Algorithme de construction d'un treillis des concepts formels et de détermination des générateurs minimaux

S. Ben Tekaya* — S. Ben yahia* — Y. Slimani*

* Département des Sciences Informatique
Faculté des Sciences de Tunis, 1060, Tunis, Tunisie
sondess.bentekaya@laposte.net, {sadok.benyahia,yahya.slimani}@fst.rnu.tn

RÉSUMÉ. Dans cet article, nous présentons un algorithme pour le calcul et la construction du treillis des concepts formels et l'identification des générateurs minimaux qui leur sont associés. L'objectif principal de cet algorithme est de produire des bases génériques des règles d'associations, notamment celles de Duquenne-Guigues et Luxemburger. La principale caractéristique de cet algorithme est la construction, de manière simultanée, de l'ensemble des concepts, de leur ordre et de l'ensemble des générateurs minimaux associés à chaque concept.

ABSTRACT. In this article, we present an algorithm to build a formal concept lattice and the identification of its minimal generators. The objective of this algorithm is to produce bases generic of association rules, specifically those of Duquenne-Guigues and Luxemburger. The main characteristic of this algorithm is the construction, in a simultaneous manner, of the set of concepts, their order and of the set of minimal generators associated to each concept.

MOTS-CLÉS : Fouille de Données, Analyse de Concepts Formels, Règles d'associations, Générateur.

KEYWORDS : Data Mining, Formal Concept Analysis, Association rules, Generator.

1. Introduction

La fouille de données est une discipline qui vise à découvrir des régularités, dans des bases de données volumineuses, exprimées sous forme de règles d'associations [1]. Les méthodes ou algorithmes de découverte de règles associatives se caractérisent par leur aspect itératif, ce qui leur confère une complexité exponentielle [1]. Plusieurs approches cherchent à améliorer les performances des méthodes d'extraction et à accroître leur capacité à produire un ensemble réduit mais pertinent de concepts et de règles [7]. Des études ont notamment souligné le nombre prohibitif des règles d'associations extraites et la nécessité de définir des bases pour ces règles [7]. Ces bases constituent des ensembles réduits de règles informatives (prémisse minimale, conséquence maximale) permettant de ne conserver que les règles les plus pertinentes, sans perte d'information. La génération de ces règles d'associations informatives peut se faire par une extraction efficace des concepts et de leurs générateurs minimaux associés.

Plusieurs algorithmes calculent l'ensemble des concepts formels [3], et certains vont jusqu'à la construction du treillis des concepts formels [3]. D'autres algorithmes ne s'intéressent qu'à la détermination des générateurs minimaux, supposant que le treillis est déjà construit [4].

Dans cet article, nous proposons un algorithme qui calcule, au fur et à mesure, l'ensemble des concepts, leur ordre ainsi que l'ensemble des générateurs minimaux associés à chaque concept, et ce en un seul balayage de la base. Le reste de l'article est structuré comme suit : La section 2 introduit les principales notions relatives au problème de construction d'un treillis des concepts formels et rappelle des notions liées à l'identification des générateurs. La section 3 décrit le principe de l'algorithme proposé dans cet article et qui s'inspire des travaux de Nourine *et al.* [5] et Le Floc'h *et al.* [4]. La section 4 illustre le comportement de l'algorithme sur un exemple. La complexité de l'algorithme proposé est étudiée dans la section 5. Enfin, la section 6 conclut l'article en rappelant les principales caractéristiques de l'algorithme proposé.

2. Notions de base

Treillis de concepts formels (de Galois) : Un treillis des concepts (ou treillis de Galois) correspond à un ordre partiel induit par une relation binaire R entre deux ensembles : un ensemble d'objets $\mathcal{O}$ et un ensemble d'attributs $\mathcal{A}$. La relation oRa indique que l'objet o possède l'attribut a [2].

Connexion de Galois : La fonction ϕ associe l'ensemble des attributs à un ensemble d'objets, tandis que ψ, qui représente la fonction duale de ϕ, associe l'ensemble des objets communs à un ensemble d'attributs.

$$\phi : 2^{\mathcal{O}} \rightarrow 2^{\mathcal{A}}, \phi(O) = \{a \in \mathcal{A} | \forall o \in \mathcal{O} : (o, a) \in \mathcal{R}\}$$

$$\psi : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{O}}, \psi(A) = \{o \in \mathcal{O} | \forall a \in \mathcal{A} : (o, a) \in \mathcal{R}\}$$

Le couple d'applications (ϕ,ψ) est une *connexion de Galois* entre l'ensemble des parties de $\mathcal{O}$ et l'ensemble des parties de $\mathcal{A}$ [2].

Générateur minimal : Un générateur minimal g d'un ensemble fermé F est le sous ensemble minimal de F tel que $\phi \circ \psi(g) = F$ et qu'il n'existe pas un générateur g' tel que $g' \subset g$, avec $\phi \circ \psi(g') = F$.

Face : Soit $N(X,Y)$ un concept formé de l'extent X et de l'intent Y et soit $pred_i(N)$ le i^{eme} prédécesseur immédiat de N dans le treillis de Galois. La i^{eme} face du concept N correspond à la différence entre son intent et l'intent de son i^{eme} prédécesseur [6].

Soit p le nombre de prédécesseurs immédiats du concept N. La famille des faces F_N du concept N s'exprime par la relation suivante : $F_N = \{Y - Intent(pred_i(N)), i \in \{1..p\}$

Bloqueur : Soit $G = \{G_1, ..., G_n\}$ une famille de n ensembles. Un bloqueur B de la famille G est un ensemble dont l'intersection avec tous les ensembles $G_i \in G$ est non vide [6]. Un bloqueur B est dit minimal si $B \notin G$ et s'il n'existe aucun bloqueur B' de G inclus dans B [6].

Générateur : Soit le concept N et F_N la famille de ses faces. L'ensemble G des générateurs associés à l'intent Y du concept N, correspondent aux bloqueurs minimaux de la famille des faces F_N [6].

3. Algorithme proposé

Dans cette section, nous présentons un nouvel algorithme, qui s'inspire notamment de la première partie de l'algorithme de Nourine et al. [5], relative à la construction du trie des concepts. Le choix de cet algorithme est basé sur le fait que celui-ci possède la meilleure complexité [3, 5] par rapport aux algorithmes existants à l'heure actuelle. En plus de sa complexité linéaire, l'originalité de cet algorithme réside dans sa façon de découvrir les concepts formels et leur stockage dans un trie. D'un autre côté, cet algorithme présente un certain nombre d'inconvénients que l'on peut résumer comme suit : le calcul séquentiel des concepts puis de leur ordre, le retour systématique à la base pour calculer le graphe de couverture des concepts formels et enfin, sa limitation au calcul de l'ordre.

Pour pallier à ces inconvénients, nous utilisons, dans l'algorithme que nous proposons dans cet article, les étapes de construction du trie pour calculer, au fur et à mesure, l'ensemble des concepts formels, l'ordre entre ces concepts (treillis des concepts formels), ainsi que leurs générateurs minimaux. Les notations utilisées par notre algorithme sont données dans le tableau 1, alors que son pseudo-code est défini dans *Algorithme* 1.

Structure	Champs	Description
$\mathcal{F}$		Famille des concepts formels C
L		Liste des concepts calculés dans une itération i
C		Concept formel
	intent	Intent du concept C
	extent	Extent du concept C
	ImmSucc	Liste des successeurs immédiats du concept C (les concepts qui le couvrent)
	liste_face	Liste des faces du concept C (la différence avec ses prédécesseurs immédiats)
	liste_gen	Liste des générateurs minimaux du concept C

Tableau 1. *Notations*

A chaque itération, l'algorithme construit l'ensemble des concepts, l'ensemble des générateurs minimaux candidats et un ordre potentiel entre les concepts trouvés. Chaque tuple est lu une seule fois et un seul accès à la base de données est nécessaire. Une itération est constituée de deux phases. La première calcule les concepts formels et la génération d'un ordre potentiel des concepts, alors que la deuxième a pour objectif de finaliser l'ordre et calculer les générateurs minimaux.

1: **Algorithme** GENALL ($\mathcal{B}$)
Input : La Base $\mathcal{B}$
Output : L'arbre lexicographique de $\mathcal{F}, ImmSucc, liste\_gen$
Debut
2: $\mathcal{F} = M$ {/*Initialisation de la famille des concepts à l'ensemble des attributs*/}
3: **Pour** chaque tuple $t \in \mathcal{B}$ **Faire**
4: $L = \emptyset$ {/*Initialisation de la liste de l'ensemble des concepts trouvés dans une itération*/}
5: **Pour** chaque concept $C_i \in \mathcal{F}$ **Faire**
6: $C.intent = C_i.intent \cap t$
7: **Si** $C.intent \notin \mathcal{F}$ **Alors**
8: {/*Nouveau concept*/}
9: $\mathcal{F} = \mathcal{F} \cup C$
10: $C.extent = C_i.extent \cup t.TID$
11: $C.ImmSucc = \{t\} \cup \{C_i\} \setminus \{C\}$
12: $L = L \cup C$ {/*Insertion triée*/}
13: **Sinon**
14: {/*Concept existant*/}
15: $C.extent = C.extent \cup C_i.extent \cup t.TID$
16: **Si** $C \notin L$ **Alors**
17: $L = L \cup C$ {/*Insertion triée*/}
18: $C.LP = \{t\} \cup \{C_i\} \setminus \{C\}$
{/*Mise à jour de $C.ImmSucc$*/}
19: **Pour** chaque $P_i \in C.LP$ **Faire**
20: **Pour** chaque $Succ \in C.ImmSucc$ **Faire**
21: COMPARE-CONCEPT($Succ, P_i$)
{/*Finalisation de l'ensemble des successeurs immédiats et calcul des générateurs minimaux*/}
22: **Pour** chaque concept $C_i \in L$ **Faire**
23: $C_i.ImmSucc$ = FIND-SUCC(C_i, L)
24: **Pour** Chaque $Succ \in C_i.ImmSucc$ **Faire**
25: $face = Succ \setminus C_i$ {/*Calcul de la face du successeur immédiat*/}
26: **Pour** Chaque $face_i \in Succ.liste\_face$ **Faire**
27: $Succ.liste\_face$ = COMPARE_FACE($face, face_i$)

Algorithme 1: Construction du treillis décoré par les générateurs minimaux

3.1. Calcul des concepts

Dans cette phase (lignes 4-21), à chaque itération, nous initialisons une liste L à l'ensemble vide. Cette liste sera utile pour la finalisation de l'ensemble des successeurs immédiats de chaque concept

trouvé lors d'une itération. Pour calculer l'ensemble des concepts, nous faisons une intersection entre l'intent de chaque concept de la famille $\mathcal{F}$ et chaque tuple de la base. Deux cas sont possibles :

1) L'intent n'existe pas dans la famille (un nouveau concept est trouvé) : il doit alors être ajouté. Ensuite, l'extent est calculé d'une façon incrémentale (ligne 10). Les successeurs immédiats potentiels de ce nouveau concept seront les concepts qui l'ont produit (ligne 11). Par la suite, ce nouveau concept est ajouté à la liste L (ligne 12). Notons que l'insertion dans cette liste L assure que la liste L est triée selon l'ordre croissant des intents des concepts.

2) L'intent existe dans la famille : l'extent du concept doit être mis à jour (ligne 15), et il faut vérifier si le concept existe déjà dans la liste L (lignes 16-17). Dans le but de mettre à jour la liste des successeurs immédiats du concept, et vu que le concept possède une liste $ImmSucc$, nous construisons une liste LP contenant les concepts qui l'ont produit une nouvelle fois. Cette liste est nécessaire pour pouvoir faire les comparaisons et la mise à jour de la liste $ImmSucc$. En effet, pour chaque élément appartenant à LP et pour chaque élément de la liste $ImmSucc$ du concept, nous testons l'inclusion des concepts à l'aide de la fonction COMPARE-CONCEPT (ligne 21).
Cette fonction a pour objectif de faire la mise à jour de la liste $ImmSucc$ du concept trouvé. Cette liste sera modifiée dans deux cas :

1) L'élément de la liste LP est plus petit (en terme d'inclusion) que celui de $ImmSucc$: dans ce cas, l'ancien successeur sera remplacé par le nouveau,

2) Les deux éléments sont incomparables : le nouveau successeur sera alors ajouté à la liste $ImmSucc$ de ce concept.

3.2. Finalisation des successeurs immédiats et calcul des générateurs minimaux

Nous remarquons que les concepts, trouvés lors d'une itération, représentent une branche du treillis, c'est à dire que pour chaque concept (sauf le plus grand), nous trouvons un autre concept, calculé dans cette même itération, qui le couvre. Pour cela, nous parcourons la liste L des concepts trouvés lors d'une itération et pour chacun des concepts de L nous ferons appel à la fonction FIND-SUCC (lignes 22-23). Cette fonction, est basée sur le fait qu'un concept de cardinalité n est couvert par un concept de cardinalité $(n+1)$ ou plus. Pour cela, et étant donné que la liste L est triée selon l'ordre croissant de l'intent, nous cherchons pour chaque concept de cardinalité n, un concept qui le couvre dans la liste L de cardinalité $(n+1)$. En cas de défaut, nous passons à la cardinalité $(n+2)$, jusqu'à ce que l'on trouve un concept qui le couvre. Ensuite, nous comparons les concepts en vu de mettre à jour la liste $ImmSucc$. Une fois la liste des successeurs immédiats ($ImmSucc$) du concept courant construite, nous parcourons cette liste et pour chaque successeur nous calculons l'ensemble de ses générateurs minimaux. Pour obtenir un tel résultat, nous calculons la face du successeur (ligne 25), puis nous la comparons avec la liste des faces qui lui correspondent en faisant appel à la fonction COMPARE-FACE (ligne 27). L'ensemble des faces du concept (successeur immédiat) peut être modifié dans deux cas définis comme suit :

1) La $face$ est plus petite (en terme d'inclusion) qu'un élément de la $liste\_face$: dans ce cas, nous calculons la $difference\_face$, et nous remplaçons l'ancienne face par la nouvelle. Si

$difference\_face$ n'existe pas dans une des faces de ce concept, alors nous supprimons chaque générateur contenant cette différence car un attribut qui n'existe pas dans les faces ne peut pas exister dans les générateurs.

2) La $face$ est non comparable avec toutes les faces de $liste\_face$: dans ce cas, elle sera ajoutée dans $liste\_face$.

Lorsque la liste des faces est mise à jour, nous utilisons l'algorithme JEN décrit dans [4] pour le calcul des générateurs minimaux.

4. Exemple d'application

Soit la base de transactions donnée par la figure 1 avec l'ensemble d'attributs $\mathcal{A} = \{a, b, c, d, e, f, g, h, i\}$ et l'ensemble des tuples de la base, notés de 1 à 8.

$\mathcal{R}$	a	b	c	d	e	f	g	h	i
1	×	×					×		
2	×	×					×	×	
3	×	×	×				×	×	
4	×		×				×	×	×
5	×	×		×		×			
6	×	×	×	×		×			
7	×		×	×	×				
8	×		×	×		×			

Figure 1. *Base de transactions B.*

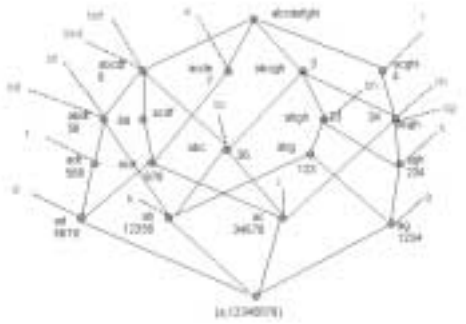

Figure 2. *Treillis des concepts formels décoré par l'ensemble des générateurs minimaux.*

A titre d'exemple, nous considérons le tuple $4 = \{acghi\}$ et la famille obtenue lors des trois itérations précédentes :
$\mathcal{F} = \{\{abcdefghi, \emptyset\}, \{abg, 123\}, \{abgh, 23\}, \{abcgh, 3\}\}$ avec l'ordre suivant :
$\{abg\}.ImmSucc = \{abgh\}$; $\{abgh\}.ImmSucc = \{abcgh\}$; $\{abcgh\}.ImmSucc = \{abcdefghi\}$.
L'application de l'algorithme 1 donne les résultats partiels suivants :

Première phase : Calcul des concepts

Pour $C_1 = \{abcdefghi\}$ l'application de l'opération d'intersection avec le tuple donne un nouveau concept $\{acghi\}$.
La famille devient donc : $\mathcal{F} = \{\{abcdefghi, \emptyset\}, \{abg, 123\}, \{abgh, 23\}, \{abcgh, 3\}, \{acghi\}\}$.
Le calcul de l'extent du concept trouvé donne : $\{acghi\}.extent = \{4\}$.
$\{acghi\}.ImmSucc = \{abcdefghi\}$
$L = \{acghi\}$.

Nous répétons ce processus avec tous les concepts existants dans la famille.
A la fin de cette première phase, nous obtenons la famille suivante :
$\mathcal{F} = \{(abcdefghi, \emptyset), (abg, 123), (abgh, 23), (abcgh, 3), (acghi, 4), (ag, 1234), (agh, 234), (acgh, 34)\}$.
L'ordre obtenu est : $(acghi).ImmSucc = (abcdefghi)$; $(ag).ImmSucc = (abg), (acghi)$;
$(agh).ImmSucc = (abgh), (acghi)$; $(acgh).ImmSucc = (abcgh), (acghi)$
La liste L contient $\{(ag), (agh), (acgh), (acghi)\}$.

Deuxième phase : Finalisation de l'ordre et calcul des générateurs minimaux

Pour le premier concept de la liste L, nous avons $(ag).ImmSucc = (abg), (acghi)$. Le concept (agh), qui est de cardinalité $(n+1)$, couvre (ag) et $(agh) \subset (acghi)$. Il faut donc remplacer $(acghi)$ par (agh) ; en effet l'ancien lien avec $(acghi)$ est transitif $\Rightarrow (ag).ImmSucc = (abg), (agh)$.
Pour chaque successeur nous calculons $liste\_faces$ et $liste\_gen$, ce qui nous donne :
$(abg).liste\_face = (b)$ d'où $(abg).liste - gen = (b)$.
Le même traitement est répété pour le successeur (agh) : $(agh).liste\_face = (h)$ et $(agh).liste\_gen =$
$(agh).ImmSucc = (abgh), (acghi)$ or $(acgh) \subset (acghi) \Rightarrow$ remplacer $(acghi)$ par $(acgh)$ ce qui donne $(agh).ImmSucc = (abgh), (acgh)$.
$(abgh).liste\_face = (h) \cup (b)$
$(h) \cap (b) = \emptyset$: donc (h) n'est pas bloqueur pour l'ensemble des faces $\{(b), (b)\}$
Construction de l'ensemble des bloqueurs de cette nouvelle famille :
$Gen = (bh) \Rightarrow (abgh).liste\_gen = (bh)$.
Le même travail est répété pour l'autre successeur.
$(acghi).ImmSucc = (abcdefghi)$.
$(abcdefghi).liste\_face = (defi) \cup (bdef) \Rightarrow$ plusieurs fils $(abcdefghi).liste\_gen = (d, e, f, i)$.
Pour chaque générateur g de $liste\_gen$ nous calculons son intersection avec la nouvelle face.
Pour $(i) \cap (bdef) = \emptyset$ d'où (i) n'est pas bloqueur $\Rightarrow$ calcul de l'ensemble $Gen = (bi, di, ei, fi)$.
Elimination des bloqueurs non minimaux de Gen : $d \subset di, e \subset ei, f \subset fi \Rightarrow$
$(abcdefghi).liste\_gen = (d, e, f, bi)$

Dans la figure 2, nous présentons le treillis des concepts formels complet avec indication de quelques générateurs minimaux représentés en pointillé.

5. Complexité

Dans cette section, nous allons étudier la complexité dans le pire des cas de l'algorithme que nous avons proposé. Soient, $\mathcal{B}$ l'ensemble des tuples d'une base, X l'ensemble des attributs de la base et $\mathcal{F}$ l'ensemble des concepts trouvés. L'étude de la boucle **Pour** (ligne 5) donne : l'instruction 6 s'effectue en $O(|X|)$ étant donné que dans le pire des cas, nous allons effectuer une intersection avec tous les attributs de la base. L'instruction 7 se fait en $O(|X|)$ puisqu'on peut avoir au maximum $|X|$ branches. Par ailleurs, la complexité du bloc d'instructions 9-12 (partie **alors**) étant inférieure à celle du bloc d'instructions 15-21, nous ne comptabiliserons que cette dernière : l'instruction 15 peut se faire dans le pire des cas en $O(|B|)$. La recherche dans la liste L (instruction 16) est réalisée en $O(\frac{|\mathcal{F}|}{2})$ dans le pire des cas. En effet, une itération peut donner autant de nouveaux concepts qu'il y a dans $|\mathcal{F}|$. La boucle

Pour (ligne 19) peut se répéter au maximum deux fois. Celle de la ligne 20 se répète $|ImmSucc|$ fois et la fonction COMPARE_CONCEPT se fait en $O(|X|)$. De ce fait, la complexité des instructions 5-21 est en $O(|\mathcal{F}| * 2|X| + |\mathcal{B}| + \frac{|\mathcal{F}|}{2} + 2|X|.|ImmSucc|)$. L'instruction 23 s'effectue en $O(|ImmSucc|)$, alors que la complexité de l'instruction 27 est en $O(|X| + |liste\_gen|)$. Par conséquent, la complexité du bloc comprenant les instructions 22 à 27 est en $O(\frac{|\mathcal{F}|}{2} * |ImmSucc|^2.(|X| + |liste\_gen|))$. Puisque notre algorithme se répète $|\mathcal{B}|$ fois (boucle **Pour** de la ligne 3), la complexité totale de notre algorithme est alors en $O(|\mathcal{B}|.|\mathcal{F}|.[\frac{1}{2}.|ImmSucc|^2.(|X| + |liste\_gen|) + |\mathcal{B}| + \frac{|\mathcal{F}|}{2}])$. Malgré le fait que notre algorithme donne plus de résultats (calcul de l'ensemble des concepts, leur ordre et l'ensemble des générateurs minimaux associés à chaque concept) que les algorithmes de Nourine et al. [5] et Le Floc'h et al. [4], sa complexité reste toujours linéaire.

6. Conclusion

Dans cet article, nous avons proposé un nouvel algorithme pour la construction **à la fois** de l'ensemble des concepts, le treillis de Galois et la détermination de l'ensemble des générateurs minimaux associés à chaque concept en nous inspirant des travaux de Nourine et al. [5] et Le Floc'h et al. [4]. Les caractéristiques de ce nouvel algorithme sont : **un balayage unique de la base** et l'obtention des **concepts formels**, le calcul de leur **ordre** ainsi que les **générateurs minimaux** associés. En utilisant cet algorithme, la dérivation des bases génériques pour les règles d'association devient évidente.

7. Bibliographie

[1] R. AGRAWAL , R. SKIRANT, « Fast algorithms for mining association rules », *In Proceeding of the 20th International Conference on Very Large Database, pages 478-499, June 1994.*

[2] B. GANTER , R. WILLE, « Formal Concept Analysis : Mathematical Foundations », *Springer-Verlag, Berlin Heidelberg, 1999.*

[3] S.O. KUZNETSOV, S.A. OB"EDKOV,« Comparing Performance of Algorithms for Generating Concept Lattices » *ICCS'01 Int'l. Workshop on Concept Lattices-Based KDD, pages 35-47,Stanford USA July 2001.*

[4] A. LE FLOC"H , C. FISETTE, R. MISSAOUI, P. VALTCHEV, R. GODIN, « JEN : un algorithme efficace de construction de générateurs pour l'identification des règles d'association », *In Proceedings of the XXXVemes Journees de Statistique, Lyon, France Juin 2003.*

[5] L. NOURINE, O. RAYNAUD, « A fast algorithm for building lattices », *Information Processing Letters, 71, pages 199-204, 1999.*

[6] J. L. PFALTZ, C. M. TAYLOR, « Scientific Discovery through Iterative Transformation of Concept Lattices », *Workshop on Discrete Applied Mathematics in conjunction with the 2nd SIAM International Conference on Data Mining, pages 65-74, Arlington, VA, April 2002.*

[7] G. STUMME, R. TAOUIL, Y. BASTIDE, N. PASQUIER, L. LAKHAL, « Intelligent structuring and reducing of association rules with formal concept analysis », *In Proceedings of KI'2001 conference, Lecture Notes in Artificial Intelligence 2174, Springer-Verlag, pages 335-350, september 2001.*