Recherche efficace de motifs fréquents dans des grilles

Romain Deville[2] Elisa Fromont[1] Baptiste Jeudy[1] Christine Solnon[2]

[1] Univ Lyon, UJM, CNRS, Lab Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France
[2] Université de Lyon, CNRS INSA-Lyon, LIRIS, UMR5205, F-69622, France

Résumé

La complexité des algorithmes de fouille de graphes généraux est telle qu'ils sont peu utilisés en pratique. Cette complexité est due à la fois aux tests d'isomorphisme et au grand nombre de combinaisons permettant d'étendre un graphe durant le processus de fouille. Dans cet article, nous proposons d'exploiter des représentations géométriques régulières (des grilles) pour rechercher efficacement des motifs fréquents dans un ensemble de grilles. Nous présentons un algorithme appelé GRIMA *qui, contrairement aux algorithmes généraux, peut passer l'échelle. Nous appliquons cet algorithme à un problème de classification d'images, pour lesquelles nous proposons une représentation par* Sac de grilles*. Les expérimentations montrent l'efficacité de notre algorithme et l'intérêt d'utiliser une représentation structurée pour représenter les images.*

Mots Clef

Fouille de graphes, grilles, classification d'images.

Abstract

General-purpose exhaustive graph mining algorithms are seldom used in real life contexts due to the high complexity of the process mostly based on costly isomorphism tests and countless expansion possibilities. In this paper, we show how to exploit grid-based representations to efficiently extract frequent grid subgraphs, and we introduce an efficient grid mining algorithm called GRIMA *designed to scale to large amount of data. We apply our algorithm on image classification problems. Experiments show that our algorithm is efficient and that adding the structure may help the image classification process.*

Keywords

Graph Mining, sub-graphs, grids, image classification.

1 Introduction

La complexité des algorithmes de fouille de graphes généraux est telle qu'ils sont peu utilisés en pratique. Cette complexité est due à la fois aux tests d'isomorphisme et au grand nombre de combinaisons permettant d'étendre un graphe durant le processus de fouille [6]. Afin de pouvoir effectivement utiliser ces algorithmes dans des applications réelles, il est souvent nécessaire de réduire la complexité en considérant des graphes particuliers ((outer)planar, clique, de degré borné, etc.), en contraignant la stratégie d'expansion, ou en relâchant la propriété de complétude.

Dans cet article, nous nous intéressons au problème de la recherche de motifs fréquents dans des graphes qui sont des grilles : les grilles ont une topologie fixée telle que le degré de chaque sommet est constant, *e.g.*, 4 dans une grille carrée 2D. Cette structure de grille est par exemple naturellement présente dans de nombreux jeux de plateau (dames, échecs, go, etc), ou encore dans les modélisations d'écosystèmes à base d'automates cellulaires [11]. En image en particulier, il est maintenant reconnu que les représentations haut niveau à base de graphes (*e.g.*, graphes d'adjacence de régions ou triangulations de points d'intérêt) sont sensibles au bruit de sorte que deux images légèrement différentes sont représentées par des graphes très différents [23, 22]. Cependant, les images sont intrinsèquement des grilles : des grilles 2D de pixels pour les images 2D, des grilles 3D de voxels pour les images tomographiques ou IRM, et des grilles 2D+t pour des vidéos. Nous proposons d'exploiter cette structure de grille naturellement présente dans les images pour caractériser les images par des histogrammes de motifs (sous-grilles) fréquents.

Dans la section 2, nous décrivons des travaux existants à base de fouille pour classer des images. Dans la section 3, nous rappelons quelques définitions sur les graphes, et dans la section 4 nous introduisons les grilles. Dans la section 5, nous décrivons notre algorithme, GRIMA, en lien avec les autres algorithmes de fouille de graphes. Dans la section 6 nous comparons expérimentalement ces différents algorithmes et nous montrons l'intérêt de cette approche pour la classification d'images.

2 Fouille pour la classification d'images

La fouille a été récemment utilisée avec succès pour faire de la classification d'images [10, 26] et permet d'obtenir des caractéristiques de plus haut niveau et plus discriminantes. Cependant, ces approches exploitent des caractéristiques telles que les sacs-de-mots (*Bags-Of-Visual-Words, BOW*), par exemple, et aucune information spa-

tiale entre ces caractéeristiques n'est exploitée. L'utilisation de sacs-de-graphes à la place des BOW a déjà été proposée dans [21] pour la classification d'images satellite : les sommets du graphe correspondent à des régions de l'image et les arêtes à des pavages de Voronoi (duaux des triangulations de Delaunay). Dans ce travail, les occurrences des sous-graphes dans chaque image sont dénombrées, et un support basé sur un ensemble indépendant maximum est utilisé. Dans notre approche, nous considérons une définition plus classique de la notion de support, basée sur le nombre d'images dans lesquelles un motif apparaît. Les auteurs de [21] ne proposent pas d'algorithme spécifique pour résoudre leur problème de fouille. La triangulation de Delaunay a également été utilisée dans [23] pour classer des images. Cependant, les triangulations ne sont pas invariantes aux changements d'illumination ou d'échelle des objets dans les images de sorte qu'elles ne permettent pas de bons résultats de classification. Des travaux récents [2, 24] mentionnent également l'utilisation de sacs-de-graphes pour de la classification en biologie. Cependant, les auteurs ne décrivent pas comment les motifs sont extraits des graphes, ni même comment les graphes sont créés à partir d'une image (ils considèrent que cela est dépendant de l'application). Dans [20], les auteurs ont montré qu'en combinant la fouille de graphes et le *boosting*, ils peuvent obtenir des règles de classification plus pertinentes qu'en utilisant simplement des ensembles de caractéristiques. Dans ces graphes, chaque sommet correspond à un point d'intérêt et est étiqueté par un descripteur discret. Le graphe est complet et chaque arête est étiquetée par un vecteur discret modélisant l'échelle, la distance et l'orientation. L'algorithme GSPAN [27] est ensuite utilisé pour extraire des sous-graphes fréquents, mais seulement un nombre limité de caractéristiques est utilisé pour chaque image afin de permettre un passage à l'échelle sur des jeux de données réels.

3 Définitions sur les graphes

Nous considérons des graphes non orientés connexes et étiquetés. Un **graphe** est défini par un triplet (N, E, L) où N est un ensemble de nœuds, $E \subseteq N \times N$ un ensemble d'arêtes et $L : N \cup E \rightarrow \mathbb{N}$ est une fonction qui associe une étiquette $L(c)$ à chaque composant (nœud ou arête) $c \in N \cup E$. Un **graphe géométrique** est un graphe dont les nœuds sont plongés dans un espace 2D. Il est défini par un tuple (N, E, L, η) tel que (N, E, L) est un graphe et $\eta : N \rightarrow \mathbb{R}^2$ associe chaque nœud à un point 2D (appelé coordonnées). Un **graphe plan** est un graphe géométrique dont les arêtes ne s'intersectent pas, à l'exception de leurs extrémités. Nous considérons des graphes plans dont les arêtes sont des segments de droites de sorte qu'un graphe géométrique (N, E, L, η) est plan si pour chaque couple d'arêtes différentes $\{(u, u'), (v, v')\} \subseteq E$, les deux segments ouverts $]\eta(u), \eta(u')[$ and $]\eta(v), \eta(v')[$ ont une intersection vide. Les arêtes d'un graphe plan partitionnent le plans en plusieurs régions, appelées **faces**. La face correspondant à la région infinie est appelée **face externe** ; les autres faces sont des **faces internes**.

Un **sous-graphe** d'un graphe $G = (N, E, L)$ est un graphe $G' = (N', E', L')$ tel que $N' \subseteq N$, $E' \subseteq E$ et L' est la restriction de L à $N' \cup E'$. L'extension de cette définition aux graphes géométriques est immédiate : nous ajoutons simplement que le plongement η' de G' est la restriction de η à N'. Cependant, l'extension aux graphes plans mérite une discussion. Nous pouvons simplement considérer la même définition que pour les graphes géométriques, comme cela est proposé dans [9]. Dans ce cas, certaines faces internes de G' peuvent ne plus correspondre à des faces internes de G (par exemple, si une arête entre deux faces adjacentes de G est supprimée de sorte que les deux faces de G sont fusionnées en une seule face dans G', comme illustré dans les figures 1(a) et 1(b)). Dans [8], une condition est ajoutée pour assurer que chaque face interne de G' est aussi une face interne de G. Plus formellement, étant donnés deux graphes plans $G = (N, E, L, \eta)$ et $G' = (N', E', L', \eta')$: G' est un **sous-graphe plan** de G si (N, E, L) est un sous-graphe de (N', E', L') et η' est la restriction de η à N' ; G' est **face-induit sous-graphe plan** de G si G' est un sous-graphe plan de G et chaque face interne de G' est aussi une face interne de G.

Deux graphes $G = (N, E, L)$ et $G' = (N', E', L')$ sont **isomorphes** s'il existe une bijection $f : N \rightarrow N'$, appelée fonction d'isomorphisme, qui préserve arêtes et étiquettes, *i.e.*, $\forall i, j \in N, (i, j) \in E \Leftrightarrow (f(i), f(j)) \in E'$; $\forall i \in N, L(i) = L'(f(i))$; et $\forall (i, j) \in E, L(i, j) = L'(f(i), f(j))$. Dans le cas de graphes géométriques, la fonction d'isomorphisme doit aussi préserver le plongement, mais les transformations rigides sont autorisées (voir [1] pour une définition des transformations rigides) : $G = (N, E, L, \eta)$ et $G' = (N', E', L', \eta')$ sont **géométriquement isomorphes** s'il existe une fonction d'isomorphisme $f : N \rightarrow N'$ et une transformation rigide $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ telle que $T(\eta(u)) = \eta'(f(u))$ pour chaque nœud $u \in N$.

Dans le cas de graphes plans, la fonction d'isomorphisme ne doit pas nécessairement préserver le plongement, mais elle doit préserver les faces, *i.e.*, G et G' sont **plan isomorphes** s'il existe une fonction d'isomorphisme f telle que, pour chaque face F de G, les arêtes bordant F sont appariées par f aux arêtes bordant une face F' de G', dans le même ordre en tournant autour des faces F et F'.

Finalement, un graphe (resp. un graphe géométrique) G_1 est **sous-isomorphe** (resp. **géométriquement sous-isomorphe**) à un graphe G_2, noté $G_1 \subseteq G_2$ (resp. $G_1 \subseteq_{\text{geo}} G_2$), si G_1 est isomorphe (resp. géométriquement isomorphe) à un sous-graphe de G_2. Dans le cas de graphes plans, nous obtenons deux relations différentes de sous-isomorphisme, selon la définition de sous-graphe considérée : G_1 est **plan sous-isomorphe** (resp. **face-induit plan sous-isomorphe**) à G_2, noté $G_1 \subseteq_{\text{plan}} G_2$ (resp. $G_1 \subseteq_{\text{face}} G_2$) si G_1 est plan isomorphe à un sous-graphe plan (resp. face-induit sous-graphe plan) de G_2.

Relation de sous-isomorphisme entre deux graphes G_1 et G_2	Carac. de G_1 préservées dans G_2			
	Faces internes	Angles	Ordre voisins	Longueur arêtes
$G_1 \subseteq G_2$ (sous-graphe)	Non	Non	Non	Non
$G_1 \subseteq_{\text{plan}} G_2$ (sous-graphe plan)	Non	Non	Oui	Non
$G_1 \subseteq_{\text{face}} G_2$ (face-induit sous-graphe plan)	Oui	Non	Oui	Non
$G_1 \subseteq_{\text{geo}} G_2$ (sous-graphe géométrique)	Non	Oui	Oui	Non
$G_1 \subseteq_{\text{grid}} G_2$ (sous-grille)	Non	Oui	Oui	Oui

TABLE 1 – Comparaison des relations de sous-isomorphisme.

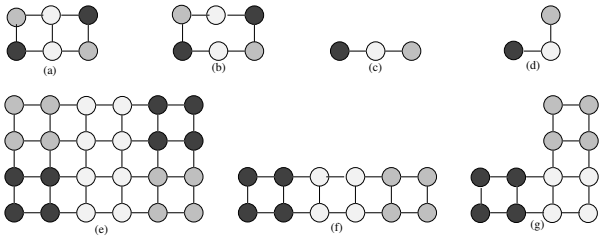


FIGURE 1 – Exemples de graphes. (b) est un sous-graphe plan de (a), mais pas un sous-graphe plan face-induit de (a). (c) et (d) sont des graphes isomorphes et sont des sous-graphes de (a). Cependant, (c) et (d) ne sont pas grille-isomorphes, et (c) est une sous-grille de (a) tandis que (d) n'est pas une sous-grille de (a).

4 Grilles

Soient u et v deux nœuds de coordonnées $\eta(u) = (x_u, y_u)$ et $\eta(v) = (x_v, y_v)$. u et v sont voisins dans le plan si $|x_u - x_v| + |y_u - y_v| = 1$. Une **grille** est un graphe géométrique $G = (N, E, L, \eta)$ tel que $\forall u \in N, \eta(u) \in \mathbb{Z}^2$ et $\forall (u, v) \in E$, u et v sont voisins dans le plan. Par définition, les grilles sont aussi des cas particuliers de graphes plans (les arêtes ne peuvent se croiser puisqu'elles ne relient que des nœuds voisins dans le plan).

Chercher des motifs dans des grilles revient à chercher des isomorphismes de sous-grilles. Dans un contexte d'analyse d'images, ces motifs doivent être invariants aux translations et aux rotations de sorte que deux motifs équivalents à une rotation et/ou une translation près doivent être isomorphes. Par conséquent, comme pour l'isomorphisme de graphes géométriques, l'isomorphisme de grilles autorise les translations et les rotations [1]. Par exemple, les graphes plans (c) et (d) de la figure 1 sont plan-isomorphes. Cependant, (d) ne peut être obtenu en translatant et/ou tournant (c), car l'angle entre les arêtes de (d) est différent de l'angle entre les arêtes de (c). Par conséquent, (c) et (d) ne sont pas grille-isomorphes.

Finalement, une grille G_1 est **sous-grille-isomorphe** à une grille G_2 (noté $G_1 \subseteq_{\text{grid}} G_2$) si G_1 est grille-isomorphe à un sous-graphe plan de G_2. La table 1 résume les propriétés des cinq relations de sous-isomorphisme définies précédemment. Pour chacune de ces relations $\subseteq_x \in \{\subseteq, \subseteq_{\text{geo}}, \subseteq_{\text{plan}}, \subseteq_{\text{face}}, \subseteq_{\text{grid}}\}$, si $G_1 \subseteq_x G_2$ alors une fonction d'isomorphisme f de G_1 vers un sous-graphe de G_2 est appelée **occurrence** de G_1 dans G_2.

Complexité. L'isomorphisme de sous-graphes (planaires) est un problème $\mathcal{NP}$-complet. Cependant, il existe des cas particuliers de graphes planaires pour lesquels ce problème devient polynomial, *e.g.*, les graphes 2-connexes outerplanar, et les graphes outerplanar graphs avec la contrainte block-and-bridge [12]. Le sous-isomorphisme plan face-induit est polynomial quand le graphe motif est 2-connexe [8], mais il devient NP-complet quand on relâche cette contrainte [25]. L'isomorphisme de sous-grilles n'est pas un cas particulier de l'isomorphisme de sous-graphes plans face-induits (car il n'impose pas que les faces soient préservées), et il n'est pas non plus une généralisation de ce problème (car il ajoute la contrainte que les angles doivent être préservés). L'isomorphisme de sous-grilles est un cas particulier de l'isomorphisme de sous-graphes géométriques, où les arêtes doivent relier les nœuds voisins dans le plan. Dans [1] (cf section 5), il est montré que nous pouvons énumérer tous les sous-graphes géométriques maximaux en temps incrémental polynomial, et en espace polynomial.

1. Pour les graphes géométriques, il est aussi possible d'appliquer une transformation de changement d'échelle. Cependant, cette transformation n'a pas de sens dans un contexte de grille où les arêtes ne peuvent relier que des nœuds voisins dans le plan.

5 Algorithme de fouille de grilles

Nous présentons maintenant notre algorithme de fouille de grilles, appelé GRIMA, en lien avec les autres algorithmes de fouille de graphes existants. Des algorithmes de fouille de graphes précurseurs sont GSPAN [27], et les algorithmes similaires de fouille exhaustive de graphes [13]. Ces algorithmes ne tiennent pas compte des angles relatifs des arêtes pendant la fouille de sorte que GSPAN considère comme isomorphes des graphes qui sont différents du point de vue des grilles, comme illustré dans la figure 1 (c et d). Par conséquent, des algorithmes non spécialisés comme GSPAN peuvent considérer comme fréquents un grand nombre de motifs, et mal passer à l'échelle. Par ailleurs, PLAGRAM [22] a été conçu pour fouiller des graphes plans et passe mieux à l'échelle sur ces graphes. Cependant, dans PLAGRAM, la stratégie d'extension implique une limite forte : les motifs sont composés de faces et le plus petit motif est une face. Il est possible d'utiliser PLAGRAM pour fouiller des grilles mais cela impose de transformer les grilles de sorte que chaque nœud de la grille initiale corresponde à une face dans le graphe fourni à PLAGRAM. Cette transformation est illustrée dans la figure 1 : les deux graphes isomorphes (c) et (d) (qui sont des sous-graphes de (a)) sont transformés en deux graphes non isomorphes (f) et (g) (de sorte que seul (f) est sous-graphe de (e)). Cependant, cette transformation augmente artificiellement le nombre de nœuds et d'arêtes, ce qui peut de nouveau causer des problèmes de passage à l'échelle pour PLAGRAM. Les grilles ont été introduites dans [17], mais les auteurs ne tiennent pas compte des informations d'angles entre arêtes. Enfin, les grilles sont des cas par-

Algorithme	Relation	Motif	Extension
GSPAN	$\subseteq$	sous-graphe connexe	une arête
FREQGEO	$\subseteq_{\text{geo}}$	sous-graphe géométrique	une arête/nœud
PLAGRAM	$\subseteq_{\text{face}}$	sous-graphe plan 2-connexe	une face
GRIMA	$\subseteq_{\text{grid}}$	sous-grille connexe	une arête

TABLE 2 – Caractéristiques des quatre algorithmes de fouille de graphes. Le type de motif est une conséquence de la relation d'isomorphisme et du choix d'extension.

ticuliers de graphes géométriques, pour lesquels un algorithme de fouille a été proposé dans [1]. Cet algorithme, appelé FREQGEO, peut être vu comme une généralisation de GRIMA mais, du fait de cette généralisation, il n'est pas optimisé pour le cas des grilles. De plus, les auteurs ne fournissent aucune implémentation ni ne rapportent de résultats expérimentaux, dans leur article comme dans leurs publications suivantes, ce qui ne nous permet pas de nous comparer à eux. MAXGEO [1] est une amélioration de FREQGEO qui ne fouille que des motifs fermés (ou maximaux). Cependant, dans notre application, les expérimentations ont montré que quasiment tous les motifs extraits sont fermés. Par conséquent, fouiller des motifs clos ne change pas les résultats, et nous ne parlerons pas de MAXGEO ni de graphes clos dans la suite.

Afin de mieux mettre en évidence les points communs et différences entre GRIMA, GSPAN, PLAGRAM, et FREQGEO, nous décrivons ces différents algorithmes comme des instanciations d'un unique algorithme générique. Cet algorithme générique calcule des sous-graphes fréquents, *i.e.*, des sous-graphes qui apparaissent au moins une fois dans au moins σ graphes de la base de graphes, où σ est un seuil fixé par l'utilisateur. Plus formellement, étant donnés un ensemble de graphes D, un seuil de fréquence σ, et une relation de sous-isomorphisme $\subseteq_x \in \{\subseteq, \subseteq_{\text{face}}, \subseteq_{\text{geo}}, \subseteq_{\text{grid}}\}$, le **problème de fouille de graphes** consiste à trouver tous les graphes motifs P dont la fréquence dans D est plus grande ou égale à σ. À l'exception de FREQGEO, la **fréquence** d'un graphe dans D est le nombre de graphes $G_i \in D$ qui contiennent au moins une occurrence de P, *i.e.*, fréquence$(P) = |\{G_i \in D \mid P \subseteq_x G_i\}|$. Cela est différent du nombre total d'occurrences dans D dans la mesure où un motif P peut avoir plusieurs occurrences dans un même graphe cible G_i. Pour FREQGEO, D est constitué d'un seul graphe et la fréquence d'un motif P est le nombre d'occurrences de P dans D.

5.1 Algorithme générique

L'algorithme utilise des codes pour représenter les graphes : le **code** d'un graphe peut être une liste d'arêtes rencontrées pendant le parcours du graphe [27, 22] ou une liste de nœuds et d'arêtes en ordre lexicographique [1]. Un graphe peut avoir plusieurs codes, mais un seul est choisi comme signature : le **code canonique**. Pour GSPAN et FREQGEO, c'est le plus petit code tandis que pour PLAGRAM et GRIMA c'est le plus grand.

Algorithm 1 Fouille de sous-graphes fréquents générique

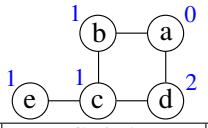
Entrée : Une base de graphes $D = \{G_1, ..., G_n\}$ et un seuil de fréquence σ.
Sortie : L'ensemble des motifs fréquents dans D.

1: **for all** extension e fréquente dans D **do** Extend(e, D)
2:
3: **function** EXTEND(code P, base de graphes D)
4: Output P
5: $E \leftarrow \emptyset$
6: **for all** graphe G_i de D **et chaque** occurrence o du motif P dans G_i **do**
7: $E \leftarrow E \cup$ ValidExtensions(P, G_i, o)
8: **for all** extension $e \in E$ **do**
9: **if** fréquence$(P.e) > \sigma$ **et** $P.e$ est canonique **then** Extend($P.e$, D)

L'algorithme 1 décrit l'algorithme de fouille générique (voir [27, 22, 1] pour plus de détails), qui explore l'espace des codes canoniques en profondeur d'abord et récursivement. Chaque fois qu'un code correspondant à un graphe fréquent est trouvé, il est étendu en ajoutant une arête et/ou un nœud ou une face, en fonction de la stratégie d'expansion (voir Table 2). L'algorithme calcule d'abord toutes les extensions fréquentes et appelle la fonction `Extend` pour chacune de ces extensions (ligne 1).
La fonction `Extend` a deux paramètres en entrée : un code P fréquent et canonique, et une base de graphes D. Elle retourne en sortie tous les codes fréquents canoniques dont P est préfixe. Pour cela, elle calcule l'ensemble E de toutes les extensions valides de toutes les occurrences de P dans D : une extension valide est le code e d'un composant (nœud, arête ou face) tel que $P.e$ apparaisse dans D. Ensuite, `Extend` est récursivement appelée pour chaque extension e telle que $P.e$ est fréquent et canonique. Ainsi, à chaque appel récursif, le motif grossit. Dans les paragraphes qui suivent, nous décrivons l'instanciation des étapes de `Extend` pour GRIMA.

5.2 Code d'une grille

Dans les algorithmes de fouille de graphes, un point clé est d'éviter de générer le même motif plusieurs fois. En effet, un motif avec n arêtes peut être généré en ajoutant n'importe laquelle de ses n arêtes à un motif de $n-1$ arêtes. Nous utilisons pour cela la notion de code canonique [27] pour représenter des motifs de sorte que nous explorons l'espace des codes canoniques. Dans cet espace, il existe un seul chemin pour atteindre un code canonique depuis la racine (*i.e.*, un seul ordre pour ajouter les arêtes au motif).
Pour GRIMA, un **code** d'une grille G est une séquence de n codes d'arêtes $C(G) = \langle ec_0, ..., ec_{n-1} \rangle$, associée à un parcours en profondeur de G, au départ d'un nœud donné. Pendant ce parcours, chaque arête est parcourue une fois, et les nœuds sont numérotés : le nœud initial a le numéro 0 ; à chaque fois qu'un nouveau nœud est découvert, il est numéroté avec le plus petit entier pas encore utilisé dans le parcours. Chaque code d'arête correspond à une arête différente de G et l'ordre des codes d'arêtes dans $C(G)$ correspond à l'ordre dans lequel les arêtes sont parcourues.



Code 1								Code 2								Code 3							
edge	d	i	j	a	L_i	L_j	L_{ij}	edge	d	i	j	a	L_i	L_j	L_{ij}	edge	d	i	j	a	L_i	L_j	L_{ij}
(a,b)	0	0	1	0	0	1	0	(a,b)	0	0	1	0	0	1	0	(d,c)	0	0	1	0	2	1	0
(b,c)	0	1	2	3	1	1	0	(b,c)	0	1	2	3	1	1	0	(c,e)	0	1	2	2	1	1	0
(c,e)	0	2	3	1	1	1	0	(c,d)	0	2	3	3	1	2	0	(c,b)	0	1	3	1	1	1	0
(c,d)	0	2	4	3	1	2	0	(d,a)	1	3	0	3	2	0	0	(b,a)	0	3	4	1	1	0	0
(d,a)	1	4	0	3	2	0	0	(c,e)	0	2	4	1	1	1	0	(a,d)	1	4	0	1	0	2	0

FIGURE 2 – Exemples de codes pour une grille (les étiquettes des nœuds sont affichées en bleu : toutes les arêtes ont la même étiquette 0). Les codes 1 et 2 correspondent à des parcours commencés depuis l'arête (a,b), et diffèrent pour la troisième arête. Le code 3 correspond à un parcours commencé depuis l'arête (d,c) et il est canonique.

Ainsi, ec_k est le code de la k^e arête parcourue. Ce code d'arête ec_k est le tuple $(d,i,j,a,L_i,L_j,L_{(i,j)})$ tel que

- i et j sont les numéros associés aux nœuds de la k^e arête parcourue ;
- $d \in \{0,1\}$ est la direction de la k^e arête parcourue :
  - $d=0$ si elle est "en avant", *i.e.*, i apparaît déjà dans le préfixe $\langle ec_0,...,ec_{k-1}\rangle$ du code et j est un nouveau nœud qui vient d'être découvert ;
  - $d=1$ si elle est "en arrière", *i.e.*, à la fois i et j apparaissent déjà dans le préfixe ;
- $a \in \{0,1,2,3\}$ est l'angle de la k^e arête parcourue :
  - $a=0$ si $k=0$ (première arête) ;
  - sinon, $a = 2A/\pi$ où $A \in \{\pi/2, \pi, 3\pi/2\}$ est l'angle entre (i,j) et l'arête utilisée pour atteindre i la première fois ;
- L_i, L_j, $L_{(i,j)}$ sont les étiquettes de nœuds et d'arête.

Considérons par exemple le code 1 de la figure 2. La quatrième arête parcourue est (c,d). C'est une arête en avant (car d n'a pas encore été découvert) de sorte que $d=0$. L'angle entre (b,c) et (c,d) est $3\pi/2$ de sorte que $a=3$. La cinquième arête parcourue est (d,a) qui est une arête en arrière (car a a déjà été découvert) de sorte que $d=1$.

Étant donné un code, nous pouvons reconstruire la grille correspondante dans la mesure où les arêtes sont listées dans le code, avec leurs angles et leurs étiquettes. Cependant, il existe plusieurs codes différents pour une même grille, comme cela est illustré dans la figure 2 : chaque code correspond à un parcours différent (partant d'un nœud différent ou parcourant les arêtes dans un ordre différent).

Nous définissons maintenant un ordre total sur l'ensemble de tous les codes pouvant être associés à une grille. Nous définissons un ordre total sur les codes des arêtes, en considérant l'ordre lexicographique induit par l'ordre des composants du tuple (qui sont des valeurs entières). L'ordre sur les codes de grille est l'ordre lexicographique induit par l'ordre sur les codes des arêtes. Le **code canonique** d'une grille est le plus grand de tous ses codes, selon l'ordre ainsi défini, et il est unique. Nous pouvons prouver (par manque de place, la preuve est omise) qu'il n'est pas possible d'étendre un code non canonique en un code canonique de sorte que *tout préfixe d'un code canonique est également canonique*. Ainsi, nous pouvons élaguer les codes non canoniques de l'espace de recherche.

5.3 Extensions et canonicité d'un code

La fonction `Extend` de l'algorithme 1 retourne tous les motifs fréquents canoniques P' tels que le motif P en entrée est préfixe de P'. Cela est fait en deux étapes : d'abord elle recherche toutes les extensions possibles de P (lignes 6-7), où une extension est le code e d'une arête tel que $P.e$ apparaît dans D ; ensuite, pour chaque extension e, si $P.e$ est fréquent et canonique, elle étend récursivement $P.e$.

Pour chaque code motif P et chaque grille $G_i \in D$, nous mémorisons la liste $L(P,G_i)$ de toutes les occurrences de P dans G_i (pour chaque occurrence, nous mémorisons uniquement l'appariement de la première arête). Ces listes sont maintenues incrémentalement : pour chaque code e tel que $P.e$ est canonique, la liste $L(P.e,G_i)$ est une sous-liste de $L(P,G_i)$. Nous utilisons des *sparse sets* [15] pour maintenir efficacement ces listes à chaque appel récursif : les *sparse sets* permettent de rétablir en temps constant $L(P,G_i)$ à partir de $L(P.e,G_i)$ au moment d'un retour-arrière. Les fréquences sont également maintenues incrémentalement.

Ainsi, dans la fonction `ValidExtensions`, pour chercher toutes les extensions de P dans une grille G_i, nous parcourons la liste $L(P,G_i)$ et pour chaque occurrence $o \in L(P,G_i)$, nous cherchons toutes les arêtes de G_i qui n'appartiennent pas à o mais sont incidentes à au moins un nœud de o. Cela est fait en temps linéaire par rapport au nombre de nœuds de P. Comme le nombre d'occurrences de P dans G_i est borné par $2|G_i|$ (pour chaque arête de G_i, il y a au plus 2 occurrences de P partant de cette arête), la complexité des lignes 6–7 pour un motif P est $\mathcal{O}(|P|.\sum_i |G_i|)$.

Enfin, nous devons vérifier si $P.e$ est canonique (ligne 9) pour chaque code d'arête e tel que $P.e$ est fréquent dans D. Ce test de canonicité est réalisé en deux temps : d'abord nous reconstruisons la grille motif à partir du code P ; ensuite, pour chaque arête (i,j) dans la grille, nous construisons les deux codes $P_{(i,j)}$ et $P_{(j,i)}$ correspondant aux deux parcours différents partant de i et de j, respectivement, et commençant par l'arête (i,j). À chaque étape du parcours, si nous avons le choix entre plusieurs arêtes, nous choisissons toujours les arêtes en arrière d'abord, puis nous départageons les ex-aequo en choisissant celle dont l'angle maximise le code. Par exemple, dans la figure 2, si nous commençons le parcours à partir de a et choisissons en premier l'arête (a,b), la deuxième arête traversée est (b,c). Pour l'arête suivante, nous devons choisir entre (c,e) et (c,d). Comme aucune des deux n'est en arrière, et que l'angle entre (b,c) et (c,e) est $\pi/2$ et l'angle entre (b,c) et (c,d) est $3\pi/2$, nous choisissons l'arête (c,d) qui donne

un plus grand code (Code 2) et nous ne générons pas le code obtenu en choisissant l'arête (c, e) (Code 1) car nous savons qu'il sera nécessairement plus petit. Si nous trouvons un code $P_{(i,j)}$ plus grand que P, nous concluons que P n'est pas canonique, tandis que si tous les codes $P_{(i,j)}$ sont plus petits ou égaux à P, nous concluons que P est canonique.

Si P contient p arêtes, nous devons faire au plus $2p$ parcours, et chaque parcours est fait en $\mathcal{O}(p)$, de sorte que la complexité du test de canonicité est $\mathcal{O}(p^2)$. Cependant, chaque parcours peut être arrêté dès qu'un code d'arête dans $P_{(i,j)}$ est différent du code d'arête à la même position dans P, i.e., ce pire des cas n'est quasiment jamais atteint en pratique (il n'est atteint que si le motif est complètement symétrique de sorte que tous les codes sont égaux, quelle que soit la première arête).

Pour un motif P, nous devons tester la canonicité de $P.e$ pour chaque extension fréquente e. Le nombre d'occurrences de P dans un graphe G_i avec p_i arêtes est borné par $2p_i$ et chaque occurrence de P a au plus $2p_i$ extensions. Le nombre total d'extensions de P dans D est donc borné par $4\sum_{G_i \in D} p_i^2$. Comme les fréquences sont maintenues incrémentalement (lignes 5–7), le test de fréquence de la ligne 9 est réalisé en temps constant. Ainsi, la complexité des lignes 8–9 est $\mathcal{O}(p^2 \cdot \sum_{G_i \in D} p_i^2)$.

5.4 Propriétés de Grima et optimisations

Nous pouvons prouver (non détaillé ici par manque de place) que GRIMA est à la fois *correct*, *i.e.*, il ne retourne que des sous-grilles fréquentes (voir lignes 1 et 9 de l'algorithme 1) et *complet*, *i.e.*, il retourne tous les motifs fréquents. Si la base D de grilles contient k grilles chacune ayant au plus n arêtes, alors GRIMA énumère tous les motifs fréquents en $\mathcal{O}(kn^4)$ pour chaque motif P. En effet, la complexité d'un appel à `Extend` pour un motif P est $\mathcal{O}(n^2)$ pour les lignes 6–7 de l'algorithme 1 et $\mathcal{O}(kn^4)$ pour les lignes 8–9. Cela représente une réelle amélioration par rapport à FREQGEO et MAXGEO qui ont une complexité en temps de $\mathcal{O}(k^2n^4.\ln n)$ [1] par motif (quand D est une grille).

Motifs induits par les nœuds. Dans nos expérimentations, il n'y a pas d'étiquette sur les arêtes, seulement sur les nœuds. Par conséquent, la liste des occurrences d'un motif ne change pas si nous lui ajoutons des arêtes sans ajouter de nouveau nœuds. Ainsi, nous avons conçu une variante de GRIMA, appelée node-induced-GRIMA, qui calcule des sous-grilles induites par les nœuds (ce qui peut être vu comme une forme relâchée de grilles fermées).

6 Évaluation expérimentale

6.1 Protocole expérimental

Nous avons évalué notre approche sur la base d'images *Oxford-Flowers17* [18], qui contient 17 classes de fleurs, chaque classe contenant 80 images. Pour évaluer l'intérêt d'une structure de grille pour la classification d'images, nous proposons de représenter chaque image par une grille de mots visuels, et d'extraire des sous-grilles fréquentes pour obtenir des sacs-de-grilles (BOG). Nous comparons ces BOG avec une représentation par sacs-de-mots (BOW).

Modélisation par BOW. Étant donné un ensemble de descripteurs, associés à des régions d'images extraites d'une base d'images, les mots visuels sont créés en regroupant les descripteurs similaires en utilisant l'algorithme K-means [7, 4]. L'ensemble des mots visuels est appelé vocabulaire visuel. Chaque image est décrite par l'histogramme de fréquence de ce vocabulaire, appelé sac-de-mots. Jusqu'à il y a une dizaine d'années, les régions utilisées pour créer le vocabulaire visuel étaient sélectionnées en utilisant des détecteurs de points d'intérêt ou des méthodes de segmentation. Cependant, [19] a montré que les résultats de classification sont aussi bons, et même souvent meilleurs, lorsque les régions sont sélectionnées aléatoirement sur une grille (*dense sampling*). Cette modélisation est donc particulièrement bien adaptée pour notre approche. Dans nos expériences, nous avons utilisé des descripteurs SIFT [16], qui sont des vecteurs à 128 dimensions décrivant des informations de gradient dans un carré centré autour d'un pixel donné. Les descripteurs SIFT sont extraits régulièrement sur une grille telle que les centres des carrés sont espacés de 8 pixels. Nous sommes conscients que les méthodes basées sur des BOW ne sont plus l'état de l'art pour la classification d'images, mais notre but est de comparer un ensemble non structuré de descripteurs et un ensemble de descripteurs structurés en grille, afin d'évaluer l'apport de la structuration pour la classification d'images.

Modélisation par BOG. Nous commençons par calculer un vocabulaire visuel en utilisant la méthode décrite précédemment. Nous décrivons ensuite chaque image par une grille obtenue en connectant chaque mot avec ses 4 voisins (sauf sur les bords). Nous recherchons ensuite les motifs fréquents avec GRIMA, pour chaque classe séparément. Enfin, nous faisons l'union de tous les motifs fréquents trouvés dans chaque classe, et décrivons chaque image par l'"histogramme de fréquence (nombre d'occurrences) de ces motifs dans l'image.

Paramètres. Le paramètre K de l'algorithme K-means détermine le nombre de mots visuels, et nous avons fait varier K dans l'ensemble $\{100, 200, 250, 500, 1000\}$. Pour la classification, nous utilisons l'implémentation Libsvm [3] de SVM avec le noyau d'intersection de [10]. Nous effectuons une validation croisée en 10 parties : pour chaque partie, nous avons 72 images par classe dans la base d'entrainement (utilisée pour calculer les BOW, extraire les motifs fréquents pour BOG, et entrainer SVM), et 8 images par classe dans la base de test (utilisée pour calculer le pourcentage d'images bien classées).

6.2 Comparaison de l'efficacité

La figure 3 compare GRIMA, node-induced-GRIMA, GSPAN et PLAGRAM pour les 1360 images de notre base, avec $K = 100$ mots. Rappelons que la stratégie d'expansion

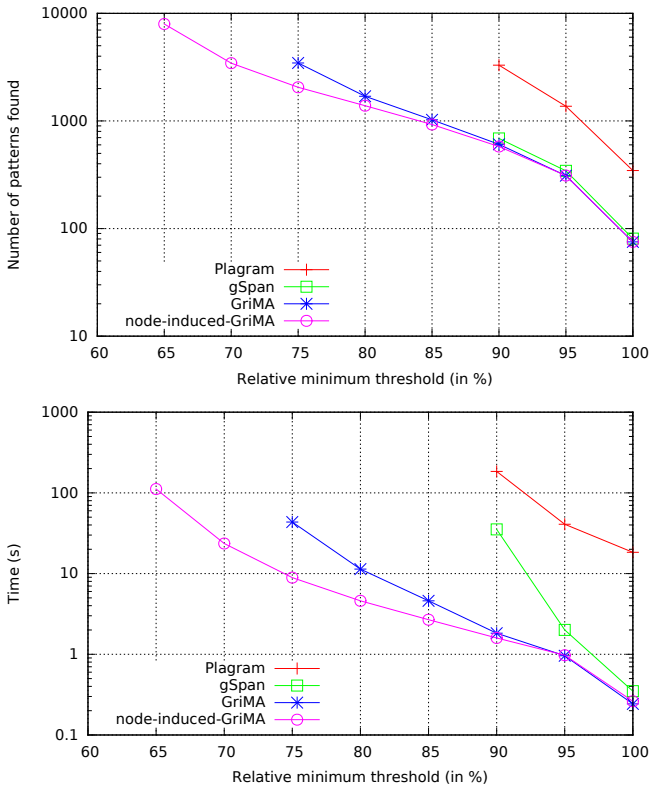


FIGURE 3 – Nombre de motifs extraits (en haut) et temps de fouille (en bas) en fonction du seuil de fréquence σ, en moyenne pour toutes les classes de *Oxford-Flowers17*, pour GSPAN, PLAGRAM, GRIMA et node-induced-GRIMA. Le temps est limité à 1 heure par classe.

de PLAGRAM ne permet pas de calculer des motifs sans face. Pour permettre à PLAGRAM de fouiller les mêmes motifs que les autres algorithmes, chaque nœud de la grille originale est remplacé par une face à quatre nœuds, de sorte que chaque motif fréquent dans la grille initiale soit trouvé par PLAGRAM. Cependant, certains motifs trouvés par PLAGRAM ne correspondent pas à des motifs dans la grille initiale (*e.g.*, des faces dans la grille étendue, correspondant à des arêtes dans la grille initiale). Ainsi, PLAGRAM trouve plus de motifs que les autres algorithmes, et est donc logiquement plus lent. GSPAN ne tient pas compte des angles entre arêtes au moment de la fouille, et deux sous-graphes isomorphes peuvent ne pas correspondre à des sous-grilles isomorphes, comme illustré dans la figure 1. Ainsi, GSPAN et GRIMA calculent des ensembles différents de motifs fréquents. Cependant, la figure 3 montre que le nombre de motifs est relativement similaire (légèrement supérieur pour GSPAN). GRIMA et plus particulièrement sa version induite par les nœuds passe bien mieux à l'échelle que GSPAN, et est capable d'extraire tous les motifs fréquents en moins de 10 secondes quand $\sigma = 85\%$ alors que GSPAN n'a pas terminé au bout d'une heure. Le nombre de motifs trouvés par PLAGRAM et GSPAN pour un support inférieur à 90% n'est donc pas pertinent et n'est pas montré dans la figure. Node-induced-GRIMA extrait moins de motifs et est donc plus efficace que GRIMA, et bien plus efficace que GSPAN et PLAGRAM.

	100		250		500		750		1000	
σ	%	#pat	%	#pat	%	#pat	%	#pat	%	#pat
BOG 0.9	**59.9**	2136	57.9	658	59.9	244	61.1	115	61.3	57
BOG 0.8	58.7	8483	59.6	1623	60.0	683	61.2	390	61.6	241
BOG 0.7	53.9	62016	59.6	4142	60.6	1635	60.7	998	61.1	686
BOG 0.6	51.3	302111	**59.7**	11624	**61.2**	3878	**61.5**	2106	**62.1**	1444
BOW	48.5		55.9		58.7		60.7		61.5	

TABLE 3 – Résultats de classification. Les quatre premières lignes donnent les résultats de BOG, quand σ est fixé à 0.9, 0.8, 0.7 et 0.6, respectivement. La dernière ligne donne les résultats de BOW. Chaque ligne donne le pourcentage d'images bien classées (%), quand K est fixé à 100, 250, 500, 750, et 1000 mots, respectivement. Pour BOG, la colonne #pat donne le nombre de motifs extraits par node-induced-GRIMA.

6.3 Comparaison des taux de classification

La table 3 donne les résultats de classification pour différents nombres de mots K, compris entre 100 et 1000. Pour BOG, nous donnons les résultats obtenus avec différents seuils de fréquence σ, compris entre 0.9 et 0.6. Le nombre de motifs extrait augmente quand on diminue σ ou K. Par exemple, quand $\sigma = 0.9$, le nombre de motifs passe de 57 (quand $K = 1000$) à 2136 (quand $K = 100$) ; quand $K = 100$, il augmente de 2136 (quand $\sigma = 0.9$) à 302111 (quand $\sigma = 0.6$). Quand le nombre de mots (correspondant au nombre d'étiquettes sur les nœuds des grilles) est petit (*e.g.*, 100), nous avons besoin de moins de motifs pour obtenir des gains importants pour BOG (59.9) par rapport à BOW (48.5). Pour de plus grands vocabulaires (*e.g.* 1000 mots), nous devons baisser le seuil de fréquence pour obtenir des motifs plus pertinents. De façon générale, ces résultats montrent qu'ajouter une information structurelle sous la forme de grilles améliore la représentation des images, et que notre algorithme de fouille peut être utilisé dans un contexte réel.

Nous avons effectué des expériences similaires sur la base *15-Scenes* [14]. Cependant, sur cette base, les résultats de BOW et BOG sont très similaires. Notons enfin que ces résultats ne sont pas compétitifs avec l'état de l'art. En particulier, [5] montre que les techniques d'apprentissage profond permettent d'obtenir de bien meilleurs résultats que les BOW. Cependant, notre méthode est générique et peut être appliquée à n'importe quelle grille (dont les nœuds peuvent être étiquetés par n'importe quelle caractéristique, y compris des caractéristiques calculées à l'aide de réseaux de convolution profonds.

7 Conclusion

Nous avons présenté GRIMA, un algorithme de fouille de grilles permettant de faire de la classification à partir de sacs-de-grilles. GRIMA exploite les informations d'angle

entre arêtes de la grille et a été conçu pour mieux passer à l'échelle sur de gros volumes de données. Nous pensons que cette structure de grille apporte des informations pertinentes non seulement dans l'application de classification d'images présentée ici, mais également dans d'autres applications telles que les jeux de plateaux (dames, échecs, go, etc) ou les modélisations d'écosystèmes à base d'automates cellulaires. Notons que pour ces applications, nous avons non pas une grille, mais une séquence temporelle de grilles (*i.e.*, une grille 2D+t). Ainsi, nous prévoyons d'étendre GRIMA à ce type de grilles. Par ailleurs, nous envisageons d'ajouter une étape de post-traitement, après l'extraction des motifs, afin de sélectionner les motifs les plus discriminants dans le cas où un trop grand nombre de motifs a été extrait, comme proposé dans [10].

Remerciements. Ce travail a été en partie financé par le projet ANR SoLStiCe (ANR-13-BS02-0002-01).

Références

[1] Hiroki Arimura, Takeaki Uno, and Shinichi Shimozono. Time and space efficient discovery of maximal geometric graphs. In *International Conference on Discovery Science*, pages 42–55, 2007.

[2] F. Brandao Da Silva, S. Goldenstein, S. Tabbone, and R. Da Silva Torres. Image classification based on bag of visual graphs. In *IEEE SPS*, pages 4312–4316, 2013.

[3] Chih-Chung Chang and Chih-Jen Lin. LIBSVM : A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2 :27 :1–27 :27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details : an evaluation of recent feature encoding methods. In *BMVC*, pages 76.1–76.12, 2011.

[5] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details : Delving deep into convolutional nets. In *BMVC*, 2014.

[6] Diane Cook and Lawrence Holder. *Mining Graph Data*. J. Wiley & Sons, 2006.

[7] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *ECCV*, pages 1–22, 2004.

[8] C. de la Higuera, J.-C. Janodet, E. Samuel, G. Damiand, and C. Solnon. Polynomial algorithms for open plane graph and subgraph isomorphisms. *TCS*, 498 :76–99, 2013.

[9] F. Dorn. Planar subgraph isomorphism revisited. In *STACS*, pages 263–274, 2010.

[10] B. Fernando, É. Fromont, and T. Tuytelaars. Mining mid-level features for image classification. *IJCV*, 108(3) :186–203, 2014.

[11] P. Hogeweg. Cellular automata as a paradigm for ecological modelling. *Applied Mathematics and Computation*, 27 :81–100, 1988.

[12] T. Horváth, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. *DMKD*, 21(3) :472–508, 2010.

[13] C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *KER*, 28 :75–105, 2013.

[14] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features : Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.

[15] V. Le Clément de Saint-Marcq, P. Schaus, C. Solnon, and C. Lecoutre. Sparse-Sets for Domain Implementation. In *TRICS*, pages 1–10, 2013.

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2) :91–110, 2004.

[17] R. Marinescu-Ghemeci. Maximum induced matchings in grids. In *Optimization Theory, Decision Making, and Operations Research Applications*, pages 177–187. Springer, 2013.

[18] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *ICVGIP*, pages 722 –729, 2008.

[19] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *ECCV*, volume 3954, pages 490–503, 2006.

[20] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *IEEE CVPR*, pages 1–8, 2007.

[21] B. Ozdemir and S. Aksoy. Image classification using subgraph histogram representation. In *ICPR*, pages 1112–1115, 2010.

[22] A. Prado, B. Jeudy, E. Fromont, and F. Diot. Mining spatiotemporal patterns in dynamic plane graphs. *IDA*, 17 :71–92, 2013.

[23] É. Samuel, C. de la Higuera, and J.-C. Janodet. Extracting plane graphs from images. In *SSPR*, pages 233–243, 2010.

[24] F. Brandão Silva, S. Tabbone, and R. Torres. Bog : A new approach for graph matching. In *ICPR*, pages 82–87, 2014.

[25] C. Solnon, G. Damiand, C. de la Higuera, and J.-C. Janodet. On the complexity of submap isomorphism and maximum common submap problems. *PR*, 2014.

[26] W. Voravuthikunchai, B. Crémilleux, and F. Jurie. Histograms of Pattern Sets for Image Classification and Object Recognition. In *CVPR*, pages 1–8, 2014.

[27] X. Yan and J. Han. gSpan : graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.