

# GraphMDL : sélection de motifs de graphes avec le principe MDL

Francesco Bariatti, Peggy Cellier, Sébastien Ferré

Univ Rennes, INSA, CNRS, IRISA  
Prénom.Nom@irisa.fr

**Résumé.** Plusieurs algorithmes de fouille de motifs ont été proposés pour identifier des structures récurrentes dans les graphes. Le principal défaut de ces approches est qu'elles produisent généralement trop de motifs pour qu'une analyse humaine soit possible. Récemment, des méthodes de fouille de motifs ont traité ce problème sur des données transactionnelles, séquentielles et relationnelles en utilisant le principe MDL (*Minimum Description Length*). Dans ce papier, nous proposons une approche MDL pour sélectionner un sous-ensemble représentatif de motifs sur des graphes étiquetés. Une notion clé de notre approche est l'introduction de *ports* pour encoder les connections entre occurrences de motifs, sans perte d'information. Nos expériences montrent que le nombre de motifs est drastiquement réduit et que les motifs sélectionnés peuvent avoir des formes complexes.

## 1 Introduction et état de l'art

Baucoup de domaines utilisent des données représentées par des graphes. Par exemple, en chimie et biologie, les molécules sont représentées par des graphes avec des atomes et des liaisons ; en linguistique, les phrases sont représentées par des graphes avec des mots et des liens de dépendance ; en web sémantique, la connaissance est représentée par un graphe avec des entités et des relations. En fonction du domaine, le jeu de données est un graphe unique ou une collection de graphes. Ces graphes sont intrinsèquement complexes à analyser pour en extraire de la connaissance, par exemple pour identifier des sous-structures fréquentes.

Dans le domaine de la fouille de motifs, plusieurs approches de *fouille de graphes* pour extraire des sous-graphes fréquents ont été proposées. Les approches classiques de fouille de graphes telles que MoFa (Borgelt et Berthold, 2002), gSpan (Yan et Han, 2002), Gaston (Nijssen et Kok, 2005) ou FFSM (Huan et al., 2003), génèrent tous les motifs possibles par rapport à une fréquence minimale. Le défaut majeur de ce type d'approches est le grand nombre de motifs générés, qui rend difficile leur analyse. Certaines approches comme CloseGraph (Yan et Han, 2003) réduisent le nombre de motifs en ne générant que les *motifs clos*. Cependant, le nombre de motifs reste généralement trop élevé, avec beaucoup de redondance entre motifs. Les algorithmes *s'appuyant sur les contraintes* comme gPrune (Zhu et al., 2007), essaient quant à eux de réduire la quantité de motifs générés en n'extrayant que des motifs qui suivent une certaine règle d'acceptation. Ces algorithmes réussissent généralement à limiter le nombre

de motifs, cependant ils limitent aussi le type de motifs générés. De plus, si les règles d'acceptation doivent être définies par l'utilisateur, celui-ci doit avoir une connaissance *a priori* du domaine.

Les méthodes les plus efficaces en terme de réduction du nombre de motifs sont les approches s'appuyant sur le principe MDL (*Minimum Description Length*) (Grünwald, 2000). Ce principe tient ses origines de la théorie de l'information et définit que le *modèle* qui décrit le mieux les données est celui qui les compresse le plus. Ces approches ont montré leur efficacité pour sélectionner des ensembles de motifs sur des données ensemblistes (Vreeken et al., 2011), séquentielles (Tatti et Vreeken, 2012) et relationnelles (Koopman et Siebes, 2009). Peu d'approches utilisant le principe MDL ont été proposées pour des graphes. SUBDUE (Cook et Holder, 1993) compresse itérativement un graphe en remplaçant chaque occurrence de motif par un unique sommet. À chaque itération, le motif choisi est celui qui compresse le plus. Un inconvénient de SUBDUE est que le remplacement d'occurrences de motifs par des sommets entraîne des pertes d'information. VoG (Koutra et al., 2015) «résume» les graphes comme étant une composition de motifs dont la forme est prédéfinie (e.g. chemins, étoiles). Similairement à SUBDUE, VoG vise à n'extraire que des motifs «intéressants», mais contrairement à SUBDUE il évalue l'ensemble des motifs générés comme un tout, plutôt que d'évaluer chaque motif individuellement. Ceci permet à l'algorithme de chercher un «bon ensemble» de motifs, plutôt qu'un ensemble de «bons motifs». Une limitation de VoG est que le type de motifs extraits est limité à des formes de motifs prédéfinies. Une deuxième limitation est qu'il travaille avec des graphes non étiquetés (e.g. des réseaux), alors que les graphes qui nous intéressent sont étiquetés.

La contribution de ce papier (section 3) est une nouvelle approche, appelée GRAPHMDL, s'appuyant sur le principe MDL pour sélectionner des motifs de graphes étiquetés. Contrairement à SUBDUE, GRAPHMDL garantit qu'il n'y a pas de perte d'information grâce à l'introduction de *ports* associés aux motifs de graphes. Ces ports représentent comment les occurrences adjacentes de motifs sont inter-connectées. Nous évaluons notre approche de manière expérimentale (section 4) sur deux jeux de données de nature différente : un jeu de données sur des molécules (peu d'étiquettes, beaucoup de cycles) et un jeu de données représentant des arbres de dépendances (beaucoup d'étiquettes, pas de cycles). Les expériences valident notre approche en montrant que les données sont drastiquement compressées et que le nombre de motifs sélectionnés est nettement inférieur au nombre de motifs candidats. De plus, nous observons que les motifs peuvent avoir des formes complexes et variées.

## 2 Préliminaires

Nous rappelons tout d'abord quelques définitions concernant le principe MDL, les graphes et les motifs de graphe.

### 2.1 Principe MDL

Le principe de *longueur de description minimale* ou MDL (Grünwald, 2000) est souvent résumé par le slogan «l'induction par la compression». Le principe MDL définit que le meilleur modèle  $M$  pour décrire des données  $D$  est celui qui minimise la *longueur de description*  $L(M, D) = L(M) + L(D|M)$  où  $L(M)$  est la longueur du modèle et  $L(D|M)$  la

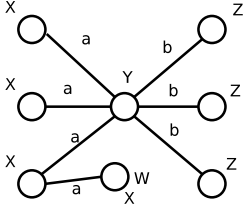


FIG. 1 – Un graphe simple non orienté étiqueté.

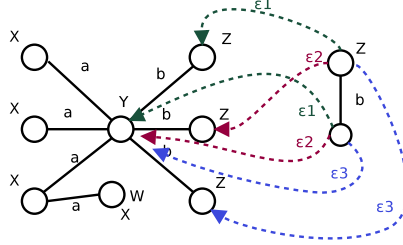


FIG. 2 – Occurrences d'un motif dans le graphe de la figure 1.

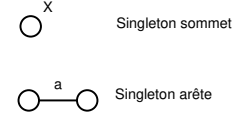


FIG. 3 – Deux motifs singletons.

longueur des données encodées avec le modèle. Le principe MDL ne dit pas comment calculer les longueurs de description. Toutefois, il existe quelques primitives usuelles pour des données et distributions élémentaires (Lee, 2001) :

- Un élément  $x$  d'un ensemble  $\mathcal{X}$  avec distribution uniforme a un code de  $\log(|\mathcal{X}|)$  bits.
- Un élément  $x$  d'un ensemble  $\mathcal{X}$  qui apparaît  $usage(x, D)$  fois dans des données  $D$  a un code de  $L_{usage}^{\mathcal{X}}(x, D) = -\log\left(\frac{usage(x, D)}{\sum_{x_i \in \mathcal{X}} usage(x_i, D)}\right)$  bits. Ce codage est optimal.
- Un entier  $n$  non borné est encodé avec un *encodage universel d'entiers*. Nous notons  $L_{\mathbb{N}}(n)$  la taille en bits de cet encodage<sup>1</sup>.

Dans le domaine de la fouille de motifs, l'algorithme Krimp (Vreeken et al., 2011) utilise le principe MDL pour sélectionner un ensemble «descriptif» de motifs sur des données transactionnelles. Une base de données transactionnelle est une liste de transactions, chaque transaction étant un ensemble d'éléments (d'un vocabulaire propre à la base). Les motifs extraits par Krimp sont des ensembles d'éléments qui apparaissent fréquemment ensemble dans les transactions. Au vu de son efficacité, Krimp a été adapté à d'autres types de données, comme des séquences (Tatti et Vreeken, 2012) et des données relationnelles (Koopman et Siebes, 2009). Notre approche s'inspire de Krimp pour la sélection de motifs, tout en redéfinissant ses concepts-clés sur des graphes.

## 2.2 Graphes et motifs de graphe

**Définition 1** Un graphe étiqueté  $G = (V, E, l_V, l_E)$  défini sur deux ensembles d'étiquettes  $\mathcal{L}_V$  et  $\mathcal{L}_E$  est une structure composée d'un ensemble de sommets  $V$ , d'un ensemble d'arêtes  $E \subseteq V \times V$  qui les connectent, et de deux fonctions d'étiquetage  $l_V \in V \rightarrow 2^{\mathcal{L}_V}$  et  $l_E \in E \rightarrow \mathcal{L}_E$  qui associent un ensemble d'étiquettes à chaque sommet et une étiquette à chaque arête.  $G$  est dit non orienté si  $E$  est symétrique et simple si  $E$  est non réflexive.

Bien que notre approche s'applique à tout graphe étiqueté, dans la suite nous considérons des graphes simples non orientés, pour nous comparer avec les outils existants. La figure 1 montre un exemple de graphe, avec 8 sommets et 7 arêtes, défini sur un ensemble  $\{W, X, Y, Z\}$  d'étiquettes de sommet et un ensemble  $\{a, b\}$  d'étiquettes d'arête. Dans notre définition les sommets peuvent avoir entre 0 et  $|\mathcal{L}_V|$  étiquettes. Ceci n'est pas toujours le cas

<sup>1</sup>. Dans notre implémentation nous utilisons le «Elias gamma encoding» (Elias, 1975), décalé de 1 pour qu'il puisse encoder 0. Soit  $L_{\mathbb{N}}(n) = 2\lfloor \log(n+1) \rfloor + 1$ .

GraphMDL : Sélection de motifs de graphes avec le principe MDL

dans les définitions usuelles de graphe : nous adoptons cette définition car plus générique et donc potentiellement applicable à un plus grand nombre de jeux de données.

**Définition 2** Soient  $G^P$  et  $G^D$  des graphes. Une occurrence de  $G^P$  dans  $G^D$  est une fonction injective  $\varepsilon \in V^P \rightarrow V^D$  telle que

$$\begin{aligned} - l_V^P(v) &\subseteq l_V^D(\varepsilon(v)) && \forall v \in V^P \\ - (u, v) \in E^P &\Rightarrow (\varepsilon(u), \varepsilon(v)) \in E^D && \forall (u, v) \in E^P \\ - l_E^P(e) &= l_E^D(\varepsilon(e)) && \forall e \in E^P \end{aligned}$$

Les *motifs de graphe*<sup>2</sup> sont définis comme étant des graphes  $G^P$  ayant des occurrences dans le graphe  $G^D$  qui représente les données. La figure 2 montre les trois occurrences  $\varepsilon_1, \varepsilon_2, \varepsilon_3$  dans le graphe de la figure 1 d'un motif ayant deux sommets, dont un étiqueté par  $Z$ , et un arc les reliant étiqueté par  $b$ .

Les *motifs singletons* sont nos briques élémentaires pour construire des motifs. Il y a deux types de singletons. Un *motif singleton sommet* est un graphe avec un seul sommet ayant une seule étiquette. Un *motif singleton arête* est un graphe avec deux sommets sans étiquette, connectés par une seule arête étiquetée. La figure 3 montre un exemple de motif singleton pour chaque type.

### 3 GRAPHMDL : le principe MDL appliqué aux graphes

Dans cette section nous présentons notre contribution : l'approche GRAPHMDL. Cette approche prend en entrée un graphe —le *graphe original*  $G^o$ — et un ensemble de motifs extraits de ce graphe —les *motifs candidats*— et elle retourne le sous-ensemble le plus descriptif de ces motifs selon le principe MDL. Les motifs candidats peuvent être générés avec tout algorithme de fouille de graphes déjà existant comme gSpan (Yan et Han (2002)).

L'intuition derrière GRAPHMDL est que, comme les données et les motifs sont tous deux des graphes, les données peuvent être vues comme une composition d'occurrences de motifs. Informellement, nous voulons qu'un utilisateur analysant les résultats de GRAPHMDL puisse dire : «Les données sont composées d'une occurrence du motif A, connectée à une occurrence du motif B, elle-même connectée à une occurrence du motif C». De plus, nous voulons que l'utilisateur soit capable de dire *comment* ces motifs sont connectés : quels sommets de chaque motif font la connexion avec les autres motifs.

#### 3.1 Une «code table» pour graphes

Similairement à Krimp, nous définissons notre modèle comme étant une *Code Table* (CT), où un ensemble de motifs  $\mathcal{P}$  a des informations d'encodage associées. La première différence avec les CT de Krimp est que les motifs sont des motifs de graphe. La deuxième différence est le besoin d'informations supplémentaires associées à chaque motif : un code seul ne serait pas suffisant, car toute l'information liée à la connectivité entre occurrences de motifs serait perdue. Par exemple, le graphe de la figure 1 a quatre sommets  $X$ , un sommet  $Y$ , etc : il y a une multitude de graphes possibles avec ces mêmes caractéristiques.

2. Appelés «patterns» en anglais, d'où l'utilisation de l'indice  $P$  pour les indiquer

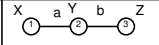











P	$G^P$	Usage motif	Code motif	$c_P$ Longueur code motif (bits)	Nombre de ports	$v_\Pi$		$c_\Pi$	
						Id port	Usage port	Code port	Longueur code port (bits)
P1		3		1	2	v1 v2	1 3	 	2 0.42
Pa		1		2.58	2	v1 v2	1 1	 	1 1
Pw		1		2.58	1	v1	1		0
Px		1		2.58	1	v1	1		0

FIG. 4 – Exemple d’une code table de GRAPHMDL associée au graphe de la figure 1. Les usages des motifs et ports et les longueurs de code ont été ajoutés à des fins illustratifs. Les motifs singletons inutilisés ont été omis.

C’est pourquoi nous introduisons la notion de *ports*, pour représenter comment les occurrences des motifs se connectent entre elles pour former le graphe original. Les ports d’un motif sont un sous-ensemble des sommets de celui-ci. Intuitivement, un sommet du motif est un port si au moins une des occurrences du motif le projette sur un sommet du graphe original qui est aussi utilisé par une autre occurrence de motif (que ce soit du même motif ou d’un autre). Par exemple, dans la figure 5a les trois occurrences du motif  $P1$  se superposent sur leur deuxième sommet : ce sommet est un port. Puisque les ports augmentent la longueur de description, nous nous attendons à ce que l’approche MDL sélectionne des motifs avec peu de ports.

La figure 4 montre un exemple de CT associée au graphe de la figure 1. Chaque ligne de la CT est composée de trois parties et contient les informations d’un motif  $P \in \mathcal{P}$  (e.g. la première ligne contient les informations d’un motif  $P1$ ). La première partie d’une ligne est le graphe  $G^P$  représentant la structure du motif (e.g.  $P1$  est un motif avec trois sommets et deux arêtes). La deuxième partie est le code  $c_P$  associé au motif (e.g.  $P1$  a un code de 1 bit, dû à son usage). La troisième partie est la description de l’ensemble de ports  $\Pi_P$  du motif. La description de chaque port  $\pi \in \Pi_P$  est elle-même composée de deux parties : le numéro du sommet du motif  $v_\pi$  et le code  $c_\pi$  associé au port (e.g.  $P1$  a 2 ports, ses deux premiers sommets, avec des codes respectivement de 2 et 0.42 bits<sup>3</sup>). On note  $\Pi$  l’ensemble de tous les ports de tous les motifs. Comme dans Krimp, la longueur du code d’un motif ou celle d’un port dépend de son usage dans l’encodage des données, c’est-à-dire combien de fois cette structure est utilisée pour décrire le graphe original  $G^o$  (voir sections 3.2 et 3.3).

### 3.2 Encoder des données avec une code table

L’intuition derrière GRAPHMDL est que nous pouvons représenter le graphe original  $G^o$  (i.e. les données) comme un ensemble d’occurrences de motifs, connectées au travers de ports. Encoder les données avec une CT consiste à créer une structure qui explicite quelles occurrences sont utilisées et comment elles se connectent pour former le graphe original. On appelle cette structure le *graphe réécrit*  $G^r$ .

**Définition 3** Un graphe réécrit  $G^r = (V^r, E^r, l_V^r, l_E^r)$  est un graphe où  $V^r = V_{emb}^r \cup V_{port}^r$  est l’ensemble des sommets :  $V_{emb}^r$  est l’ensemble des sommets occurrence et  $V_{port}^r$  est l’ensemble

3. Les approches MDL s’intéressent aux longueurs *théoriques*, qui peuvent donc avoir des valeurs non entières.

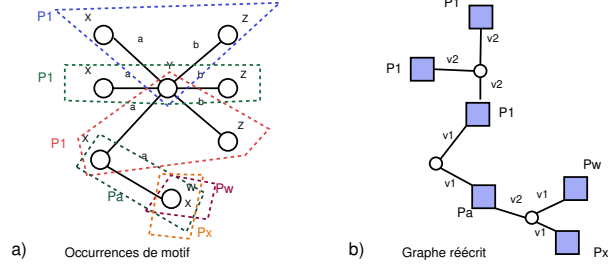


FIG. 5 – Graphe de la figure 1 encodé avec la code table de la figure 4. a) Les occurrences retenues des motifs de la CT. b) Le graphe réécrit. Les carrés bleus sont des occurrences de motif (leur étiquette indique quel motif), les cercles blancs sont les sommets ports. Les étiquettes d’arêtes indiquent à quel port du motif les sommets ports correspondent.

des sommets port.  $E^r \subseteq V_{emb}^r \times V_{port}^r$  est l’ensemble des arêtes des occurrences vers les ports,  $l_V^r \in V_{emb}^r \rightarrow \mathcal{P}$  et  $l_E^r \in E^r \rightarrow \Pi$  sont les fonctions d’étiquetage.

Le calcul de l’encodage du graphe original avec une CT commence avec un graphe réécrit vide. L’un après l’autre, les motifs de la CT sont sélectionnés. Pour chaque motif, les occurrences de son graphe  $G^P$  sont calculées. Similairement à Krimp, nous limitons les superpositions d’occurrences : nous autorisons la superposition sur les sommets (puisque c’est le cœur de la notion de ports), mais pas les superpositions sur les arêtes. Chaque occurrence retenue est représentée dans le graphe réécrit par un *sommet occurrence* : un sommet  $v_e \in V_{emb}^r$  avec une étiquette  $P \in \mathcal{P}$  qui indique le motif qu’il instancie.

Les sommets du graphe original qui sont partagés par plusieurs occurrences sont représentés dans le graphe réécrit par un *sommet port*  $v_p \in V_{port}^r$ . Une arête  $(v_e, v_p) \in E^r$ , entre le sommet occurrence  $v_e$  d’un motif  $P$  et le sommet port  $v_p$ , est ajoutée au graphe réécrit quand l’occurrence associée à  $v_e$  projette le port  $v_\pi \in \Pi_P$  du motif sur  $v_p$ . Cette arête est étiquetée  $v_\pi$ .

Toute CT doit inclure tous les motifs singletons, pour garantir que tout sommet ou arête du graphe original puisse être encodé.

La figure 5 montre le résultat de l’encodage du graphe de la figure 1 avec la CT de la figure 4. Les occurrences des motifs de la code table deviennent des sommets occurrence dans le graphe réécrit (carrés bleus). Les sommets qui sont à la frontière entre plusieurs occurrences de motif sont traduits en sommets ports dans le graphe réécrit (cercles blancs). Quand une occurrence a un port, dans le graphe réécrit son sommet occurrence est connecté au sommet port correspondant et l’étiquette sur l’arête indique de quel port du motif il s’agit. Par exemple, les trois occurrences retenues du motif  $P1$  partagent toutes le même sommet étiqueté  $Y$  (au centre du graphe original), donc dans le graphe réécrit les trois sommets occurrence correspondants sont tous connectés au même sommet port.

### 3.3 Longueurs de description

Dans cette section nous définissons comment calculer la longueur de description de la CT et du graphe réécrit. Ces longueurs de description sont utilisées pour comparer les CT entre elles. Les formules sont expliquées ci-dessous et regroupées dans la figure 6.

$$\begin{aligned}
L(c_P) &= L_{usage}^P(P, G^r) \quad \text{avec } usage(P_i, G^r) = |\{v_e \in V_{emb}^r \mid l_V^r(v_e) = P_i\}| \\
L(c_\pi, P) &= L_{usage}^{\Pi_P}(\pi, G^r) \quad \text{avec } usage(\pi_i, G^r) = |\{e \in E_{emb}^r \mid l_E^r(e) = \pi_i\}| \\
L(M) = L(CT) &= \sum_{\substack{P \in \mathcal{P} \\ usage(P, G^r) \neq 0}} \underbrace{L(G)}_{\text{structure}} + \underbrace{L(c_P)}_{\text{code}} + \underbrace{L(\Pi_P)}_{\text{ports}} \\
\left| \begin{aligned}
L(G) &= \underbrace{L_{\mathbb{N}}(|V|)}_{\text{nb sommets}} + \sum_{v \in V} [ \underbrace{L_V(v, G)}_{\text{étiquettes du sommet}} + \underbrace{L_E(v, G)}_{\text{arêtes du sommet}} ] \\
L_V(v, G) &= \underbrace{L_{\mathbb{N}}(|l_V(v)|)}_{\text{nb étiquettes}} + \sum_{l \in l_V(v)} \underbrace{L_{usage}^{\mathcal{L}_V}(l, G^o)}_{\text{code étiquette}} \\
L_E(v, G) &= \underbrace{\log(|V|)}_{\text{nb arêtes}} + \sum_{(v,w) \in E \mid v < w} [ \underbrace{\log(|V|)}_{\text{destination}} + \underbrace{L_{usage}^{\mathcal{L}_E}(l_E(v, w), G^o)}_{\text{étiquette}} ] \\
L(\Pi_P) &= \underbrace{\log(|V| + 1)}_{\text{nb ports } |\Pi_P|} + \underbrace{\log(C_{|V|}^{|\Pi_P|})}_{\text{identités ports}} + \sum_{\pi \in \Pi_P} \underbrace{L(c_\pi, P)}_{\text{code port}}
\end{aligned} \right. \\
L(D|M) = L(G^r) &= \underbrace{L_{\mathbb{N}}(|V_{port}^r|)}_{\text{nb sommets ports}} + \sum_{v \in V_{emb}^r} L_{emb}(v, P, G^r) \quad \text{avec } P = l_V^r(v) \\
\left| \begin{aligned}
L_{emb}(v, P, G^r) &= \underbrace{L(c_P)}_{\text{code motif}} + \underbrace{\log(|\Pi_P| + 1)}_{\text{nb arêtes}} + \sum_{\substack{(v,w) \in E^r \\ \pi = l_E^r(v,w)}} \underbrace{\log(|V_{port}^r|)}_{\text{id sommet port}} + \underbrace{L(c_\pi, P)}_{\text{code port}}
\end{aligned} \right.
\end{aligned}$$

FIG. 6 – Formules utilisées dans le calcul des longueurs de description. La structure  $G^P = (V^P, E^P, l_V^P, l_E^P)$  est abrégée en  $G = (V, E, l_V, l_E)$  pour faciliter la lecture.

**Code table.** La longueur de description  $L(M) = L(CT)$  d'une CT est la somme des longueurs de ses lignes ayant un usage non nul. Chaque ligne est composée de trois parties : la structure du motif, le code du motif et la description des ports du motif.

Pour décrire la structure  $G = G^P$  du motif ( $L(G)$ ), nous encodons en premier son nombre de sommets. Puis nous encodons les sommets  $v$  les uns après les autres. Pour chacun, il faut encoder ses étiquettes puis ses arêtes. Pour encoder les étiquettes d'un sommet ( $L_V(v, G)$ ), nous indiquons leur nombre en premier, puis les étiquettes elles-mêmes. Pour encoder les arêtes d'un sommet ( $L_E(v, G)$ ), nous indiquons leur nombre (entre 0 et  $|V| - 1$  dans un graphe simple), puis pour chacune son sommet destination et son étiquette. Il faut éviter d'encoder deux fois une même arête, nous avons donc choisi —dans un graphe non orienté— de ne l'encoder que dans son sommet de plus petit identifiant. Les étiquettes de sommet et arête sont encodées avec des codes basés sur leur usages dans le graphe original  $G^o$  ( $L_{usage}^{\mathcal{L}_V}(l, G^o)$  et  $L_{usage}^{\mathcal{L}_E}(l_E(v, w), G^o)$ ). Puisque ces codes sont construits à partir du graphe original, commun à toutes les CT, ils peuvent être utilisés pour calculer la longueur de description des CT.

Le deuxième élément d'une ligne de CT est le code  $c_P$  associé au motif ( $L(c_P)$ ). Ce code est basé sur l'usage du motif dans le graphe réécrit.

Le dernier élément d'une ligne de la CT est la description de l'ensemble  $\Pi_P$  des ports du motif ( $L(\Pi_P)$ ). Premièrement nous encodons le nombre de ports du motif (entre 0 et  $|V|$ ). En-

suite nous indiquons quels sommets sont ports : s'il y a  $k$  ports, il y a alors  $C_{|V|}^k$  combinaisons possibles. Enfin nous encodons les codes des ports ( $L(c_\pi, P)$ ) : leur code est basé sur l'usage du port dans le graphe réécrit par rapport aux autres ports du motif.

**Graphe réécrit.** Le graphe réécrit contient deux types de sommets : les ports et les occurrences de motifs. Les ports n'ont pas d'information associée, il suffit donc juste d'encoder leur nombre. La longueur de description  $L(D|M) = L(G^r)$  du graphe réécrit est alors la longueur de l'encodage du nombre de ports plus la somme des longueurs  $L_{emb}(v, P, G^r)$  des encodages des sommets occurrence  $v$ . Chaque sommet occurrence a une étiquette  $l_V^v(v)$  qui indique son motif  $P$ , encodée avec le code  $c_P$ . Nous encodons ensuite son nombre d'arêtes, c-à-d. le nombre de ports de cette occurrence en particulier (entre 0 et  $|\Pi_P|$ ), puis pour chaque arête le sommet port auquel elle est connectée et à quel port du motif elle correspond (au moyen de son code  $c_\pi$ ).

### 3.4 L'algorithme GRAPHMDL

Dans les sous-sections précédentes nous avons présenté les définitions MDL que GRAPHMDL utilise pour comparer des ensembles de motifs (CT). Un algorithme naïf pour trouver le meilleur sous-ensemble de motifs candidats (au sens MDL) créerait une CT pour chaque sous-ensemble possible et de retenir celui avec la plus petite longueur de description. Mais une telle approche n'est pas praticable à cause du très grand nombre de sous-ensembles possibles. C'est pourquoi GRAPHMDL utilise une approche gloutonne basée sur des heuristiques, en adaptant l'algorithme Krimp à nos définitions (Vreeken et al., 2011).

Comme dans Krimp, l'algorithme démarre avec une CT formée de singletons, que nous appelons  $CT_0$ . Un par un, les motifs candidats sont ajoutés à celle-ci seulement s'ils permettent de diminuer la longueur de description. Deux heuristiques guident GRAPHMDL : l'ordre des candidats et l'ordre des motifs dans la CT. Nous utilisons les mêmes heuristiques que Krimp, à différence que nous définissons la taille d'un motif comme le nombre total d'étiquettes (sommets et arêtes) de celui-ci. Nous implémentons aussi la phase de « pruning » de Krimp : après qu'un motif soit accepté, GRAPHMDL vérifie si la suppression de certains motifs déjà sélectionnés permet de diminuer la longueur de description  $L(M, D)$ .

## 4 Évaluation expérimentale

Pour évaluer notre proposition nous avons développé un prototype de GRAPHMDL. Le prototype a été développé en Java 1.8 et est disponible comme dépôt git<sup>4</sup>.

### 4.1 Jeux de données

Les deux premiers jeux de données que nous utilisons (AIDS-CA, AIDS-CM) sont extraits du « National Cancer Institute AIDS antiviral screen data<sup>5</sup> ». Il s'agit d'une collection de graphes utilisée dans la littérature pour évaluer les algorithmes de fouille de graphes (Wörlein et al., 2005). Les graphes de cette collection représentent des molécules : les sommets sont des

4. <https://gitlab.inria.fr/fbariatt/graphmdl>

5. <https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data>



TAB. 1 – *Caractéristiques des jeux de données utilisés dans l'évaluation expérimentale*

Nom	Nombre de graphes	$ V $	$ E $	$ \mathcal{L}_V $	$ \mathcal{L}_E $
AIDS-CA	423	16714	17854	21	3
AIDS-CM	1082	34387	37033	26	3
UD-PUD-En	1000	21176	20176	17	46

TAB. 2 – *Résultats sur différents ensembles de candidats*

Données	Support	Candidats	Temps	$ CT $	$\frac{L(CT,D)}{L(CT_0,D)}$	Médiane nb	
	gSpan		calcul			étiq.	ports
AIDS-CA	20%	2194	19m	115	24.42%	9	3
AIDS-CA	15%	7867	1h47m	123	21.64%	10	4
AIDS-CA	10%	20596	3h36m	148	19.03%	11	3
AIDS-CM	20%	433	22m	111	28.91%	7	4
AIDS-CM	15%	779	32m	131	27.44%	9	4
AIDS-CM	10%	2054	1h10m	163	24.94%	9	4
AIDS-CM	5%	9943	5h02m	225	20.43%	9	4
UD-PUD-En	10%	164	1m	162	39.55%	5	2
UD-PUD-En	5%	458	3m	249	34.45%	5	2
UD-PUD-En	1%	6021	19m	523	28.14%	7	2
UD-PUD-En	0%	233434	9h57m	773	26.25%	7	2

atomes et les arêtes des liaisons. Nous avons enlevé les atomes d'hydrogène, dont les positions peuvent être déduites.

Nous avons extrait notre troisième jeu de données du projet « Universal Dependencies<sup>6</sup> ». Ce projet gère une collection d'arbres décrivant les relations de dépendance syntaxique entre les mots de phrases de différents corpus en plusieurs langues. Nous avons utilisé les arbres correspondant à la version anglaise du corpus PUD (UD-PUD-En).

La table 1 présente les caractéristiques principales des jeux de données utilisés : le nombre de graphes élémentaires dans le jeu de données, le nombre total de sommets et d'arêtes, le nombre total d'étiquettes de sommet et d'arête. Puisque GRAPHMDL travaille sur un seul graphe plutôt qu'une collection, nous agrégeons les données (qui deviennent un seul graphe avec plusieurs composantes connexes) quand nécessaire. Nous générons des motifs candidats en utilisant une implémentation de gSpan disponible sur le site de l'auteur<sup>7</sup>.

## 4.2 Évaluation quantitative

La table 2 présente les résultats des nos expériences. Par exemple, la première ligne montre que nous avons exécuté GRAPHMDL sur le jeu de données AIDS-CA, avec comme candidats les 2194 motifs générés par gSpan pour un support minimum de 20%. En 19 minutes notre algorithme a sélectionné une CT composée de 115 motifs, donnant une longueur de description

6. <https://universaldependencies.org/>

7. <https://sites.cs.ucsb.edu/~xyan/software/gSpan.htm>

## GraphMDL : Sélection de motifs de graphes avec le principe MDL

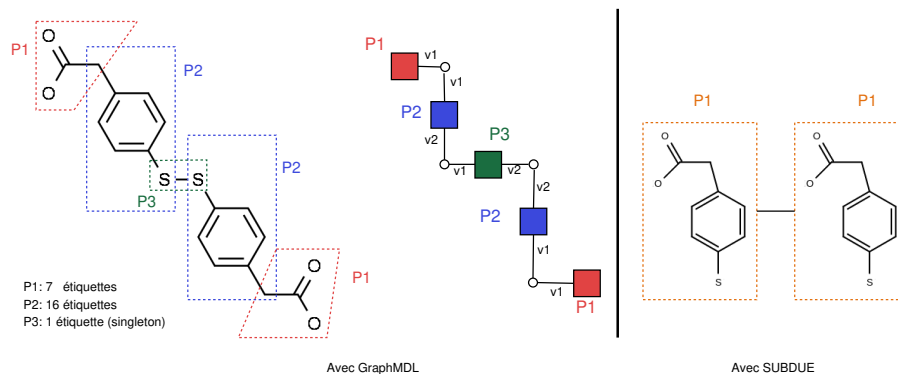


FIG. 7 – Comment GRAPHMDL (à gauche) et SUBDUE (à droite) encodent un des graphes du jeu de données AIDS-CM.

de 24% la longueur de description générée par la code table  $CT_0$  (composée seulement de singletons). Les motifs sélectionnés ont une médiane de 9 étiquettes et 3 ports.

Nous observons que le nombre de motifs d'une CT est souvent significativement inférieur au nombre de candidats. Ceci est particulièrement remarquable pour les petites valeurs de support, où GRAPHMDL réduit le nombre de motifs jusqu'à 300 fois : les motifs générés à ce support contiennent probablement beaucoup de redondance, qui est éliminée par GRAPHMDL.

De plus, nous observons que les longueurs de description des CT trouvées par GRAPHMDL sont entre 20% et 40% celles de la code table de référence  $CT_0$ , ce qui montre que notre algorithme est efficace à trouver des régularités dans les données qui permettent de les compresser. Les longueurs de descriptions sont plus petites quand le nombre de candidats augmente. Nous attribuons ceci au fait qu'avec un plus grand nombre de candidats il y a plus de chances qu'il y ait de « bons » motifs qui permettent de réduire les longueurs de description.

Nous observons que GRAPHMDL peut trouver des motifs de taille conséquente, comme indiqué par le nombre médian d'étiquettes dans la table 2. De plus, la plupart des motifs ont seulement quelques ports, ce qui montre que GRAPHMDL réussit à trouver des modèles où le graphe original est décrit comme un ensemble de composantes peu inter-connectées. Nous pensons qu'un humain aura plus de facilité à analyser un tel modèle, plutôt qu'un modèle formé de motifs très entremêlés.

### 4.3 Évaluation qualitative

La figure 7 montre comment GRAPHMDL utilise certains des motifs qu'il a sélectionnés sur le jeu de données AIDS-CM pour encoder un des graphes de celui-ci. L'image illustre l'idée clé de notre approche : trouver un ensemble de motifs tels que chacun décrit une partie des données, avec leurs occurrences connectées ensemble au travers de sommets ports.

Nous observons que GRAPHMDL sélectionne bien des gros motifs (tels que P2), qui décrivent de grandes parties des données, mais aussi des petits motifs (tels que P3, motif singleton arête) pour faire le lien entre les occurrences de motifs. Les gros motifs accroissent la longueur de description de la code table, mais décrivent une plus grosse partie des données en une seule

fois, alors que les petits motifs font le contraire. En suivant le principe MDL, GRAPHMDL trouve un bon équilibre entre les deux types de motifs.

Il est intéressant de remarquer que le motif P1 de l'image correspond au groupe fonctionnel carboxyle, courant en chimie organique. GRAPHMDL a sélectionné ce motif sans connaissance a priori en chimie, en utilisant uniquement le principe MDL.

**Comparaison avec SUBDUE** Dans la partie droite de la figure 7 nous pouvons voir l'encodage proposé par SUBDUE. Le désavantage principal de SUBDUE est sa perte d'information : nous pouvons voir que les données sont composées de deux occurrences du motif P1, mais il est impossible de savoir comment ces deux occurrences sont connectées entre elles. Grâce à l'introduction de la notion de ports, GRAPHMDL n'a pas ce défaut : un utilisateur analysant ses résultats peut savoir exactement par quels atomes les occurrences de motifs sont connectées.

## 5 Conclusion

Dans ce papier nous avons proposé GRAPHMDL, une approche utilisant le principe MDL pour sélectionner un sous-ensemble descriptif de motifs de graphes sur des graphes étiquetés. Nous avons proposé des définitions MDL permettant de calculer les longueurs de description nécessaires pour appliquer le principe MDL. L'originalité de notre approche est l'introduction du concept de *ports*, qui permet d'encoder les données sans perte d'informations. Nos expériences montrent que GRAPHMDL réduit significativement le nombre de motifs par rapport à des approches complètes. De plus, les motifs sélectionnés peuvent avoir des formes complexes avec des connexions simples. L'introduction de la notion de ports facilite l'interprétation des motifs par rapport à SUBDUE, une autre approche utilisant le principe MDL. Dans le futur nous projetons d'étendre cette approche à des graphes plus complexes : les graphes dirigés et les multi-graphes.

## Références

- Borgelt, C. et M. R. Berthold (2002). Mining molecular fragments : Finding relevant substructures of molecules. In *Proc. of the 2002 IEEE Int. Conf. on Data Mining (ICDM '02)*, pp. 51–58. IEEE Computer Society.
- Cook, D. J. et L. B. Holder (1993). Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research* 1, 231–255.
- Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* 21(2), 194–203.
- Grünwald, P. (2000). Model selection based on minimum description length. *Journal of Mathematical Psychology* 44(1), 133–152.
- Huan, J., W. Wang, et J. Prins (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. In *IEEE Int. Conf. Data Mining (ICDM)*, pp. 549–. IEEE Computer Society.

- Koopman, A. et A. Siebes (2009). Characteristic Relational Patterns. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09*, pp. 437–446. ACM.
- Koutra, D., U. Kang, J. Vreeken, et C. Faloutsos (2015). Summarizing and understanding large graphs. *Statistical Analysis and Data Mining : The ASA Data Science Journal* 8(3), 183–202.
- Lee, T. C. M. (2001). An Introduction to Coding Theory and the Two-Part Minimum Description Length Principle. *International Statistical Review* 69(2), 169–183.
- Nijssen, S. et J. N. Kok (2005). The Gaston tool for frequent subgraph mining. *Electron. Notes Theor. Comput. Sci.* 127(1), 77–87.
- Tatti, N. et J. Vreeken (2012). The long and the short of it : Summarising event sequences with serial episodes. In *Proc. of the Int. Conf. on Knowledge Discovery and Data Mining (KDD'12)*, pp. 462–470. ACM.
- Vreeken, J., M. van Leeuwen, et A. Siebes (2011). Krimp : mining itemsets that compress. *Data Min. Knowl. Discov.* 23(1), 169–214.
- Wörlein, M., T. Meinl, I. Fischer, et M. Philippsen (2005). A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston. In *Knowledge Discovery in Databases : PKDD 2005*, Volume 3721 of LNCS, pp. 392–403. Springer Berlin Heidelberg.
- Yan, X. et J. Han (2002). gSpan : Graph-based substructure pattern mining. In *Proc. of the 2002 IEEE Int. Conf. on Data Mining (ICDM '02)*, pp. 721–724. IEEE Computer Society.
- Yan, X. et J. Han (2003). CloseGraph : Mining closed frequent graph patterns. In *ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 286–295. ACM.
- Zhu, F., X. Yan, J. Han, et P. S. Yu (2007). gPrune : A Constraint Pushing Framework for Graph Pattern Mining. In *Advances in Knowledge Discovery and Data Mining*, LNCS, pp. 388–400.

## Summary

Many graph pattern mining algorithms have been designed to identify recurring structures in graphs. The main drawback of these approaches is that they often extract too many patterns for human analysis. Recently, pattern mining methods using the *Minimum Description Length* (MDL) principle have been proposed to select a characteristic subset of patterns from transactional, sequential and relational data. In this paper, we propose a MDL-based approach for selecting a characteristic subset of patterns on labeled graphs. A key notion in this paper is the introduction of *ports* to encode connections between pattern occurrences without any loss of information. Experiments show that the number of patterns is drastically reduced, and the selected patterns can have complex shapes.