canonical codes in some graph mining algorithms. Also, we only consider connected graphs, such that all vertices can be reached from the initial vertex v_0.

DSF uses colors to keep track of the progress of the traversal and associates a unique number to each vertex, corresponding to the order of visit of the vertices. It also marks every edge used to visit a new vertex as a *forward edge*. Other edges (that have not been used to visit vertices) are called *backward edges*.

DFS is described in algorithm 1. The input is an undirected graph $G = \langle V, E, L \rangle$ and an initial vertex $v_0 \in V$. At the beginning of the search, all vertices are white, and the counter $nb_{visited}$ that is used to number vertices is initialized to 0 (line 2). The first vertex to be visited is the initial vertex v_0 (line 3). Vertices are recursively visited by the VISIT procedure. When a vertex v_i is visited, it is colored in gray and is given a new number (lines 5-6). Then, for each white neighbor v_j of v_i, the edge (v_i, v_j) is marked as forward (line 9), and v_j is recursively visited (line 10). Once all neighbors of v_i are either black or gray, v_i is colored in black and the visit of v_i ends (line 11).

Algorithm 1 Depth-first search traversal.

```
1:  procedure DFS(G = ⟨V, E, L⟩, v_0)
2:      Color all vertices of V in white and initialize nb_visited to 0
3:      VISIT(v_0)

4:  procedure VISIT(v_i)
5:      Color v_i in gray
6:      Increment nb_visited, and give number nb_visited to v_i
7:      for all vertex v_j which is neighbor of v_i in G do
8:          if v_j is white then
9:              Mark edge (v_i, v_j) as forward
10:             VISIT(v_j)
11:     color v_i in black
```

At the end of $DFS(G, v_0)$, each vertex has a unique visit number, and edges used to visit vertices are marked as forward edges. Figure 2.5 illustrates step by step the progress of DFS on a graph.

The set of forward edges defines a tree, called *DFS tree*. The root of this tree is the initial vertex v_0; each other vertex $v_i \in V \setminus \{v_0\}$ has a unique parent which

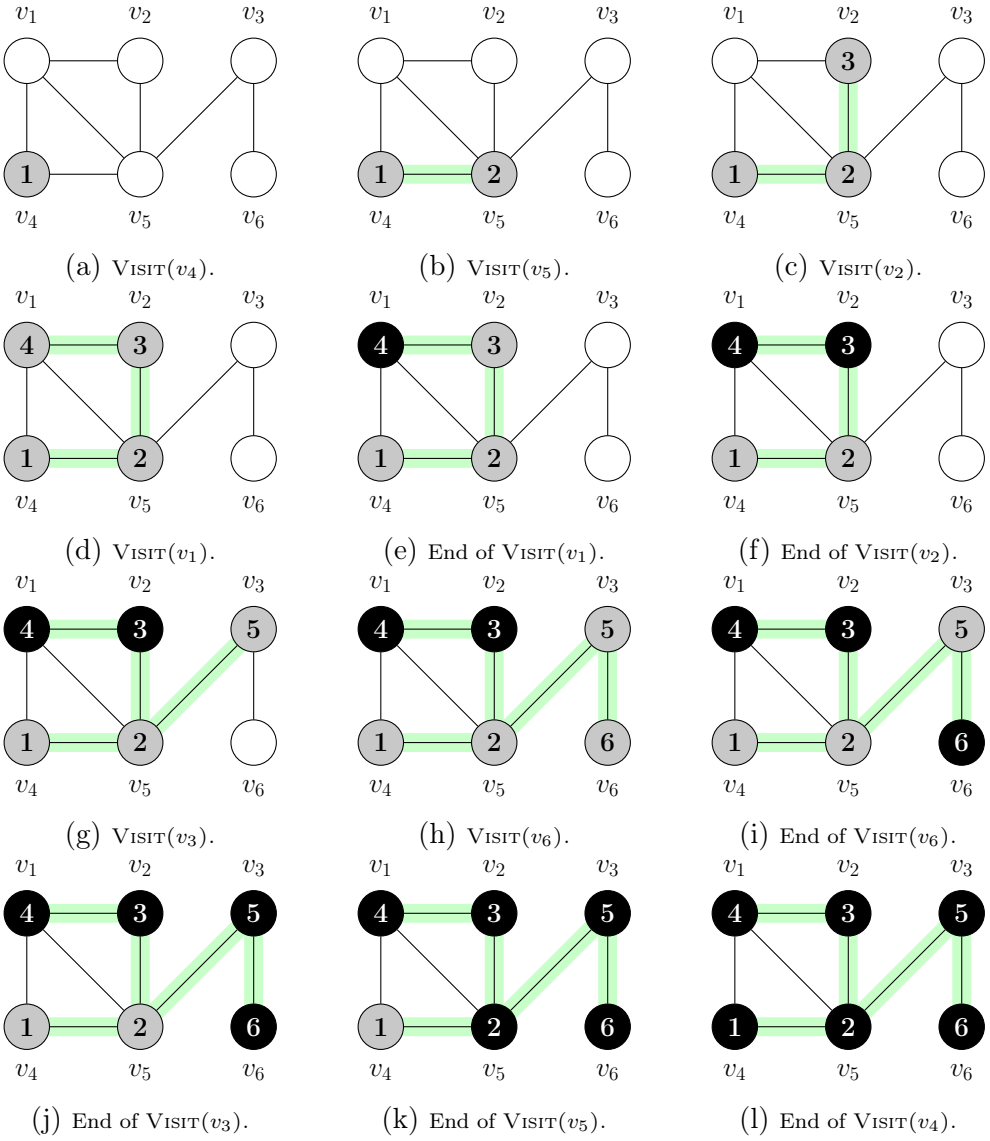
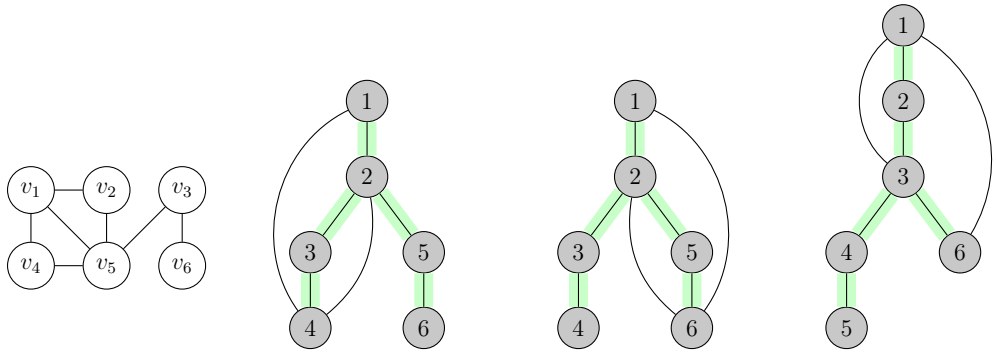


Figure 2.5 – Illustration of the different steps of a DFS algorithm starting from vertex v_4. Numbers assigned to vertices are displayed in the circles, and forward edges are displayed in green.

is the vertex that has called Visit(v_i). Note that the DFS tree depends on the initial vertex v_0 and on the order used to select the neighbors of v_i (line 7). Different results may be obtained when changing them. For instance, Fig. 2.6 displays three examples of DFS trees for a same graph.

Properties of DFS. The time complexity of algorithm 1 is linear with respect to the number of edges in the graph (provided that the graph is represented with adjacency lists).

For each forward edge (v_i, v_j), such that Visit(v_i) has called Visit(v_j), the number associated with v_i is always smaller than the number associated with v_j. This is a straightforward consequence of the fact that numbers are given



(a)

(b) DFS tree when starting from v_4 and successively visiting v_5, v_2, v_1, v_3, and v_6.

(c) DFS tree when starting from v_2 and successively visiting v_5, v_3, v_6, v_4, and v_1.

(d) DFS tree when starting from v_1 and successively visiting v_4, v_5, v_3, v_6, and v_2.

Figure 2.6 – A graph (a) and three possible depth-first tree (b), (c) and (d). Forward edges are represented in green, and visit numbers are displayed inside the circles.

to vertices at the beginning of the VISIT procedure, and that numbers are incremented after each number assignment.

2.6 Discussion

In this first chapter, we have defined geometric graphs and plane graphs which are special cases of labeled graphs. These graphs are of particular interest for this thesis because they are used to model the geometry of objects. Grid graphs, which are defined in Chapter 4 and for which we propose a new mining algorithm in Chapter 5, are closely related to these graphs.

Graph mining algorithms search for frequent patterns in graph databases. These patterns are sub-isomorphic to the database graphs. We have seen in this chapter that we may consider different definitions for the sub-isomorphism relation, depending on the properties that must be preserved by the sub-isomorphism. Table 2.1 summarizes these different sub-isomorphism relations.

Finally, graph mining algorithms usually represent graph patterns by canonical codes which are built by performing DFS traversals, and are a compact representation of DFS trees. Hence, we have described in this chapter the DFS algorithm and introduced DFS trees.

Table 2.1 – Different kinds of graphs and subisomorphism relations introduced in this chapter.

Kind of graph	Sub-isomorphism relation
Graph	Partial subgraph isomorphism ($G' \subseteq^{p} G$) preserves labels and pattern edges
	Induced subgraph isomorphism ($G' \subseteq^{i} G$) preserves labels and both pattern and target edges
Geometric graph	Partial geometric subgraph isomorphism ($G' \subseteq^{p}_{geo} G$) preserves labels, pattern edges, and geometry (up to a rigid transformation)
	Induced geometric subgraph isomorphism ($G' \subseteq^{i}_{geo} G$) preserves labels, both pattern and target edges, and geometry (up to a rigid transformation)
Plane graph	Face-induced plane subgraph isomorphism ($G' \subseteq_{face} G$) preserves labels, both pattern and target edges, and internal faces

Chapter 3

Existing Graph Mining Algorithms

Contents

In this chapter, we first define the general problem of pattern mining and focus on the particular case of graph patterns which are the main objects studied in this document. In order to understand our graph mining algorithm (presented in chapter 5), we also introduce in this chapter the related graph mining algorithms through their key ingredients: 1) the graph canonical representation used to avoid pattern duplicates; 2) the pattern search space exploration strategy; 3) the pattern expansion strategy and 4) the completeness and the correctness of the algorithm according to the general pattern mining problem.

3.1 The Graph Mining Problem

Data mining methods aim at processing large datasets to extract relevant information/pieces of knowledge for further use in a knowledge discovery process [Fayyad et al. 1996]. In this thesis, we focus on a particular subfield of data mining called *pattern mining*. The pattern mining goal is to find regularities in data, usually without using any knowledge about what to find in the data (no supervised information) but with a strong assumption on the shape of the regularities that should be found.

In this document, we are interested in graph patterns, which means that regularities take the shape of graphs (as defined in the previous chapter). Most of the definitions related to graph patterns are relevant for all types of patterns. We thus first present the generic definitions about pattern mining before discussing graph mining algorithms.

3.1.1 General Pattern Mining Problem

The pattern mining problem has been formally described in [Mannila et al. 1997]:

Definition 3.1 General Pattern Mining Problem
Given a database D, a language L for expressing the type of regularities we want to find and, a set of constraints C used for evaluating whether a pattern $P \in L$ is interesting. The task is to find $Th(L, D, C)$ where:

$$Th(L, D, C) = \{P \in L \mid C(P, D) \text{ is true}\}$$

In other words, we want to find **all** patterns $P \in L$ satisfying the constraint C in D.

One of the first successful algorithm to tackle this task was *Apriori* [Agrawal et al. 1994]. This algorithm was originally developed to analyze supermarket customer baskets and in particular, to understand for marketing purposes, which food items (e.g. *beer*, *pizza*, etc.) were bought together. The target language in this case was the set of all possible subsets of food items that could be bought together. This target class of patterns, called *itemsets*, has been used, since then, in many types of applications where the goal is to find unstructured subsets of elements (*e.g.*, gene expression levels in micro-arrays, co-occurring words in texts, co-occurring patches in images, etc.) that often occur together in the data. However, if n is the number of *items* in a given application (*e.g.*, food products in a supermarket), the number of possible subsets that can be extracted is 2^n (the number of possible subsets of a set with n elements). This number is usually much larger than the number of transactions considered (*e.g.*,

higher than the number of baskets ever bought in the supermarket). Finding such a huge number of patterns is thus at the same time, computationally intractable, but also useless to analyze.

The literature in pattern mining thus focuses on three core research problems: 1) defining relevant constraints to limit the number of patterns that should be output by the mining algorithm, 2) implementing algorithms that can make use of constraints in the most efficient way, 3) defining new types of regularities (itemsets, sequences, episodes, intervals, graphs, etc.). Itemsets are the "simplest" type of patterns tackled by existing pattern mining algorithms. Finding other types of regularities turned out to be even more complex, making essential the research works on the two first points.

3.1.2 Constraint-based Pattern Mining

A naive approach to tackle the general pattern mining problem could be to implement a simple "generate and test" algorithm: generate all possible patterns $P \in L$, and test if the constraint C is satisfied. However, as explained before, the number of possible candidate patterns is exponential in the dimension of the data. A huge effort has been made to define interestingness measures that can be used to constrain the space of candidates and reduce the number of produced patterns. The interestingness measures can be divided into two categories (see [Geng et al. 2006, Leeuwen et al. 2016] for survey papers): 1) *objective measures*, that depend only on the structure of the class of patterns and on the data, and 2) *subjective measures*, that can take into account user preferences to return more specific type of patterns.

In this document, we consider the measures, thus the constraints, in another way:

- **Pruning Constraints** which are tractable constraints that can be pushed during the mining process and allow to prune the search space of possible patterns.
- **Post-processing Constraints** which allow to refine the amount of interesting patterns found after the mining process.

Pruning Constraints

Mining algorithms usually explore the space of candidate patterns from the smallest ones to the largest ones, with respect to some size measure (*e.g.*, number of items in itemsets, number of edges in graphs, etc.): they iteratively expand current acceptable patterns to obtain larger ones, with respect to some expansion strategy (*e.g.*, adding an item to an itemset, adding an edge to a

graph, etc). Considering the huge size of this search space (see complexity arguments before), it is crucial to stop considering an expansion branch (*i.e.*, to prune the search space) as early as possible if the expansion branch has no chance to lead to patterns that fulfill the given constraints.

This pruning step can be done efficiently with constraints that present special properties such as monotonicity or anti-monotonicity properties. In this thesis, we mainly focus on anti-monotone constraints.

Definition 3.2 Anti-monotone constraint
A constraint C is anti-monotone if and only if, for all patterns p and p':

- if $p \subseteq p'$ and p' satisfies C, then p satisfies C.

where $\subseteq$ is a sub-pattern relation such that $p \subseteq p'$ if p' may be obtained from p by a sequence of expansions.

In other words, if a pattern p' satisfies a constraint C, any sub-pattern p of p' also satisfies this constraint C. Conversely, if a pattern p does not satisfy a constraint C, then all its super-patterns p' do not satisfy this constraint C either. When exploring the search space, an anti-monotone constraint C is used to safely prune a branch rooted at a pattern p if p does not satisfy the constraint C since none of the expansions of p satisfy C.

The most commonly used anti-monotone constraint is the support/frequency of a pattern p.

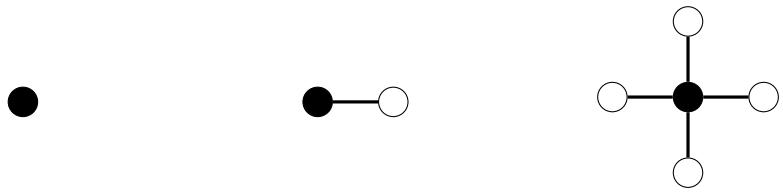
Definition 3.3 Pattern Support & Frequency
The support of a pattern p in a database $\mathcal{D} = \{e_0, e_1, ..., e_n\}$ corresponds to the number of transactions e_i in the database $\mathcal{D}$ which contain p:

$$support_{\mathcal{D}}(p) = |\{e_i | p \subseteq e_i \text{ and } e_i \in \mathcal{D}\}|$$

The frequency of p in $\mathcal{D}$ is the ratio of its support by the number of transactions (or examples) in the considered database:

$$frequency_{\mathcal{D}}(p) = \frac{support_{\mathcal{D}}(p)}{|\mathcal{D}|}$$

In the particular case of graph mining, it is possible to either mine patterns in a single graph or in a database of graphs. These two settings lead to two different definitions of the support of a graph pattern [Bringmann et al. 2008]. In figure 3.1, the pattern p_1 has only one occurrence in the graph G while the pattern p_2 has 4 different occurrences. Therefore in this case, the support is usually considered for non overlapping occurrences. This can be done by first building the overlap graph of the occurrences. Then, the support of a pattern p is the size of the Maximum Independent Set (MIS) of the overlap graph of the occurrences of p, which is anti-monotonic [Gudes et al. 2006]. In this thesis, we



(a) A pattern p_1 (b) A pattern $p_2 \supset p_1$ (c) A graph G

Figure 3.1 – Patterns with non-monotonic support. If the support is defined as the number of occurrences, then the support of p_1 in G is less than the support of p_2 even though $p_1 \subset p_2$.

consider the second setting and we define the anti-monotonic constraint *support of a graph pattern p* as:

Definition 3.4 Support of a graph pattern
The support of a graph pattern p in an database of graphs $\mathcal{D} = \{G_0, G_1, ..., G_n\}$ is the number of graphs $G_i \in \mathcal{D}$ to which the graph pattern is subgraph isomorphic:

$$support_{\mathcal{D}}(p) = |\{G_i | P \subseteq G_i \text{ and } G_i \in \mathcal{D}\}|$$

The support/frequency constraint can be combined with other constraints to further prune the pattern space [Yan et al. 2006]. In this line, some algorithms propose to search for *maximal patterns*. A pattern is said to be *maximal* when it has no super-pattern that is frequent. For example, SPIN [Huan et al. 2004] is one of the first graph mining algorithm that uses the maximal graph pattern constraint on graphs. Another constraint is the *closedness* of a pattern. A pattern is said to be *closed* when it has no super-pattern with the same support. Maximal patterns are a subset of closed ones. In [Yan et al. 2003], authors proposed an algorithm to find closed graph patterns. More recently, [Bendimerad et al. 2017] used the closedness constraint in urban data analysis context. With some supervised information about the targeted problem, one can constrain the patterns to be frequent in some part of the dataset (*e.g.*, for some classes) but not in others. Patterns that rely on this type of contraint, which are not especially anti-monotonic, are called *emerging* or *contrast* patterns [Borgelt et al. 2002].

Other anti-monotinic constraints have been designed to constrain the structure of the targeted patterns such as the size constraint, the total number of vertices or edges in graph patterns, planarity constraints, etc. Some constraints are not anti-monotone but can be converted into anti-monotone ones by preprocessing the attributes: they are called *convertible constraints*. [Pei et al. 2000, 2004] have characterized some constraints (among them, the convertible ones) that can be pushed into the mining process to prune the search space of patterns.

In [Soulet et al. 2005], authors propose a framework, as well as a new algorithm based on it, that use constraint based on SQL-like and syntactic primitives. By introducing the use of lower and uppper bounds of the constraint on an interval, it allows to use complexe constraints (such as *area* of a pattern) by using boolean combinations of the usual constraint (*e.g.* monotonic, anti-monotonic and convertible ones). On the other hand, authors of [Cerf et al. 2008] propose a new class of constraint to mine closed n-set, the *piecewise (anti-)monotonic* constraint which includes monotonic and ant-monotic constraints. To be able to use nonmonotonic contrasints during the mining process, authors of [Buzmakov et al. 2015] introduced the notion of "generalized monotonicity". They introduced a new class of constraints which are monontonic w.r.t. a chain of projections as well as an algorithm which use this constraint to be able to find patterns in polynomial time for interval tuple data.

Post-processing Constraints

Unfortunately, even when using pruning constraints, the number of output patterns is often too important to be humanly processed as noted by [Piatetsky-Shapiro et al. 1994]. Thus, it may be needed to further reduce the output and select only relevant patterns by applying constraints *a posteriori*, *i.e.*, after the mining step.

In this line of work, researchers have tried to find constraints that do not require any expert knowledge but are entirely data dependent: this is the case of Minimum Description Length-based constraints (MDL), that are used for example in KRIMP [Vreeken et al. 2011] or in SLIM [Smets et al. 2012], or entropy-based constraints [Bie et al. 2010, Bie 2011, Cheng et al. 2007, Fernando et al. 2014]. In MDL-based works, the aim is to find patterns that well compress the data. The intuition is that redundant patterns will not improve the compression criterion during the selection step and will thus not be selected. [Vreeken et al. 2011] have shown that, in practice, the compression criterion is also good to obtain patterns that have a high discriminative power, *i.e.*, patterns that can be successfully used as inputs for supervised machine learning (classification) algorithms. In the entropy-based framework [Bie et al. 2010, Bie 2011], algorithms rely on prior information on the data to choose the most interesting patterns. For example, they can use the class information available for each example when the targeted application is a classification problem.

In the following, we detail the post-processing constraints used in [Cheng et al. 2007] and [Fernando et al. 2014], as we have used them in our experiments (see Chapter 7) to select a set of discriminative patterns.

Let C be a set of classes, such that each class $c \in C$ is composed of a set I_c of elements/examples (images in [Fernando et al. 2014]). Let $I = \cup_{c \in C} I_c$ be the set of all elements. Each element $i \in I$ is represented by a set of features F, and for every feature $f \in F$ and every element $i \in I$, $i[f]$ is the value associated with feature f in i (for example, the number of occurrences of feature f in element i). The authors of [Fernando et al. 2014] propose two criteria to select the most discriminative features of P: *redundancy* and *relevance* which further allow to sort the features of F according to a combined redundancy/relevance interestingness measure.

Definition 3.5 Feature Redundancy
A feature redundancy measure is a function $R : F \times F \to \mathbb{R}$ such that, the higher $R(f, f')$, the more redundant the two features f and f'.

If the redundancy measure $R(f, f')$ is high enough (depending on a user-defined threshold), it is useless to select both features f and f' since they will express similar information according to R. In [Fernando et al. 2014], the redundancy between two features f and f' is defined as follows:

$$R(f, f') = \exp\left(p(f) \cdot \sum_{i \in I} p(i|f) \log \frac{p(i|f)}{p(i|\{f, f'\})} + p(f') \cdot \sum_{i \in I} p(i|f') \log \frac{p(i|f')}{p(i|\{f, f'\})}\right) \tag{3.1}$$

where

$$\forall f \in F, \forall i \in I, \quad p(i|f) = \frac{i[f]}{\sum_{i' \in I} i'[f]} \tag{3.2}$$

$$\forall f, f' \in F, \forall i \in I, \quad p(i|\{f, f'\}) = \frac{i[f] + i[f']}{\sum_{i' \in I} (i'[f] + i'[f'])} \tag{3.3}$$

$$\forall f \in F, \quad p(f) = \frac{\sum_{i \in I} i[f]}{\sum_{f' \in F, i \in I} i[f']} \tag{3.4}$$

Note that, in this case, $0 \leq R(f, f') \leq 1$ and $R(f, f') = R(f', f)$.

Definition 3.6 Relevance of a feature
The relevance measure is a function $S : F \to \mathbb{R}$ such that the higher $S(f)$ the more relevant the feature $f \in F$.

Authors of [Fernando et al. 2014] define a relevance score S derived from the Shannon entropy proposed in [Cheng et al. 2007] as:

$$S(f) = D(f) \times O(f) \tag{3.5}$$

where the discriminating score $D(f)$ is:

$$D(f) = 1 + \frac{\sum_{c \in C} p(c|f) \cdot log\ p(c|f)}{log\ |C|} \tag{3.6}$$

where $p(c|f)$ is the probability of a class $c \in C$ given a feature $f \in F$:

$$p(c|f) = \frac{\sum_{i \in I_c} i[f]}{\sum_{i' \in I} i'[f]} \tag{3.7}$$

Note that the use of the term $log\ |C|$, where $|C|$ is the number of class c, from equation 3.6 ensures that $0 \leq D(f) \leq 1$.

The second part of Equation 3.5, $O(f)$, is the representativity score of a feature: it "says" how well a feature represents a given class. To compute this, [Fernando et al. 2014] compare each feature f distribution over all elements $i \in I_c$ which belong to a given class $c \in C$, to the feature, f_c^*, with the "optimal" distribution for the class c. f_c^* is such that:

- The feature f only appears in the elements of class c, *i.e*, $p(c|f_c^*) = 1$ and $\forall c' \neq c, p(c|f_{c'}^*) = 0$,
- The feature follows a uniform distribution for all elements i in class c, *i.e.* $\forall i, i' \in I_c$, $p(i|f_c^*) = p(i'|f_c^*) = (1/|I_c|)$.

Thus, the representativity score of a feature f, $O(f)$, is the divergence between the feature distribution $p(i|f)$ and the optimal one $p(i|f_c^*)$. It is defined by:

$$O(f) = max_{c \in C}(exp\{\sum_{i \in I_c} p(i|f_c^*) \log \frac{p(i|f_c^*)}{p(i|f)}\}) \tag{3.8}$$

Finally, given a subset of features $\mathcal{F}$ that have already been selected, [Fernando et al. 2014] define the gain $G(f)$ brought by a feature $f \in F \backslash \mathcal{F}$ by combining relevancy and redundancy measures as follows:

$$G(f) = S(f) - max_{f' \in \mathcal{F}}\{R(f, f') \cdot min(S(f), S(f'))\} \tag{3.9}$$

This gain may be used to greedily select k features from F as follows (where k is a user-defined parameter). First, $\mathcal{F}$ is initialized to the empty set. Then, we iteratively select the feature $f \in F \setminus \mathcal{F}$ that maximizes the gain $G(f)$ and add it to $\mathcal{F}$, until $\mathcal{F}$ contains k features. Finally, we return the set $\mathcal{F}$.

3.2 Graph Mining Strategies

After having defined the general constraint-based pattern mining problem, we now focus on the particular case of graph mining which is the main topic of this

thesis. We present the state-of-the-art graph mining algorithms through their mining strategies.

3.2.1 Graph Canonical Representation

In graph mining, for efficiency reasons, it is crucial to avoid discovering the same graph pattern multiple times during the mining process. This prevents the mining algorithm from considering multiple times the same expansion branch and thus, it prunes the graph pattern space drastically. To do so, one needs to compare a candidate graph pattern with all the previously discovered patterns using a costly (see Chapter 2) isomorphism test. To make this comparison as efficient as possible, a usual trick in graph mining consists in converting each graph into a unique code (usually a list of elements of the graph) called a *canonical code*, such that two graphs are isomorphic if and only if their canonical codes are equals. Then, the graph pattern space becomes a code space and avoiding duplicates boils down to comparing codes. Building such a code for a given graph has the same theoretical complexity as doing an isomorphism test (in practice, heuristics are used to alleviate this step): given the canonical codes of two graphs, we can decide whether the two graphs are isomorphic or not by comparing their canonical codes, and this is done in linear time with respect to the length of the codes.

For instance, in [Inokuchi et al. 2000], the AGM algorithm represents a graph by concatenating the elements in the upper triangle of its adjacency matrix into a sequence. For example, suppose that a graph with n vertices has the following adjacency matrix :

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \cdots \\ x_{n-1,1} & x_{1,2} & \cdots & x_{n-1,n-1} & x_{n-1,n} \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n-1} & x_{n,n} \end{pmatrix}$$

The corresponding code of such a graph is:

$$code(X_n) = x_{1,1}x_{1,2}x_{2,2}x_{1,3}x_{2,3}x_{3,3}\cdots x_{n-1,n}x_{n,n}$$

It is possible to represent a graph by different sequences by permuting rows (and columns accordingly) of the adjacency matrix. Thus, before building the graph code, the indexes of the matrix are sorted based on the labels of the vertices. An order is defined on those codes and the minimum one is defined as the *canonical code* of the graph.

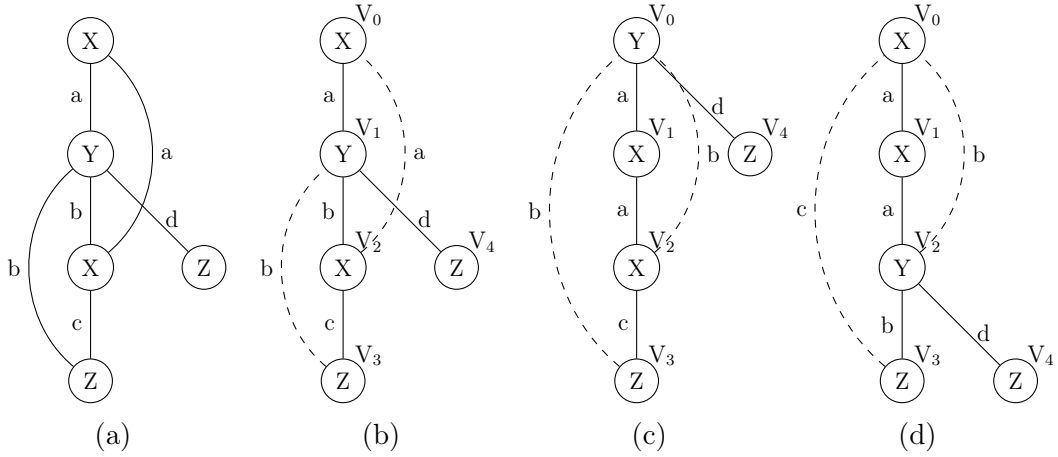


Figure 3.2 – A graph (a) and three possible DFS tree (b), (c) and (d). Forward edges are represented in plain lines and backward edges are represented in dashed ones.

edge	(b)	(c)	(d)
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,0,Z,b,Y)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)

Table 3.1 – DFS codes for the corresponding DFS trees of Figure 3.2, [Yan et al. 2002].

gSpan [Yan et al. 2002], one of the most popular graph mining algorithm, introduced another canonical representation: the code of a graph is built by performing a DFS traversal of the graph (see Section 2.5 for the definition of a DFS traversal). Figure 3.2 shows an example of a graph (a) and three possible DFS trees (b), (c) and (d), among all possible DFS trees. The code associated with a DFS tree is a sequence of m edge codes, where m is the number of edges of the graph, such that the kth edge code describes the kth traversed edge and is a tuple $(i, j, l_i, l_{(i,j)}, l_j)$ where:

- i and j are the visit numbers associated with the vertices of the kth edge,
- l_i and l_j are the labels of the vertices of the kth edge,
- $l_{(i,j)}$ is the label of the kth edge.

There is one possible code per different depth-first traversal. The respective codes of the three DFS trees of Figure 3.2 are reported in Table 3.1.

In [Yan et al. 2002], the *canonical code* is defined as the smallest code (according to a lexicographic order), among all codes corresponding to all possible DFS traversals of the graph. For example, let us consider the codes of Table 3.1

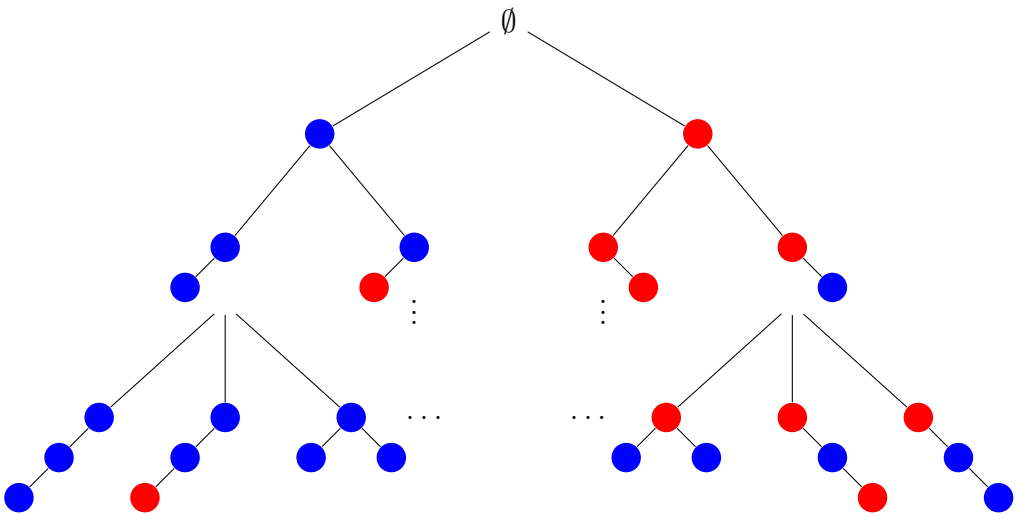


Figure 3.3 – Partial search space of all graph patterns with only two different vertex labels (red and blue).

and let us consider the alphabetical order to compare labels. The code of the first edge of the DFS code (b) is smaller than the code of the first edge of the DFS code (c), because $X < Y$. In this example, the code (d) is the minimum code in lexicographical order among those presented: it is the the canonical code associated to the graph (a) of Figure 3.2.

3.2.2 Exploring a Search Space of Canonical Codes

The search space of all possible patterns may be explored by building a search tree. Each node of this search tree corresponds to a pattern, and the parent of a pattern of size k is the pattern of size $k-1$ from which it has been expanded. For instance, Figure 3.3 presents a part of the search tree built to explore all possible graph patterns when there are only two vertex labels (red and blue), when the size of a graph pattern is defined as the number of its vertices. Each node is associated with a different graph pattern P, and has one child for every graph that may be obtained by adding a vertex to P.

As explained in Section 3.1.2, the size of the search tree is exponential in the number of vertices of the patterns and in the number of edge and vertex labels. *Anti-monotone constraints* such as the *minimum frequency constraint* are thus mandatory to prune the search tree.

There exist two main approaches to explore the search tree, the *Apriori*-like one and the *pattern growth*-based one. They respectively explore the search tree in a Breadth-First and in a Depth-First manner.

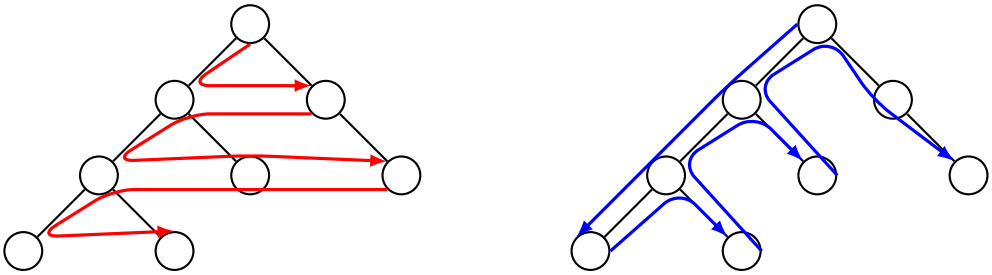


Figure 3.4 – Scheme of the BF exploration (on the left) and of the DF exploration (on the right) of a search space of patterns.

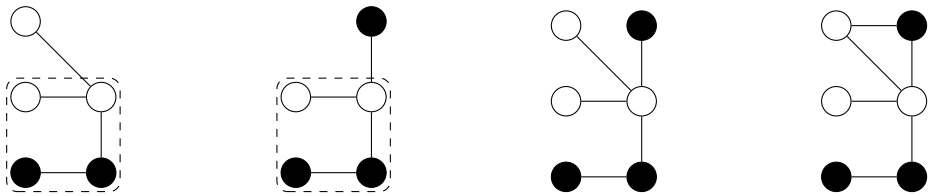
The Breadth-First strategy, popularized on itemsets by the *Apriori* algorithm [Agrawal et al. 1994], computes all patterns of size k before generating candidate patterns of size $k+1$ that share a common subgraph of size k previously computed. This approach is represented by the scheme on the left of Figure 3.4. One of the main drawback of this method is that it needs to generate and store all candidate patterns of size $k+1$ before computing their frequency, thus it requires a lot of memory. Some of the first graph mining algorithms using this approach are AGM [Inokuchi et al. 2000] and FSG [Kuramochi et al. 2001]

The *pattern growth* strategy generates new candidate patterns by recursively adding an edge or a vertex to an already found pattern until all super-patterns are discovered. It explores a branch of the tree until no further expansion is possible, then backtracks to explore another branch as shown by the scheme on the right of Figure 3.4. This approach is less costly in memory than the *Apriori* one as it only needs to store the graph patterns of the currently explored branch. The first graph mining algorithm to use this approach was gSpan [Yan et al. 2002]. It was later used by many other graph mining algorithms such as FreqGeo [Arimura et al. 2007] or Plagram [Prado et al. 2013].

Note that there is no clear best strategy to explore the search space of patterns. The performances of Breadth-First and Depth-First strategies depend on the data and are usually assessed through experimental studies.

3.2.3 Expansion Strategies

Once the exploration strategy set, one has to decide how to expand a particular graph pattern. This depends on the exploration strategy but also on the type of graphs that are mined or on the type of targeted patterns. AGM [Inokuchi et al. 2000] algorithm uses a Breadth-First strategy and grows patterns of size $k+1$ at each iteration by adding two vertices and at least two edges that connect the two new vertices to the original common pattern of size $k-1$ as shown in Figure 3.5. Because it is undetermined whether there is an edge connecting the two additional vertices, AGM can actually generate two possible candidates.



(a) A graph pattern p_1 of size 5 (b) Another graph pattern p_5 of size 5 (c) A possible candidate of size 6 (d) Another possible candidate of size 6

Figure 3.5 – Example of the behaviour of a level-join based expansion done by AGM where pattern p_1 and p_2 share a common subgraph, represented inside the dot line, and allow to construct two possible candidates

Figure 3.5 displays two patterns of size 5 and the two possible candidates of size 6 that AGM is able to generate. FSG [Kuramochi et al. 2001] uses an edge expansion strategy. In this paper, the size of a pattern is defined as its number of edges and two patterns of size k are merged if and only if they share the same subgraph having $k-1$ edges. The joining operation between graphs can be complex as shown in [Kuramochi et al. 2001].

Algorithms that use the pattern-growth exploration strategy usually expand a pattern by adding one elementary component. Usually, an elementary component is an edge or a vertex. One of the first graph mining algorithm using this approach was gSpan [Yan et al. 2002]. gSpan uses a *right-most extension* strategy to extend a graph pattern with an *edge*. Let v_0 and v_n be the first and the last vertices visited when performing the DFS traversal of a pattern graph to build its *DFS code*. Vertex v_n is called the *right-most vertex* and the direct path between v_0 and v_n in the DFS tree is called the *right-most path*. To ensure a valid expansion, the expansions are done either with *forward edges* on vertices of the *right-most path* or with *backward edges* on the *right-most vertex* of the DFS tree. With this property, gSpan avoids the generation of invalid extensions, thus reducing the size of the search space. Also any non canonical DFS code can be pruned without missing any canonical DFS code, similarly to the anti-monotonicity property of the frequency constraint.

Some algorithms can use multiple expansion strategies to build the search space such as FFSM [Huan et al. 2003] that uses both level-wise join operations between pairs of patterns of size k, where k is the number of edges, and a pattern-growth approach by adding one edge to a pattern of size k to produce a candidate pattern of size $k+1$. In this category, DPMine [Vanetik et al. 2002] reduces the number of candidate graph patterns by extending patterns with edge-disjoint paths instead of a simple vertex or edge. It starts by searching all frequent paths in the database and all frequent graph patterns with two paths. Then, using an Apriori-like exploration strategy, it merges pairs of graph

patterns with k paths that have $k-1$ paths in common to generate candidates with $k+1$ paths.

3.3 Most Related Algorithms

Our algorithm is presented in details in Chapter 5. It is closely related to gSpan [Yan et al. 2002], a general graph mining algorithm, Plagram [Prado et al. 2013], a plane graph mining algorithm, and FreqGeo [Arimura et al. 2007], a geometric graph mining algorithm. These algorithms are presented in this section as instantiations of a same generic depth-first graph mining algorithm to better highlight their common points and differences.

3.3.1 Generic Depth-First Algorithm

gSpan, Plagram, and FreqGeo all consider a same mining scheme: they use DFS canonical codes to represent graphs and perform a recursive Depth-First exploration of the search space of all canonical codes as described in Section 3.2.1. Algorithm 2 describes a generic graph mining algorithm that is a generalization of these algorithms. It enumerates all frequent patterns by recursively growing pattern codes: starting from basic pattern codes composed of a single elementary component which is frequent and canonical (lines 1-2), it recursively calls the *extend* procedure to grow pattern codes by adding elementary components. More precisely, given the canonical code P of a pattern, *extend*(P) outputs all canonical codes P' such that P is a prefix of P'. It first outputs P, and then builds the set E of all elementary components that may be added at the end of P (lines 6-9): for each graph G_i of D, it computes the set of every elementary component e such that the pattern graph corresponding to the code $P.e$ occurs at least once in G_i and adds all these extensions to E. Finally, *extend* is called recursively for each extension $e \in E$ such that $P.e$ is frequent and canonical (lines 9-10).

For each instantiation of Algorithm 2, the key points are (i) to define codes that describe elementary components, (ii) to define what is a valid extension of a pattern in a graph, and (iii) to implement a canonicity test to decide whether a given code is canonical or not. The following sections present the instantiations of the generic algorithm for gSpan, Plagram and FreqGeo and discuss differences between these algorithms around four axes:

- Type of graph mined
- Elementary component and extension strategy
- Canonical code

Algorithm 2 General DFS Frequent Subgraph Mining

Input: A database of graph $D = \{G_1, ..., G_n\}$ and a frequency threshold σ.
Output: All patterns P such that $freq(P, D) \geq \sigma$.

```
1: for all elementary component e do
2:     if The elementary component is frequent and is canonical then extend(e)
3:
4: procedure EXTEND(Pattern code P)
5:     Output P                                          ▷ P is canonical and frequent
6:     Initialization of an empty set E of possible extensions
7:     for all graph G_i in D do
8:         Compute all possible and valid extensions of P in G_i
9:         Add these extensions to E
10:    for all possible extensions e ∈ E do
11:        if the pattern P with the extension e is frequent and is canonical then
12:            extend(P.e)
```

Table 3.2 – Characteristics of subgraph mining algorithms. The type of mined patterns is a consequence of the sub-isomorphism relation and extension choices.

Algorithm	Relation	Mined Patterns	Extensions
gSpan	$\subseteq$	connected subgraphs	one edge
FreqGeo	$\subseteq_{\text{geo}}$	geometric subgraphs	one edge/node
Plagram	$\subseteq_{\text{face}}$	2-connected plane subgraphs	one face

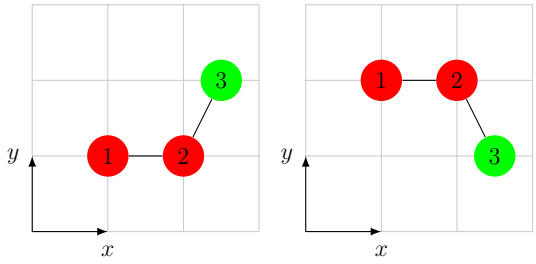
- Computational complexity

Table 3.2 summarizes the kind of mined patterns and elementary components considered by gSpan, FreqGeo, and Plagram as an extension.

3.3.2 gSpan

Graph mined and subgraph relation. gSpan mines general connected labeled graphs without any restriction on them. It uses the classical subgraph relation $\subseteq$ and does not consider structural informations hold by the graphs, *i.e.* it does not consider the topology neither the geometry of the graphs. Thus, gSpan consider two graphs as isomorphic whereas they can be geometrically different. For instance, for gSpan, graphs of Figures 3.6(a) and 3.6(b) are isomorphic whereas they have different geometric structures.

Elementary component and extension strategy. For gSpan an elementary component from which the mining process starts (line 1-2) is the code of a frequent edge in the database of graphs. The code of an edge is represented by a tuple $(i,j,l_i,l_{(i,j)},l_j)$ as described in section 3.2.1.



(a) A graph G_1 (b) A graph G_2

Figure 3.6 – For gSpan, graph G_1 and G_2 are isomorphic. However, their geometries are different, and they are considered as non isomorphic for FreqGeo.

Then, gSpan uses the *right-most path* extension strategy to construct a set of valid extensions from each occurrence of a pattern P in the database (line 8). It searches for incident edges from each occurrence of pattern P, ensures that they are on the right-most path, if so, adds edge codes to the set of valid extensions and update their frequency.

Canonical code. At each call of the function *extend*, the code of the pattern P grows by a new edge code. The canonical code of a pattern for gSpan is the minimal code as described in section 3.2.1. To check if the pattern code is canonical, gSpan constructs every possible DFS code for the pattern P. If one of these DFS codes is smaller than the code of the pattern P, it means that this code is not canonical and the branch can be pruned.

Computational complexity. The subgraph isomorphism problem that gSpan has to solve to find all pattern occurrences is an $\mathcal{NP}$-complete problem. Therefore, the runtime of gSpan is exponential. Still, if measured by the number of subgraph and/or graph isomorphism tests, the runtime of gSpan can be bounded by $\mathcal{O}(kFS+rF)$, where k is the maximum number of subgraph isomorphisms existing between a frequent graph pattern and a graph in the database, F is the number of frequent patterns, S the size of the database, *i.e.*, the number of graphs, and r is the maximum number of possible codes of a frequent pattern that grow from other minimum codes.

3.3.3 Plagram

Graph mined and subgraph relation. Plagram only considers 2-connected plane graphs and plane subgraph patterns. Moreover, it searches for face-induced patterns (described in section 2.4). Thus, it considers the face-induced

plane subgraph relation $\subseteq_{\text{face}}$ and cannot mine graphs depicted in Figure 3.6 as they are not 2-connected.

Elementary component and extension strategy. The elementary component for PLAGRAM (line 1-2) is the code of a frequent edge. Like GSPAN, the code of an edge is represented by a tuple $(i,j,l_i,l_{(i,j)},l_j)$ as described in section 3.2.1. Contrary to GSPAN, these elementary components do not represent the first possible patterns.

Each call to *extend* finds occurrences of a pattern P (for the first call, it finds occurrences of frequent edges) and searches for valid extensions. A valid extension is a sequence of incident edges such that the end vertices of this sequence are two different vertices on the external face of the pattern. Thus, adding this extension to the pattern adds a new face to the pattern. As a consequence, all patterns are composed of faces and the smallest possible subgraph pattern is a single face. For the first call, when starting from a frequent edge, this creates a pattern with only one internal face.

Canonical code. At each call of *extend*, the code of pattern P grows by a sequence of edge codes forming a face lying in the outer face of the pattern (the pattern grows by one face at each level of the search space instead of one edge for GSPAN). The lexicographic order is used to compare DFS codes. The canonical code for a pattern is the maximal one for this order.

Computational complexity. The canonical code of a pattern is computed in quadratic time with respect to the number of edges of the pattern. As a consequence, PLAGRAM has an incremental polynomial time complexity, which is $\mathcal{O}(p^3kn^2)$ per pattern, where k is the number of graphs in D, and p and n are, respectively, the number of edges in the pattern and in the largest graph in D.

3.3.4 FreqGeo and MaxGeo

Graph mined and subgraph relation. FREQGEO and MAXGEO consider only geometric graphs as described in section 2.3: graphs of the database and mined patterns hold geometric information (vertex coordinates). The subgraph relation used by the algorithm is $\subseteq_{\text{geo}}$ (two graphs are isomorphic if one is the image of the other by a composition of a translation, a rotation and a scaling function). For instance, graphs of Figures 3.6(a) and 3.6(b) are not isomorphic for FREQGEO.

Moreover, for FreqGeo, the frequency of a pattern is defined as the number of its occurrences in the database D as discussed in section 3.1.2 (for gSpan and Plagram, the frequency of a pattern is defined as the number of graphs of D which contain the pattern). MaxGeo is the restriction of FreqGeo to maximal patterns.

Elementary component and extension strategy. The initial component from which FreqGeo starts mining is an empty graph, thus the line (1-2) can be replaced by a single call of *extend* with an empty pattern in the place of elementary component.

An extension is defined as either a new vertex or an edge connecting two vertices already in the pattern.

Canonical code. The code of a pattern for FreqGeo differs from the one used by gSpan and Plagram. It is a sorted list of edges and vertices. The canonical code is the lexicographically minimum code.

Computational complexity. Computing extensions and testing the canonicity are done in polynomial time. As a consequence, FreqGeo has an incremental polynomial time complexity, which is $\mathcal{O}(k^2n^4 \cdot \log n)$ per pattern, where k is the number of graphs in D, and p and n are the number of edges in the pattern and the number of edges in the largest graph in D, respectively. Both FreqGeo and MaxGeo can be seen as a generalization of our grid mining algorithm. Since they mine geometric graphs, they are not optimized for cases where the graph structure is known (such as grids) and we show in Chapter 5 that their complexity is higher than the complexity of our approach. Besides, the authors did not provide any implementation (thus, also no experiment) of their proposed algorithm neither in [Arimura et al. 2007] nor in further publications which could allow an experimental comparison with our method.

3.4 Other Graph Mining Algorithms

According to Definition 3.1 given at the beginning of this chapter, pattern mining algorithms output all the patterns that fulfill the given constraints C. If most algorithms stick to this definition (the *exact* mining algorithms), some algorithms have relaxed this definition to either output some (ideally interesting) patterns or to fulfill the constraints only up to a certain extent (the *inexact* mining algorithms). Before discussing the inexact mining algorithms and for the sake of completeness, we present the existing exact graph mining algorithms that were not mentioned in the previous sections.

3.4.1 Exact Mining algorithms

MoFa [Borgelt et al. 2002] is a well known Depth-First graph mining algorithm. It stores all occurrences of the frequent graph patterns to generate only graph patterns that appear in the database. Moreover, when a new graph pattern is generated, finding its occurrences to compute its support is based on the occurrences of the sub-pattern from which it has been grown and which only match the newly added edge. MoFa also defines an order on the vertices of a graph pattern based on their time of addition to it. Thus, when a vertex is added to a pattern, an extension can only happen at this vertex or the ones added after it. This strategy allows to limit the generation of duplicate candidates but do not avoid it. GASTON [Nijssen et al. 2004] also stores the list of occurrences but its main feature is that it speeds up the mining process by splitting it into path mining, then subtree mining and finally subgraph mining.

An extensive comparison of gSpan, MoFa, FFSM and GASTON was presented in [Wörlein et al. 2005]. It concluded that storing occurrences does not considerably speed up the mining process. Even if gSpan does not use it, it remains competitive with other algorithms unless graph patterns become too large. Authors noted that the candidate generation and the computation of the isomorphism tests (to compute occurrences list or to compute the support) cost more in computational time than the pruning of duplicate candidates. The use of a canonical representation to detect duplicates is more efficient than directly performing the isomorphism test, and GASTON strategy to delay the generation of graph to later stages is even more efficient.

Previously described algorithms mine all frequent graph patterns without any constraints on the target patterns. gFSG, proposed in [Kuramochi et al. 2007], focuses on mining frequent geometric subgraphs. It uses an Apriori-like search strategy to explore the search space, also stores lists of occurrences of patterns to decrease the number of isomorphism tests and constructs a k-edge pattern by joining pairs of $(k-1)$-edge patterns that share a common subgraph. However, it only allows three rigid transformations, described in section 2.3, with the possibility to set a tolerance threshold on the vertex coordinates when testing if two graph patterns are isomorphic.

CloseGraph [Yan et al. 2003] is based on gSpan but mines closed subgraphs only. It explores the search space in a Depth-First manner, uses DFS codes to represent graphs and uses the right-most extension strategy. However, contrary to gSpan, it prunes the search space by detecting if an extension can lead to a new closed pattern.

SPIN [Huan et al. 2004] is inspired by GASTON but mines maximal subgraphs. It first mines frequent trees which are later extended into graphs by

adding edges to them.

Some other graph mining algorithms focus on discovering clique patterns, which are graphs where every two distinct vertices are adjacent. For example, CLAN [Wang et al. 2006] mines frequent closed cliques in a database of graphs with labels only on vertices. It uses the fact that cliques are fully connected sets of vertices, thus any two cliques with the same vertices have the exact same structure. Therefore the representation of a clique can be a simple sequence of the labels of its vertices and the canonical code is the minimum sequence according to a lexicographic order. It explores the search space in a Depth-First way and an extension is a vertex adjacent to all other vertices already in a pattern. It also benefits of the fact that for a clique composed of k vertices, each vertex must have a degree of at least $k-1$. Thus, when scanning the database of graphs to find possible extensions of a pattern, only vertices with a degree of at least $k-1$ are considered.

An extension of CLAN was proposed by [Zeng et al. 2006]: the algorithm Cocain, which mines closed γ-quasi-cliques from large and dense graph. A graph of size k is a γ-quasi-clique if the degree of all its vertices is above $\gamma(k-1)$, where $0.5 \leq \gamma \leq 1$ is a user parameter. Similarly to CLAN, Cocain uses canonical representations for quasi-cliques, explores the search space in a Depth-First manner and prunes the exploration using structural properties of quasi-cliques, *i.e.* the fact that in cliques, there exists an edge between every pair of vertices.

Finally, mSpan [Gosselin et al. 2011] is another algorithm that mines frequent pattern from combinatorial maps. A combinatorial map can be seen as the generalization of plane graphs in $n\mathcal{D}$: it describes the subdivision of an nD object in cells (vertices, edges, faces, volumes, etc), and incidence and adjacency relations between these cells. mSpan starts from all frequent patterns composed of one face, then combine these one face patterns to create new patterns. It also uses a signature as a canonical code that allows to do the isomorphism test in linear complexity.

Instead of mining graph databases, some algorithms search for frequent graph patterns in a single graph. [Kuramochi et al. 2004, 2005] presented two algorithms (called HSIGRAM and VSIGRAM) that mine a single graph. They respectively explore the search space using a Breadth-First and Depth-First approach. The support of a graph pattern is defined as its number of non overlapping occurrences as described in section 3.1.2 using the MIS measure. Authors implemented several variations of the MIS measures, exact and approximate, thus leading to HSIGRAM and VSIGRAM to be able to mine exact and approximate frequent graph patterns. They show that both algorithms scale well on large sparse graphs, although VSIGRAM is faster than HSIGRAM. This

is due to the fact that it stores occurrences of the frequent graph patterns along the DFS path resulting in less isomorphism tests.

More recently, the GRAMI algorithm, presented in [Elseidy et al. 2014], proposed an alternative approach that does not maintain a complete list of pattern occurrences. It models the graph pattern evaluation as a constraint satisfaction problem (CSP). During each iteration, it solves the CSP until it finds the minimal set of occurrences that satisfy the minimum support threshold and ignores other occurrences.

Interested readers can refer to the survey on frequent graph mining in [Jiang et al. 2013] to learn more about the frequent graph mining algorithms described previously and other techniques not reported here such as tree mining algorithms or relational pattern mining.

More recently, researchers have investigated the use of parallelization and GPUs to improve the efficiency of existing graph mining algorithms [Kessl et al. 2014] such as GSPAN or GASTON. Like GSPAN, they mine frequent patterns in a Depth-First manner and like GASTON they store a list of all occurrences of patterns. They observed that GPU memory allocation and I/O transfers between the CPU and the GPU induce a large overhead.

3.4.2 Inexact and Incomplete Mining Algorithms

Sometimes, user can accept minor variations between graph patterns mined. Thus, inexact frequent graph mining algorithm allows patterns with some minor variation to be considered as one same pattern. While this increases the possible number of frequent graph patterns, these techniques are usually not interested in finding all of them. Instead, inexact mining algorithms focus on extracting more interesting patterns that capture information in the data. They are able to find patterns that would have not been found by exact frequent graph mining algorithm because of small variations in the graphs of the databases or in the patterns.

One of the first popular of such algorithm is SUBDUE [Holder et al. 1994]. It uses a graph edit distance to measure the similarity between two graphs. The edit distance is the minimal number of operations needed, in terms of vertex and edge additions, deletions and label modifications, to transform a graph into another. It performs a Beam Search of the search space.

Another well known approximate graph mining algorithm is GREW [Kuramochi et al. 2004]. It focuses on finding graph patterns which do not have vertex in common in a single large graph. It explores the search space in a Breadth-First manner. New possible graph patterns are built by merging frequent graph patterns of previous iterations connected by one or several vertices. It also uses

some heuristics that can rewrite the input graph by collapsing vertices of occurrences of frequent patterns into a single vertex, reducing the relative size of the graph mined at each iteration. As a consequence, GREW underestimates the frequency of a pattern leading to missed patterns that can actually be frequent. Experiments show that GREW outperforms SUBDUE as it can discover larger patterns in less time.

Monkey is another approximate subgraph mining algorithm presented in [Zhang et al. 2007] that mines a dabatase of graphs. Authors define a β-edge isomorphism relation. Two graphs G_1 and G_2 are β-edge isomorphic it there is a subgraph isomorphism between G_1 and G_2 for which at most β-edges are not preserved. Unlike GREW and SUBDUE, it explores the search space with a Depth-First strategy to discover frequent approximate trees. Then, those approximate frequent trees are recursively extended by adding an edge connecting two of their vertices. Authors show that Monkey was able to find interesting patterns that exact frequent graph mining algorithms could not find. But increasing β drastically slows it down due to the explosion of the number of approximate frequent patterns.

This concept of β-edge isomorphism is also used by RAM algorithm [Zhang et al. 2008]. It uses feature vectors instead of canonical codes to check if a candidate graph pattern has already been found. Like using canonical codes, if a pattern is found and has an equivalent feature vector of another already found pattern, then the search space is pruned. The use of this representation implies that two isomorphic patterns have the same feature vector, but it is also possible that two different patterns share the same feature vector. Thus, some pattern might be eluded during the mining process. To limit this impact, the algorithm extends a pattern by adding edges in a random order. This means that multiple runs of the algorithm can lead to different outputs. Experiments showed that RAM could discover interesting patterns that are missed by exact mining algorithms and multiple runs of the algorithm can allow to miss less patterns.

gApprox algorithm proposed in [Chen et al. 2007] mines frequent patterns in a single large graph. It allows differences in edges and, if the user provides a list of admissible replacements, differences in labels. But structural differences in vertices, *i.e.*, the facts that some vertices are not present in some occurrences, are not allowed as it requires a bijective function between the vertex sets of matching graphs. Authors use a variant of the graph edit distance measure to define if two patterns are similar. Finally, the search space is explored in a Depth-First manner where the pattern support is determined by computing an upper bound of the maximum number of vertex disjoint occurrences.

The authors of APGM [Jia et al. 2011] proposed to use a compatibility real

matrix M, indexed by the vertex labels and where an entry $M(i, j)$ represents the probability of the label i to be mistaken with label j. With this representation, authors define a similarity measure and define that two graphs are approximately isomorphic if their similarity is below a user defined threshold σ. This algorithm mines a database of graphs in a Depth-First way and the support of a graph pattern is defined as the number of graphs G in the database of graphs that hold an approximate occurrence of the pattern.

Authors of [Jia et al. 2011] mentioned that their idea can be extended to take into account edge labels. This extension was proposed in [Acosta-Mendoza et al. 2012] with the algorithm VEAM. Similar to APGM, VEAM defines an approximate sub-isomorphism in which they take into account the probability of vertex and edge label substitutions. Thus, VEAM allows variations of labels but it requires that two graphs are approximately isomorphic if they have the same topology.

More recently, [Flores-Garrido et al. 2015] presented AGraP, an algorithm that focuses on searching frequent patterns in a single graph using inexact matching allowing structural differences in vertices and edges. They handle the fact that pattern occurrences can differ in their edge and vertex labels, similarly to gApprox, but also in their number of edges and vertices, *i.e.*, they can have different numbers of edges and vertices. Authors propose two similarity functions to compare graphs using inexact matching. Their algorithm explores the search space in a Depth-First manner using these similarity functions and a new search strategy to identify patterns that can have structural differences with respect to their occurrences. Experiments showed that AGraP is able to find all patterns found by gApprox and additional patterns up to twelve times as more as gApprox. Moreover, these additional patterns positively affected the accuracy in a classification experiment. However, AGraP generally requires more time that gApprox to mine a given database.

Other approaches focus on mining *probabilistic graphs*, *i.e.*, graphs with a probability of existence assigned to each edge. In this case, uncertain graphs are graphs $G = \langle V_G, E_G, P\rangle$ where $P(e)$ is the probability of existence of an edge $e \in E_G$. An uncertain graph implicates a set of exact graphs $I(G) = \{I = \langle V_G, E_i\rangle | E_I \subseteq E_G\}$. The probability that an uncertain graph G implies an exact graph I is :

$$P(G \implies I) = \prod_{e \in E_I} P(e) \prod_{e' \in E_G \setminus E_I} (1 - P(e'))$$

In [Zou et al. 2009], authors present MUSE, an algorithm that searches for frequent graph patterns in a database of uncertain graphs. In this case, for a given database of uncertain graphs $D = \{G_1, G_2, ..., G_n\}$, the probability that

a pattern p occurs in one of the uncertain graph G_i is :

$$P(p \subseteq G_i) = \sum_{I \in I_(G_i)} P(G_i \implies I).\Phi(I, p)$$

where $\Phi(I, p) = 1$ if p is subgraph isomorphic to I, 0 otherwise. The expected support of p in the database D is defined as:

$$esup_D(p) = \frac{1}{|D|} \sum_{i=1}^{|D|} P(p \subseteq G_i)$$

MUSE performs a Depth-First exploration of the search space of possible graph patterns and approximates their expected support. Experiments showed that the approximation of the expected support leads to few false positive, *i.e.*, patterns considered as frequent while they are not. Moreover, authors also showed that MUSE has a time complexity that increases linearly with the number of uncertain graphs in the database.

Other algorithms mine graphs to find a specific subset of graph patterns. SpiderMine presented in [Zhu et al. 2011] focuses on mining Top-K largest patterns mainly in a single graph, *i.e.*, finding the K largest frequent patterns with a maximum diameter. The diameter of a graph is the maximal distance between any pair of vertices, where the distance between two vertices is the length of the shortest path between them. As finding all exact Top-K largest patterns means to compute all possible pattern sets, authors use a randomized framework to compute the top-K largest patterns with a user defined probability $1 - \epsilon$. To do so, they propose a new graph representation they called *spider*. These *spiders* are all frequent patterns with a specified diameter from which a subset of them are randomly selected and used to construct pattern candidates. Authors conducted experiments mainly on a single graph but also on a database of graphs and show the efficiency and the scalability of their method.

3.5 Discussion

In this chapter, we defined the graph mining problem, and described existing graph mining algorithms, with a specific focus on algorithms that explore a search space of canonical codes in a Depth-First way. We have described a generic algorithm that follows this basic principle, and show how it may be instantiated to define well-known algorithms dedicated to general graphs (gSpan), plane graphs (Plagram), and geometric graphs (FreqGeo and MaxGeo).

Our new algorithm, GriMA, is described in Chapter 5 as an instantiation of this generic algorithm, and it is dedicated to graphs that have a grid geometry. This allows us to compare our new algorithm with existing ones, and better highlight their common points and differences.

Chapter 4

Definitions on Grids

Contents

Many objects have a regular structure, as illustrated in Figure 4.1: images are regular grids of pixels; game boards for playing go or chess are regular grids of squares on which pieces are placed; cellular automata are regular grids of cells that have a state. In Section 4.1, we formally define these spatial regular structures, called $2\mathcal{D}$ grids, as well as isomorphism and sub-isomorphism relations that are used to compare them.

In some applications, these grids evolve through time. This is the case for the three examples of Figure 4.1: a video is a temporal sequence of grids of pixels; go boards evolve during a game; cell states of a cellular automata evolve through time. In Section 4.2, we formally define these spatio-temporal regular structures, called $2\mathcal{D} + t$ grids, and extend isomorphism and sub-isomorphism relations to them.

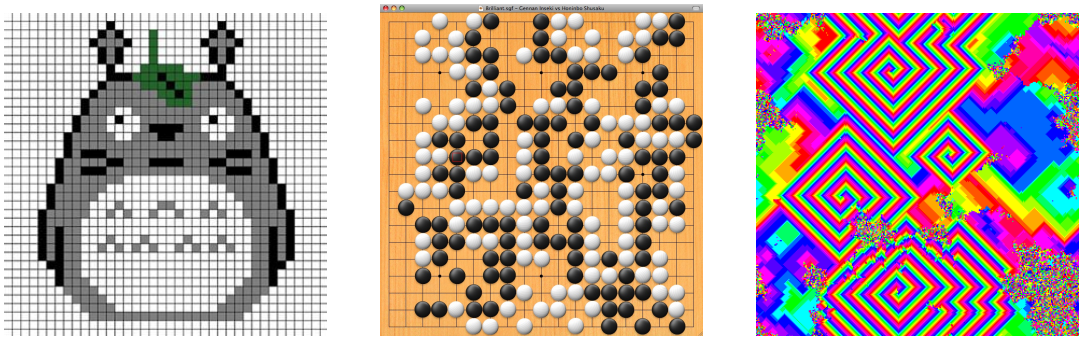
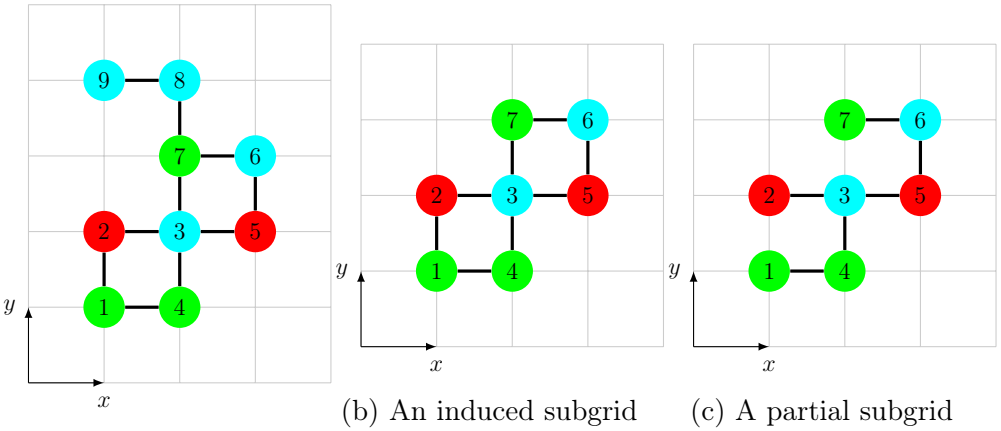


Figure 4.1 – Examples of objects modeled by grids: An image (left), a go board (middle), and a cellular automata (right)



(a) Example of $2\mathcal{D}$ grid. (b) An induced subgrid of (a). (c) A partial subgrid of (a).

Figure 4.2 – Example of a $2\mathcal{D}$ grid, and induced and partial $2\mathcal{D}$ subgrids

4.1 $2\mathcal{D}$ grids

$2\mathcal{D}$ Grids are particular cases of geometric graphs, *i.e.*, they are labeled undirected graphs whose vertices have $2\mathcal{D}$ coordinates. However, these coordinates must have integer values, and edges can only connect vertices which have neighbor coordinates. By definition, grids are also particular cases of plane graphs because edges cannot intersect as they only connect vertices that have neighbor coordinates.

Definition 4.1 $2\mathcal{D}$ **grid**
A $2\mathcal{D}$ grid (or grid for short) is defined by a tuple $G = \langle V, E, L, x, y\rangle$ such that:

- $\langle V, E, L\rangle$ is a graph,
- $x : V \to \mathbb{Z}$ and $y : V \to \mathbb{Z}$ are two functions that map each vertex $v \in V$ to coordinates in $2\mathcal{D}$, *i.e.*, $(x_v, y_v) \in \mathbb{Z}^2$,
- $\forall (u, v) \in E$, u and v have neighbor coordinates, *i.e.*,

$$|x_u - x_v| + |y_u - y_v| = 1.$$

In other words, each vertex v is linked by an edge with at most four vertices which are at distance 1 from v. Figure 4.2(a) displays an example of $2\mathcal{D}$ grid. Vertex 1 has coordinates $x_1 = 1$ and $y_1 = 1$, and is connected by an edge with vertices 2 (at coordinates $x_2 = 1$ and $y_2 = 2$), and 4 (at coordinates $x_4 = 2$ and $y_4 = 1$). It cannot be connected by an edge with vertex 3 as $|x_1 - x_3| + |y_1 - y_3| = 2$.

Definitions of induced subgraph and partial subgraph can be applied to $2\mathcal{D}$ grid: an induced subgrid is obtained by deleting vertices (and every incident edge to a deleted vertex) from a grid, and a partial subgrid is obtained by

deleting vertices and edges. For example, the grid of figure 4.2(b) is an induced subgrid of the grid of figure 4.2(a) obtained by removing vertices 8 and 9 whereas the grid of figure 4.2(c) is a partial subgrid obtained by removing vertices 8 and 9, and edges $(7, 3)$ and $(1, 2)$.

Looking for patterns in a grid amounts to searching for subgrid isomorphisms. Patterns in grid mining context should be invariant to translations and rotations. Therefore, similarly to geometric graph isomorphism, grid isomorphism allows rigid transformations which are translations and rotations.

Definition 4.2 Translation of a 2$\mathcal{D}$ grid
Let $G = \langle V, E, L, x, y\rangle$ be a 2$\mathcal{D}$ grid and $\mathcal{T} \in \mathbb{Z}^2$ be a translation vector. The translation of G by t, denoted $G + t$, is the grid obtained by moving all vertices of G by t, *i.e.*, $G + \mathcal{T} = \langle V, E, L, x', y'\rangle$ where $\forall v \in V, (x'_v, y'_v) = (x_v, y_v) + \mathcal{T}$.

Definition 4.3 Rotation of a 2$\mathcal{D}$ grid
Let $G = \langle V, E, L, x, y\rangle$ be a 2$\mathcal{D}$ grid and $\Theta_{2\mathcal{D}} = \{\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right), \left(\begin{smallmatrix}0 & 1\\ -1 & 0\end{smallmatrix}\right), \left(\begin{smallmatrix}-1 & 0\\ 0 & -1\end{smallmatrix}\right), \left(\begin{smallmatrix}0 & -1\\ 1 & 0\end{smallmatrix}\right)\}$ be the set of rotation matrices corresponding to 0, $\pi/2$, π and $3\pi/2$ angles, respectively. The rotation of G by $\theta \in \Theta_{2\mathcal{D}}$, denoted $\theta \cdot G$, is the 2$\mathcal{D}$ grid obtained by applying rotation θ to all vertices of G, *i.e.*, $\theta \cdot G = \langle V, E, L, x', y'\rangle$ where $\forall v \in V, (x'_v, y'_v) = (x_v, y_v) \cdot \theta$.

For geometric graphs, it is also possible to apply scaling transformations, besides rotations and translations. However, this transformation is meaningless in a grid context as edges only connect vertices that have neighbor coordinates.

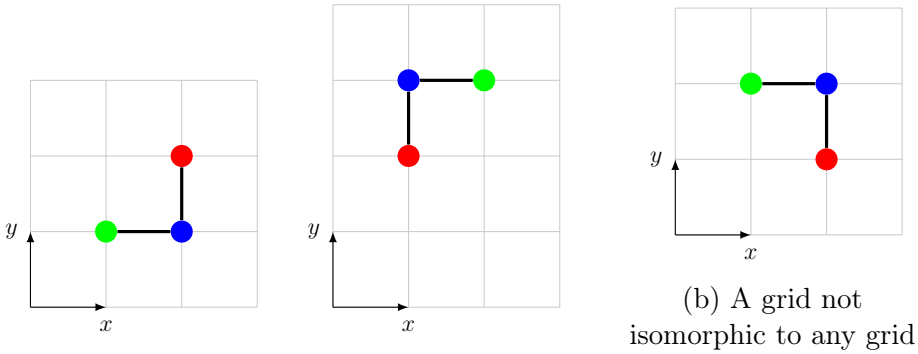
2$\mathcal{D}$ grid isomorphism searches for an isomorphism function between two grids that preserves the geometry up to a translation and a rotation.

Definition 4.4 2$\mathcal{D}$ grid isomorphism
Let $G = \langle V, E, L, x, y\rangle$ and $G' = \langle V', E', L', x', y'\rangle$ be two 2$\mathcal{D}$ grids. G and G' are 2$\mathcal{D}$ grid isomorphic if:

- There exists an isomorphism function f between the graphs $\langle V, E, L\rangle$ and $\langle V', E', L'\rangle$.
- There exist a translation $t \in \mathbb{Z}^2$ and a rotation $\theta \in \Theta_{2\mathcal{D}}$ which preserves vertex coordinates, *i.e.* $G' = \theta \cdot G + \mathcal{T}$

For example, the two grids displayed in Figure 4.3(a) are 2$\mathcal{D}$ grid isomorphic as the right-hand side grid may be obtained from the left-hand side grid by applying rotation π and translation $t = (3, 4)$. However, the grid of Figure 4.3(b) is not 2$\mathcal{D}$ grid isomorphic to any of the grids of Figure 4.3(a) as it cannot be obtained by rotating or translating them.

Finally, the definition of subgrid isomorphism follows from the definition of subgrids and isomorphism.



(a) Two isomorphic grids.

(b) A grid not isomorphic to any grid of (a).

Figure 4.3 – Example of $2\mathcal{D}$ grid isomorphisms

Definition 4.5 $2\mathcal{D}$ **subgrid isomorphism**
A $2\mathcal{D}$ grid G is subgrid isomorphic to a $2\mathcal{D}$ grid G', denoted $G' \subseteq_{\text{grid}_{2D}} G'$, if there exists a subgrid G'' of G' such that G is $2\mathcal{D}$ grid isomorphic to G''.

Depending on whether the subgrid relation is induced or partial, we obtain two different definitions of subgrid isomorphism, respectively denoted $G \subseteq^{i}_{\text{grid}_{2D}} G'$ and $G \subseteq^{p}_{\text{grid}_{2D}} G''$.

4.2 $2\mathcal{D} + t$ grids

A $2\mathcal{D} + t$ grid is a temporal sequence of $2\mathcal{D}$ grids, where vertices in two consecutive $2\mathcal{D}$ grids of the sequence may be linked with temporal edges. $2\mathcal{D} + t$ grids may be used to model the temporal evolution of structured objects such as, for example, videos, board games or cellular automata.

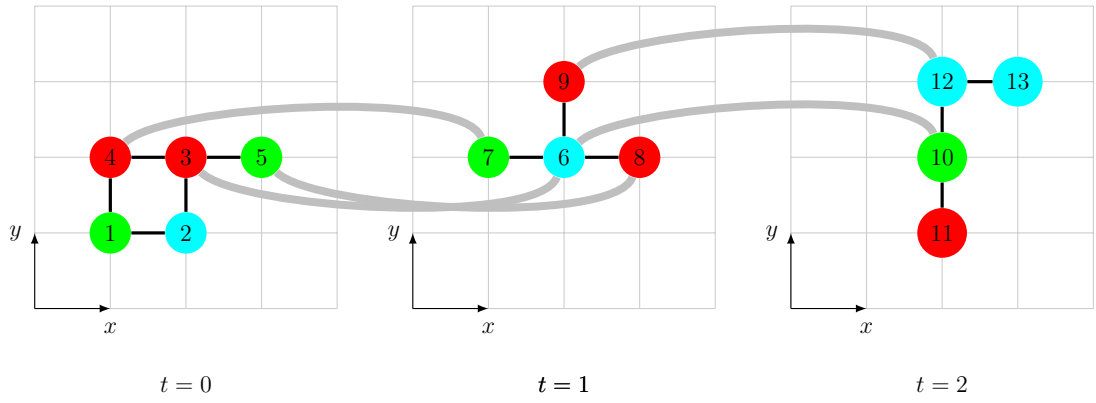
Definition 4.6 $2\mathcal{D} + t$ **grid**
A $2\mathcal{D} + t$ grid is defined by a tuple $G = \langle V, E, L, x, y, t \rangle$ such that:

- $\langle V, E, L \rangle$ is a graph.
- $x : V \to \mathbb{Z}$, $y : V \to \mathbb{Z}$, and $t : V \to \mathbb{Z}$ are functions that map each vertex $v \in V$ to $2\mathcal{D} + t$ coordinates $(x_v, y_v, t_v) \in \mathbb{Z}^3$. (x_v, y_v) are *spatial coordinates* whereas t_v is a *temporal coordinate*.
- $\forall (u, v) \in E$, u and v have neighbor coordinates, *i.e.*,

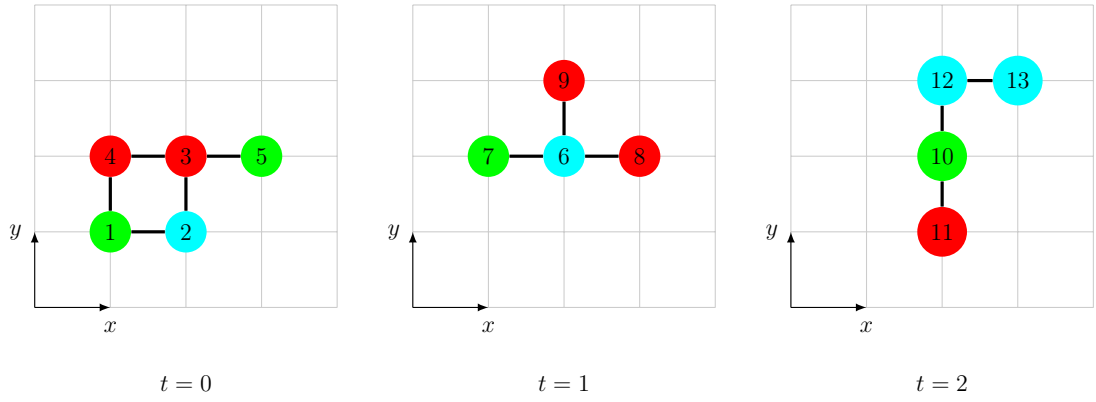
$$|x_u - x_v| + |y_u - y_v| + |t_u - t_v| = 1.$$

  If $|x_u - x_v| + |y_u - y_v| = 1$, then (u, v) is a *spatial edge* (and in this case $t_u = t_v$). If $|t_u - t_v| = 1$, then (u, v) is a *temporal edge* (and in this case $x_u = x_v$ and $y_u = y_v$).

Figure 4.4(a) displays a $2\mathcal{D} + t$ grid with three time steps. Edges $(4, 7)$, $(3, 6)$, $(5, 8)$, $(6, 10)$ and $(9, 12)$ are temporal edges and are displayed in gray. In



(a) A $2\mathcal{D}+t$ grid with three time steps. Temporal edges are displayed in gray.



(b) Same $2\mathcal{D}+t$ grid as (a), but where temporal edges are not displayed: We implicitely assume that whenever two vertices have the same spatial coordinates, and neighbor temporal coordinates, then they are linked by a temporal edge.

Figure 4.4 – Example of $2\mathcal{D}+t$ grid

this thesis, we often consider $2\mathcal{D}+t$ grids such that there is always a temporal edge between two vertices u and v that have the same spatial coordinates (*i.e.*, $x_u = x_v$ and $y_u = y_v$), and neighbor temporal coordinates (*i.e.*, $|t_u - t_v| = 1$). In this case, we shall not display temporal edges (that are implicit) as illustrated in Figure 4.4(b).

Definitions of induced and partial subgraphs may be applied to $2\mathcal{D}+t$ grids, and Figure 4.5 displays the induced subgrid of the $2\mathcal{D}+t$ grid displayed in Figure 4.4 obtained by removing vertices 5, 7, 10, 11, 12 and 13, and the partial subgrid obtained by further deleting edges $(4,3)$ and $(1,2)$.

The extension of $2\mathcal{D}$ grid isomorphism to $2\mathcal{D}+t$ grids deserves a discussion. Indeed, in many applications the temporal dimension is directed: a vertex label that is turned from red to blue from time step t to time step $t+1$ (such as, for example, vertex 3 at time step 0 which becomes vertex 6 at time step 1 in Figure 4.4) may have a different meaning than when a vertex is turned from blue to red. Therefore, we have chosen to consider that a temporal edge connecting a vertex with label l at time t_1 with a vertex with label l' at time t_1+1 is not

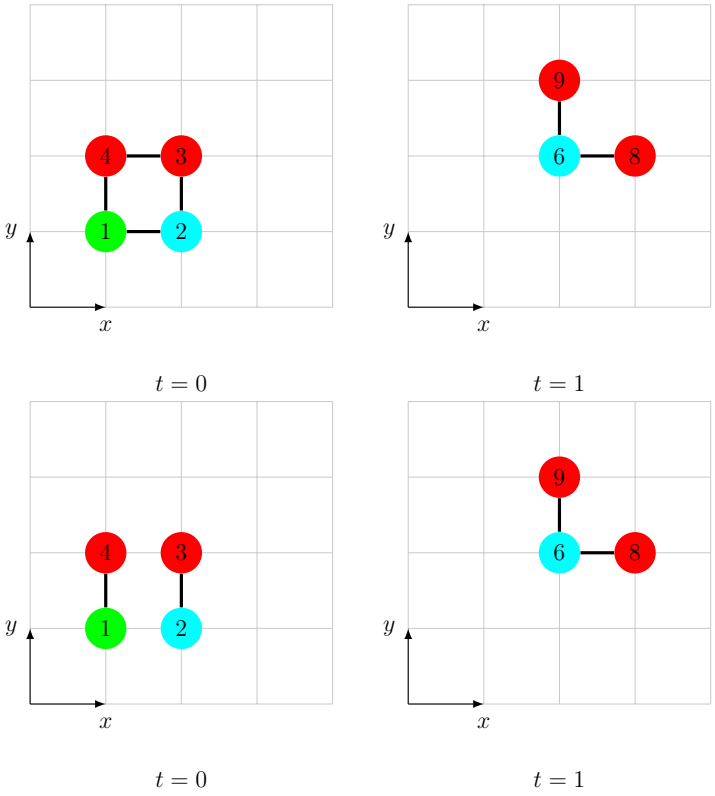


Figure 4.5 – Examples of induced (top) and partial (bottom) $2\mathcal{D}+t$ subgrids of the $2\mathcal{D}+t$ grid displayed in Figure 4.4(b)

isomorphic to a temporal edge connecting a vertex with label l' at time t_2 with a vertex with label l at time t_2+1. For example, we consider that edges $(4,7)$ and $(5,8)$ are not isomophic, whereas we consider that edges $(4,1)$ and $(3,5)$ are isomorphic. By means of rotations, this implies that we only allow rotations around the temporal axis. This leads us to the following definition of $2\mathcal{D}+t$ grid rotations.

Definition 4.7 Rotation of $2\mathcal{D}+t$ grids
Let $G=\langle V,E,L,x,y,t\rangle$ be a $2\mathcal{D}+t$ grid and $\Theta_{2D+t}=\{\left(\begin{smallmatrix}1&0&0\\0&1&0\\0&0&1\end{smallmatrix}\right),\left(\begin{smallmatrix}0&1&0\\-1&0&0\\0&0&1\end{smallmatrix}\right),\left(\begin{smallmatrix}-1&0&0\\0&-1&0\\0&0&1\end{smallmatrix}\right),\left(\begin{smallmatrix}0&-1&0\\1&0&0\\0&0&1\end{smallmatrix}\right)\}$ be the set of rotation matrices around the temporal axis. The rotation of G by $\theta\in\Theta_{2D+t}$, denoted $\theta\cdot G$, is the $2\mathcal{D}+t$ grid obtained by applying rotation θ to every vertex of G, *i.e.*, $\theta\cdot G=\langle N,E,L,x',y',t\rangle$ where $\forall v\in V,(x'_v,y'_v,t)=(x_v,y_v,t_v)\cdot\theta$.

We extend the translation definition to $2\mathcal{D}+t$ grids in a straightforward way.

Definition 4.8 Translation of $2\mathcal{D}+t$ grids
Let $G=\langle V,E,L,x,y,t\rangle$ be a $2\mathcal{D}+t$ grid and $\mathcal{T}\in\mathbb{Z}^3$ be a translation vector. The translation of G by $\mathcal{T}$, denoted $G+\mathcal{T}$, is the grid obtained by moving every

vertex of G by $\mathcal{T}$, *i.e.* $G + \mathcal{T} = \langle V, E, L, x', y', t' \rangle$ where $\forall v \in V, (x'_v, y'_v, t'_v) = (x_v, y_v, t_v) + \mathcal{T}$.

Like in the $2\mathcal{D}$ case, $2\mathcal{D}+t$ grid isomorphism searches for an isomorphism function between two grids that preserves the topology up to a translation and a rotation.

Definition 4.9 $2\mathcal{D}+t$ **grid isomorphism**
Let $G = \langle V, E, L, x, y, t \rangle$ and $G' = \langle V', E', L', x', y', t' \rangle$ be two $2\mathcal{D}+t$ grids. G and G' are $2\mathcal{D}+t$ grid isomorphic if:

- There exists an isomorphism function f between the graphs $\langle V, E, L \rangle$ and $\langle V', E', L' \rangle$.
- There exist a translation $T \in \mathbb{Z}^3$ and a rotation $\theta \in \Theta_{2D+t}$ such that $G' = \theta_{2D+t} \cdot G + \mathcal{T}$

Finally, the definition of subgrid isomorphism follows from the definition of subgrids and isomorphism.

Definition 4.10 $2\mathcal{D}+t$ **subgrid isomorphism**
A $2\mathcal{D}+t$ grid G is subgrid isomorphic to a $2\mathcal{D}+t$ grid G', denoted $G \subseteq_{\text{grid}_{2D+t}} G'$, if there exists a subgrid G'' of G' such that G' is $2\mathcal{D}+t$ grid isomorphic to G''.

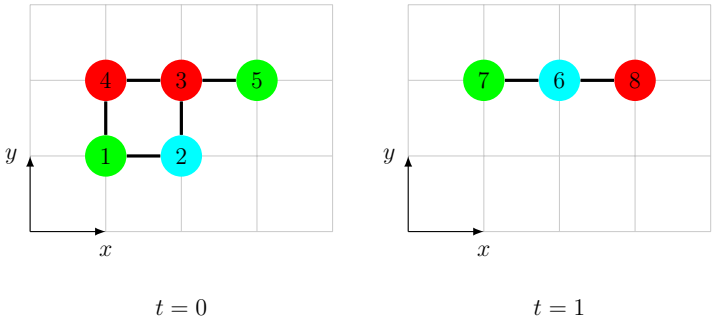
Again, depending on whether the subgrid relation is induced or partial, we obtain two different definitions of subgrid isomorphism, respectively denoted $G \subseteq^i_{\text{grid}_{2D+t}} G'$ and $G \subseteq^p_{\text{grid}_{2D+t}} G''$. Figure 4.6 gives an example of induced subgrid isomorphism.

In this thesis, if the type of grid ($2\mathcal{D}$ or $2\mathcal{D}+t$) is not explicitly specified, then both definitions may be used. This also applies to notations. For instance, the notation $\subseteq_{\text{grid}}$ means that both $\subseteq_{\text{grid}_{2D}}$ and $\subseteq_{\text{grid}_{2D+t}}$ may be used.

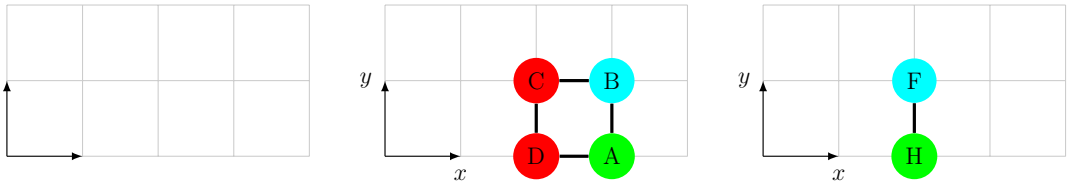
4.3 Discussion

In this chapter, we have introduced regular grids. We have considered two different kinds of regular grids: $2\mathcal{D}$ grids, such that each vertex has at most 4 neighbors, and $2\mathcal{D}+t$ grids, such that each vertex has at most 4 spatial neighbors, and at most 2 temporal neighbors. $2\mathcal{D}$ grids may be viewed as special cases of $2\mathcal{D}+t$ grids with no temporal edges. Hence, our grid mining algorithm (described in the next chapter) will be described for $2\mathcal{D}+t$ grids, but may be applied to $2\mathcal{D}$ grids as well.

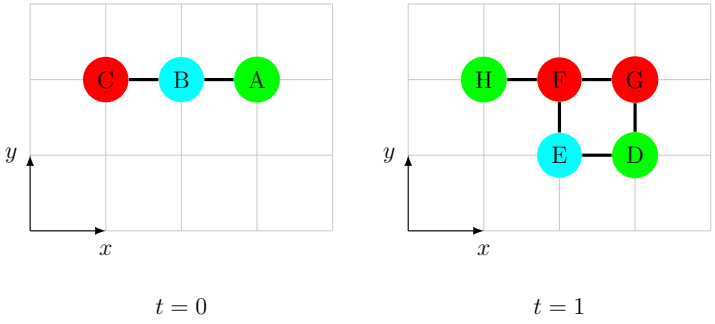
When mining grids, we search for frequent patterns, where a pattern is defined as a sub-isomorphic grid. Hence we have defined subgrid isomorphism. Subgrid isomorphism is close to geometric isomorphism, as it allows rotations



(a) A $2\mathcal{D}+t$ grid G_1.



(b) A $2\mathcal{D}+t$ grid G_2 such that $G_2 \subseteq^{i}_{grid} G_1$: G_2 is obtained from G_1 by removing vertices 5 and 8, by rotating vertices of $\pi/2$ around the temporal axis and translating them by $(4,-1,1)$.



(c) A $2\mathcal{D}+t$ grid which is not grid isomorphic to G_1 because rotations around spatial axis are not allowed (it is isomorphic to G_1 with a rotation of π around y axis and a translation).

Figure 4.6 – Example of $2\mathcal{D}+t$ subgrid isomorphism.

and translations. This allows our mining algorithm to be invariant to rotations and translations. However, the rigid transformations we consider are different from those defined for geometric graphs: we do not allow scaling transformations, as this is meaningless in a grid context, and we only consider rotations of multiples of $\pi/2$ around the temporal axis.

Table 4.1 summarizes properties which are preserved by subgrid isomorphism, and compares subgrid isomorphism with other sub-isomorphism relations described in section 2.1. Given a pattern graph G_p, a target graph G_t, and a sub-isomorphism function f from G_p and G_t, we consider the following properties:

Edges: for each edge (u,v) of G_p, $(f(u),f(v))$ is also an edge of G_t;

Table 4.1 – Comparison of sub-isomorphism relations.

Sub-isomorphism relation between a pattern graph G_p and a target graph G_t	Features of G_p preserved in G_t						
	Edges	Non edges	Internal faces	Angles	Neighbor order	Distance ratio	Edge length
$G_p \subseteq^p G_t$ (partial subgraph)	X						
$G_p \subseteq^i G_t$ (induced subgraph)	X	X					
$G_p \subseteq_{\text{face}} G_t$ (face-induced plane subgraph)	X	X	X		X		
$G_p \subseteq^p_{\text{geo}} G_t$ (partial geometric subgraph)	X			X	X	X	
$G_p \subseteq^i_{\text{geo}} G_t$ (induced geometric subgraph)	X	X		X	X	X	
$G_p \subseteq^p_{\text{grid}} G_t$ (partial subgrid)	X			X	X	X	X
$G_p \subseteq^i_{\text{grid}} G_t$ (induced subgrid)	X	X		X	X	X	X

Non edges: for each couple (u, v) of vertices of G_p such that (u, v) is not an edge, $(f(u), f(v))$ is not an edge of G_t;

Internal faces: for each internal face bounded by the vertex cycle $(v_1, \ldots, v_n)$ in G_p, there is an internal face in G_t which is bounded by the vertex cycle $(f(v_1), \ldots, f(v_n))$;

Angles: for each vertex triple (u, v, w) of G_p, the angle between $[u, v]$ and $[v, w]$ is equal to the angle between $[f(u), f(v)]$ and $[f(v), f(w)]$ in G_t;

Neighbor order: for each vertex u of G_p, the order we encounter the neighbors of u when turning around u in clockwise order is preserved when turning around $f(u)$ in clockwise order (*i.e.*, if the neighbors of u in clockwise order are $v_1, \ldots v_k$, then the neighbors of $f(u)$ in clockwise order are $f(v_1), \ldots, f(v_k)$);

Distance ratio: for each couple of edges (u_1, v_1) and (u_2, v_2) in G_p, we have $\frac{d(u_1,v_1)}{d(u_2,v_2)} = \frac{d(f(u_1),f(v_1))}{d(f(u_2),f(v_2))}$ (where $d(u, v)$ is the euclidian distance between u and v);

Edge length: for each edge (u, v) in G_p, we have $d(u, v) = d(f(u), f(v)) = 1$.

Subgrid isomorphism is not a special case of face-induced plane subgraph isomorphism (because it relaxes the constraint that faces must be preserved), neither it is a generalization of it (because it adds the constraint that angles must be preserved, for example). Subgrid isomorphism is a special case of geometric subgraph isomorphism, where edges are constrained to connect vertices with neighbor coordinates.

Chapter 5

Description of GriMA

Contents

As mentioned in the previous chapter, many objects exhibit a regular structure and may be modeled by grids, which are particular cases of geometrical graphs. Mining algorithms that are dedicated to geometric graphs (such as FreqGeo and MaxGeo, for example) may be used to mine grids. However, they do not exploit the fact that grids have regular structures. In this chapter, we introduce a novel **Gri**d **M**ining **A**lgorithm, called GriMA, in order to find frequent subgrid structures in a database of $2\mathcal{D} + t$ grids.

In Section 5.1, we define the grid mining problem, and relate it with the general graph mining problem (solved by gSpan), the plane graph mining problem (solved by Plagram), and the geometric graph mining problem (solved by FreqGeo).

Then, we describe GriMA as an instantiation of the generic graph mining algorithm (Algorithm 2) introduced in Section 3.3: canonical codes for identifying grids are defined in Section 5.2; the canonicity test procedure is defined in Section 5.3; the extension strategy of a pattern during the mining process is described in Section 5.4. The complete GriMA algorithm is summarized in Section 5.5, and its theoretical properties are studied.

In Section 5.6, we introduce node-induced GRIMA, which is an optimization of the mining process when all grids of the database are complete and all edges have the same label.

When using frequent patterns to characterize grids (to classify them, for example), we also need to count the number of occurrences of a given pattern in a grid. In Section 5.7, we describe an algorithm for achieving this task.

5.1 Definition of the grid mining problem

Let us recall that the frequency of a pattern is the number of graphs of the database in which it occurs at least once. More precisely, in our grid context, the frequency of a pattern grid G_P in a database D of $2\mathcal{D} + t$ grids is defined by

$$freq(G_P, D) = |\{G_T \in D : G_P \subseteq_{grid} G_T\}|$$

The grid mining problem aims at finding all frequent patterns in a database of grids.

Definition 5.1 Grid Mining Problem
Let D be a database of $2\mathcal{D} + t$ grids, and σ be a positive integer. The grid mining problem is to find every pattern grid G_P such that $freq(G_P, D) \geq \sigma$

The difference with other graph mining problems lays in the kind of graphs that belong to the database D and the kind of subgraph isomorphism considered to match patterns. In the general graph mining problem solved by GSPAN, D is a database of general graphs and the sub-isomorphism relation is $\subseteq^p$ (partial subgraph isomorphism). As a consequence, GSPAN does not consider any geometrical information when comparing patterns and it considers as isomorphic two subgraphs that are different from a grid point of view as shown in Figure 5.1(b) and 5.1(c)

In the plane graph mining problem solved by PLAGRAM, D is a database of plane graphs and the sub-isomorphism relation is $\subseteq_{face}$ (face-induced plane subgraph isomorphism). In this case, the smallest possible subgraph is a single face. Using PLAGRAM to mine grids is possible but the problem needs to be transformed such that each grid vertex becomes a face. This transformation is illustrated in Figure 5.1. However, this artificially increases the number of vertices and edges which may cause scalability problems for PLAGRAM.

In the geometrical graph mining problem solved by FREQGEO and MAXGEO, D is a database of geometrical graphs and the sub-isomorphism relation is $\subseteq^p_{geo}$ (partial geometric subgraph isomorphism). If D only contains $2\mathcal{D}$ grids, then the grid mining problem may be solved by FREQGEO and MAXGEO.

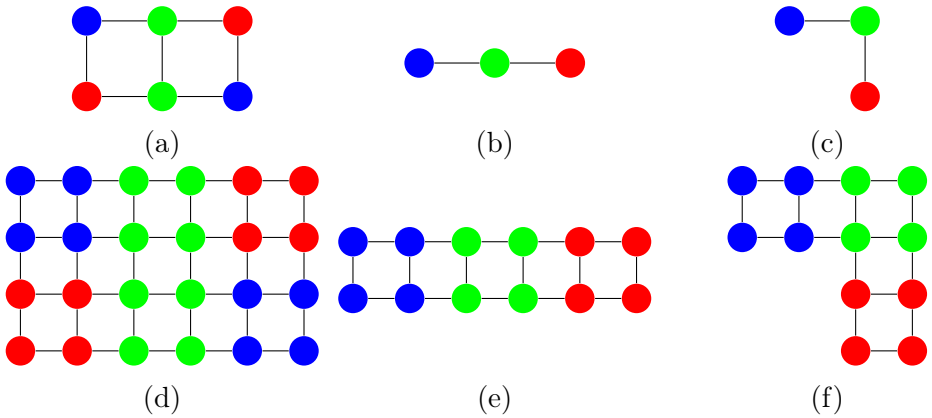


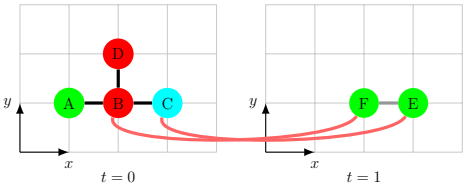
Figure 5.1 – Examples of sub-isomorphism relations. If we consider that (a), (b), and (c) are graphs, then (b) and (c) are isomorphic, and they are both subgraphs of (a). If we consider that (a), (b), and (c) are grids, then angles must be preserved by isomorphism functions, so that (b) and (c) are not grid isomorphic, and (b) is a subgrid of (a) whereas (c) is not. The plane graphs (d), (e), and (f) are obtained from (a), (b), and (c), respectively, by replacing each vertex with a 4-vertex face (with the same label on the 4 vertices). (e) and (f) are not isomorphic, and (e) is a face-induced plane subgraph of (d) whereas (f) is not.

However, these algorithms are not optimized for grids and have higher time-complexities. When D contains $2\mathcal{D}+t$ grids, the grid mining problem cannot be solved by FreqGeo and MaxGeo because rotations around the spatial axis are not allowed when mining $2\mathcal{D}+t$ grids (to take into account the fact that the temporal axis is oriented).

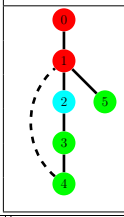
These differences motivate us to design a new algorithm, GriMA, which is dedicated to grids. A first interest is that GriMA takes advantage of the regular grid structure to lower the mining complexity. A second interest is that the mined patterns are more relevant for applications where objects have a regular grid structure as they fully preserve angle information, while being tolerant to translations and rotations along the temporal axis.

5.2 Canonical Code

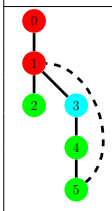
A key point when mining graphs is to efficiently detect isomorphic patterns. Similarly to gSpan and Plagram, this is done in GriMA by using canonical codes to represent patterns. A code of a grid G is a sequence of n edge codes $C(G) = \langle ec_0, ..., ec_{n-1} \rangle$ which is associated with a depth-first traversal of G starting from a given initial vertex. During this traversal, each edge is visited once, and vertices are numbered (as described in Section 2.5): the initial vertex has number 0, and each time a new vertex is discovered, it is numbered with the smallest integer not already used in the traversal. Each edge code corresponds



Code 1

DFS tree	edge	δ	i	j	a	L_i	L_j	L_{ij}
	(D,B)	0	0	1	0	●	●	0
	(B,C)	0	1	2	$3\pi/2$	●	●	0
	(C,E)	0	2	3	-1	●	●	0
	(E,F)	0	3	4	0	●	●	0
	(F,B)	1	4	1	-2	●	●	0
	(B,A)	0	1	5	$\pi/2$	●	●	0

Code 2

DFS tree	edge	δ	i	j	a	L_i	L_j	L_{ij}
	(D,B)	0	0	1	0	●	●	0
	(B,A)	0	1	2	$\pi/2$	●	●	0
	(B,C)	0	1	3	$3\pi/2$	●	●	0
	(C,E)	0	3	4	-1	●	●	0
	(E,F)	0	4	5	0	●	●	0
	(F,B)	1	5	1	-2	●	●	0

Code 3

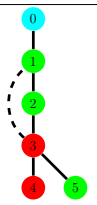
DFS tree	edge	δ	i	j	a	L_i	L_j	L_{ij}
	(C,E)	0	0	1	-1	●	●	0
	(E,F)	0	1	2	0	●	●	0
	(F,B)	0	2	3	-2	●	●	0
	(B,D)	0	3	4	$\pi/2$	●	●	0
	(B,A)	0	3	5	π	●	●	0
	(B,C)	1	3	0	0	●	●	0

Figure 5.2 – Top left: A $2\mathcal{D}+t$ grid (temporal edges are displayed in red, all edges have the same label 0, and vertex labels are represented by colors with ● $= 0$, ● $= 1$, and ● $= 2$). Top right and bottom: three examples of codes for this grid with the corresponding DFS trees (forward edges are represented with plain lines, and backward edges with dashed lines.

to a different edge of G and the order of edge codes in $C(G)$ corresponds to the order edges are visited. Hence, ec_k is the code associated with the k^{th} visited edge. This edge code ec_k is the tuple $(\delta, i, j, a, l_i, l_j, l_{(i,j)})$ where:

- i and j are the numbers associated with the vertices of the visited edge.
- $\delta \in \{0, 1\}$ is the direction of the visited edge:
  - $\delta = 0$ if it is forward, *i.e.*, j is a new vertex reached for the first time;
  - $\delta = 1$ if it is backward, *i.e.*, j already appears in $ec_{k'}$ with $k' < k$.
- $a \in \{-2, -1, 0, \pi/2, \pi, 3\pi/2\}$ is the angle value of the visited edge (i, j):
  - if (i, j) is a temporal edge, then $a = -2$ if $t_i = t_j + 1$, and $a = -1$ if $t_i = t_j - 1$;
  - else, (i, j) is a spatial edge:
    - If (i, j) is the first spatial edge encountered since the beginning of the traversal, then $a = 0$.
    - Else, let (l, m) be the first spatial edge in $\langle ec_0, ..., ec_{k-1} \rangle$ such that $(x_i, y_i) = (x_m, y_m)$. a is the angle between $[(x_l, y_l), (x_m, y_m)]$ and $[(x_i, y_i), (x_j, y_j)]$.
- l_i, l_j, $l_{(i,j)}$ are labels of i, j, and (i, j), respectively.

For instance, Figure 5.2 displays three possible codes (among all possible codes) for a grid. Code 1 corresponds to a traversal starting from edge (D, B), and successively visiting edges (B, C), (C, E), (E, F), (F, B), and (B, A). For each visited edge, code 1 contains a tuple $(\delta, i, j, a, L_i, L_j, L_{(i,j)})$. i and j correspond to the order of discovery of the edge endnodes during the traversal. In this traversal, this order is $D = 0, B = 1, C = 2, E = 3, F = 4$, and $A = 5$. For all edges but (F, B), $\delta = 0$ because all edges but (F, B) are forward edges, *i.e.*, they reach new vertices that have not yet been discovered before. (F, B) is a backward edge because B has been discovered before F. Finally, a is the angle value. For the first visited edge, (D, B), which is a spatial edge, we have $a = 0$. For the other spatial edges $e \in \{(B, C), (E, F), (B, A)\}$, the value of a corresponds to the angle between e and the first spatial edge (X, Y) such that Y has the same spatial coordinates as the first endnode of e. For (B, C) (resp. (E, F) and (B, A)), this edge is (D, B) (resp. (B, C) and (D, B)), and therefore $a = 3\pi/2$ (resp. $a = 0$ and $\pi/2$). For the temporal edges, (C, E) and (F, B), a is equal to -1 and -2 because $t_C = t_E - 1$ whereas $t_F = t_B + 1$.

There exist different possible codes for a given grid, as illustrated in Figure 5.2: each code corresponds to a different traversal that can be obtained by starting from a different initial vertex and choosing edges in a different order. We define a total order on the set of all possible codes by considering a lexicographic order (as all code components have numerical values). Among all the possible codes for a grid, the largest one according to this order is the canonical code of this grid and it is unique.

Let us consider the codes displayed in Figure 5.2. As code 1 and code 2 share the same first edge code, we compare the second edge code, corresponding to (B, C) and (B, A), respectively. They have a different angle value ($\widehat{DBC} = 3\pi/2$ and $\widehat{DBA} = \pi/2$) and code 1 is greater than code 2. The same consideration can be done with the first edge of code 1 and code 3. Moreover, code 1 is also greater than all other possible codes for this grid (not shown in the figure), thus code 1 is canonical.

Proposition 5.1
Every prefix of a canonical code is canonical.

Proof. Given a code C with n vertices, the right most path of C (denoted $rmp(C)$) is the path from vertex number 0 to vertex number $n - 1$ using only forward edges. For example, in Figure 5.2, $rmp(\text{Code1}) = \langle D, B, A\rangle$ as the last vertex is A, and edges (D, B) and (B, A) are forward edges. As the right-most path is built from a DFS traversal and only contains forward edges, it has the property that numbers associated with vertices are necessarily increasing (see Section 2.5).

Let G be a grid with $n > 1$ edges and $C = \langle ec_0, \dots ec_{n-1} \rangle$ be its canonical code. Let $D = \langle ec_0, \dots ec_{n-2} \rangle$ be the prefix of C that contains all edges of C but the last one, and let e be this last edge (whose edge code in C is ec_{n-1}). D is a code corresponding to the grid G without edge e. Suppose that D is not canonical, *i.e.*, it is not the greatest possible code. This implies that there exists a code $D' = \langle ec'_0, \dots ec'_{n-2} \rangle$ such that $D' > D$ and D' represents the same grid as D. We have to consider two cases: either e is incident to one vertex of $rmp(D')$, or not.

Case 1: Edge e is incident to one vertex of $rmp(D')$. In this case, we can build the code $C' = D'.\langle ec'_{n-1} \rangle$ corresponding to the same depth first traversal as the one used to build D', but where we traverse edge e just after traversing the edge associated with ec'_{n-2} and where ec'_{n-1} corresponds to the edge code of e in this traversal. In this case, we have $C' > C$, because $D' > D$, and C' is a code for G. This is in contradiction with the assumption that C is canonical.

Case 2: Edge e is not incident to $rmp(D')$. In this case, there necessarily exists a prefix $P = \langle ec'_0, \dots ec'_k \rangle$ of D', with $k < n - 2$, such that e is incident to a vertex i in $rmp(P)$ but not to any vertex in $rmp(P.\langle ec'_{k+1} \rangle)$. Let us consider a depth-first traversal identical to the one that has been done to build P, but where edge e is traversed just after the edge associated with ec'_k and then, all remaining edges are traversed. Let M be the code associated with this traversal. M is a code for G, and we have $M = P.\langle ec_e \rangle.Q$, where ec_e is the code associated with edge e when it is traversed just after the edge associated with ec'_k, and Q is the code associated with the end of the traversal. Let j be the number of the vertex of $rmp(P)$ which is incident to the edge associated with ec'_{k+1}. By definition of P, e is not incident to a vertex in $rmp(P.\langle ec'_{k+1} \rangle)$ and thus: i) ec'_{k+1} is forward and ii) i is "after" j in $rmp(P)$. As the numbers associated with the vertices in $rmp(P)$ are increasing, $i > j$. Thus, if edge e is forward, ec_e is either $(0, i, ...)$, or if edge e is backward, ec_e is $(1, i, ...)$ which are both larger than ec'_{k+1} which is $(0, j, ...)$, since ec'_{k+1} is forward. Therefore, $M = P.\langle ec_e \rangle.Q$ is greater than C, and this is in contradiction with the assumption that C is the canonical code of G. □

5.3 Canonicity Test

The *canonicity* test, called line 11 in Algorithm 2, decides whether a code P is canonical. This is done in two steps. First we reconstruct the grid pattern G from the code P. This step is done in linear time with respect to the number of edges in P since edges are listed in P together with angles and labels. Then, for each edge (i, j) in G, we build two codes $P_{(i,j)}$ and $P_{(j,i)}$ corresponding to

two different depth-first traversals: $P_{(i,j)}$ (resp. $P_{(j,i)}$) is obtained by starting the traversal on i (resp. j) and first choosing edge (i,j) (resp. (j,i)). At each step of the traversal, if we have to choose between different edges, we always choose first according to the direction of the edge (backward edges first as the first value of edge code is δ) then according to the angle to maximize the code (as choosing another edge will necessarily lead to a smaller code so that it will not be canonical).

In Figure 5.2, for example, when starting the traversal on D and first choosing edge (D,B), for the second traversed edge we have to choose between (B,A) and (B,C). As none of them is backward and the angle $\widehat{DBA}$ is $\pi/2$ while the angle $\widehat{DBC}$ is $3\pi/2$, we choose edge (B,C) which leads to a larger code (Code 1) and we do not generate the code obtained when choosing (B,A) (Code 2) because we know by construction that it will be smaller.

Whenever we find a code $P_{(i,j)}$ which is strictly greater than P, we conclude that P is not canonical, whereas if all codes $P_{(i,j)}$ are smaller than or equal to P, we conclude that P is canonical.

Complexity of the canonicity test. If P contains p edges, we have to perform at most $2p$ traversals, and each traversal is done in $\mathcal{O}(p)$. So, the complexity of the canonicity test is $\mathcal{O}(p^2)$. However, each traversal may be stopped as soon as an edge code in $P_{(i,j)}$ is different from the edge code at the same position in P. So this worst-case complexity is nearly never reached in practice: it is reached only when the pattern is fully symmetric and all codes are equals, whatever the first edge is. Furthermore, the first edge of the code must be one with maximal labels on vertices and edges. Therefore, we do $2p$ traversals only if all labels are equals. For instance, on Figure 5.2, there are only two vertices (D and B) with the greatest label (*i.e.*, 2, represented by the red colour). Therefore, the canonicity test will only build the two codes $P_{(D,B)}$ and $P_{(B,D)}$ corresponding to traversals starting from edges (D,B) and (B,D), respectively.

5.4 Extension strategy

Like gSpan, Plagram and FreqGeo, GriMA uses the pattern-growth approach to explore a search space of canonical codes in a depth-first manner: each time a frequent pattern is found, it is extended into a bigger candidate pattern. The elementary component used to start the mining process (line 2 of Algorithm 2) is an edge, which is also the smallest pattern considered by

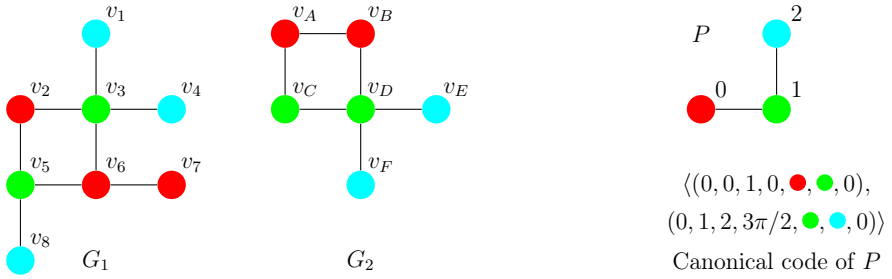


Figure 5.3 – Left: A database of grids $D = \{G_1, G_2\}$ (all edges have the same label 0, and vertex labels are represented by colors with $\bullet = 0$, $\bullet = 1$, and $\bullet = 2$). Right: A pattern P together with its canonical code

GRIMA. Then at each recursive call to EXTEND (lines 12 of Algorithm 2), the pattern given as input to EXTEND is grown with one edge.

EXTEND(P) only generates promising candidate patterns, *i.e.*, patterns that actually occur in the grid database D. More precisely, for each grid $G_i \in D$, it gathers in a list Ext_i all possible and valid extensions of P in $G_i \in D$ (line 8 of algorithm 2). To this aim, we first need to know the list of all occurrences of P in G_i. To save space, this list, called Occ_i, only contains the first edge of each occurrence of P as all other edges may be obtained from the pattern code P.

For example, let us consider the database of grids D and the pattern P displayed in Figure 5.3. P occurs twice in G_1 and once in G_2, and the three corresponding sub-isomorphism functions are:

$$\begin{aligned} 0 \to v_2, \quad 1 \to v_3, \quad 2 \to v_1 \\ 0 \to v_6, \quad 1 \to v_5, \quad 2 \to v_8 \\ 0 \to v_B, \quad 1 \to v_D, \quad 2 \to v_E \end{aligned}$$

Hence, the two occurrence lists of P in G_1 and G_2, respectively, are:

$$\begin{aligned} Occ_1 &= ((v_2, v_3), (v_6, v_5)) \\ Occ_2 &= ((v_B, v_D)) \end{aligned}$$

Given the list Occ_i of all occurrences of a pattern P in a grid G_i, we build the list Ext_i of all possible extensions of P in G_i as follows: for each edge (u, v) in Occ_i, we build the sets $V_{(u,v)}$ and $E_{(u,v)}$ of vertices and edges that belong to the occurrence of P starting from edge (u, v) in G_i, and we search for every edge (u', v') of G_i such that $\{u', v'\} \cap V_{(u,v)} \neq \emptyset$ and $(u', v') \notin E_{(u,v)}$. For each of these edges, we compute the corresponding edge code (*i.e.*, the tuple $(\delta, u', v', a, L_{u'}, L_{v'}, L_{(u',v')})$ as described in Section 5.2), and add it to the list Ext_i of extensions of P in G_i (while removing duplicates if any).

On our running example, for the occurrence (v_2, v_3) of pattern P in G_1, we have:

$$\begin{aligned} V_{(v_2,v_3)} &= \{v_2, v_3, v_1\} \\ E_{(v_2,v_3)} &= \{(v_2, v_3), (v_3, v_1)\}. \end{aligned}$$

The set of edges of G_1 that are possible extensions of this occurrence in G_1 is $\{(v_3, v_4), (v_3, v_6), (v_2, v_5)\}$ and the corresponding edge codes respectively are $(0, 1, 3, \pi,$  $, 0)$, $(0, 1, 3, \pi/2,$  $, 0)$ and $(0, 0, 3, 3\pi/2,$ $, 0)$.

For the occurrence (v_6, v_5) of pattern P in G_1, we have:

$$\begin{aligned} V_{(v_6,v_5)} &= \{v_6, v_5, v_8\} \\ E_{(v_6,v_5)} &= \{(v_6, v_5), (v_5, v_8)\}. \end{aligned}$$

The set of edges of G_1 that are possible extensions of this occurrence in G_1 is $\{(v_5, v_2), (v_6, v_3), (v_6, v_7)\}$ and the corresponding edge codes respectively are $(0, 1, 3, \pi/2,$  $, 0)$, $(0, 0, 3, 3\pi/2,$  $, 0)$ and $(0, 0, 3, \pi,$  $, 0)$.

Hence, the list of all possible extensions of P in G_1 is:

$$\begin{aligned} Ext_1 = \langle &(0, 1, 3, \pi,\ \bullet,\ \bullet, 0), (0, 1, 3, \pi/2,\ \bullet,\ \bullet, 0), \\ &(0, 0, 3, 3\pi/2,\ \bullet,\ \bullet, 0), (0, 0, 3, \pi,\ \bullet,\ \bullet, 0)\rangle \end{aligned}$$

Similarly, we build the list of all possible extensions of P in G_2:

$$Ext_2 = \langle (0, 1, 3, \pi,\ \bullet,\ \bullet, 0), (0, 1, 3, \pi/2,\ \bullet,\ \bullet, 0), (0, 0, 3, 3\pi/2,\ \bullet,\ \bullet, 0)\rangle$$

Finally, all the Ext_i lists are merged into a single list (line 9 of algorithm 2), and for each different extension we memorise its frequency (*i.e.*, the number of lists Ext_i in which the extension appears). This final list is used to iterate on all possible extensions (line 10 of Algorithm 2) and, for each of these extensions that leads to a frequent and canonical pattern, `extend` is recursively called (line 12 of Algorithm 2).

On our running example, for example, we merge Ext_1 and Ext_2 to obtain the following possible extensions of P in D:

- $(0, 1, 3, \pi,$  $, 0)$, whose frequency is 2 as it both occurs in Ext_1 and Ext_2,
- $(0, 1, 3, \pi/2,$  $, 0)$, whose frequency is 1 as it only occurs in Ext_1,
- $(0, 1, 3, \pi/2,$  $, 0)$, whose frequency is 1 as it only occurs in Ext_2,
- $(0, 0, 3, 3\pi/2,$ $, 0)$, whose frequency is 1 as it only occurs in Ext_1,
- $(0, 0, 3, 3\pi/2,$  $, 0)$, whose frequency is 1 as it only occurs in Ext_1,
- $(0, 0, 3, \pi,$  $, 0)$, whose frequency is 1 as it only occurs in Ext_2,

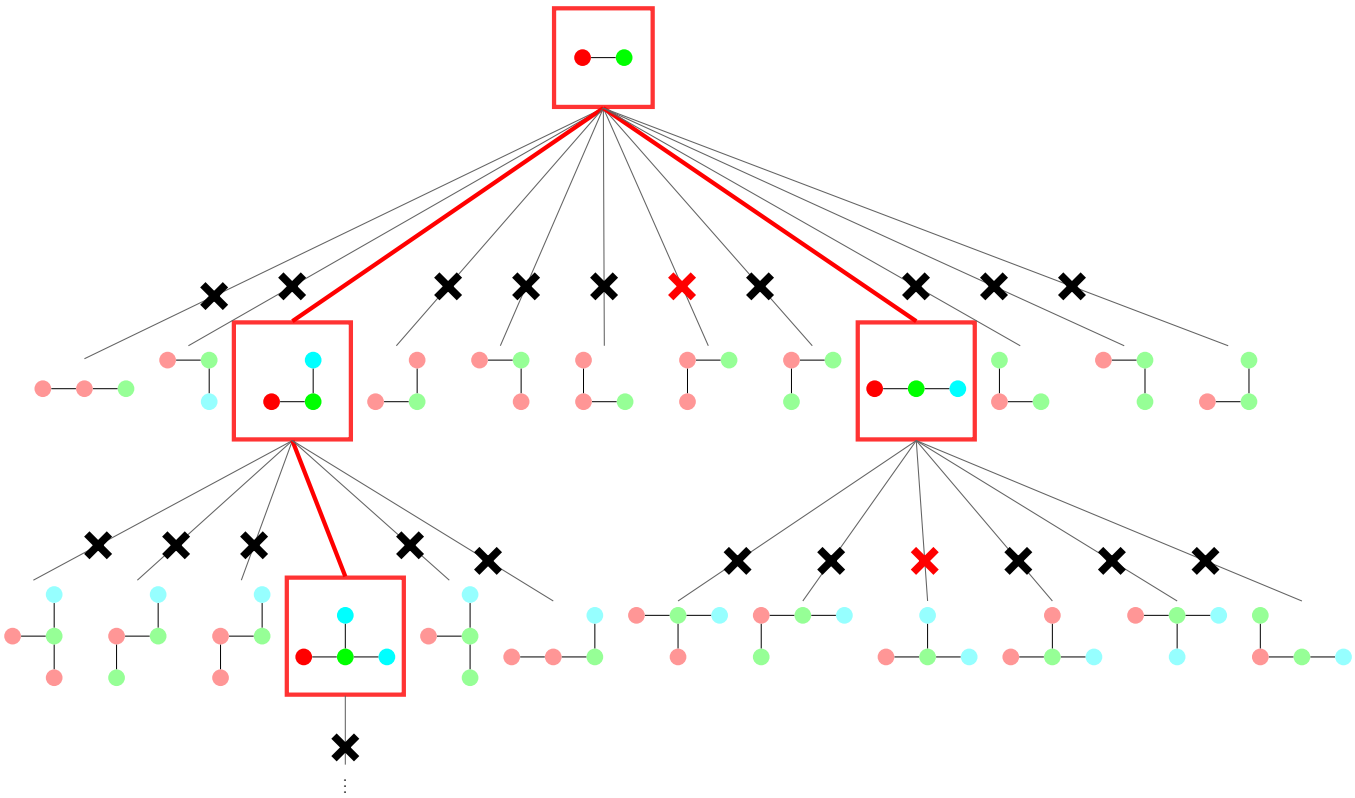


Figure 5.4 – Search space explored by GRIMA when starting from edge for the database $\mathcal{D} = \{G_1, G_2\}$ displayed in Figure 5.3, with a frequency threshold $\sigma = 2$. Black (resp. red) crosses indicate branches that are pruned because the pattern is not frequent (resp. not canonical). Red branches are not pruned and patterns in red rectangles are frequent and canonical.

Illustration of the extension strategy. Figure 5.4 illustrates the extension strategy on our running example, composed of the two grids displayed in Figure 5.3, when starting from the initial edge . Many branches are pruned because the current pattern is no longer frequent (branches with black crosses). Two branches are pruned because the current pattern is frequent but it is not canonical (branches with red crosses):

- Pattern is not canonical, and it is output when starting from the initial edge ;
- Pattern is not canonical and has already been output in a previous branch of the tree.

Finally, four patterns are frequent and canonical (those displayed in red rectangles).

Incremental management of occurrence lists. A key point for an efficient implementation is to manage occurrence lists incrementally. More precisely, before each recursive call to `Extend(`$P.e$`)`, the list Occ_i contains all occurrences of P in G_i. It is updated by removing occurrences which do not contain edge e (we test whether an occurrence contains edge e in constant time as we know the coordinates of e relatively to P).

We use sparse sets [Briggs et al. 1993, Saint-Marcq et al. 2013] to restore occurrence lists in constant time when backtracking, *i.e.*, to restore the occurrence list of P in G_i from the occurrence list of $P.e$ in G_i when returning back from the recursive call to `extend(`$P.e$`)`. When using sparse sets, the current occurrence list of P in G_i is represented by a couple (T_i, S_i) where T_i is an array that contains occurrences, and S_i is the number of occurrences. T_i is initialized before the first call to `extend` (line 2 of Algorithm 2) when the pattern P is only composed of a single edge e: it is initialized to the set of all occurrences of edge e in G_i, and S_i is initialized to the number of occurrences of e in G_i. Each time we recursively call the `extend` procedure (line 12 of Algorithm 2), some occurrences must be removed from the occurrence list. However, instead of removing them from T_i, we push them at the end of the array. More precisely, for each occurrence to remove, we swap it with the last occurrence, which is at index $S_i - 1$ in T_i and decrease S_i. Hence, when we return back from the recursive call `extend(`$P.e$`)`, we restore the list of occurrences of P from the list of occurrences of $P.e$ in constant time, by increasing S_i by the number of occurrences that have been removed before the recursive call.

On our running example, let us assume that the frequency threshold σ is equal to 2. The single edge pattern $P = \langle(0, 0, 1, \color{red}\bullet, \color{green}\bullet, 0)\rangle$ is frequent, and before trying to extend it, we build the following sparse sets:

$$T_1 = \begin{array}{|c|c|c|c|}\hline (v_2, v_3) & (v_2, v_5) & (v_6, v_3) & (v_6, v_5) \\ \hline \end{array} \qquad T_2 = \begin{array}{|c|c|}\hline (v_A, v_C) & (v_B, v_D) \\ \hline \end{array}$$

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \qquad \begin{array}{cc} 0 & 1 \end{array}$$

$$S_1 = 4 \qquad S_2 = 2$$

During the execution of `extend(`P`)`, we find that $e = (0, 1, 2, 3\pi/2, \color{green}\bullet, \color{cyan}\bullet)$ is a valid extension and that the pattern $P.e$ is canonical and frequent. Hence, before recursively calling `extend(`$P.e$`)`, we update occurrence lists. There are two occurrences of $P.e$ in G_1, and their first edges are (v_2, v_3) and (v_6, v_5). To remove (v_2, v_5) from the occurrence list associated with G_1, we swap it with the last element (which is at index $S_1 - 1 = 3$) of T_1 and decrement S_1 to 3. We

obtain:

$T_1 =$	(v_2, v_3)	(v_6, v_5)	(v_6, v_3)	(v_2, v_5)	$T_2 =$	(v_A, v_C)	(v_B, v_D)
	0	1	2	3		0	1
$S_1 =$	3				$S_2 =$	2	

To remove (v_6, v_3) from the occurrence list, we swap it with the last element (which is at index $S_1 - 1 = 2$). However, as (V_6, v_3) already is the last element of T_1, this does not change anything. We decrement S_1 to 2, and we obtain:

$T_1 =$	(v_2, v_3)	(v_6, v_5)	(v_6, v_3)	(v_2, v_5)	$T_2 =$	(v_A, v_C)	(v_B, v_D)
	0	1	2	3		0	1
$S_1 =$	2				$S_2 =$	2	

There is only one occurrence of $P.e$ in G_2, and its first edge is (v_B, v_D). To remove (v_A, v_C), we swap it with the last element (which is at index $S_2 - 1 = 1$) of T_2 and decrement S_2 to 1. We obtain:

$T_1 =$	(v_2, v_3)	(v_6, v_5)	(v_6, v_3)	(v_2, v_5)	$T_2 =$	(v_B, v_D)	(v_A, v_C)
	0	1	2	3		0	1
$S_1 =$	2				$S_2 =$	1	

When returning back from the recursive call to `extend(`$P.e$`)`, the initial occurrence lists are restored in constant time by setting S_1 and S_2 to 4 and 2, respectively.

Note that sparse sets also allow us to reduce memory complexity: we do not need to save the full list of occurrences at each recursive call, but we only save the sizes of the sparse sets.

5.5 Theoretical analysis of GRIMA

Algorithm 3 gives a more detailed description of GRIMA: it basically follows the generic algorithm displayed in Algorithm 2, but includes more details related to the management of sparse sets, and to the computation of extensions, as described in the previous sections.

The time complexity of this algorithm depends on the number of frequent patterns. In some cases, this number may be null, allowing the algorithm to process the mining more quickly. This is the case, for example, when $\sigma > 1$ and, for every pair of grids $\{G_1, G_2\} \subseteq D$, the labels in G_1 are all different from the labels in G_2. However, in some other cases, the number of frequent patterns may be exponential in the size of the grids in D. This is the case, for

Algorithm 3 GriMA.

Input: A database of grids $D = \{G_1, ..., G_n\}$ and a frequency threshold σ.
Output: Print all patterns P such that $|\{G_i \in D : P \subseteq_{grid} G_i\}| \geq \sigma$.

```
1: for all canonical edge code e do
2:     for all grid G_i ∈ D do
3:         Build the sparse set (T_i, S_i) that contains all occurrences of e in G_i
4:     if at least σ sparse sets contain e then extend(e, {(T_i, S_i) : G_i ∈ D})
5:
6: procedure EXTEND(P, {(T_i, S_i) : G_i ∈ D}) ▷ P is a frequent canonical code and
   ∀G_i ∈ D, (T_i, S_i) is a sparse set that contains all occurrences of P in G_i.
7:     Output P
8:     for all grid G_i in 𝒟 do
9:         Initialise Ext_i to en empty set
10:        for all edge (u, v) in the sparse set (T_i, S_i) do
11:            Let G_(u,v) be the occurrence of P starting from (u, v) in G_i
12:            for all vertex u' of G_(u,v) do
13:                for all edge (u', v') of G_i st (u', v') is not an edge of G_(u,v) do
14:                    Add to Ext_i the edge code corresponding to (u', v')
15:    FreqExt ← {e ∈ ⋃_{G_i∈D} Ext_i : |{G_i ∈ D : e ∈ Ext_i}| ≥ σ}
16:    for all edge code e ∈ FreqExt do
17:        if the pattern P.e is canonical then
18:            for all grid G_i in D do
19:                Save S_i
20:                Remove from (T_i, S_i) every edge that is not an occ. of P.e in G_i
21:            extend(P.e, {(T_i, S_i) : G_i ∈ D})
22:            for all grid G_i in D do
23:                Restore S_i to the value saved line 19
```

example, when $\sigma = 1$ and all grids in D are such that all their vertices have different labels. Hence, to study the time complexity of GriMA, we evaluate its time complexity per pattern. The next proposition shows us that it has a polynomial delay time complexity, *i.e.*, the time complexity between the output of two frequent patterns is polynomial with respect to the size of the database.

Proposition 5.2
The time complexity of GriMA (Algorithm 3) is $\mathcal{O}(knp^3)$ per frequent and canonical pattern P, where k is the number of grids in the base D, n the number of edges of the largest grid in D and p the number of edges in P.

Proof. Lines (1-3) basically involve traversing all grids of D to collect all different kinds of edges and build their initial occurrence lists (using sparse sets). This is done once before searching for patterns and the time complexity of lines (1-3) is $\mathcal{O}(kn)$.

Each call to `extend`(P) outputs exactly one frequent and canonical pattern (*i.e.*, P). Let us study the time complexity of one of these calls.

Lines 8-14 build the list Ext_i of all possible extensions of P in G_i, for each grid G_i in D. There are k grids in D and at most $2n$ occurrences of P in each grid G_i (because for each edge of G_i there are at most two occurrences of P that start from this edge, depending on the edge direction considered). Therefore, we iterate $\mathcal{O}(kn)$ times on lines 11-14. For each of these iterations, we first build the subgrid $G_{(u,v)}$ that starts from edge (u, v) in G_i (line 11). This is done in $\mathcal{O}(p)$, by performing a traversal that starts from (u, v) and is guided by P. Then, lines (12-14) collect all possible extensions of this subgrid in G_i, and this is done in $\mathcal{O}(p)$ as there are at most 5 possible extensions for each vertex of $G_{(u,v)}$.

Hence, the time complexity of lines 8-14 is $\mathcal{O}(pkn)$.

Line 15 merges the k Ext_i lists into a single list $FreqExt$, while removing edge codes that are not frequent. This is done in linear time with respect to the size of all Ext_i lists (provided that we use a map that allows us to decide in constant time whether an edge code already belongs to $FreqExt$). Each list Ext_i contains at most $10pn$ extensions as there are at most $2n$ occurrences of P in G_i, and each of these occurrences has at most $5p$ possible extensions.

Hence, the time complexity of line 15 is $\mathcal{O}(kpn)$.

Lines 16-17 test whether $P.e$ is canonical, for each extension e in $FreqExt$. As each pattern $P.e$ has $p+1$ edges, each canonicity test is done in $\mathcal{O}(p^2)$ (see Section 5.3). $FreqExt$ contains $\mathcal{O}(pkn)$ different extensions, as there are k lists Ext_i, and each of these lists contains at most $10pn$ extensions.

Hence, the time complexity of lines 16-17 is $\mathcal{O}(p^3kn)$.

Lines 18-20 and 22-23 are done exactly once for each recursive call to `extend`. Lines 18-20 update sparse sets by removing edges that do not correspond to occurrences of $P.e$. There are k sparse sets, and each sparse set contains at most n edges. We test in constant time whether an edge corresponds to an occurrence of $P.e$, because we can compute the coordinates of e in G_i. Lines 22-23 simply restore sizes in $\mathcal{O}(k)$.

Hence the time complexity of lines 18-20 and 22-23 is $\mathcal{O}(kn)$ for each call to `extend`.

Therefore, the complexity of GriMA is $\mathcal{O}(knp^3)$ per pattern P. □

Proposition 5.3
GriMA is correct.

Proof. GRIMA is correct if it only outputs frequent subgrids. This is ensured by the fact that patterns are output at the first line of `extend` (line 7), and `extend` is called only if the pattern is frequent:

- when `extend` is called with a single edge pattern (line 4), we first check that there are at most σ sparse sets such that $S_i > 0$, implying that there are at least σ non empty occurrence lists;
- when `extend` is called to extend a pattern $P.e$ (line 22), we ensure that $P.e$ is frequent because $FreqExt$ only contains extensions that occur in at least σ grids (line 15).

□

Proposition 5.4
GRIMA is complete.

Proof. To show that GRIMA is complete, we must prove that it cannot miss any frequent subgrid. Since GRIMA explores a search-space of grid codes, and since each grid has exactly one canonical grid code, we must prove that GRIMA does not miss any frequent canonical code. Given a frequent canonical code P, we already know from proposition 5.1 that every prefix of P is canonical. If G' is a subgrid of G, then it is obvious from the definition of frequency that $\text{freq}(G', D) \geq \text{freq}(G, D)$. When there are two canonical codes P and P' such that $P = P'.e$, this implies that the grid G_P corresponding to P is a subgrid of the grid $G_{P'}$ corresponding to P': $G_{P'}$ is obtained from G_P by removing one edge (and also one vertex if e is a forward edge). As a consequence, if $P = P'.e$, then $\text{freq}(G_{P'}, D) \geq \text{freq}(G_P, D)$ and thus if P is frequent then every prefix of P is also frequent. Thus any prefix of a frequent canonical code is also a frequent canonical code. This means that GRIMA will find P because only infrequent codes and non-canonical codes are pruned from the search-space (lines 15 and 17). □

5.6 Node Induced GRIMA

Some graph mining algorithms may be improved by mining only closed patterns. For this, a closure operator is used to expand patterns: in line 11 of Algorithm 2, $P.e$ is replaced by $Closure(P.e)$. In general, the closure of a pattern is the maximal super-pattern with the same occurrence list. This closure operator has been adapted for general graphs in [Yan et al. 2003] and geometric graphs in MAXGEO [Arimura et al. 2007]. Closure computation adds some complexity but closed patterns are less numerous and thus sometimes faster to enumerate.

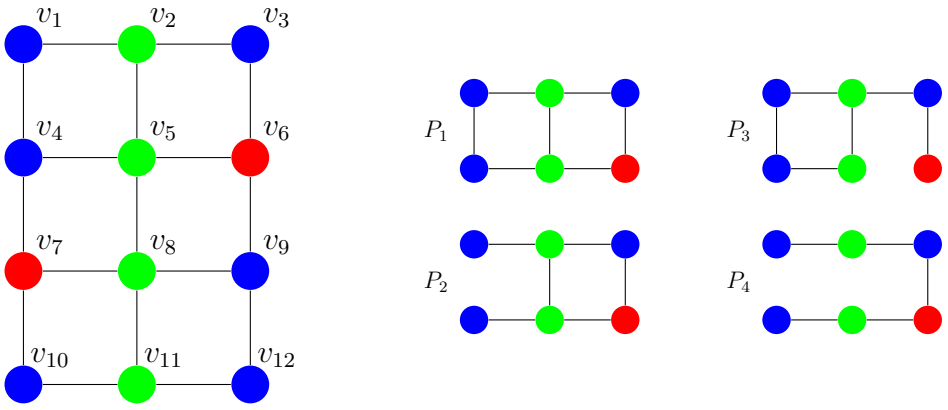


Figure 5.5 – Example of a complete $2\mathcal{D}$ grid, and of four patterns that all occur twice in it and that all share the same subset of vertices. P_1 is an induced subgrid of the complete grid whereas P_2, P_3, and P_4 are partial subgrids of it.

In the two applications considered in our experiments (reported in the next two chapters), there is no label on edges (or edges all have the same label). However, with the canonical code used by GRIMA, it is possible to use grid data that have label on edges. Furthermore, the mined grids are complete, *i.e.*, whenever two vertices have neighbor coordinates, they are connected by an edge. In this case, only vertices are important in patterns. More precisely, given two patterns P_1 and P_2 such that P_1 is a partial grid of P_2 (*i.e.*, P_1 has the same vertices as P_2, but the set of edges of P_1 is a subset of the set of edges of P_2), and given a grid G, P_1 and P_2 have the same occurrence lists in G. As a consequence, if one of these two patterns is frequent, then the other one is also frequent. Therefore, when mining frequent pattern, we only consider patterns which are induced subgrids of the initial grids or, in other words, that have a maximal number of edges. The resulting algorithm is called I-GRIMA.

For example, let us consider Figure 5.5. P_2, P_3, and P_4 are partial grids of P_1, obtained by removing some of its edges. These four patterns have the same occurrence lists in the grid G, and each of them is frequent if and only if all others are frequent. I-GRIMA only outputs P_1, whereas GRIMA outputs P_1, P_2, P_3, and P_4.

The implementation of I-GRIMA is derived from the implementation of GRIMA in a rather straightforward way. Before calling `extend(`$P.e$`)`, we check whether $P.e$ is a complete grid, *i.e.*, whether all couples of vertices which have neighbour coordinates are connected by an edge. If this is not the case, we add edges to $P.e$. More precisely, we add to $P.e$ every edge (u, v) such that u and v are vertices of $P.e$ and have neighbour coordinates, but (u, v) is not an edge of $P.e$.

I-GRIMA may be viewed as a loosening of a grid mining algorithm that

Algorithm 4 Enumerating pattern occurrences.

```
1: function enumOcc(a grid G, a pattern code P)
2:     Initialise the counter to 0
3:     for all edge (u, v) of G do
4:         if there is an occurrence of P in G that starts from u → v then
5:             increment the counter
6:         if there is an occurrence of P in G that starts from v → u then
7:             increment the counter
       Return c
```

only mines closed patterns, as it considers a relaxation of the closure operator restricted to edges. The interest is that this relaxed closure is very fast to compute and significantly reduces the number of patterns, as shown in our experiments. Moreover, experiments showed us that in our datasets almost all node-induced grids are actually closed. Therefore, computing the "full" closure operator would be more expensive and would not decrease significantly the number of patterns.

5.7 Algorithm for enumerating occurrences

In the two applications considered in the next two chapters, we show how to use GriMA for classifying objects that have a regular grid structure (images in Chapter 6 and cellular automata in Chapter 7). In both applications, we consider a similar process: we first use GriMA to extract frequent patterns; then we use these patterns to characterise objects by means of occurrence histograms. These occurrence histograms give, for each frequent pattern, the number of its occurrences in the grids. This occurrence enumeration task is actually achieved by GriMA during the mining process, as it maintains occurrence lists for each pattern and each grid of the database. However, in a classification context, we need to enumerate occurrences of patterns in new grids, that have not been used to compute frequent patterns. Hence, in this section we describe an algorithm for achieving this task.

This algorithm is displayed in Algorithm 4. Given the canonical code P of a pattern and a grid G, it iterates on each edge (u, v) of G and performs at most two traversals of P for each of these edges: a traversal that starts from u and visits v just after u (line 4), and a traversal that starts from v and visits u just after v (line 6). Each traversal uses the code P to deterministically choose edges of G to visit. More precisely, given the angle a of the next edge in P, if this angle corresponds to a temporal edge ($a < 0$), then we choose the corresponding temporal edge in G. If this angle corresponds to a spatial edge ($a > 0$), then we use this angle to choose the corresponding spatial edge in G.

For each visited edge, if vertex or edge labels are different from those in P, or if there is no edge in G corresponding to the current edge of P, then the traversal stops and we conclude that there is no occurrence of P from this starting point of G. If all edges match and all labels are equal, then we conclude that there is an occurrence of P from this starting point of G and we increment the counter.

The time complexity of Algorithm 4 is $\mathcal{O}(np)$ where n and p are the number of edges in G and P, respectively, as we perform at most $2n$ traversals and each traversal has a linear time complexity with respect to p. This complexity is an upper bound that considers a worst case where there is an occurrence of P for each possible starting edge in G. In practice, we stop the traversal of P as soon as a difference occurs, and this usually happens very soon in the traversal. For example, in the application described in Chapter 7, traversals are stopped at the fourth edge of P, on average, when P has 70 vertices.

As we usually have to enumerate occurrences of a large number of different patterns in a same grid G, we optimize the enumeration process by precomputing, for each possible kind of initial edge (*i.e.*, each possible label triple $(l_i, l_j, l_{(i,j)})$ associated with the two endnodes and the edge, respectively), the list of its occurrences in G. Then, given a pattern P, we search for occurrences of P only from the list associated with the label triple of the first edge of P.

5.8 Discussion

In this chapter, we have described GriMA, a **Gri**d **M**ining **A**lgorithm. This algorithm follows the same DFS exploration of the search space than gSpan, Plagram, and FreqGeo: to explore the search space, it recursively grows patterns by adding edges. However, it uses a different canonical code for uniquely representing patterns. This canonical code contains angle information that allows us to reconstruct a grid given its code, or to test for isomorphism in quadratic time with respect to the number of edges. We also use sparse sets to manage occurrence lists: this allows us to restore occurrence lists in constant time when returning back from a recursive call; this also allows us to reduce memory complexity as we do not need to save the current state of each occurrence list before each recursive call.

We have proven that GriMA is complete and correct, and that its time complexity is $\mathcal{O}(knp^3)$ per pattern P, where k is the number of grids in the base D, n the number of edges of the largest grid in D and p the number of edges in P. This is a significant improvement over FreqGeo which has a complexity of $\mathcal{O}(k^2n^4.\ln n)$ per pattern, and over Plagram which has a complexity of $\mathcal{O}(kn^2p^3)$ per pattern.

Chapter 6

Application of GriMA to Image Classification

Contents

In the previous chapter, we have described GriMA, an algorithm for mining frequent patterns in $2\mathcal{D}+t$ grids. In this chapter, we describe a first application of GriMA to image classification. A first goal is to study scale-up properties for extracting frequent patterns from large sets of $2\mathcal{D}$ grids that model images. A second goal is to study the interest of using grid patterns to classify images.

First, we present a quick background overview on image classification in section 6.1. We introduce a new image representation, based on frequent grid patterns, and called *Bag-of-Grids (BoG)* in section 6.2. We describe the experimental setup in section 6.3. We study scale-up properties of GriMA and

compare it with gSpan and Plagram in section 6.4. Finally, we present accuracy results of our approach for image classification in section 6.5.

6.1 Background on image classification

Image classification is a subfield of computer vision that aims at associating predefined classes to images. In this section, we first briefly recall basic background on supervised classification. Then, we describe the Bag-of-Words approach, where images are described by means of visual word histograms prior to classifying them. Finally, we briefly overview existing approaches that use data mining for image classification

6.1.1 Supervised classification

Given a set of objects, a set of classes, and a subset of objects (called the learning set) such that we know the class of each object in the learning set, the goal of supervised classification is to build a classifier which is able to predict the class of every object. This classifier is evaluated on a testing set of objects whose intersection with the learning set is empty, and for which we have a ground truth, *i.e.*, we know the class of each object in the testing set. Different performance indicators may be considered. In this thesis, we consider the classification rate, which is the percentage of objects in the testing set for which the predicted class is equal to the ground truth class.

When there are not enough objects for which we know the ground truth class, we may consider an n fold cross-validation process: Given the set of objects for which we know the class, we partition it into n subsets $O_1, \ldots, O_n$, and build n classifiers such that each classifier is trained on the learning set composed of all subsets but O_i and evaluated on the testing set O_i, with $i \in [1, n]$. In this case, we report the average accuracy for the n classifiers.

There exist many different approaches for learning a classifier such as, for example, Neural Networks, or k Nearest Neighbours. In this thesis, we use Support Vector Machines (SVMs). SVMs are binary classifiers, dedicated to the specific case where there are 2 classes.

To classify a vector x with a linear SVM, a linear combination of dot products $< x_i, x >$ between learning vectors x_i and x is computed, and the sign of the result determines the class. The separator is thus an hyperplane. The coefficients of this linear combination are optimized to maximize the margin m (i.e., no training point is allowed to lie within distance m of the separator). Most of the coefficient are zero, the non zero coefficients corresponds to so-called support vectors. However, learning data are generaly not linearly separable.

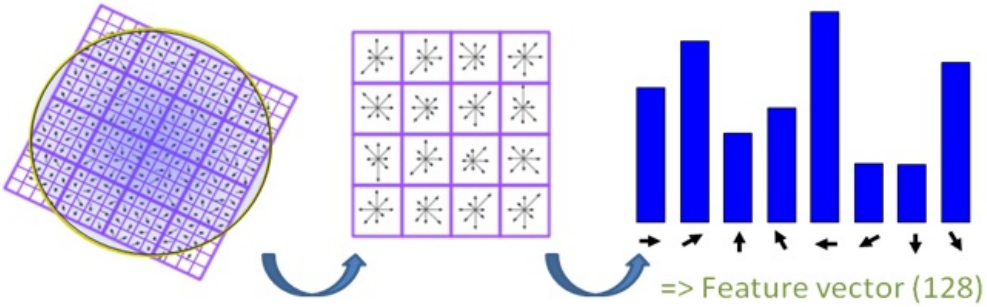


Figure 6.1 – Standard 16×16 SIFT descriptor with 4×4 bins.

Thus, errors are allowed (errors are learning vectors x_i lying within the margin or miss-classified) and the sum of errors is used as a regularization (weighted by an hyper-parameter C) in the optimization process.

The optimization of the coefficients of this linear combination only involves the computation of dot products between pairs of training vectors $< x_i, x_j >$. Indeed, it is possible to replace these dot products $< x, y >$ by any positive semi-definite function $k(x, y)$ (which is the same as doing a dot product in another vector space implicitly defined by k, this is the so-called kernel trick). Such functions k are called kernels and allow to learn non linear SVM classifiers (the final classifier will be a linear combination of $k(t_i, x)$, but k can be non-linear).

When there are more than 2 classes, we use SVMs to build an n-ary classifier as follows: for each class C_i, we build a binary SVM classifier which is trained to discriminate objects of C_i against objects of all other classes. To predict the class of an object of the testing set, we collect the answer of each binary SVM, which is a probability of belonging to the class associated with the binary SVM, and select the class for which the probability is the highest.

6.1.2 BoW-based representation of images

A key point of image classification is the choice of image representation. In this thesis, we consider Bag-of-Words (BoW) based representations. These representations no longer give state-of-the-art results. In particular, [Chatfield et al. 2014] reports that the features discovered using deep learning techniques give much better accuracy results than the BoWs and all their extensions on classification problems. However, our aim is to compare an unstructured set of descriptors and a set of descriptors structured by the grid topology. The method presented in this thesis is generic and may be used with any low-level features (*e.g.* deep-learned features) as labels.

The idea of BoWs is to describe images by frequency histograms of visual words [Chatfield et al. 2011, Csurka et al. 2004].

Construction of a visual vocabulary. Given a learning set of images, the set of visual words is called the visual vocabulary and is built in two steps.

In a first step, patches (*i.e.*, small connected sets of pixels corresponding to image regions) are extracted from images, and each of these patches is represented by a descriptor. In this thesis, we use SIFT descriptors [Lowe 2004] which are numerical vectors describing gradient information in a square pixel patch as represented in Figure 6.1. The standard SIFT descriptor is extracted from a patch of 16×16 pixels which is split into 16 areas, called bins, of 4×4 pixels. In each of these bins, image gradient is represented by a vector of 8 values, thus leading to a SIFT vector in $16 \times 8 = 128$ dimensions.

In a second step, the set of all SIFT descriptors (associated with all patches extracted from all learning images) is partitioned into k clusters. For each cluster, a representative SIFT descriptor is chosen. These k representative SIFT descriptors are the visual words that constitute the visual vocabulary. Here, k is a parameter which defines the size of the visual vocabulary.

Construction of BoWs. Given this vocabulary of k visual words, each image is described by a frequency histogram of k values which is computed as follows: first patches are extracted from the image, and the SIFT descriptor of each patch is computed; then, for each SIFT descriptor, the closest visual word in the vocabulary is added to the bag of visual words associated with the image; finally, the image is represented by a vector of k values, such that the i^{th} value in this vector is the number of occurrences of the i^{th} visual word in the bag.

6.1.3 Pattern mining for image classification

Pattern mining techniques have recently been very successfully used in image classification [Fernando et al. 2014, Voravuthikunchai et al. 2014] as a mean to obtain more discriminative mid-level features. However, these approaches consider the extracted features used to describe images (*e.g.*, BoWs) as spatially independent from each other.

The problem of using bag-of-graphs instead of BoWs has been mentioned in [Silva et al. 2013, Özdemir et al. 2010, Silva et al. 2014] for satellite image classification and biological applications. However, none of these papers provide a general graph representation nor a graph mining algorithm to extract the patterns. In their case, graphs associate vertices with image patch regions and edges are computed using Voronoi tessellations. Besides, they are interested in counting how many times a given subgraph occurs in an image graph using a maximum independent set support, whereas we are interested in counting the number of image graphs where a given subgraph occurs which defines a different, maybe

more usual, pattern support. Delaunay triangulation has already been used for image classification in [Samuel et al. 2010]. However, the triangulation structure is unstable to illumination changes, scale or rotation of objects in images and thus can hardly be used in practice for general image classification purposes.

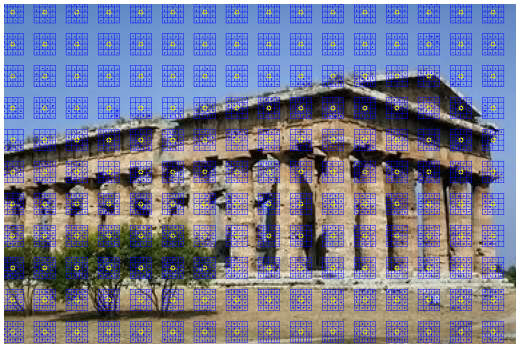
Recent works from [Silva et al. 2013, Silva et al. 2014] have also mentioned the use of bag-of-graphs for classification purposes. However, the authors mainly focus on describing the entire bag-of-graph creation process on an abstract level and do not precisely discuss the automated extraction of subgraph patterns nor the creation of the original graphs (they consider it as being driven by the application and handcrafted). Besides, they apply their method on biological graph problems and not on image processing. In [Nowozin et al. 2007], authors have already shown that, by combining graph mining and boosting, they can obtain classification rules based on subgraph features that contain more information than sets of features. In their graph, each interest point is represented by one vertex labeled with a discretized descriptor of the point as label. All vertices are connected by undirected edges to obtain a complete graph. For each edge, a discretized temporary continuous-valued edge label vector is used to represent the ratio of scales, a normalized distance and a horizontal orientation measure, respectively. gSpan algorithm is then used to compute the subgraph patterns but a limited number of features per image is used to be able to scale on real-life datasets.

6.2 BoG-based representation of images

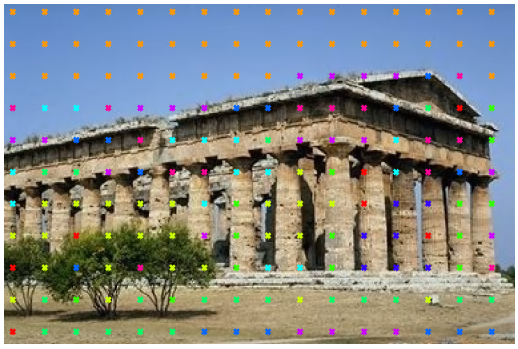
More than a decade ago, patches used to create the visual vocabulary were selected by using interest point detectors or segmentation methods. However, [Nowak et al. 2006] has shown that randomly sampling patches on grids (called dense sampling) gave as good (and often better) results for image classification than when using complex detectors. In this thesis, we propose to go one step further, and to model images by $2\mathcal{D}$ grids of visual words. Our goal is to study the interest of extracting structural information, by means of frequent subgrids, for characterizing images.

6.2.1 Construction of $2\mathcal{D}$ grids

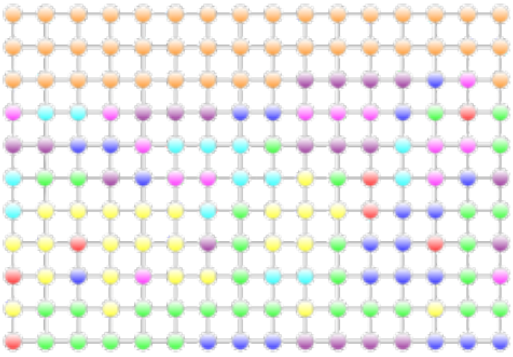
More precisely, given a visual vocabulary (computed on the set of learning images as described in the previous section), each image is described by a $2\mathcal{D}$ grid which is computed as follows. First, patches are extracted regularly, every s pixels, from the image, and the SIFT descriptor of each patch is computed, as shown in figure 6.2(a). Then, for each SIFT descriptor, we search for the closest



(a) SIFT descriptors densely applied on an image with a step s of 16 pixels between each SIFT center.



(b) Visual words associated with the same image as (a).



(c) $2\mathcal{D}$ grid associated with the same image as (a).

Figure 6.2 – Representation of an image by a $2\mathcal{D}$ grid of visual words.

visual word in the vocabulary (according to the squared euclidean distance), as shown in figure 6.2(b). Finally, we build a $2\mathcal{D}$ grid such that each vertex corresponds to a pixel patch and is labeled with the visual word associated with this patch. In this grid, two vertices are connected by an edge iff the two corresponding patches are neighbors, *i.e.*, they are separated by exactly s pixels. As a result, every vertex is connected to at most 4 vertices (it is connected to less than 4 vertices if it is on a border), as shown in figure 6.2(c). Note that edges do not have labels.

When the number k of visual words in the vocabulary is low, we noticed that, for many images, we obtain $2\mathcal{D}$ grids with large connected components that have a same label on all their vertices. In this case, we obtain a huge number of frequent patterns such that all vertices in the pattern have the same label. These patterns are not relevant for a classification purpose. To prevent us from mining these uniform patterns, for each $2\mathcal{D}$ grid associated with an image, we remove every vertex v such that all neighbours of v have the same label as v. This create holes in our grids, where each hole corresponds to a connected subset of vertices that all have a same label and that is surrounded by vertices

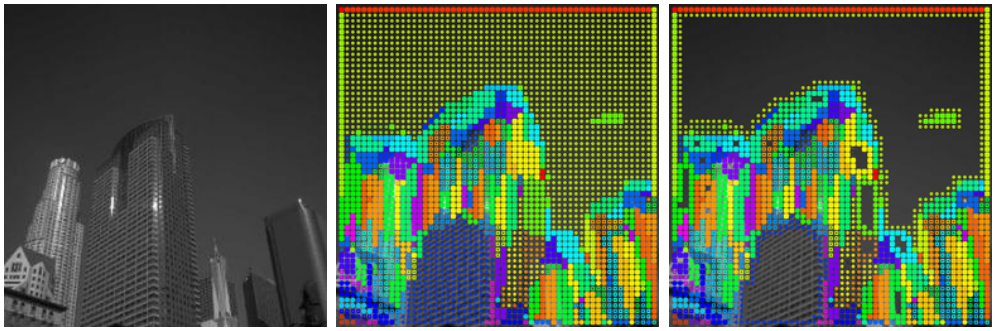


Figure 6.3 – Example of image (left), its associated $2\mathcal{D}$ grid (left) and its $2\mathcal{D}$ grid with holes (right), such that every vertex surrounded by vertices with the same label is removed.

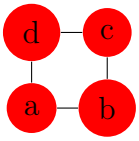
with the same label. An example of a $2\mathcal{D}$ grid with holes is given in Figure 6.3.

6.2.2 Construction of BoGs

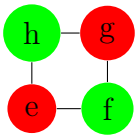
We propose to use GriMA to mine frequent subgrids, and to use these frequent subgrids to characterize images. As $2\mathcal{D}$ grids generated from images do not have labels on their edges, we use i-GriMA for mining frequent patterns: this allows us to reduce the number of mined patterns, and to speed-up the mining process. The mining process is done on each class separately. By mining each class separately, we ensure finding all relevant patterns for each class separately. For instance, let us assume that there are two classes, and that a pattern occurs in all images of the first class whereas it never occurs in the second class. In this case, if we mine all images of both classes together and if the frequency threshold σ is greater than 50%, then i-GriMA will not output this pattern, whereas it is relevant for distinguishing images of the two classes.

More precisely, let m be the number of classes, and $L_1, \ldots, L_m$ be the m learning sets of images, such that all images in L_i belong to the i$^{\text{th}}$ class, for $i \in [1, m]$. For each class i, we use i-GriMA to build the set P_i of grid patterns that are frequent in L_i. Then, all these patterns are merged to build the set of frequent patterns $P = \cup_{i=1}^{m} P_i$. Finally, these frequent patterns are used to characterize images by means of Bags of Grids (BoGs), corresponding to grid frequency histograms: each image is characterized by a vector of $|P|$ values such that the jth value corresponds to the number of occurrences of the jth pattern of P in the $2\mathcal{D}$ grid associated with the image.

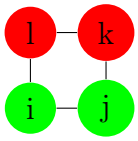
However, if a pattern has $a > 1$ automorphisms, then there are a occurrences of the pattern for each subset of vertices S in the target grid such that the subgrid induced by S is isomorphic to the pattern. For example, let us consider pattern (a) in Fig. 6.4. It has four automorphisms, corresponding to the four isomorphism functions: $(a \to a, b \to b, c \to c, d \to d)$, $(a \to d, b \to a, c \to$



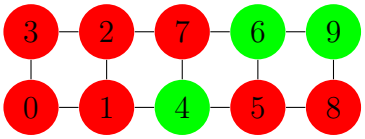
(a) Pattern with 4 automorphisms.



(b) Pattern with 2 automorphisms.



(c) Pattern without automorphism (different from the identity).



(d) Target grid.

Figure 6.4 – Pattern (a) has four occurrences in (d), corresponding to the four subisomorphism functions: $(a \to 0, b \to 1, c \to 2, d \to 3)$, $(a \to 1, b \to 2, c \to 3, d \to 0)$, $(a \to 2, b \to 3, c \to 0, d \to 1)$, and $(a \to 3, b \to 0, c \to 1, d \to 2)$. Pattern (b) has 2 occurrences in (d), corresponding to the two subisomorphism functions: $(h \to 4, g \to 7, f \to 6, e \to 5)$, and $(h \to 6, g \to 5, f \to 4, e \to 7)$. Pattern (c) has 1 occurrence in (d), corresponding to the subisomorphism function: $(j \to 6, i \to 9, l \to 8, k \to 5)$.

$b, d \to c)$, $(a \to c, b \to d, c \to a, d \to b)$, and $(a \to b, b \to c, c \to d, d \to a)$. As a consequence, there are four different occurrences of (a) in (d). Similarly, pattern (b) has two automorphisms and therefore it has two occurrences in (d). Finally, pattern (c) has no automorphism different from the identity and therefore it has only one occurrence in (d).

Hence, patterns that have automorphisms have a number of occurrences which is artificially increased, and this may introduce a bias. Therefore, we divide the number of occurrences of each pattern by its number of automorphisms. The number of automorphisms of a pattern is computed by searching for all occurrences of the pattern in itself, with the algorithm described in Section 5.7.

6.3 Experimental setup

BoW- and BoG-based classification processes are experimentally evaluated and compared in the next two sections. In this section, we first describe the datasets considered for these experiments. Then, we give an overview of the full classification process, and describe the parameter settings considered in our experiments.

Figure 6.5 – Examples of images from classes bluebell (a), iris (b), snowdrop (c), and tigerlily (d) of the Flowers dataset [Nilsback et al. 2008].

6.3.1 Datasets

We consider three datasets.

Flowers [Nilsback et al. 2008] is composed of 17 classes where each class corresponds to a different kind of flower. Each class contains 80 different images of this kind of flower. Figure 6.5 displays five images of the classes bluebell, iris, snowdrop and tigerlily.

15-Scenes [Lazebnik et al. 2006] is composed of 15 classes where each class corresponds to a natural scene category such as, for example, kitchen, street, and forest. Each class contains between 210 and 410 images. The complete dataset contains 4485 images. Figure 6.6 displays five images of the classes bedroom, highway, kitchen and street.

Caltech-101 [Li et al. 2004] is composed of 101 classes of various kinds. There are 40 to 800 images per class. As we mine frequent patterns for each class separately, if a class only contains a few images in the learning set, then the number of frequent patterns may be very large for this class. Hence, we consider a subset of *Caltech-101* which is composed of the 26 classes of *Caltech-101* that contain at least 80 pictures per class. These classes are: airplaines, bonsai, brain, buddha,

Figure 6.6 – Examples of images from classes bedroom (a), highway (b), kitchen (c), and street (d) of the 15-Scenes dataset [Lazebnik et al. 2006].

butterfly, car_side, chandelier, ewer, faces, faces_easy, grand_piano, hawksbill, helicopter, ibis, kangaroo, ketch, laptop, leopards, menorah, motorbikes, revolver, scorpion, starfish, sunflower, trilobite and watch. For each class, we randomly select 80 images. This subset will be referred as *Caltech-26*. Figure 6.7 displays five images of the classes watch, starfish, kangaroo and brain.

Table 6.1 summarizes informations on our three datasets.

For *Flowers* and *Caltech-26*, we consider a 10-fold cross-validation process since the number of images is the same for each class. For *15-Scenes*, we have created 10 different folds: For each fold, we randomly select 100 images per class for training and 50 images per class for testing.

Table 6.1 – Dataset information summary: For each dataset, #Class gives the number of classes, #Images/Class gives the minimum, average and maximum number of images per class, and #Pixels/Image gives the minimum, average and maximum number of pixels per image.

	#Class	# Images/Class			# Pixels/Image		
		Min	Avg	Max	Min	Avg	Max
Flowers	17	80	80	80	249,001	319,381	546,500
15Scene	15	210	298	410	44,660	66,328	121,440
Caltech-26	26	80	80	80	24,576	71,782	313,040

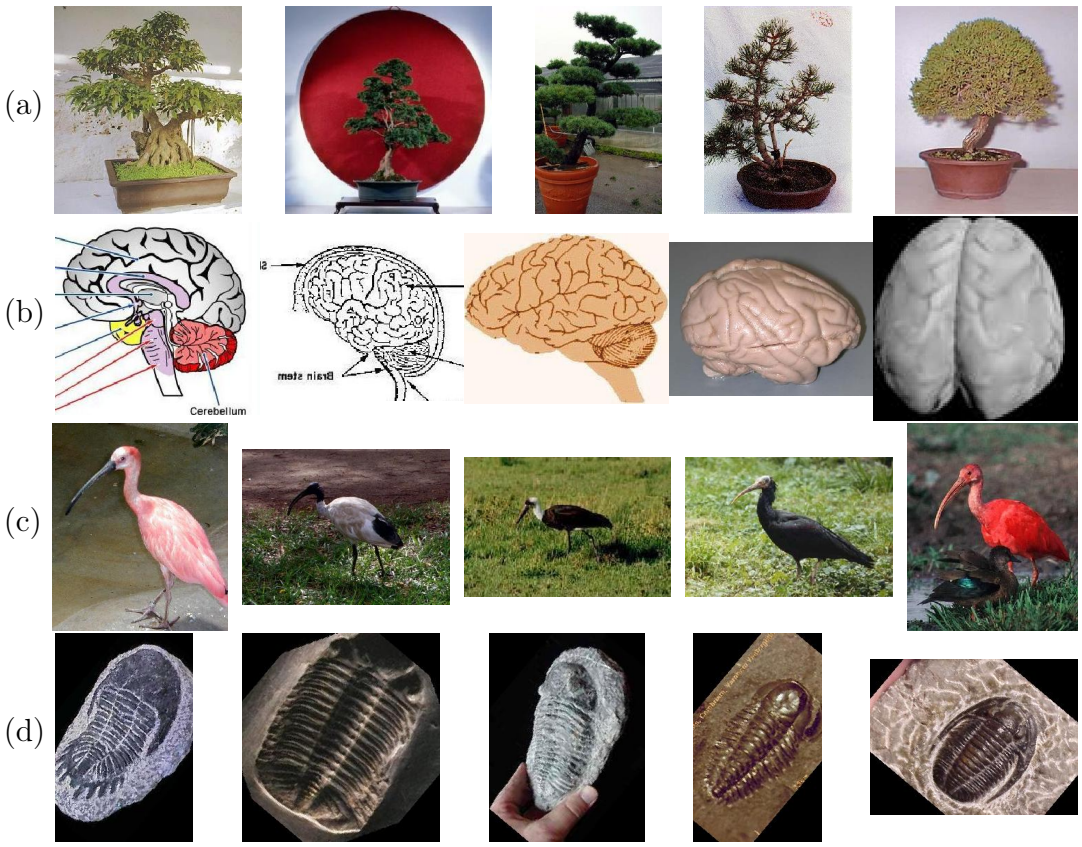


Figure 6.7 – Examples of images from classes bonsai (a), brain (b), ibis (c), and trilobite (d) from the Caltech-26 dataset [Li et al. 2004].

6.3.2 Overview of the learning process and parameter settings

Figure 6.8 gives an overview of the whole process for building BoW- and BoG-based classifiers from the learning set L of images.

Construction of the visual vocabulary. The first step for building BoW- and BoG-based classifiers is to build a visual vocabulary V. This is done by extracting a patch of 16×16 pixels every s pixels and computing the SIFT descriptor in 128 dimensions corresponding to this patch. In our experiments, we set s to 8. This way, there is an overlapping between patches such that every pixel occurs in 4 patches (except for pixels that are close to image borders). Then, the resulting set of SIFT descriptors is partitioned into k clusters. In our experiments, we have computed clusters with the k-means algorithm, and we have used the squared euclidean distance to compare SIFT descriptors.

Depending on the dataset, it may happen that there are too many SIFT descriptors for running k-means within a reasonable amount of time. Hence, we limit the maximum number of SIFT descriptors to $100,000$: When there

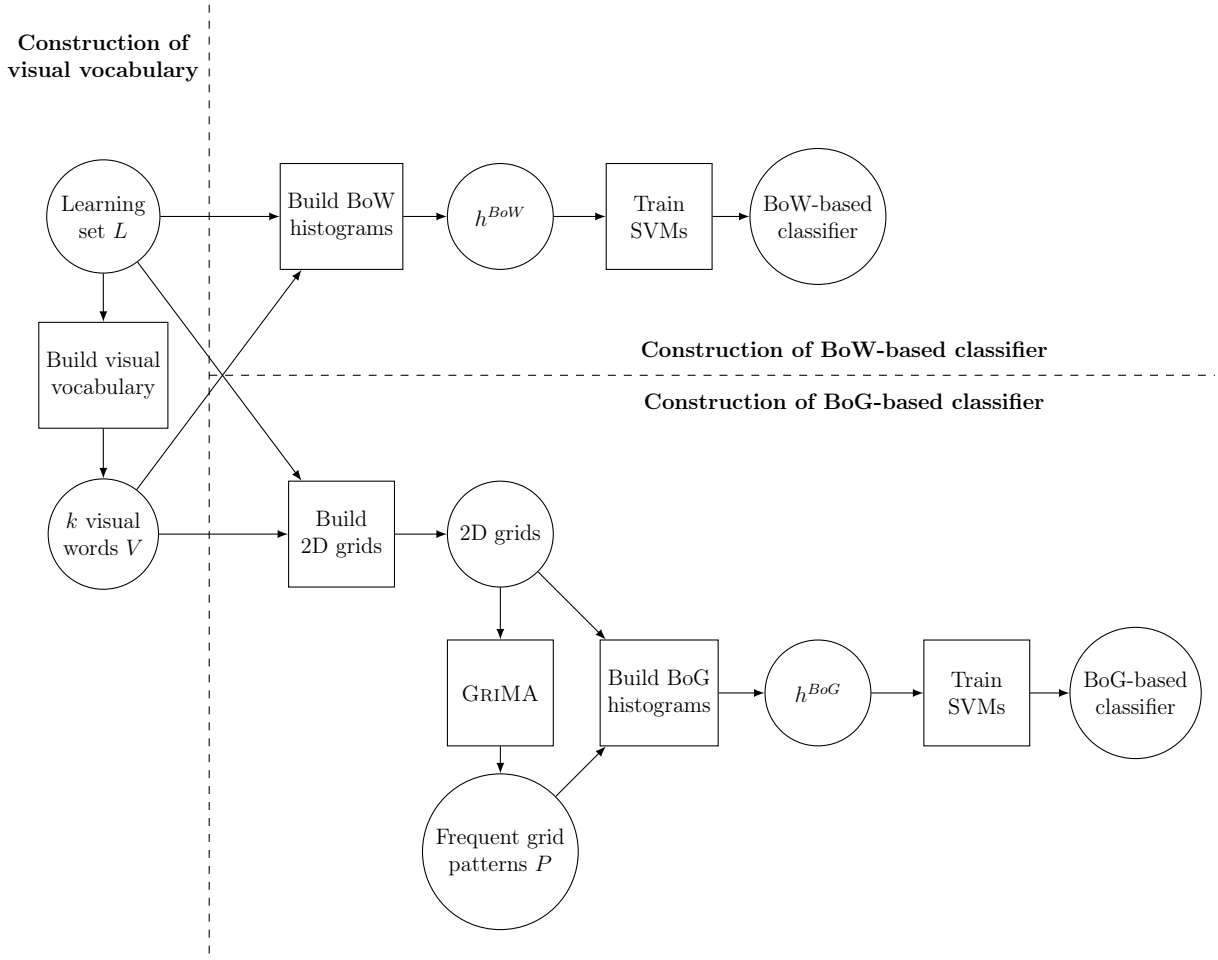


Figure 6.8 – Overview of the learning process for BoW (top right) and BoG (bottom right). The construction of the visual vocabulary (left) is shared by BoW and BoG.

are more than $100,000$ SIFT descriptors, we randomly select $100,000$ of them according to a uniform distribution. At the end of the clustering process, the visual vocabulary V contains the k SIFT descriptors which are the centroids of the k clusters.

The value of k has a strong impact on the results, and the value which gives the best results both depends on the dataset and on the classification process (BoW or BoG). Hence, we report results with the following set of values: $k \in \{100, 500, 1000, 2000, 4000\}$.

Construction of a BoW-based Classifier. For each image $i \in L$ of the learning set, we first extract a patch of 16×16 pixels every s pixels, with $s = 8$, and compute the corresponding SIFT descriptors in 128 dimensions, to obtain a collection C_i of SIFT descriptors. Then, we describe i by an histogram h_i^{BoW} in k dimensions, such that for each $j \in [1, k]$, the j^{th} value of h_i^{BoW} is the number of SIFT descriptors of C_i that belong to the j^{th} cluster, *i.e.*,

$h_i^{BoW}[j] = |\{s \in C_i : \forall l \in [1,k], d(s,V_j) \leq d(s,V_l)\}|$ where V_j is the j^{th} SIFT descriptor in the visual vocabulary V, and d is the squared euclidean distance.

All h_i^{BoW} histograms are normalized to obtain real values ranging between 0 and 1: for each dimension $j \in [1,k]$ and each image $i \in L$, $h_i^{BoW}[j]$ is replaced with $\frac{h_i^{BoW}[j]-\min_{l\in L} h_l^{BoW}[j]}{\max_{l\in L} h_l^{BoW}[j]-\min_{l\in L} h_l^{BoW}[j]}$.

Finally, normalized histograms are used to train an SVM classifier. We have used the SVM implementation of Libsvm [Chang et al. 2011] with the intersection kernel presented in [Odone et al. 2005] as it is reported to be one of the best kernels for image classification. The value of the C parameter (corresponding to the cost of constraint violations) is optimized by 5-fold cross-validation on the learning set, for all powers of 2 ranging between 2^{-10} and 2^{10}

Construction of a BoG-based Classifier. For each image $i \in L$ of the learning set, we build a $2\mathcal{D}$ grid with holes G_i as described in Section 6.2.1. Then, for each class, we use i-GriMA to mine frequent patterns in the grids associated with images of this class. The frequency threshold parameter σ has a strong impact on the results: when σ is set to very high values, such as 95%, the number of frequent patterns may not be large enough to allow SVM to learn a good model; when σ is set to very low values, such as 5%, the number of frequent patterns is usually so huge that neither the mining process nor the learning process can be completed within a reasonable amount of time. Hence, we report results with different values for σ.

Given the set P of all frequent patterns, in all classes, we build a frequency histogram h_i^{BoG} in $|P|$ dimensions such that the j^{th} value of h_i^{BoG} is the number of occurrences of the j^{th} pattern of P in G_i, divided by the number of automorphisms of the pattern.

Like for the BoW-based classification, h_i^{BoG} histograms are normalised to obtain real values ranging between 0 and 1: for each dimension $j \in [1,|P|]$ and each image $i \in L$, $h_i^{BoG}[j]$ is replaced with $\frac{h_i^{BoG}[j]-\min_{l\in L} h_l^{BoG}[j]}{\max_{l\in L} h_l^{BoG}[j]-\min_{l\in L} h_l^{BoG}[j]}$.

Finally, normalised histograms are used to train an SVM classifier, and we consider the same process as for the BoW-based classifier.

6.3.3 Overview of the classification process

Figure 6.9 gives an overview of the process used to classify a test image with BoW- and BoG-based classifiers. In both cases, we use the visual vocabulary V built during the learning process.

BoW-based classification. Given an image $i \in T$ of the testing set and the visual vocabulary V, we build a normalized histogram h_i^{BoW} as described in the

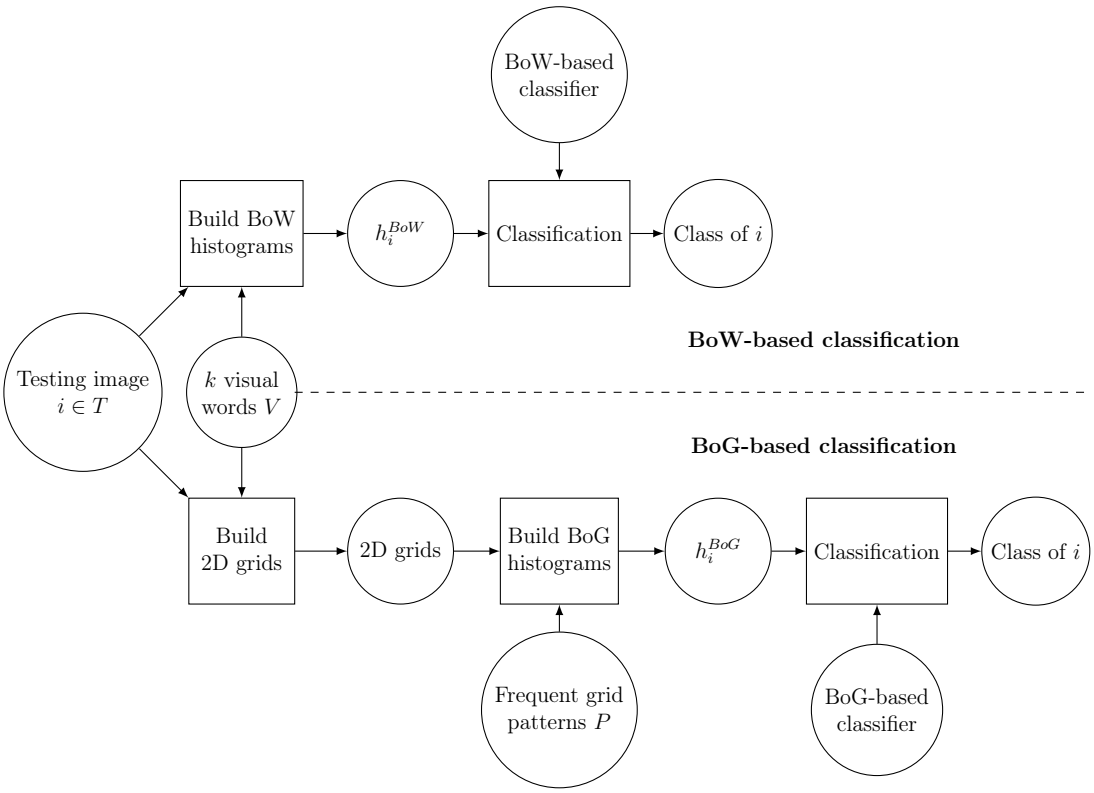


Figure 6.9 – Overview of the testing process for BoW (top) and Bog (bottom).

previous section. The only difference is during the normalization step. Indeed, it may happen that a value in h_i^{BoW}, before normalization, is greater than the maximal value for all histograms associated with images of the learning set. In this case, the normalized value is set to 1. This normalized histogram h_i^{BoW} is given as input to the BoW-based classifier which returns in output the predicted class for i.

BoG-based classification. Given an image $i \in T$ of the testing set and the visual vocabulary V, we build a normalized histogram h_i^{BoG} as described in the previous section. Like for BoW-based classification, the only difference is in the normalization step, and values greater than 1 are set to 1. This normalized histogram h_i^{BoG} is given as input to the BoG-based classifier which returns in output the predicted class for i.

6.4 Efficiency analysis

Let us first evaluate scale-up properties of GriMA, and compare it with gSpan and Plagram. For this first experiment, we consider the *Flowers* dataset. We consider the $2\mathcal{D}$ grids computed with a visual vocabulary that contains $k = 100$

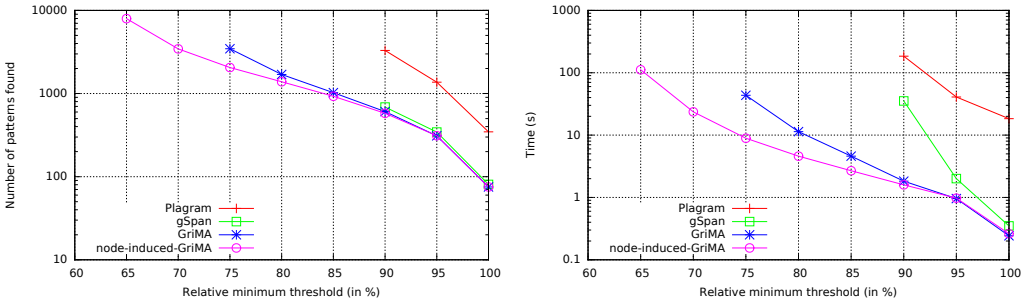


Figure 6.10 – Number of frequent patterns (left) and time to compute all frequent patterns (right) with respect to different frequency threshold values of σ (average for all classes of *Flowers*), for gSpan, Plagram, GriMA and i-GriMA. Time is limited to 1 hour per class.

words (*i.e.*, the number of different vertex labels is 100), and with the parameter s (that defines the number of pixels between two vertices) set to 4. Therefore, for each class of the dataset, we have 80 $2\mathcal{D}$ grids that have 17,963 vertices on average (31,388 for the largest grid, and 13,689 for the smallest one).

Figure 6.10 compares GriMA, i-GriMA, gSpan and Plagram and reports both the CPU time needed to mine frequent patterns, and the number of mined patterns, for different values of the frequency threshold σ ranging from 100% to 65% by steps of 5%. Each class is mined separately, and we report average results for the 17 classes. We set a timeout of 1 hour, *i.e.*, if an algorithm needs more than one hour to mine at least one class of the dataset, then we report a time-out for this algorithm. This occurs when $\sigma \leq 85\%$ for Plagram and gSpan, when $\sigma \leq 70\%$ for GriMA, and when $\sigma \leq 60\%$ for i-GriMA. In this case, we do not display the number of mined patterns with this algorithm.

Note that, as explained in Section 3.3.3, the face-based expansion strategy of Plagram does not allow it to find patterns with no face. To allow Plagram to mine the same patterns as its competitors, each vertex of the original grid graph is replaced by a four-vertex face (as illustrated in Figure 5.1). This way, each frequent subgrid in the original grid is found by Plagram in the expanded grid. However, some patterns found by Plagram do not correspond to patterns in the original grid, *e.g.*, faces in the expanded grid which correspond to edges without their end vertices in the original grid. For this reason, the number of patterns that are mined by Plagram and its computation time are higher than the ones reported for its competitors.

gSpan does not consider edge angles when mining subgraphs, and two isomorphic subgraphs may have different edge angles so that they do not correspond to isomorphic subgrids, as illustrated in Figure 3.6. Therefore, gSpan and GriMA compute different sets of frequent patterns. However, Figure 6.10 shows us that the number of patterns is rather similar (slightly higher for gSpan

Table 6.2 – Classification results. Each line reports results on the three datasets for a different number of visual words k: classification rates obtained with BoW and BoG, followed by the frequency threshold σ used for BoG (*i.e.*, the largest value that allows i-GriMA to mine at least 4000 patterns) and the number #pat of frequent patterns extracted by i-GriMA with this value of σ.

K	15 Scenes				Flowers				Caltech-26			
	BoW	BoG	#pat.	σ	BoW	BoG	#pat.	σ	BoW	BoG	#pat.	σ
100	70.7%	70.0%	4,190	60%	48.0%	63.3%	5,284	80%	71.7%	73.7%	6,473	80%
500	73.7%	72.0%	4,027	40%	59.6%	63.9%	4,377	55%	76.3%	75.4%	4,240	60%
1000	73.8%	73.1%	4,877	30%	63.7%	64.6%	4,545	45%	77.3%	76.9%	6,044	45%
2000	73.2%	73.8%	6,128	20%	67.5%	66.9%	4,640	35%	77.0%	77.2%	4,502	40%
4000	74.2%	75.0%	4,218	20%	66.9%	67.0%	5,822	25%	77.2%	75.7%	16,345	35%

than for GriMA). GriMA clearly scales better than gSpan: it is able to compute all frequent patterns within one hour per class for values of σ down to 75%, whereas gSpan has time-outs when $\sigma \leq 85\%$. Finally, the node-induced version of GriMA, i-GriMA, scales better than GriMA, as explained in Section 5.6, and it is able to find all frequent patterns in less than 100s on average for each class when $\sigma = 65\%$ whereas GriMA has not completed its execution after 3600s for at least one class.

6.5 Accuracy results

Global accuracy results. We report in Table 6.2 the results obtained for our three datasets with different values for the number k of visual words. As pointed out in the previous section, the value of the frequency threshold σ has a strong impact on the number of mined patterns, and this number of mined patterns also depends on the size k of the vocabulary: when k is small, many vertices share the same labels, and therefore the number of frequent patterns is higher. First experiments have shown us that better classification results are obtained when the number of mined patterns (which corresponds to the dimension of the vectors given as input to SVM) is close to 4000. Therefore, we automatically set the value of σ (for each dataset and each value of k) as follows: starting from $\sigma = 100\%$, we progressively decrease σ, by steps of 5, and mine frequent patterns with i-GriMA for each value of σ until the number of mined patterns becomes greater than 4000. We report in Table 6.2 the largest value of σ for which the number of mined patterns is greater than 4000, together with the actual number of mined patterns for this value of σ. As expected, we need to lower the value of σ to obtain at least 4000 patterns when the number of visual words increases: When $k = 100$, σ is set to 60%, 80%, and 80% for

	BoW	BoG											
		75%	70%	65%	60%	55%	50%	45%	40%	35%	30%	25%	20%
suburb	99.1	99.1	99.1	99.1	99.1	**99.2**	**99.2**	**99.2**	**99.2**	**99.3**	**99.3**	**99.3**	**99.3**
coast	97.4	97.4	**97.5**	**97.5**	**97.5**	**97.5**	**97.5**	**97.5**	**97.5**	**97.5**	97.2	97.1	97.2
forest	98.8	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	**98.9**	98.8	98.7
highway	97.5	**97.7**	**97.7**	**97.6**	**97.7**	**97.6**	97.5	97.5	97.4	97.3	97.3	97.2	97.1
moutain	97.7	97.6	97.6	97.6	97.7	97.6	97.6	97.5	97.5	97.5	97.5	97.4	97.3
street	97.8	**97.9**	97.8	97.8	97.7	97.7	97.8	97.7	97.7	97.5	97.3	97.1	96.8
industrial	94.6	**94.7**	**94.9**	**94.8**	**95.0**	**95.1**	**95.1**	**95.1**	**95.2**	**95.3**	**95.6**	**95.4**	**95.4**

Table 6.3 – Classification rates of binary SVMs for 7 classes of the *15-scenes* dataset with a vocabulary of $k = 1000$ visual words and a threshold $\sigma \in [75\%, 20\%]$. For each class, we highlight in bold BoG results that are better than BOW, and in green (resp. red) BoG results that are significantly better (resp. worse) than BOW. For the other 8 classes, BOW and BoG are never significantly different.

15 scenes, *Flowers*, and *Caltech-26*, respectively, whereas when $k = 4000$, σ is decreased to 20%, 25%, and 35%, respectively.

We provide a Student two-tailed t-test (with p-value set to 0.05) to statistically compare classification rates of BoW and BoG for each dataset: results in green (resp. red) are statistically better (resp. worse). On *15-Scenes*, the use of structural information statistically improves accuracy only for $k = 4000$, but this also gives the best accuracy results. On *Flowers* on the contrary, BoG greatly improves the results for low numbers of visual words ($k \leq 500$) but it is not statistically better (nor worse) for higher numbers ($k \geq 1000$) where the classification rate is the best ($k = 4000$). For *Caltech-26*, BoG slightly improves the results when $k = 100$, and it is not significantly different for higher values of k. Overall, those results show that adding structural information does not harm and sometimes improves the representation of images and that our grid mining algorithm can be used in a real-life context.

Binary classification results. Table 6.3 provides an insight into each class separately for the *15-scenes* dataset (similar results were observed for the two other datasets). Only the 7 classes with statistically significant differences are shown in the table.

Note that these classification rates are not comparable with the ones of Table 6.2, where we report n-ary classification results: Table 6.3 reports binary classification results, *i.e.*, for each class C, we report the percentage of images of the whole dataset which are well classified with respect to the two classes "$\in C$" and "$\notin C$". We can see that some classes really benefit from the use of structured patterns for almost all frequency thresholds (*e.g.*, *industrial*, *coast*, *suburb*, *forest*) whereas for some classes, using unstructured information gives better results (*e.g.*, *street* or *moutain*). This is due to the fact that for some

classes, the structure is too similar from this class to another to use it to discriminate classes.

6.6 Discussion

In this chapter, we have described a first application of GriMA to mine frequent patterns in $2\mathcal{D}$ grids that represent images. Our first goal was to evaluate scale-up properties of GriMA, and to show that it may be used to efficiently mine frequent patterns within real-world data. Experimental results have shown us that GriMA scales better than gSpan and Plagram, and that the node-induced variant of GriMA, i-GriMA, scales even better.

Our second goal was to evaluate the interest of integrating structural information, by means of frequent grid patterns, for image classification. Experiments on three datasets have shown us that these patterns may improve classification accuracies compared to an unstructured BoW-based approach. However, these experiments do not give state-of-the-art results on image classification as mentioned in Section 6.3. If BoW-based approaches were state-of-the-art approaches a decade ago, they are now clearly outperformed by other approaches, such as Convolutional Neural Networks [Rastegari et al. 2016, Huang et al. 2017]. Hence, we plan to improve our approach by integrating new image descriptors. Indeed, our grid-based model is general, and it is possible to use other descriptors than SIFT descriptors, such as SURF or descriptors computed with deep learning approaches, for instance. It is also possible to combine different descriptors: each image may be represented by multiple grids, such that each grid considers a different kind of descriptor, and GriMA may be used to extract frequent patterns from these different grids.

Another possibility is to consider multiple step sizes between descriptors for each image. For instance, each image may be represented by 3 grids obtained with a step s set to 4, 8, and 16, respectively. This leads us to the idea of multi-scale grids, and the mining process may be performed on grids at different scales, thus allowing us to discover patterns that do not have the same scale on different images.

Similarly to the multi-scale representation, another approach that could be studied is the representation of image using Spatial Pyramid Matching [Lazebnik et al. 2006]. This approach aims at partitioning images into sub-regions and representing them by a combination of BoW-based histograms of each sub-regions. This approach could be extended using our BoG-based method in order to add grid-based structural information that could benefit to image classification.

Finally, all these proposed improvements could lead to a huge number of patterns. Hence, it could be interesting to select a smaller number of relevant patterns using post-processing techniques such as those described in Section 3.1.2.

Chapter 7

Application of GriMA to Cellular Automata Analysis

Contents

In the previous chapter, we have presented a first application of GriMA to image classification, where frequent patterns in $2\mathcal{D}$ grids are used to characterize images. In this chapter, we describe a second application of GriMA to cellular automata analysis. A first goal is to study scale-up properties of GriMA for extracting frequent patterns from large sets of $2\mathcal{D}+t$ grids that represent the temporal evolution of cellular automata. A second goal is to study the interest of characterizing cellular automata by means of frequent $2\mathcal{D}+t$ patterns for predicting their outcomes.

In Section 7.1, we briefly introduce cellular automata, with a specific focus on Conway's Game of life which is the cellular automata considered in our experiments. In Section 7.2, we describe the datasets considered in our experiments, and introduce our experimental setup for predicting the outcomes of

cellular automata. In Section 7.3, we study scale-up properties of GRIMA and compare it with GSPAN for mining and with a subgraph isomorphism algorithm for enumerating pattern occurrences. Finally, in Section 7.4 we present accuracy results of our approach to predict cellular automata outcomes.

7.1 Background on Cellular Automata

Cellular Automata (CA) are discrete models that are often used to model the temporal evolution of complex systems such as, for example, ecosystems [Wolfram 1984, Wootton 2001, Breckling et al. 2011]. Indeed, biodiversity of ecosystems is increasingly recognized as an important element of global change. CA-based models are used to understand, predict and control spatio-temporal spread of species which is a key issue to preserve biodiversity [Marco et al. 2002].

More formally, a CA is a regular grid of cells. This grid defines a neighborhood relation between cells, such that each cell has a finite number of neighbor cells. Initially, each cell has some given state. Then, cell states evolve through time: at each time step, the new state of each cell is computed by using some given transition rule. Usually, the same transition rule is used for all cells, and it computes the new state of a cell given its current state as well as the states of its neighbor cells.

When executing a CA from a given initial state, one may observe the emergence of spatio-temporal patterns, and these patterns are characteristic of different outcomes. In the context of CA that model ecosystems, for example, [Wolfram 1984] distinguishes four possible outcomes: (1) development of a homogeneous fixed pattern, (2) development of a periodic pattern, (3) development of a chaotic pattern, and (4) development of patterns composed of homogeneous regions and regions containing complex localized structures.

In this chapter, we study the scale-up properties of GRIMA and assess the relevance of the mined patterns to predict outcomes of one of the most famous CA which is the Game of Life [Conway 1970]. In this CA, grids are in two dimensions, and are usually toric. Hence, if a grid has $n \times m$ cells, then each cell at coordinates $(x, y) \in [0, n-1] \times [0, m-1]$ has exactly eight neighbors: two horizontally, at coordinates $((x-1)\%n, y)$ and $((x+1)\%n, y)$, two vertically, at coordinates $(x, (y-1)\%m)$ and $(x, (y+1)\%m)$, and four diagonally, at coordinates $((x-1)\%n, (y-1)\%m)$, $((x-1)\%n, (y+1)\%m)$, $((x+1)\%n, (y-1)\%m)$, and $((x+1)\%n, (y+1)\%m)$. There are two possible cell states which are *alive* and *dead*. Initially (at time $t = 0$), each cell is either alive or dead. The state at time $t+1$ of a cell depends on its state and on

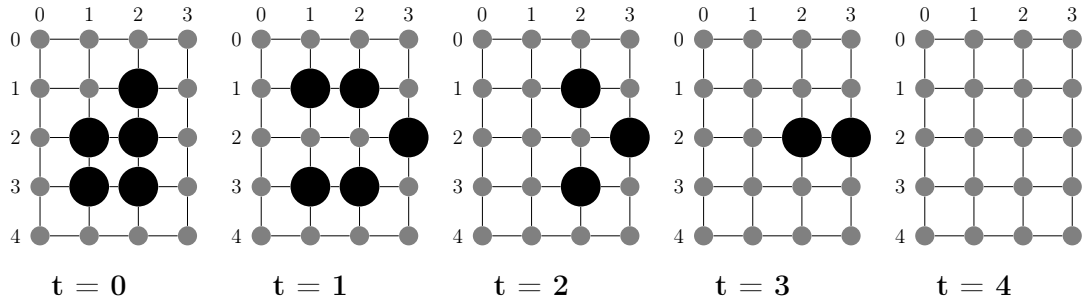


Figure 7.1 – Example of initial state of a game of life of size 4×5 (t=0), and its next four states (t=1 to t=4). Dead cells are represented by small gray circles and living cells are represented by large black circles. Grids are toric, though we do not represent edges that connect every vertex at coordinates $(0, j)$ (resp. $(i, 0)$) with vertex at coordinates $(3, j)$ (resp. $(i, 4)$).

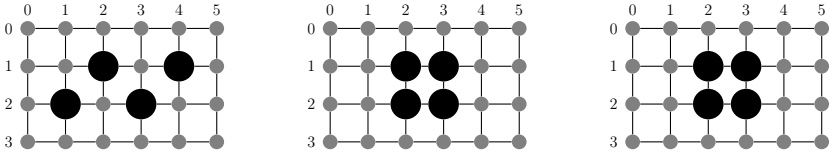
the state of its 8 neighbors at time t. It is computed by applying the following rules:

1. an alive cell with less than two alive neighbors at time t becomes dead at time $t + 1$, as if caused by underpopulation;
2. an alive cell with more than three alive neighbors at time t becomes dead at time $t + 1$, as if caused by overcrowding;
3. an alive cell with two or three alive neighbors at time t stays alive at time $t + 1$;
4. a dead cell with exactly three alive neighbors at time t becomes alive at time $t + 1$;
5. a dead cell with a number of alive neighbors different from three at time t stays dead at time $t + 1$.

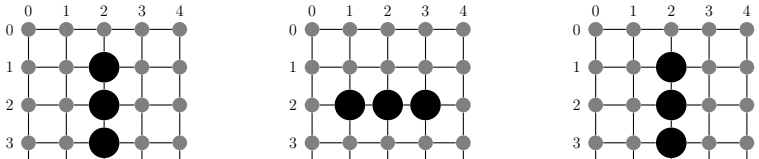
Figure 7.1 displays an example of initial state, and its next four states obtained when applying the rules of the Game of Life. For example, at time $t = 0$, the cell on $(1, 1)$ is dead and has exactly three living neighbors, at coordinates $(2, 1)$, $(2, 2)$, and $(1, 2)$, so that it becomes dead at time $t = 1$.

Different outcomes may be observed for a Game of Life CA. Examples of outcomes are:

- *Dead outcome*, where all cells are dead at some time step t. In this case, all further states, at any time step $t' > t$ will be dead. This is the case, for example, for the initial state displayed in Figure 7.1: all cells are dead at any time step $t \geq 5$.
- *Stable outcome*, where at some time step t the grid state becomes stable, *i.e.*, the state of each cell at time $t' > t$ is the same as its state at time t



(a) Example of initial state with stable outcome: states at all time steps $t' \geq 1$ are all equal.



(b) Example of initial state with periodic outcome of period $p = 2$: the state at time $t = 0$ is equal to the state at time $t = 2$, and different from the state at time $t = 1$.

Figure 7.2 – Examples of initial states with stable and periodic outcomes.

(and there is at least one cell which is alive, to distinguish this outcome from the dead outcome). This is the case, for example, for the initial state displayed in Figure 7.2(a).

- *Periodic outcome*, where there exist two time steps t_1 and $t_2 = t_1 + p$ with $p > 1$ such that the grid state at time t_1 is the same as the grid state at time t_2 whereas for every other time step t_3 such that $t_1 < t_3 < t_2$, the grid state at time t_3 is different from the grid state at time t_1. The integer $p > 1$ is called the period. For example, Figure 7.2(b) displays an initial state with periodic outcome of period $p = 2$.

The Game of Life has been shown to be Turing-complete, *i.e.*, it has the same computational capacity as Turing machines [Rendell 2014]. This implies that the problem of deciding whether a given initial state leads to some given final state (such as, for example, a state where all cells are dead) is undecidable.

The Game of Life is well suited to evaluate our grid mining algorithm GriMA. Indeed, the successive states of a grid may be modeled with $2\mathcal{D} + t$ grids in a straightforward way. Mining these grids is challenging, as they may be very large. Furthermore, frequent patterns may be useful to characterize outcomes, and our goal is to evaluate their interest for predicting final outcomes given the first states.

7.2 Experimental Setup

The goal of the experiments reported in this chapter is to predict the outcome at time $t = 1000$, given the first k states of a Game of Life, from time $t = 1$ to time $t = k$. In this section, we first describe the datasets considered in our experiments. Then, we show how to represent the first k states of a Game of Life by means of frequent $2\mathcal{D} + t$ grid pattern histograms. Finally, we describe the learning and the classification processes used to predict outcomes.

7.2.1 Dataset construction

We consider four sizes of $n \times n$ grids, with $n \in \{20, 30, 40, 50\}$. For each size, we have generated four sets of initial states, corresponding to four different outcomes at time $t = 1000$: (D) Dead, *i.e.*, all cells are dead at time $t = 1000$; (S) Stable, *i.e.*, the state at time $t = 1000$ is equal to the state at time $t = 999$, and at least one cell is alive; (P) Periodic with a period $p = 2$, *i.e.*, the state at time $t = 1000$ is equal to the state at time $t = 998$, but different from the state at time $t = 999$; and (O) Other, *i.e.*, the states at times $t = 1000$, $t = 999$, and $t = 998$ are all different.

Each initial state is generated by randomly and independently choosing the initial state (dead or alive) of each cell with respect to a probability p_n. We have empirically chosen p_n in such a way that the outcome at time $t = 1000$ is either D or one of the three other possible outcomes with equal probabilities: this probability p_n is 74%, 78%, 80%, and 81% for $n = 20, 30, 40$, and 50, respectively. Besides, to avoid trivial predictions due to the fact that outcomes may be reached before the k^{th} time step, we only select initial states such that there is at least one cell alive at time $t = 50$, and such that states at times $t = 50$, $t = 49$ and $t = 48$ are all different. For each size $n \in \{20, 30, 40, 50\}$, and for each possible outcome $X \in \{D, S, P, O\}$, we have generated a set S_n^X of 1000 initial states such that the outcome at time $t = 1000$ is X. We note $S_n = \cup_{X \in \{D,S,P,O\}} S_n^X$.

Figure 7.3 displays the first four states (from $t = 0$ to $t = 3$), and the last 3 states (from $t = 998$ to $t = 1000$), of one element of each set S_n^X with $X \in \{D, S, P, O\}$. The state at time $t = 0$ is very different from all other states, at times $t > 0$: if the percentage of living cells is higher than 70% at time $t = 0$, it drops down to less than 4% at time $t = 1$, for every grid size n, and this percentage stays rather low for all time steps until $t = 1000$. Hence, the initial states that are stored in S_n actually are states at time $t = 1$, instead of states at time $t = 0$.

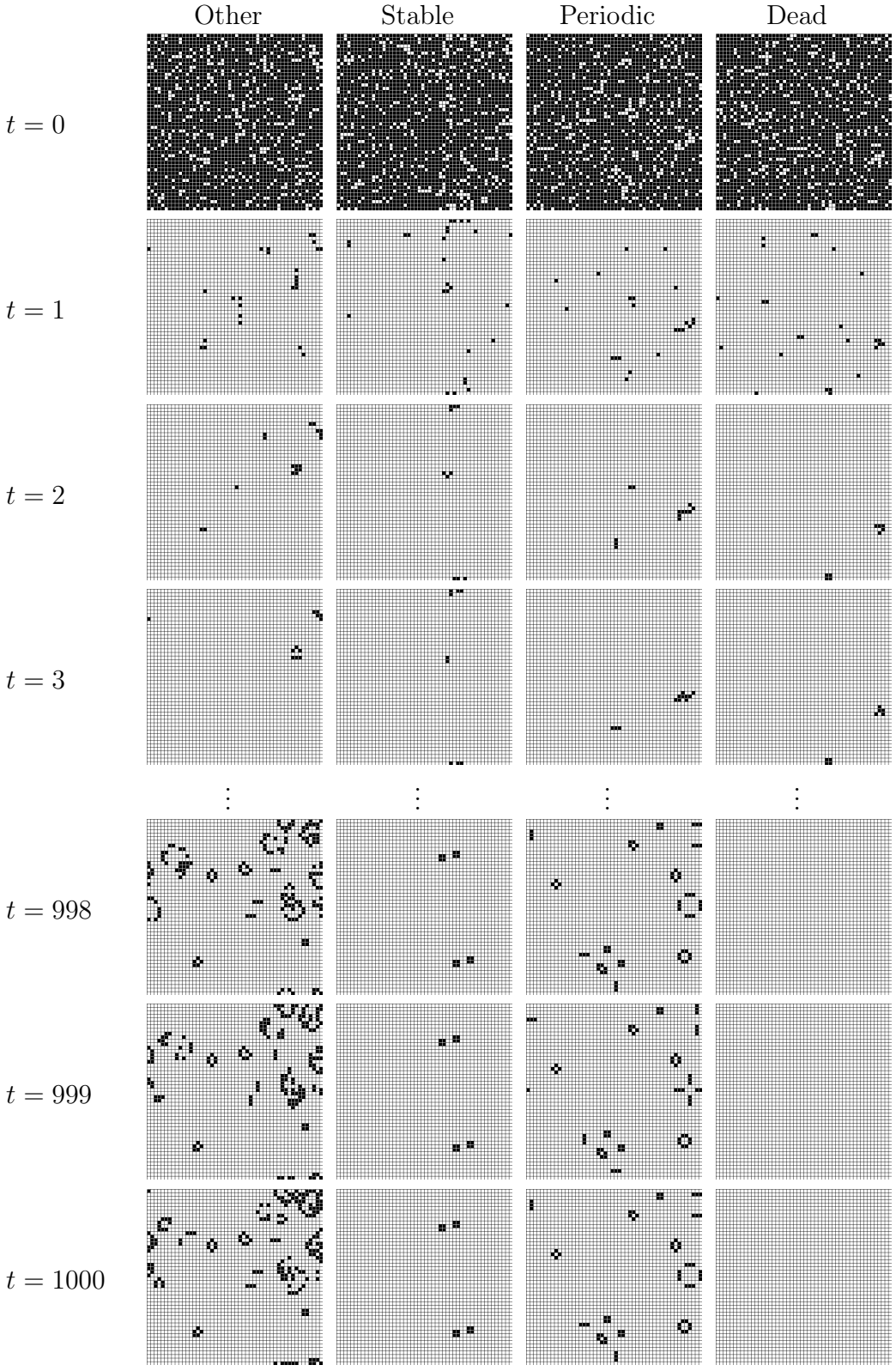


Figure 7.3 – Examples of grids: 4 first states and last 3 states, for each class.

We have split each set S_n^X into two equal parts for Learning (L_n^X) and Testing (T_n^X), and we note $L_n = \cup_{X \in \{D,S,P,O\}} L_n^X$ and $T_n = \cup_{X \in \{D,S,P,O\}} T_n^X$.

7.2.2 Representation of Game of Life initial states by histograms of frequent patterns

The goal of our experiments is to predict the outcome of a Game of Life at time $t = 1000$ given the states from time $t = 1$ to time $t = k$. We consider different values for the temporal horizon $k \in \{1, 2, 5, 10, 20\}$ used for predicting the outcome. This allows us to evaluate the interest of using spatio-temporal patterns compared to purely spatial patterns. Hence, for each state $s_i \in S_n$ (with $n \in \{20, 30, 40, 50\}$), and for each temporal horizon $k \in \{1, 2, 5, 10, 20\}$ we build a $2\mathcal{D} + t$ grid $G(s_i, k)$ which is a temporal sequence of k $2\mathcal{D}$ grids, such that each grid is build in a straightforward way from the corresponding state: grid vertices correspond to cells, and are labeled with either 0 (if the cell is dead) or 1 (if the cell is alive). All edges have the same label.

Like for the image classification application, we create holes in grids to prevent mining a huge number of uniform patterns, *i.e.*, patterns where all vertices have the same labels. Indeed, preliminary experiments showed us that the number of frequent patterns is huge because all grids contain a lot of patterns composed of dead cells only. For example, there are more than five millions of frequent patterns in the grids associated with L^O_{20} when setting the time horizon to $k = 1$ and the frequency threshold σ to 100% (*i.e.*, patterns must occur in all the grids). Hence, to reduce the number of frequent patterns, and avoid mining useless patterns only composed of dead cells, we remove every vertex corresponding to a dead cell and whose neighbors are all dead cells. When mining the resulting grids with holes, the number of frequent patterns associated with L^O_{20} (with $k = 1$ and $\sigma = 100\%$) is reduced to 24.

The second step is to mine frequent patterns from $2\mathcal{D} + t$ grids. As all edges have the same label, we use i-GriMA for mining frequent patterns: this allows to reduce the number of mined patterns and to speed-up the mining process. The mining process is done on each class separately to avoid missing relevant patterns relative to a single class. The set of all frequent patterns is the union of the result of each mining process. More formally, for each size $n \in \{20, 30, 40, 50\}$, each temporal horizon $k \in \{1, 2, 5, 10, 20\}$, and each outcome $X \in \{D, S, P, O\}$, we use i-GriMA to build the set $P^X_{n,k}$ of grid patterns that are frequent in the database of grids $D = \{G(s_i, k) : s_i \in L^X_n\}$. Then, we construct the set $P_{n,k} = \cup_{X \in \{D,S,P,O\}} P^X_{n,k}$ that contains the union of all frequent patterns mined for each class.

The last step is to build a frequency histogram for each Game of Life initial state (that may either belong to the learning set, during the training process, or to the testing set, during the classification process). More precisely, given an initial state $s \in S_n$, a temporal horizon $k \in \{1, 2, 5, 10, 20\}$, and the set

$P_{n,k}$ of frequent patterns, we build an histogram h_s which is a vector of $|P_{n,k}|$ values such that $h_s[j] = o/a$ where o is the number of occurrences of the j^th pattern $P_{n,k}[j]$ in the $2\mathcal{D}+t$ grid $G(s,k)$ associated with s, and a is the number of automorphisms of $P_{n,k}[j]$. We divide the number of occurrences of $P_{n,k}[j]$ by its number of automorphisms because there are a occurrences of $P_{n,k}[j]$ for each subset of vertices in $G(s,k)$ such that the subgrid induced by this subset is isomorphic to $P_{n,k}[j]$, as discussed in Section 6.2.2

7.2.3 Overview of the learning process

To train a classifier on a learning set L_n, the first step is to build a frequency histogram h_s for every initial state in L_n, as described in Section 7.2.2. We consider two different frequency thresholds $\sigma \in \{50\%, 100\%\}$ for the mining process: When $\sigma = 50\%$ (resp. $\sigma = 100\%$), a pattern is frequent if it is present in half of the grids (resp. all the grids) of a given class. Each mining process has been limited to 12 hours of CPU time: If it is not completed after 12 hours, we stop it and consider the subset of patterns that have been extracted within this time limit. This subset of patterns depends on the order used by GRIMA to extend patterns. In particular, we may consider two different strategies: a spatial-first strategy, where GRIMA first extends patterns with spatial edges, and a temporal-first strategy, where GRIMA first extends patterns with temporal edges. When the mining process is completed within the time limit, the final set of output patterns is the same with the two strategies. However, when the mining process is stopped at the time limit, before its end, the two strategies output different subsets of patterns: the spatial-first (resp. temporal-first) strategy outputs patterns with more spatial (resp. temporal) edges and less temporal (resp. spatial) edges. Preliminary experiments have shown us that better classification results are obtained with the temporal-first strategy. Therefore, we consider this strategy during the mining process.

In some cases, the number of frequent patterns in $P_{n,k}$ is huge (larger than one million), which causes scaling issues for the learning process. Hence, when the number of mined patterns is larger than 100000, we randomly select a subset of 100000 patterns. We note $P_{n,k}^{100000}$ this subset (when $|P_{n,k}| < 100000$, $P_{n,k}^{100000} = P_{n,k}$).

We also report accuracy results obtained when considering a smaller subset of patterns which are selected in $P_{n,k}^{100000}$ by post-processing. This post-processing selection is performed using the relevance score and the greedy selection algorithm presented in [Fernando et al. 2014] and described in Section 3.1.2. We report results obtained with 1000 and 4000 selected patterns, respectively, and we note $P_{n,k}^{1000}$ and $P_{n,k}^{4000}$ the resulting subsets of selected patterns.

Once an histogram h_s has been built for each initial state $s \in L_n$, the second step is to normalize all these histograms to obtain real values ranging between 0 and 1: for each dimension $j \in [1, |P^l_{n,k}|]$ and each initial state $s \in L_n$, $h_s[j]$ is replaced with $\frac{h_s[j] - \min_{s' \in L_n} h_{s'}[j]}{\max_{s' \in L_n} h_{s'}[j] - \min_{s' \in L_n} h_{s'}[j]}$.

Finally, these normalized histograms are used to train a binary SVM classifier κ^X for each class $X \in \{D, S, P, O\}$, the same way we did for images. We have used the SVM implementation of LibSVM [Chang et al. 2011] with the intersection kernel presented in [Odone et al. 2005]. The value of the C parameter (corresponding to the cost of the constraint violation) is optimized by 5-fold cross-validation on the learning set, for all powers of 2 ranging between 2^{-10} to 2^{10}.

7.2.4 Overview of the classification process

The accuracy of the trained classifier is evaluated on the testing set of initial states T_n. Given the set of frequent patterns $P^l_{n,k}$ mined from the learning set, we build a frequency histogram h_s for every initial state $s \in T_n$, by counting occurrences of $P^l_{n,k}$ in the $2\mathcal{D} + t$ grid associated with s. This histogram is normalized, as described in Section 7.2.3, except that if a normalized value is smaller than 0 (resp. larger than 1), then it is set to 0 (resp. 1).

This normalized histogram is given as input to each binary SVM classifier κ^X with $X \in \{D, S, P, O\}$. The predicted class corresponds to the binary classifier with maximal output probability.

7.3 Efficiency analysis

7.3.1 Mining efficiency

We cannot experimentally compare GriMA with FreqGeo, as there is no known implementation of FreqGeo. Also, we do not compare GriMA with Plagram because Plagram can only mine patterns composed of faces. If it is possible to transform each grid vertex into a four-node face, as explained in Section 5.1, experimental results reported in the previous chapter show us that Plagram does not scale well compared to GriMA.

Hence, in this section we experimentally compare GriMA with gSpan to assess the advantages of using an ad hoc algorithm on $2\mathcal{D} + t$ grids. To allow gSpan to discriminate spatial and temporal edges, we label every spatial (resp. temporal) edge with label 0 (resp. 1). However, gSpan mines general graphs and does not consider the angle information between spatial edges while GriMA does. Also, grid isomorphism does not allow rotations along spatial axes. This

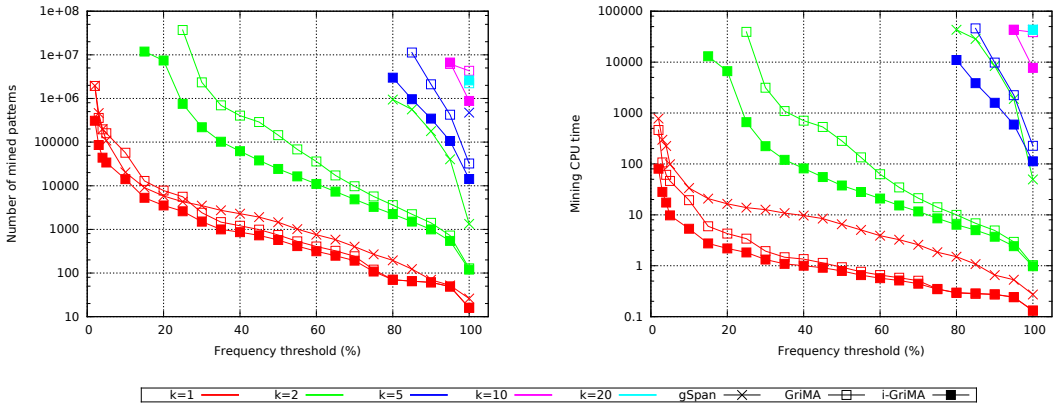


Figure 7.4 – Number of mined patterns (left) and CPU time to mine patterns (right) with respect to the frequency threshold σ for different temporal horizons $k \in \{1, 2, 5, 10, 20\}$ and three different algorithms (gSpan, GriMA, and i-GriMA). The size of the mined grids is $20 \times 20 \times k$, and there are 100 grids.

implies that a grid G_1 composed of two nodes with different labels, and a single temporal edge between these two nodes, is not grid-isomorphic to another grid G_2 obtained by swapping the two node labels. If gSpan were able to handle directed graphs, we could prevent gSpan from considering that G_1 and G_2 are isomorphic by directing temporal edges. However, gSpan considers non directed graphs and, therefore, it considers that G_1 and G_2 are isomorphic. As a consequence, for every grid G, the number of different patterns that are subgrid-isomorphic to G is greater than or equal to the number of different patterns that are subgraph-isomorphic to G. As a counterpart, given a pattern P, the number of subgrids of G that are grid-isomorphic to P is lower than or equal to the number of subgraphs of G that are isomorphic to P.

Figure 7.4 compares the number of mined patterns and the CPU time of the mining process of gSpan, GriMA and i-GriMA, for different temporal horizons $k \in \{1, 2, 5, 10, 20\}$, when changing the value of the frequency threshold σ from 2% to 100%. In all cases, the mining process has been performed on 100 grids built from the 100 first initial states of L_{20}^{O}: These grids have $20 \times 20 \times k$ nodes. The time limit has been set to 12 hours. When the mining process is not completed within this time limit, we do not display the number of mined patterns.

When $k = 1$, the number of mined patterns is comparable for gSpan, GriMA and i-GriMA whereas the time is up to 10 times as low for GriMA and i-GriMA. When k increases, these differences increase. i-GriMA mines less patterns and has the lowest computation times in all cases. All three algorithms cannot reach the lowest support threshold when $k > 1$. For $k = 2$

(resp. $k = 5$), gSpan cannot complete the mining process within the time limit when the frequency threshold σ is lower than 80 (resp. 100). For $k \in \{10, 20\}$, gSpan cannot complete the mining process within the time limit for any frequency threshold. GriMA scales better than gSpan, and it is able to complete the mining process within the time limit for all frequency thresholds when $k = 1$, and down to 25% (resp. 85% and 100%) when $k = 2$ (resp. $k = 5$ and $k = 10$).

Finally, i-GriMA mines less patterns than GriMA and scales even better. It is the only algorithm able to complete the mining process within the time limit when $k = 20$ (for $\sigma = 100\%$).

This shows that our temporal grid mining algorithm is efficient and can tackle real life problems that cannot be tackled by a general graph mining algorithm.

7.3.2 Counting the number of occurrences in grids

We also conduct experiments to study scale-up properties of the algorithm that counts the number of occurrences of a pattern in a grid, as described in Section 5.7. We compare this algorithm, dedicated to $2\mathcal{D}+t$ grids, with LAD [Solnon 2010]. LAD is a state-of-the-art algorithm for solving the subgraph isomorphism problem, and it may be used to count the number of occurrences of a pattern graph in a target graph. Grids are transformed into graphs in a straightforward way. To allow LAD to discriminate spatial and temporal edges, we label every spatial (resp. temporal) edge with label 0 (resp. 1). Also temporal edges are directed, to prevent LAD from considering that a graph composed of two nodes with different labels, and a single temporal edge between these two nodes, is isomorphic to another graph obtained by swapping the two node labels. However, LAD does not consider spatial edge angles, and two grids that are not grid-isomorphic may be graph-isomorphic if they only differ with respect to an angle between two spatial edges. As a consequence, the number of occurrences found by LAD is always larger than the number of occurrences found by our algorithm.

To evaluate scale-up properties, for each grid size $n \in \{20, 30, 40, 50\}$, we have selected in $P_{n,20}$ a subset of 100 different patterns that have 80 nodes, denoted P_n^{80}. Then, for each size $i \in [2, 79]$, we have built a set P_n^i of 100 patterns that have i nodes by randomly removing one node in each pattern of P_n^{i+1}. Finally, for each target grid size $n \in \{20, 30, 40, 50\}$, each pattern size $i \in [2, 80]$, and each pattern $p \in P_n^i$, we measure the CPU time spent by our algorithm and LAD, respectively, for counting all occurrences of p in one $n \times n \times 20$ grid.

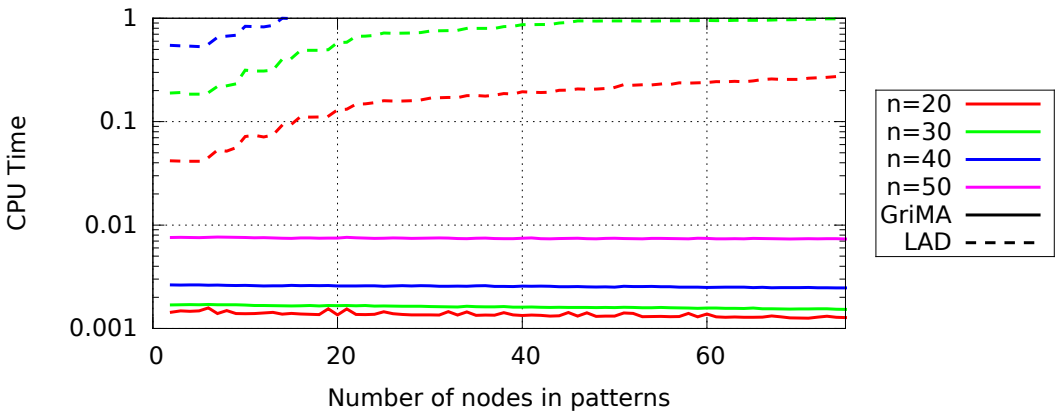


Figure 7.5 – Time (in seconds) spent by GriMA and LAD to enumerate all occurrences of a pattern in a grid of size $n \times n \times 20$ with respect to the number of nodes in the pattern (average on 100 patterns), for $n \in \{20, 30, 40, 50\}$. Time is limited to 1 second.

Figure 7.5 displays these CPU times. It shows us that GriMA is much faster than LAD, and never spends more than 0.01 second. Furthermore, the time spent by GriMA does not increase when the size of the pattern increases. The theoretical complexity $\mathcal{O}(e_1 e_2)$, where e_1 and e_2 are the number of edges in the target and pattern grids, respectively, is an upper bound that considers a worst case where there is an occurrence of the pattern for each possible starting edge in the target. In practice, we stop the traversal of the pattern code as soon as a difference occurs, and this usually happens very soon in the traversal (for example, at the fourth edge, on average, when the pattern has 70 nodes). As a comparison, the time spent by LAD increases when the pattern increases, and the time limit of 1 second is reached when the pattern has more than 15 nodes when $n = 40$, and for all patterns when $n = 50$.

7.4 Accuracy results

We report accuracy results (*i.e.*, the percentage of states for which the forecasted outcome is equal to the true outcome) in Table 7.1. When increasing the temporal horizon k (*i.e.*, the temporal size of the mined grids) from 1 to 20, accuracy results are strongly improved: from about 43% when $k = 1$ to 68% when $k = 20$ on the smallest grids of size 20×20, and from 47% when $k = 1$ to 75% when $k = 20$ on the largest grids of size 50×50. This shows the relevance of GriMA when mining spatio-temporal patterns, compared to GriMA when mining spatial $2\mathcal{D}$ grids (*i.e.*, when $k = 1$).

The number of mined patterns $|P_{n,k}|$ is smaller when the frequency threshold $\sigma = 100\%$ than when it is 50%. For small temporal horizons $k \in \{1, 2\}$, the

Table 7.1 – Accuracy results for the classification of states in T_n. For each size $n \in \{20, 30, 40, 50\}$, line $|P_{n,k}|$ reports the number of frequent patterns, and the cell is colored in red if the 12 hour time-out has been reached and green otherwise. Lines *All* , *1000* , and *4000* report accuracy results with vectors of size $min(|P_{n,k}|, 100000)$, 1000, and 4000, respectively (if $|P_{n,k}|$ is lower than 1000 or 4000, results are not given for vectors of size 1000 or 4000). Each line gives results for $k \in \{1, 2, 5, 10, 20\}$, and with $\sigma \in \{50\%, 100\%\}$.

		k=1		k=2		k=5		k=10		k=20	
n		50%	100%	50%	100%	σ=50%	100%	σ=50%	σ=100%	σ=50%	σ=100%
20	$\|P_{n,k}\|$	757	24	37055	90	1110172	3938	929039	195974	766219	555785
	All	43.40	34.30	48.80	43.85	56.70	57.65	62.55	63.70	67.20	67.65
	1000			47.90		57.40	58.30	62.30	62.50	63.55	67.70
	4000			48.85		57.90		63.55	64.45	63.60	**68.05**
30	$\|P_{n,k}\|$	696	23	35529	93	1291083	4701	1044726	295785	778995	591969
	All	44.20	33.95	54.20	45.75	62.70	63.45	65.55	67.05	69.25	71.50
	1000			51.75		62.30	62.90	65.50	68.75	67.65	**72.00**
	4000			53.25		62.90	63.55	67.50	69.75	67.40	71.30
40	$\|P_{n,k}\|$	725	27	37674	95	1320971	5563	1114497	357265	834531	618569
	All	45.05	32.45	57.45	48.60	63.10	65.35	69.25	70.05	72.40	71.90
	1000			55.85		65.50	64.45	68.70	70.15	71.75	73.50
	4000			57.25		66.30	65.40	69.25	71.00	72.25	**74.15**
50	$\|P_{n,k}\|$	760	29	42609	100	1325108	6964	1026214	554505	853811	639770
	All	46.95	36.05	58.05	48.70	65.45	67.05	70.15	71.15	74.00	73.95
	1000			54.95		66.30	65.95	69.90	71.25	72.10	74.35
	4000			56.80		67.80	66.55	71.10	72.40	72.65	**74.90**

number of mined patterns is not large enough (smaller than 29 for $k = 1$ and than 100 for $k = 2$). In this case, the classification results obtained with $\sigma = 100\%$ are much lower than those obtained with $\sigma = 50\%$. However, for larger time horizons $k \in \{5, 10, 20\}$, the number of mined patterns becomes large enough for $\sigma = 100\%$ while it becomes so large for $\sigma = 50\%$ that the mining process is never completed before reaching the timeout of 12h. In this case, accuracy results obtained with $\sigma = 100\%$ are often slightly better than those obtained with $\sigma = 50\%$.

Finally, let us compare the results obtained when all patterns of $P_{n,k}$ are used for the classification (or 100000 patterns if $|P_{n,k}| > 100000|$) with the results obtained when we only use the patterns selected by post-processing (1000 or 4000 patterns): The difference is usually rather small, and in most cases selecting patterns by post-processing improves results, and better results are obtained with 4000 selected patterns than with 1000. However, the post-processing improves the efficiency of the counting step: Even if counting all occurrences of a pattern in a grid is done in less than 0.01 second (see Figure 7.5), counting all occurrences of 100000 patterns becomes time-consuming, whereas this is done in a few seconds when the number of patterns is 4000.

Confusion matrices. Tables 7.2 displays confusion matrices for each size $n \in \{20, 30, 40, 50\}$ with a temporal horizon $k = 20$, when we use 4000 patterns selected by post-processing. These confusion matrices are quite different from

Table 7.2 – Confusion matrices when $k = 20$, $\sigma = 100\%$, and when using 4000 patterns.

(a) Confusion matrix when $n = 20$.

	O	S	P	D
O	86.0	2.0	1.4	5.6
S	2.0	73.0	19.8	11.0
P	5.4	14.0	58.4	28.6
D	6.6	11.0	20.4	54.8

(b) Confusion matrix when $n = 30$.

	O	S	P	D
O	71.4	3.6	2.8	4.4
S	3.4	67.6	18.6	9.4
P	4.2	16.0	68.0	8.0
D	21.0	12.8	10.6	78.2

(c) Confusion matrix when $n = 40$.

	O	S	P	D
O	63.8	5.2	4.0	5.8
S	3.8	69.0	11.4	6.2
P	10.4	16.2	78.4	2.6
D	22.0	9.6	6.2	85.4

(d) Confusion matrix when $n = 50$.

	O	S	P	D
O	59.4	6.8	6.8	7.0
S	6.0	75.2	10.8	3.8
P	12.2	8.8	78.4	2.6
D	22.4	9.2	4.0	86.6

Table 7.3 – Average and Maximum (in parenthesis) depth (D) and number of cells (C) for all patterns of $P_{n,k}$ (All) or only those selected with post processing (1000) when mining with a frequency threshold $\sigma = 50\%$.

k		1		2		5		10		20	
n		All	1000	All	1000	All	1000	All	1000	All	1000
20	D	0.0(0)	0.0(0)	0.9(1)	1.0(1)	2.1(4)	2.3(4)	5.7(8)	4.6(8)	10.3(17)	7.6(17)
	C	6.5(11)	6.5(11)	8.5(16)	8.5(15)	14.2(29)	12.2(25)	22.6(46)	13.8(33)	31.8(83)	14.1(39)
30	D	0.0(0)	0.0(0)	0.9(1)	0.9(1)	2.2(4)	2.2(4)	5.7(9)	5.5(8)	11.1(17)	8.1(17)
	C	6.4(11)	6.4(11)	8.3(16)	8.3(15)	13.8(28)	11.7(23)	23.8(46)	20.3(34)	34.0(78)	18.2(41)
40	D	0.0(0)	0.0(0)	0.9(1)	1.0(1)	2.2(4)	2.3(4)	5.9(9)	6.0(9)	11.8(18)	7.7(17)
	C	6.5(11)	6.5(11)	8.5(17)	8.9(16)	14.2(29)	12.1(23)	25.7(46)	20.5(34)	41.0(83)	33.8(43)
50	D	0.0(0)	0.0(0)	0.9(1)	0.9(1)	2.2(4)	2.3(4)	6.2(9)	6.1(9)	13.8(19)	7.8(15)
	C	6.5(11)	6.5(11)	8.6(17)	8.7(17)	14.2(29)	12.9(24)	25.9(46)	25.8(35)	44.1(83)	22.8(50)

a value of n to another. For example, when $n = 20$, classification errors mainly come from classes *Dead* and *Periodic*, whereas when $n = 50$ these two classes have the best results. Hence, we cannot really draw conclusions from these matrices.

Pattern statistics. Table 7.3 reports some statistics about the mined patterns (all patterns in $P_{n,k}$, or the 1000 ones selected by post-processing). We report the average and maximum (in parenthesis) number of nodes of the mined patterns as well as their depth, where the depth of a pattern P is defined by $\max_{u \in L} t_u - \min_{u \in L} t_u$. The number of nodes and the depth of the patterns logically increase with k. These values tend to be smaller when considering the subset of 1000 patterns selected by post-processing. This may come from the fact that deep patterns are not diverse enough to be selected by the post-processing step which in turn suggests that, when the timeout is reached, the diversity of the mined pattern is not high enough. To further increase this diversity, stochastic search methods such as Monte-Carlo Tree Search [Bosc et al.

2016] could be integrated in our algorithm.

7.5 Discussion

In this chapter, we have described a second application of GriMA to mine frequent patterns in $2\mathcal{D}+t$ grids that represent Conway's Game of Life. Our first goal was to evaluate scale-up properties of GriMA, and to show that it may be used to mine frequent patterns within real-world $2\mathcal{D}+t$ data. Like for $2\mathcal{D}$, experimental results have shown us that GriMA scales better that gSpan and that its node induced variant, i-GriMA, scales even better. We also conduct series of experiments to evaluate scale-up properties of our algorithm to count occurrences of patterns in $2\mathcal{D}+t$ grids. These experiments have shown us that our algorithm outperforms LAD as it benefits from the grid structure.

Our second goal was to evaluate the interest of using spatio-temporal frequent grid patterns to predict the outcome of a well know cellular automata, the Conway's Game of Life. Experimental results have shown us that we can predict the outcome, among four different possible outcomes, at time $t = 1000$ given the k first states, with a classification rate close to 75%. The reliability of the prediction increases with the temporal horizon k, thus showing the interest of mining spatio-temporal patterns instead of purely spatial patterns.

GriMA is a complete approach which exhaustively mines all frequent patterns. When the frequency threshold is low, the number of frequent patterns may become huge, so that the mining process cannot be completed within a reasonable amount of time. Furthermore, some frequent patterns may not be relevant for a classification purpose, and experimental results have shown us that we may obtain similar results (if not better) when selecting a small subset of relevant patterns. Hence, we plan to explore the interest of using heuristic algorithms that can directly mine a small number of relevant patterns, by using Monte-Carlo Tree Search as proposed in [Bosc et al. 2016], for example.

Besides, GriMA should to be evaluated on other, even more challenging, applications such as, for example, action recognition in videos represented by $2\mathcal{D}+t$ grids, or the extraction of winning spatio-temporal patterns in board games like chess or go.

Finally, we plan to extend GriMA to mine other kinds of regular structures such as, for example, $3\mathcal{D}+t$ grids, to search for spatio-temporal patterns in dynamic $3\mathcal{D}$ objects. Other types of regular structures may be studied, like honeycomb, or any other regular polygon face-based structure.

Chapter 8

Conclusion

Graph mining aims at extracting patterns from graph databases. If this problem hardly scales in the general case, there exist efficient algorithms when considering special kinds of graphs, for which there exist polynomial-time algorithms for solving (sub)graph isomorphism problems such as, for example, plane graphs and geometric graphs. In this thesis, we consider the graph mining problem when graphs have a grid structure.

Summary of our contributions

We have defined grids as a special case of geometric graphs. Like for geometric graphs, grid vertices have coordinates: $2\mathcal{D}$ coordinates for $2\mathcal{D}$ grids, and $2\mathcal{D}+t$ coordinates for $2\mathcal{D}+t$ grids. However, these coordinates are constrained to take integer values, and edges can only connect vertices that are neighbors on the grids (*i.e.*, the distance between two vertices connected by an edge must be equal to one). These regular grids are used to model objects in many real-life applications such as, for example, game boards, cellular automata, or images. Furthermore, the evolution through time of these objects may be modeled with $2\mathcal{D}+t$ grids.

We have defined grid isomorphism, to decide whether two grids are equivalent, and subgrid isomorphism, to decide whether a pattern grid occurs in a target grid. These isomorphisms are invariant to translations and to some rotations (those around the temporal axis): two patterns that have the same structure but have been translated or rotated around the temporal axis are considered as equivalent.

We have introduced new algorithms dedicated to grid mining:

- GriMA, an efficient algorithm for mining frequent patterns in a database of grids. This algorithm performs a depth-first exploration of a search space of canonical codes, like many other graph mining algorithms such as gSpan for general graphs, Plagram for plane graphs, and FreqGeo for geometric graphs. This canonical code is used to uniquely characterize

isomorphic patterns, and it is computed in polynomial time by exploiting the grid topology. Hence, the time complexity of GriMA is $\mathcal{O}(knp^3)$ per frequent and canonical pattern P, where k is the number of grids in the database, n the number of edges of the largest grid in the database, and p the number of edges in the pattern P. This is a significant improvement over gSpan (which has an exponential time complexity per pattern), FreqGeo (which has a complexity of $\mathcal{O}(k^2n^4 \cdot \ln n)$ per pattern), and Plagram (which has a complexity of $\mathcal{O}(kn^2p^3)$ per pattern). GriMA has been implemented with sparse sets, so that occurrence lists are restored in constant time when backtracking during the depth-first exploration of the canonical code search space.

- i-GriMA, a variant of GriMA dedicated to the case where all grids of the database are complete (*i.e.*, if the distance between two vertices is one, then they are connected by an edge), and all edges have the same label. In this case, if a pattern P is frequent, then every pattern obtained by adding edges to P is also frequent. Hence, i-GriMA only mines complete frequent patterns: this relaxed closure allows us to significantly reduce the number of mined patterns and speed-up the mining process.

- an efficient algorithm for enumerating all occurrences of a pattern P in a grid G in $\mathcal{O}(np)$ where n and p are the number of edges in G and P, respectively.

We have evaluated scale-up properties of our algorithms, and the interest of exploiting grid structures for characterizing objects, on two applications: image classification, and cellular automata analysis.

For the image classification application, we have introduced a new way of representing images by means of frequency histograms of $2\mathcal{D}$ grid patterns, called Bag-of-Grids. We have shown that GriMA and i-GriMA are more efficient than other existing graph mining algorithms. In particular, i-GriMA is able to find all frequent patterns in less than one hour in a database composed of 80 $2\mathcal{D}$ grids that have $18,000$ vertices on average when the frequency threshold σ is set to 65% whereas gSpan and Plagram cannot complete the mining process within one hour when σ is smaller than or equal to 85%. Also, we have shown the interest of exploiting a grid structure to represent images, compared to a classical unstructured representation based on bags of visual words. However, as discussed in Section 6.6, these results are not competitive with state-of-the-art approaches for image classification, that are usually based on convolutional neural networks.

For the cellular automata analysis application, we have used $2\mathcal{D}+t$ grids to model the temporal evolution of well-known cellular automata: Conway's Game of life. Like for the image classification application, we have studied scale-up properties of our algorithms, and we have shown that our mining algorithms scale better that GSPAN on $2\mathcal{D}+t$ grids. Also, we have shown the interest of using spatio-temporal frequent patterns to predict the outcome of a Game-of-Life CA, given its k first initial states. In particular, we have shown that the percentage of well-classified outcomes reaches 58% (resp. 67%, 72%, and 75%) when $k=2$ (resp. 5, 10, and 20), when the initial states are composed of 50×50 cells, and when considering four different outcomes.

These two different applications showed that grid mining may be useful to retrieve structural information and may be applied on real world datasets.

Future work

Our grid mining algorithm is complete and outputs all frequent patterns. In many cases, the number of frequent patterns is huge and, therefore, the mining process cannot be completed within a reasonable amount of time. This motivated us to introduce I-GRIMA, which reduces the number of patterns by only considering complete grid patterns, and our experiments (on images and cellular automata) have shown that I-GRIMA actually scales better than GRIMA. However, in many cases the number of frequent patterns found by I-GRIMA is huge (*e.g.*, larger than one million when mining game-of-life $2\mathcal{D}+t$ grids). In this case, we have shown that we may select a small number of relevant patterns, using a post-processing selection procedure, and that we obtain similar classification results (if not better) when only using these selected patterns, instead of all mined patterns. Hence, we plan to explore approaches for reducing the number of mined patterns. A first possibility to achieve this goal is to use an incomplete heuristic algorithm, able to randomly sample a subset of relevant patterns within the set of all frequent patterns, by using Monte-Carlo Tree Search approaches as proposed in [Bosc et al. 2016], for example. Another possibility could be to incrementally add new constraints during the mining process, to forbid mining patterns similar to already mined patterns: pushing these constraints during the mining process (instead of using them *a posteriori* to select relevant patterns), could allow us to find more diversified patterns quicker.

Our grid mining algorithms have been designed for tackling $2\mathcal{D}+t$ grids such that each node has at most 6 neighbors (4 spatial neighbors, and 2 temporal neighbors, all at distance one from the node). We could easily extend it to tackle other kinds of regular grids. First, we could consider other spatial neighborhoods. For example, each node could have k spatial neighbors, evenly

distributed around the node. Second, we may also consider larger dimensions such as, for example, $3\mathcal{D} + t$ grids. These grids could be used to model the evolution through time of $3\mathcal{D}$ objects modeled with voxels. The extension of GriMA and i-GriMA to these new kinds of grids is rather straightforward: we mainly have to change the angle information in edge codes.

From the applicative point of view, if the two applications studied in this thesis have shown that frequent grid patterns may be used to characterize images and CAs, there is still room for improvements. In particular, as discussed in Section 6.6, image classification results obtained with BoGs are far from state-of-the-art results. Hence, we plan to investigate the interest of using other kinds of labels instead of visual words computed by clustering SIFT descriptors. In particular, we could use labels learned with convolutional neural networks.

CAs are widely used to model ecosystems. In this case, a key point for preserving biodiversity is to understand the mechanisms that lead to the loss of some species that should be preserved or, conversely, the spread of invasive species that should be eradicated. Hence, it could be interesting to study the interest of using our grid mining algorithms to identify spatio-temporal patterns that lead to these outcomes.

Finally, many games are played on boards that have regular grid structures such as, for example, chess, draughts, or go. In this case, a game may be modeled by a $2\mathcal{D}+t$ grid in a straightforward way. Hence, it could be interesting to evaluate the interest of using GriMA to identify winning spatio-temporal patterns.

Bibliography

Acosta-Mendoza, N., A. G. Alonso, and J. E. Medina-Pagola (2012). "Frequent approximate subgraphs as features for graph-based image classification." *Knowl.-Based Syst.* 27, pp. 381–392 (cit. on p. 45).

Agrawal, R. and R. Srikant (1994). "Fast Algorithms for Mining Association Rules in Large Databases." In: *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*. Ed. by J. B. Bocca, M. Jarke, and C. Zaniolo. Morgan Kaufmann, pp. 487–499 (cit. on pp. 24, 34).

Arimura, H., T. Uno, and S. Shimozono (2007). "Time and Space Efficient Discovery of Maximal Geometric Graphs." In: *Discovery Science, 10th International Conference, DS 2007, Sendai, Japan, October 1-4, 2007, Proceedings*. Ed. by V. Corruble, M. Takeda, and E. Suzuki. Vol. 4755. Lecture Notes in Computer Science. Springer, pp. 42–55 (cit. on pp. 1, 5, 14, 15, 34, 36, 40, 73).

Audemard, G., C. Lecoutre, M. S. Modeliar, G. Goncalves, and D. C. Porumbel (2014). "Scoring-Based Neighborhood Dominance for the Subgraph Isomorphism Problem." In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by B. O'Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, pp. 125–141 (cit. on p. 13).

Babai, L. (2016). "Graph isomorphism in quasipolynomial time [extended abstract]." In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. by D. Wichs and Y. Mansour. ACM, pp. 684–697 (cit. on p. 12).

Bendimerad, A., M. Plantevit, and C. Robardet (2017). "Mining exceptional closed patterns in attributed graphs." *Knowledge and Information Systems* (cit. on p. 27).

Bie, T. D. (2011). "Maximum entropy models and subjective interestingness: an application to tiles in binary databases." *Data Min. Knowl. Discov.* 23(3), pp. 407–446 (cit. on p. 28).

Bie, T. D., K. Kontonasios, and E. Spyropoulou (2010). "A framework for mining interesting pattern sets." *SIGKDD Explorations*, 12(2), pp. 92–100 (cit. on p. 28).

Borgelt, C. and M. R. Berthold (2002). "Mining Molecular Fragments: Finding Relevant Substructures of Molecules." In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pp. 51–58 (cit. on pp. 27, 41).

Bosc, G., C. Raïssi, J. Boulicaut, and M. Kaytoue (2016). "Any-time Diverse Subgroup Discovery with Monte Carlo Tree Search." *CoRR*, abs/1609.08827. arXiv: 1609.08827 (cit. on pp. 110, 111, 115).

Breckling, B., G. Pe'er, and Y. G. Matsinos (2011). "Cellular Automata in Ecological Modelling." In: *Modelling Complex Ecological Dynamics: An Introduction into Ecological Modelling for Students, Teachers & Scientists*. Springer Berlin Heidelberg, pp. 105–117 (cit. on p. 98).

Briggs, P. and L. Torczon (1993). "An Efficient Representation for Sparse Sets." *LOPLAS*, 2(1-4), pp. 59–69 (cit. on p. 69).

Bringmann, B. and S. Nijssen (2008). "What Is Frequent in a Single Graph?" In: *Advances in Knowledge Discovery and Data Mining, 12th Pacific-Asia Conference, PAKDD 2008, Osaka, Japan, May 20-23, 2008 Proceedings*, pp. 858–863 (cit. on p. 26).

Buzmakov, A., S. O. Kuznetsov, and A. Napoli (2015). "Fast Generation of Best Interval Patterns for Nonmonotonic Constraints." In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II*. Ed. by A. Appice, P. P. Rodrigues, V. S. Costa, J. Gama, A. Jorge, and C. Soares. Vol. 9285. Lecture Notes in Computer Science. Springer, pp. 157–172 (cit. on p. 28).

Cerf, L., J. Besson, C. Robardet, and J.-F. Boulicaut (2008). "Data-Peeler: Constraint-based Closed Pattern Mining in n-ary Relations." In: *SIAM International Conference on Data Mining SDM'08*. Atlanta, United States, pp. 37–48 (cit. on p. 28).

Chang, C. and C. Lin (2011). "LIBSVM: A library for support vector machines." *ACM TIST*, 2(3), 27:1–27:27 (cit. on pp. 89, 105).

Chatfield, K., V. S. Lempitsky, A. Vedaldi, and A. Zisserman (2011). "The devil is in the details: an evaluation of recent feature encoding methods." In: *British Machine Vision Conference, BMVC 2011, Dundee, UK, August*

29 - September 2, 2011. Proceedings. Ed. by J. Hoey, S. J. McKenna, and E. Trucco. BMVA Press, pp. 1–12 (cit. on p. 79).

Chatfield, K., K. Simonyan, A. Vedaldi, and A. Zisserman (2014). "Return of the Devil in the Details: Delving Deep into Convolutional Nets." In: *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*. Ed. by M. F. Valstar, A. P. French, and T. P. Pridmore. BMVA Press (cit. on p. 79).

Chen, C., X. Yan, F. Zhu, and J. Han (2007). "gApprox: Mining Frequent Approximate Patterns from a Massive Network." In: *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*. IEEE Computer Society, pp. 445–450 (cit. on p. 44).

Cheng, H., X. Yan, J. Han, and C. Hsu (2007). "Discriminative Frequent Pattern Analysis for Effective Classification." In: *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*. Ed. by R. Chirkova, A. Dogac, M. T. Özsu, and T. K. Sellis. IEEE Computer Society, pp. 716–725 (cit. on pp. 28, 29).

Chirkova, R., A. Dogac, M. T. Özsu, and T. K. Sellis, eds. (2007). *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*. IEEE Computer Society.

Conway, J. (1970). "The game of life." *Scientific American*, 223(4), p. 4 (cit. on p. 98).

Cook, D. and L. Holder (2006). *Mining Graph Data*. J. Wiley & Sons (cit. on pp. 1, 5).

Cordella, L. P., P. Foggia, C. Sansone, and M. Vento (2004). "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs." *IEEE Trans. Pattern Anal. Mach. Intell.* 26(10), pp. 1367–1372 (cit. on p. 13).

Cordella, L. P., P. Foggia, C. Sansone, and M. Vento (2001). "An improved algorithm for matching large graphs." In: *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pp. 149–159 (cit. on p. 12).

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press (cit. on p. 18).

Csurka, G., C. Dance, L. Fan, J. Willamowski, and C. Bray (2004). "Visual categorization with bags of keypoints." In: *Workshop on statistical learning in computer vision, ECCV*. Vol. 1. 1-22. Prague, pp. 1–2 (cit. on p. 79).

Elseidy, M., E. Abdelhamid, S. Skiadopoulos, and P. Kalnis (2014). "GRAMI: Frequent Subgraph and Pattern Mining in a Single Large Graph." *PVLDB*, 7(7), pp. 517–528 (cit. on p. 43).

Fayyad, U. M. and R. Uthurusamy, eds. (1994). *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. Technical Report WS-94-03*. AAAI Press.

Fayyad, U. M., G. Piatetsky-Shapiro, and P. Smyth (1996). "From Data Mining to Knowledge Discovery: An Overview." In: *Advances in Knowledge Discovery and Data Mining*, pp. 1–34 (cit. on p. 24).

Fernando, B., É. Fromont, and T. Tuytelaars (2014). "Mining Mid-level Features for Image Classification." *International Journal of Computer Vision*, 108(3), pp. 186–203 (cit. on pp. 28–30, 80, 104).

Flores-Garrido, M., J. A. Carrasco-Ochoa, and J. F. Martínez Trinidad (2015). "AGraP: an algorithm for mining frequent patterns in a single graph using inexact matching." *Knowl. Inf. Syst.* 44(2), pp. 385–406 (cit. on p. 45).

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman (cit. on p. 13).

Geng, L. and H. J. Hamilton (2006). "Interestingness measures for data mining: A survey." *ACM Comput. Surv.* 38(3), p. 9 (cit. on p. 25).

Gosselin, S., G. Damiand, and C. Solnon (2011). "Frequent Submap Discovery." In: *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*. Ed. by R. Giancarlo and G. Manzini. Vol. 6661. Lecture Notes in Computer Science. Springer, pp. 429–440 (cit. on p. 42).

Gudes, E., S. E. Shimony, and N. Vanetik (2006). "Discovering Frequent Graph Patterns Using Disjoint Paths." *IEEE Trans. Knowl. Data Eng.* 18(11), pp. 1441–1456 (cit. on p. 26).

Higuera, C. de la, J. Janodet, É. Samuel, G. Damiand, and C. Solnon (2013). "Polynomial algorithms for open plane graph and subgraph isomorphisms." *Theor. Comput. Sci.* 498, pp. 76–99 (cit. on pp. 16–18).

Hogeweg, P. (1988). "Cellular Automata As a Paradigm for Ecological Modeling." *Appl. Math. Comput.* 27(1), pp. 81–100 (cit. on p. 5).

Holder, L. B., D. J. Cook, and S. Djoko (1994). "Substucture Discovery in the SUBDUE System." In: *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. Technical Report WS-94-03*. Ed. by U. M. Fayyad and R. Uthurusamy. AAAI Press, pp. 169–180 (cit. on p. 43).

Horváth, T., J. Ramon, and S. Wrobel (2010). "Frequent subgraph mining in outerplanar graphs." *Data Min. Knowl. Discov.* 21(3), pp. 472–508 (cit. on pp. 5, 13).

Huan, J., W. Wang, and J. Prins (2003). "Efficient Mining of Frequent Subgraphs in the Presence of Isomorphism." In: *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), 19-22 December 2003, Melbourne, Florida, USA*. IEEE Computer Society, pp. 549–552 (cit. on p. 35).

Huan, J., W. Wang, J. Prins, and J. Yang (2004). "SPIN: mining maximal frequent subgraphs from graph databases." In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. Ed. by W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel. ACM, pp. 581–586 (cit. on pp. 27, 41).

Huang, G., Z. Liu, L. van der Maaten, and K. Q. Weinberger (2017). "Densely Connected Convolutional Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 2261–2269 (cit. on p. 94).

Inokuchi, A., T. Washio, and H. Motoda (2000). "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data." In: *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*. Ed. by D. A. Zighed, H. J. Komorowski, and J. M. Zytkow. Vol. 1910. Lecture Notes in Computer Science. Springer, pp. 13–23 (cit. on pp. 31, 34).

Jia, Y., J. Zhang, and J. Huan (2011). "An efficient graph-mining method for complicated and noisy data with real-world applications." *Knowl. Inf. Syst.* 28(2), pp. 423–447 (cit. on pp. 44, 45).

Jiang, C., F. Coenen, and M. Zito (2013). "A survey of frequent subgraph mining algorithms." *Knowledge Eng. Review*, 28(1), pp. 75–105 (cit. on pp. 1, 5, 43).

Kessl, R., N. Talukder, P. Anchuri, and M. J. Zaki (2014). "Parallel Graph Mining with GPUs." In: *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems,*

Programming Models and Applications, BigMine 2014, New York City, USA, August 24, 2014. Ed. by W. Fan, A. Bifet, Q. Yang, and P. S. Yu. Vol. 36. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1–16 (cit. on p. 43).

Kim, W., R. Kohavi, J. Gehrke, and W. DuMouchel, eds. (2004). *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. ACM.

Kotthoff, L., C. McCreesh, and C. Solnon (2016). "Portfolios of Subgraph Isomorphism Algorithms." In: *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*. Ed. by P. Festa, M. Sellmann, and J. Vanschoren. Vol. 10079. Lecture Notes in Computer Science. Springer, pp. 107–122 (cit. on p. 13).

Kuramochi, M. and G. Karypis (2001). "Frequent Subgraph Discovery." In: *Proceedings of the 2001 IEEE International Conference on Data Mining, 29 November - 2 December 2001, San Jose, California, USA*. Ed. by N. Cercone, T. Y. Lin, and X. Wu. IEEE Computer Society, pp. 313–320 (cit. on pp. 34, 35).

— (2002). "Discovering Frequent Geometric Subgraphs." In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pp. 258–265 (cit. on p. 14).

— (2004). "Finding Frequent Patterns in a Large Sparse Graph." In: *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*. Ed. by M. W. Berry, U. Dayal, C. Kamath, and D. B. Skillicorn. SIAM, pp. 345–356 (cit. on pp. 42, 43).

— (2005). "Finding Frequent Patterns in a Large Sparse Graph." *Data Min. Knowl. Discov.* 11(3), pp. 243–271 (cit. on p. 42).

— (2007). "Discovering frequent geometric subgraphs." *Inf. Syst.* 32(8), pp. 1101–1120 (cit. on pp. 14, 41).

Larrosa, J. and G. Valiente (2002). "Constraint Satisfaction Algorithms for Graph Pattern Matching." *Mathematical Structures in Computer Science*, 12(4), pp. 403–422 (cit. on p. 13).

Lazebnik, S., C. Schmid, and J. Ponce (2006). "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories." In: *2006*

IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA. IEEE Computer Society, pp. 2169–2178 (cit. on pp. 85, 86, 94).

Leeuwen, M. van, T. D. Bie, E. Spyropoulou, and C. Mesnage (2016). "Subjective interestingness of subgraph patterns." *Machine Learning*, 105(1), pp. 41–75 (cit. on p. 25).

Li, F., R. Fergus, and P. Perona (2004). "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories." In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2004, Washington, DC, USA, June 27 - July 2, 2004*. IEEE Computer Society, p. 178 (cit. on pp. 85, 87).

Lowe, D. G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints." *International Journal of Computer Vision*, 60(2), pp. 91–110 (cit. on p. 80).

Mannila, H. and H. Toivonen (1997). "Levelwise Search and Borders of Theories in Knowledge Discovery." *Data Min. Knowl. Discov.* 1(3), pp. 241–258 (cit. on p. 24).

Marco, D. E., S. A. Páez, and S. A. Cannas (2002). "Species invasiveness in biological invasions: a modelling approach." *Biological Invasions*, 4(1), pp. 193–205 (cit. on p. 98).

Matula, D. W. (1978). "Subtree Isomorphism in O(n5/2)." In: *Algorithmic Aspects of Combinatorics*. Ed. by P. H. B. Alspach and D. Miller. Vol. 2. Annals of Discrete Mathematics. Elsevier, pp. 91 –106 (cit. on p. 13).

McCreesh, C. and P. Prosser (2015). "A Parallel, Backjumping Subgraph Isomorphism Algorithm Using Supplemental Graphs." In: *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*. Ed. by G. Pesant. Vol. 9255. Lecture Notes in Computer Science. Springer, pp. 295–312 (cit. on p. 13).

McGregor, J. J. (1979). "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms." *Inf. Sci.* 19(3), pp. 229–250 (cit. on p. 13).

McKay, B. D. et al. (1981). "Practical graph isomorphism" (cit. on p. 12).

Nijssen, S. and J. N. Kok (2004). "A quickstart in frequent structure mining can make a difference." In: *Proceedings of the Tenth ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004. Ed. by W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel. ACM, pp. 647–652 (cit. on p. 41).

Nilsback, M. and A. Zisserman (2008). "Automated Flower Classification over a Large Number of Classes." In: *Sixth Indian Conference on Computer Vision, Graphics & Image Processing, ICVGIP 2008, Bhubaneswar, India, 16-19 December 2008*. IEEE Computer Society, pp. 722–729 (cit. on p. 85).

Nowak, E., F. Jurie, and B. Triggs (2006). "Sampling Strategies for Bag-of-Features Image Classification." In: *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part IV*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Vol. 3954. Lecture Notes in Computer Science. Springer, pp. 490–503 (cit. on p. 81).

Nowozin, S., K. Tsuda, T. Uno, T. Kudo, and G. H. Bakir (2007). "Weighted Substructure Mining for Image Analysis." In: *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society (cit. on p. 81).

Odone, F., A. Barla, and A. Verri (2005). "Building kernels from binary strings for image matching." *IEEE Trans. Image Processing*, 14(2), pp. 169–180 (cit. on pp. 89, 105).

Özdemir, B. and S. Aksoy (2010). "Image Classification Using Subgraph Histogram Representation." In: *20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010*. IEEE Computer Society, pp. 1112–1115 (cit. on p. 80).

Pei, J. and J. Han (2000). "Can We Push More Constraints into Frequent Pattern Mining?" In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '00. Boston, Massachusetts, USA: ACM, pp. 350–354 (cit. on p. 27).

Pei, J., J. Han, and L. V. S. Lakshmanan (2004). "Pushing Convertible Constraints in Frequent Itemset Mining." *Data Min. Knowl. Discov.* 8(3), pp. 227–252 (cit. on p. 27).

Piatetsky-Shapiro, G. and C. J. Matheus (1994). "The Interingness of Deviations." In: *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, July 1994. Technical Report WS-94-03*. Ed. by U. M. Fayyad and R. Uthurusamy. AAAI Press, pp. 25–36 (cit. on p. 28).

Prado, A., B. Jeudy, É. Fromont, and F. Diot (2013). "Mining spatiotemporal patterns in dynamic plane graphs." *Intell. Data Anal.* 17(1), pp. 71–92 (cit. on pp. 1, 5, 34, 36).

Rastegari, M., V. Ordonez, J. Redmon, and A. Farhadi (2016). "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks." In: *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*. Ed. by B. Leibe, J. Matas, N. Sebe, and M. Welling. Vol. 9908. Lecture Notes in Computer Science. Springer, pp. 525–542 (cit. on p. 94).

Régin, J. (1994). "A Filtering Algorithm for Constraints of Difference in CSPs." In: *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.* Ed. by B. Hayes-Roth and R. E. Korf. AAAI Press / The MIT Press, pp. 362–367 (cit. on p. 13).

Rendell, P. W. (2014). "Turing machine universality of the game of life." PhD thesis. University of the West of England, Bristol, UK (cit. on p. 100).

Saint-Marcq, V. l. C. de, P. Schaus, C. Solnon, and C. Lecoutre (2013). "Sparse-sets for domain implementation." In: *CP workshop on Techniques foR Implementing Constraint programming Systems (TRICS)*, pp. 1–10 (cit. on p. 69).

Samuel, É., C. de la Higuera, and J. Janodet (2010). "Extracting Plane Graphs from Images." In: *Structural, Syntactic, and Statistical Pattern Recognition, Joint IAPR International Workshop, SSPR&SPR 2010, Cesme, Izmir, Turkey, August 18-20, 2010. Proceedings*. Ed. by E. R. Hancock, R. C. Wilson, T. Windeatt, I. Ulusoy, and F. Escolano. Vol. 6218. Lecture Notes in Computer Science. Springer, pp. 233–243 (cit. on pp. 5, 81).

Silva, F. B., S. Goldenstein, S. Tabbone, and R. da Silva Torres (2013). "Image classification based on bag of visual graphs." In: *IEEE International Conference on Image Processing, ICIP 2013, Melbourne, Australia, September 15-18, 2013*. IEEE, pp. 4312–4316 (cit. on pp. 80, 81).

Silva, F. B., S. Tabbone, and R. da Silva Torres (2014). "BoG: A New Approach for Graph Matching." In: *22nd International Conference on Pattern Recognition, ICPR 2014, Stockholm, Sweden, August 24-28, 2014*. IEEE Computer Society, pp. 82–87 (cit. on pp. 80, 81).

Smets, K. and J. Vreeken (2012). "Slim: Directly Mining Descriptive Patterns." In: *SDM*. SIAM / Omnipress, pp. 236–247 (cit. on p. 28).

Solnon, C. (2010). "AllDifferent-based filtering for subgraph isomorphism." *Artif. Intell.* 174(12-13), pp. 850–864 (cit. on pp. 13, 107).

Solnon, C., G. Damiand, C. de la Higuera, and J. Janodet (2015). "On the complexity of submap isomorphism and maximum common submap problems." *Pattern Recognition*, 48(2), pp. 302–316 (cit. on p. 18).

Sorlin, S. and C. Solnon (2008). "A parametric filtering algorithm for the graph isomorphism problem." *Constraints*, 13(4), pp. 518–537 (cit. on p. 12).

Soulet, A. and B. Crémilleux (2005). "An Efficient Framework for Mining Flexible Constraints." In: *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*. Ed. by T. B. Ho, D. W. Cheung, and H. Liu. Vol. 3518. Lecture Notes in Computer Science. Springer, pp. 661–671 (cit. on p. 28).

Syslo, M. M. (1982). "The Subgraph Isomorphism Problem for Outerplanar Graphs." *Theor. Comput. Sci.* 17, pp. 91–97 (cit. on p. 13).

Ullmann, J. R. (1976). "An Algorithm for Subgraph Isomorphism." *J. ACM*, 23(1), pp. 31–42 (cit. on pp. 12, 13).

Vanetik, N., E. Gudes, and S. E. Shimony (2002). "Computing Frequent Graph Patterns from Semistructured Data." In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pp. 458–465 (cit. on p. 35).

Voravuthikunchai, W., B. Crémilleux, and F. Jurie (2014). "Histograms of Pattern Sets for Image Classification and Object Recognition." In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, pp. 224–231 (cit. on p. 80).

Vreeken, J., M. van Leeuwen, and A. Siebes (2011). "Krimp: mining itemsets that compress." *Data Min. Knowl. Discov.* 23(1), pp. 169–214 (cit. on p. 28).

Wang, J., Z. Zeng, and L. Zhou (2006). "CLAN: An Algorithm for Mining Closed Cliques from Large Dense Graph Databases." In: *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*. Ed. by L. Liu, A. Reuter, K. Whang, and J. Zhang. IEEE Computer Society, p. 73 (cit. on p. 42).

Wolfram, S. (1984). "Cellular automata as models of complexity." *Nature*, 311(5985), pp. 419–424 (cit. on p. 98).

Wootton, J. T. (2001). "Local interactions predict large-scale pattern in empirically derived cellular automata." *Nature*, 413(6858), pp. 841–844 (cit. on p. 98).

Wörlein, M., T. Meinl, I. Fischer, and M. Philippsen (2005). "A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston." In: *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*. Ed. by A. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama. Vol. 3721. Lecture Notes in Computer Science. Springer, pp. 392–403 (cit. on p. 41).

Yan, X. and J. Han (2002). "gSpan: Graph-Based Substructure Pattern Mining." In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pp. 721–724 (cit. on pp. 32, 34–36).

— (2003). "CloseGraph: mining closed frequent graph patterns." In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*. Ed. by L. Getoor, T. E. Senator, P. M. Domingos, and C. Faloutsos. ACM, pp. 286–295 (cit. on pp. 27, 41, 73).

— (2006). "Discovery of frequent substructures." *Mining graph data*, 5, pp. 99–115 (cit. on p. 27).

Zeng, Z., J. Wang, L. Zhou, and G. Karypis (2006). "Coherent closed quasi-clique discovery from large dense graph databases." In: *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*. Ed. by T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos. ACM, pp. 797–802 (cit. on p. 42).

Zhang, S., J. Yang, and V. Cheedella (2007). "Monkey: Approximate Graph Mining Based on Spanning Trees." In: *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*. Ed. by R. Chirkova, A. Dogac, M. T. Özsu, and T. K. Sellis. IEEE Computer Society, pp. 1247–1249 (cit. on p. 44).

Zhang, S. and J. Yang (2008). "RAM: Randomized Approximate Graph Mining." In: *Scientific and Statistical Database Management, 20th International Conference, SSDBM 2008, Hong Kong, China, July 9-11, 2008, Proceedings*. Ed. by B. Ludäscher and N. Mamoulis. Vol. 5069. Lecture Notes in Computer Science. Springer, pp. 187–203 (cit. on p. 44).

Zhu, F., Q. Qu, D. Lo, X. Yan, J. Han, and P. S. Yu (2011). "Mining Top-K Large Structural Patterns in a Massive Network." *PVLDB*, 4(11), pp. 807–818 (cit. on p. 46).

Zou, Z., J. Li, H. Gao, and S. Zhang (2009). "Frequent subgraph pattern mining on uncertain graph data." In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*. Ed. by D. W. Cheung, I. Song, W. W. Chu, X. Hu, and J. J. Lin. ACM, pp. 583–592 (cit. on p. 45).

List of Figures

List of Tables

FOLIO ADMINISTRATIF

THÈSE DE L'UNIVERSITÉ DE LYON OPÉRÉE AU SEIN DE L'INSA LYON

Nom : Deville **Date de soutenance :** 30 Mai 2018

Prénom : Romain

Titre : Spatio-temporal grid mining applied to image classification and cellular automata analysis

Nature : Doctorat **Numéro d'ordre :** 2018LYSEI046

École doctorale : InfoMaths

Spécialité : Informatique

Résumé :

Au cours de cette thèse, nous avons proposé un nouvel algorithme de fouille de motifs fréquents dédié aux grilles spatio-temporelles : GriMA. L'usage des grilles régulières permet à notre algorithme de réduire la complexité des tests d'isomorphismes. Ces tests sont souvent utilisés par les algorithmes génériques de fouilles de graphes mais ayant une complexité importante, cela limite leur usage sur des données réelles. Deux applications ont été proposées pour évaluer notre algorithme : la classification d'images pour la fouille de grilles 2D et la prédiction d'automates cellulaires pour la fouille de grilles 2D+t.

Mots-clefs : Grids mining - Spatio-temporal patterns - Image classification - Cellular automata

Laboratoire de recherche : LIRIS

Directrice de thèse : Christine Solnon

Président de jury :

Composition du jury :

Bruno Crémilleux
Elisa Fromont
Jean-Christophe Janodet
Baptiste Jeudy
Jean-Yves Ramel
Céline Rouveirol
Christine Solnon