Université Blaise Pascal
UFR Sciences Exactes et Naturelles
Département de Mathématiques et Informatique
63177 AUBIERE CEDEX

Master 2 Recherche - M.S.I.

Symétries dans les familles d'ensembles : une étude algorithmique

Caroline Noyer

Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes

Responsables : O. Raynaud, R. Medina

Remerciements

Je souhaite tout d'abord remercier Olivier Raynaud et Raoul Medina pour m'avoir encadrée, et avoir su me transmettre la passion et la rigueur pour mener à bien un travail de recherche.

Je tiens à remercier Lhouari Nourine pour m'avoir accueillie au sein de l'équipe Algorithmique.

Merci aussi aux autres membres de l'équipe, Yoan et Alain, ainsi que Pierre, collègue de quelques temps, grâce à qui les réunions nocturnes m'ont permis d'y voir plus clair, d'avancer dans ma réflexion, et de trouver les bonnes formulations.

Je remercie également Damien et Stéphanie, mes co-équipiers du bureau C101, pour avoir pris soin de m'éviter tout courant d'air.

Enfin merci à Damien et Isabelle d'avoir supporté les derniers instants de ce stage qui furent parfois stressants.

Résumé

On souhaite définir la notion de symétrie entre attributs dans une famille d'ensembles. Une première réponse a été proposée par R. Medina et L. Nourine à travers la notion de clones. Deux attributs sont clones s'ils sont interchangeables dans les ensembles de la famille. Ici, on s'intéresse au degré de symétrie entre deux attributs : la similitude. Cette mesure est représentée dans une matrice indiquant pour tout couple d'attributs, la "distance" qui les sépare. Différents algorithmes de calcul de cette similitude sont proposés, et, sous certaines conditions, l'un d'entre eux est optimal. Enfin, nous montrons le lien qui existe entre attributs clones et sommets jumeaux d'un graphe.

Abstract

In this master thesis, we deal with attributes having a symetrical behaviour in family of sets. A first answer has been given by R. Medina and L. Nourine through the clone notion. Two attributes are said to be clones if they can be swapped in any set of the family. We present a symetry measure for attributes : the similarity. This measure is represented by a matrix which expresses the distance between two attributes. We propose different algorithms to compute this similarity measure. We show that one of these algorithms is optimal under some hypothesis we develop. Finaly we discuss the link between twins vertices of a graph and clone attributes.

Table des matières

Table des figures

Liste des Algorithmes

Notation utilisée

⋉ Par convention pour ce mémoire, on utilise les notations suivantes :
- a : un élément de J
- M : un ensemble défini sur J et un élément de $\mathcal{M}$
- $\mathcal{F}$: une collection d'ensembles

⋉ $|\mathcal{M}|$ représente le nombre d'éléments dans $\mathcal{M}$
⋉ $||\mathcal{M}||$ la taille de $\mathcal{M}$, c'est-à-dire $\sum_{M \in \mathcal{M}} |M|$.

⋉ $\mathcal{M}_a$, la restriction de $\mathcal{M}$ aux éléments contenant a.
⋉ $\mathcal{M}_{\overline{a}}$, la restriction de $\mathcal{M}$ aux éléments ne contenant pas a.

⋉ $\mathcal{M}_{a\overline{b}} = \mathcal{M}_a \cap \mathcal{M}_{\overline{b}}$
⋉ $\mathcal{M}_{ab} = \mathcal{M}_a \cap \mathcal{M}_b$

⋉ $\mathcal{M}^t$, la restriction de $\mathcal{M}$ aux éléments de taille t.

1 Introduction

De nombreuses applications se modélisent ou admettent en entrée des familles d'ensembles. Ces familles sont parfois appelées *hypergraphes* (par exemple en bases de données, en réseaux, ...), *collections d'itemsets* (par exemple en datamining), *relations binaires* (par exemple en théorie des treillis), etc.

De même, de nombreuses applications construisent ou génèrent des familles d'ensembles (par exemple les clés ou les dépendances fonctionnelles en bases de données, les fermés en théorie des treillis, ...).

La taille de ces familles d'ensembles constitue souvent un obstacle à la mise en œuvre pratique de ces applications ou au stockage de ces mêmes familles. L'idée est donc d'identifier les attributs ayant un comportement symétrique dans ces familles afin d'éliminer une forme de redondance et donc d'en réduire la taille. Les sommets jumeaux d'un graphe représentent un exemple connu de comportement symétrique.

R. Medina et L. Nourine, dans [MN], ont exhibé un comportement symétrique des attributs d'une famille à travers la notion de clones. Deux attributs a et b seront dits clones s'ils sont interchangeables dans les ensembles de la famille. Dans [MN], un algorithme de détection des classes de clones est proposé. Sa complexité est de $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$.

Dans ce mémoire, on définit une mesure de la symétrie entre deux attributs : la *similitude*. De façon duale, on définit la *distance* séparant deux attributs (donc exprimant la dissymétrie). Notre objectif est de proposer une algorithmique efficace pour calculer cette similitude entre tout couple d'attributs d'une famille donnée.

Ce mémoire s'organise en quatre parties. Dans un premier temps, on rappellera les définitions nécessaires, pour ensuite formaliser le problème traité. Puis on présentera une modélisation de ce problème par un graphe, sur laquelle on s'appuie pour proposer plusieurs stratégies de résolution. Enfin, on fera le lien entre deux types d'ensembles ayant un comportement symétrique, les sommets jumeaux d'un graphe et les classes de clones d'une famille.

2 Positionnement du problème

2.1 Définitions

Intuitivement a et b sont clones s'ils sont interchangeables dans tout ensemble de la famille. Ainsi, on note $\varphi_{a,b}$ la correspondance de 2^J dans 2^J, qui associe à chaque sous-ensemble F de J son image en échangeant a et b.

$$\varphi_{a,b}(F) = \begin{cases} (F \setminus \{a\}) \cup \{b\} & si\ b \notin F\ et\ a \in F \\ (F \setminus \{b\}) \cup \{a\} & si\ a \notin F\ et\ b \in F \\ F & sinon \end{cases}$$

Remarque : - $\varphi_{a,b}(F) = \varphi_{b,a}(F)$
- $\varphi_{a,b}(\varphi_{b,a}(F)) = F$

Définition 1

*Soit $\mathcal{F}$ une famille d'ensembles définie sur J. Les attributs a et b sont **clones**, noté $a \approx b$, si et seulement si pour tout F de $\mathcal{F}$, $\varphi_{a,b}(F)$ appartient à $\mathcal{F}$.*

Dans [MN], le problème suivant est alors défini :

Problème 1 : Détection des classes de clones
Entrée : $\mathcal{F}$ une famille d'ensembles
Sortie : $C_1, ..., C_k$ les classes de clones de $\mathcal{F}$

2.2 Distance et similitude d'attributs

Pour deux attributs a et b, on peut définir deux mesures de leur comportement symétrique dans la famille : la *similitude* et la *distance*.

La *similitude* exprime une mesure du comportement symétrique entre deux attributs. Plus cette valeur est grande plus ils sont symétriques.

La *distance* exprime un comportement dissymétrique. Plus cette valeur est grande et moins les attributs ont un comportement symétrique.

La fonction **similitude** $s_{\mathcal{M}}$ est définie comme suit :

$$\begin{aligned} J^2 &\rightarrow \mathbb{N} \\ s_{\mathcal{M}}(a,b) &\rightarrow \left|\{M \in \mathcal{M} \mid \exists M' \in \mathcal{M}\ tq\ M' = \varphi_{a,b}(M)\}\right| \end{aligned}$$

Cette notion de *similitude* pour le couple (a,b) permet de représenter le nombre d'ensembles de $\mathcal{M}$ ayant une image par $\varphi_{a,b}$.

Le problème qui lui est associé est le suivant :

Problème 2 : Similitude
Entrée : une collection $\mathcal{M}$ sur J
Sortie : la matrice $s_{\mathcal{M}} = J \times J$ d'entiers
Relation : soient $a, b \in J^2$, $s_{\mathcal{M}}(a,b) = \left|\{M \in \mathcal{M} \mid \exists M' \in \mathcal{M} \ tq \ M' = \varphi_{a,b}(M)\}\right|$

On définit également la fonction **distance** $d_{\mathcal{M}}$:

$$\begin{aligned} J^2 &\rightarrow \mathbb{N} \\ d_{\mathcal{M}}(a,b) &\rightarrow \left|\{M \in \mathcal{M} \mid \varphi_{a,b}(M) \notin \mathcal{M}\}\right| \end{aligned}$$

Cette notion de *distance* entre a et b représente le nombre d'éléments de $\mathcal{M}$ qui n'ont pas d'image par $\varphi_{a,b}$.

Plus formellement, le problème consistant à calculer les distances est le suivant :

Problème 3 : Distance
Entrée : une collection $\mathcal{M}$ sur J
Sortie : la matrice $d_{\mathcal{M}} = J \times J$ d'entiers
Relation : soient $a, b \in J^2$, $d_{\mathcal{M}}(a,b) = \left|\{M \in \mathcal{M} \mid \varphi_{a,b}(M) \notin \mathcal{M}\}\right|$

On peut trivialement remarquer qu'on a la propriété suivante :

Propriété 1
$|\mathcal{M}| = d_{\mathcal{M}}(a,b) + s_{\mathcal{M}}(a,b)$

Donc le problème de la *distance* et de la *similitude* sont équivalents. Dans cette étude on s'attachera à résoudre le problème *Distance*. Toute solution pour ce problème permet de résoudre le problème *Similitude* avec la même complexité.

Les $d_{\mathcal{M}}(a,b)$ sont stockés dans une matrice appelée *matrice des distances*.

2.3 Discussion sur la complexité du problème *Distance*

L'objectif étant de calculer cette matrice, on s'interroge, dans un premier temps, sur la manière de l'initialiser et de la mettre à jour.

Comme la notion de *distance* $d_{\mathcal{M}}(a,b)$ est symétrique, la matrice des distances est aussi symétrique. On choisit alors de la représenter par une matrice triangulaire telle que :

$$\begin{cases} d_{\mathcal{M}}(a,a) = 0 \\ d_{\mathcal{M}}(a,b) = d_{\mathcal{M}}(b,a) \end{cases}$$

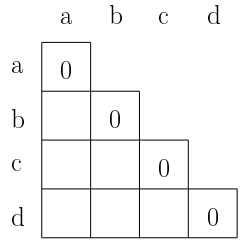


FIG. 1 – Matrice des distances

Décomposons $\mathcal{M}$ par $\mathcal{M}_{a\bar{b}} \uplus \mathcal{M}_{\bar{a}b} \uplus \mathcal{M}_{ab} \uplus \mathcal{M}_{\bar{a}\bar{b}}$, avec [1]

- $\mathcal{M}_{a\bar{b}}$: les éléments M de $\mathcal{M}$ tels que $a \in M$ et $b \notin M$
- $\mathcal{M}_{\bar{a}b}$: les éléments M de $\mathcal{M}$ tels que $a \notin M$ et $b \in M$
- $\mathcal{M}_{ab}$: les éléments M de $\mathcal{M}$ tels que $a \in M$ et $b \in M$
- $\mathcal{M}_{\bar{a}\bar{b}}$: les éléments M de $\mathcal{M}$ tels que $a \notin M$ et $b \notin M$

Discutons maintenant sur les différentes façons d'initialiser et de mettre à jour la matrice des distances. On suppose que l'on ne peut mettre à jour que par incrémentation et décrémentation de une unité à la fois. Il est clair qu'il y a deux manières de procéder :

- soit on initialise chaque case de la matrice à 0 et on incrémente de 1 à chaque fois que l'on trouve un $M \in \mathcal{M}$ tel que $\varphi_{a,b}(M) \notin \mathcal{M}$,
- soit on initialise la distance entre a et b avec la valeur maximum, puis on décrémente de 1 la distance $d_{\mathcal{M}}(a,b)$ chaque fois que l'on trouve M et $M' \in \mathcal{M}$ tels que $M = \varphi_{a,b}(M')$.

Quelle est la valeur maximum qui peut être prise par $d_{\mathcal{M}}(a,b)$? Il est évident que c'est $|\mathcal{M}_{a\bar{b}}| + |\mathcal{M}_{\bar{a}b}|$. En effet, pour chaque élément $M \in \mathcal{M}_{ab} \cup \mathcal{M}_{\bar{a}\bar{b}}$ on a $M = \varphi_{a,b}(M)$, et ainsi M ne peut pas accroître la distance entre a et b. Cette valeur maximum représente le nombre maximum d'opérations élémentaires (incrémentation ou décrémentation) nécessaires pour calculer la matrice des distances dans le pire de cas. Peu importe la stratégie retenue, le nombre d'opérations élémentaires est le même dans le pire des cas.

Si on s'intéresse à la complexité en temps pour les mises à jour, on considère que les opérations élémentaires sont exécutées en $\mathcal{O}(1)$. Ainsi la complexité globale est :

$$\mathcal{O}(\sum_{a\in J}\sum_{b\in J} d_{\mathcal{M}}(a,b)) = \mathcal{O}(\sum_{a\in J}\sum_{b\in J} |\mathcal{M}_{a\bar{b}}| + |\mathcal{M}_{\bar{a}b}|)$$

Prenons $M \in \mathcal{M}$. Pour qu'il soit pris en compte dans $d_{\mathcal{M}}(a,b)$, il faut que $a \in M$ et $b \notin M$ (donc $b \in J \setminus M$). Ainsi, dans le pire des cas, M interviendra dans $|M| \times |J \setminus M|$

[1] ici $\uplus$ correspond à l'union disjointe

distances. On obtient donc :

$$\mathcal{O}(\sum_{a\in J}\sum_{b\in J} d_{\mathcal{M}}(a,b)) = \mathcal{O}(\sum_{M\in\mathcal{M}} |M| \times (|J| - |M|))$$

Ce qu'on peut réécrire comme :

$$\mathcal{O}(\sum_{M\in\mathcal{M}} |M| \times (|J| - |M|)) = \mathcal{O}((\sum_{M\in\mathcal{M}} |M| \times |J|) - (\sum_{M\in\mathcal{M}} |M| \times |M|))$$

Puisque $|M| \leq |J|$ on a donc $|M| \times |M|$ qui est inférieur ou égal à $|M| \times |J|$. On obtient alors la complexité dans le pire des cas :

$$\mathcal{O}(\sum_{M\in\mathcal{M}} |M| \times |J|) = \mathcal{O}(|J| \times ||\mathcal{M}||)$$

Donc peu importe la stratégie que l'on retient, la complexité en temps ne peut pas être inférieure à $\mathcal{O}(|J| \times ||\mathcal{M}||)$ (avec l'hypothèse des opérations élémentaires d'incrémentation ou de décrémentation par 1). Dans la suite de ce mémoire, on privilégie l'approche par décrémentation (donc celle qui initialise la matrice avec les valeurs maximum).

Initialisation de la matrice

On initialise chaque distance $d_{\mathcal{M}}(a,b)$ avec la valeur maximum possible, c'est-à-dire $|\mathcal{M}_{a\bar{b}}| + |\mathcal{M}_{\bar{a}b}|$. Initialement, les distances sont égales à 0. Ceci peut être fait en $\mathcal{O}(|J| \times |J|)$.

Ensuite, pour chaque $M \in \mathcal{M}$ on incrémente par 1 les distances $d_{\mathcal{M}}(a,b)$, avec $a \in M$ et $b \notin M$. Comme on l'a vu précédemment ceci s'exécute en $\mathcal{O}(|J| \times ||\mathcal{M}||)$.

Algorithme 1: `InitDistance(`$\mathcal{M}$`)`

Données : Une collection d'ensembles $\mathcal{M}$ définie sur J
Résultat : La matrice des distances telle que $\forall a, b \in J^2 : d_{\mathcal{M}}(a,b) = |\mathcal{M}_{a\bar{b}}| + |\mathcal{M}_{\bar{a}b}|$
début
  pour chaque $a \in J$ **faire**
    pour chaque $b \in J$ **faire**
      $d_{\mathcal{M}}(a,b) = 0$
  pour chaque $M \in \mathcal{M}$ **faire**
    pour chaque $a \in M$ **faire**
      pour chaque $b \notin M$ **faire**
        $d_{\mathcal{M}}(a,b)++$
  retourner $d_{\mathcal{M}}$
fin

3 Modélisation par un graphe

L'idée est de modéliser le problème à l'aide d'un graphe contenant les ensembles de la famille comme sommets. Les arcs représentent la relation *"est une image de"* entre deux ensembles. Les arcs sont étiquetés par les attributs qui s'échangent entre les deux sommets.

3.1 Définition d'un φ-graphe

Plus formellement :

Définition 2

Soient $G = (V, E, \sigma)$ un graphe étiqueté et $\mathcal{M}$ une collection d'ensembles sur J. On dit que G est le **φ-graphe** *de $\mathcal{M}$ si et seulement si :*

- $V = \mathcal{M}$
- $\forall e = (M, M') \in E$ on a $M' = \varphi_{a,b}(M)$
- $\forall e = (M, M') \in E$, $\sigma(e) = (M \setminus M', M' \setminus M) = (a, b)$

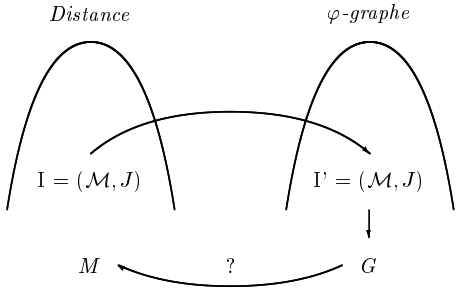
On énonce le problème suivant :

Problème 4 : φ-graphe
Entrée : une collection $\mathcal{M}$ sur J
Sortie : un graphe G étiqueté correspondant au φ-graphe de $\mathcal{M}$

Propriété 2

Soit $\mathcal{M}$ une collection d'ensembles sur J. Le problème Distance se "réduit" dans le problème φ-graphe.

Preuve de la Propriété 2



Soit I, une collection $\mathcal{M}$ sur J, une instance du problème *Distance*. Soit I' une instance du problème *φ-graphe* obtenue par la fonction identité à partir de I.

Soit G le graphe étiqueté obtenu par la résolution du problème *φ-graphe* sur I'. On peut construire la matrice $d_{\mathcal{M}}$, solution du problème *Distance*, à partir de G. En effet, la présence de l'arc $e = (M, M')$ d'étiquette $\sigma(e) = (a, b)$ dans le graphe signifie que l'on a la relation $M' = \varphi_{a,b}(M)$, autrement dit que M' est l'image de M par $\varphi_{a,b}$ et inversement puisque φ est une application symétrique. Dans ce cas, on doit décrémenter l'élément $d_{\mathcal{M}}(a, b)$ dans la matrice des distances. On va donc parcourir chacun des arcs, et ainsi, passer de G à $d_{\mathcal{M}}$ se fait en la taille de E, donc en temps linéaire.

Donc le problème *Distance* peut se réduire dans le problème *φ-graphe*.

□

Remarque : le nombre de mises à jour à effectuer dans la matrice correspond exactement au nombre d'arcs dans le graphe.

3.2 Propriétés d'un φ-graphe

La propriété suivante se déduit directement de la fonction φ.

Propriété 3

Soient $G = (\mathcal{M}, E, \sigma)$ un φ-graphe, et $e = (M, M')$ un arc de G tel que $\sigma(e) = (a, b)$. Alors les assertions suivantes sont équivalentes :

1/ $M' = \varphi_{a,b}(M)$
2/ $M = \varphi_{b,a}(M')$
3/ $|M| = |M'|$ et $M \setminus M' = \{a\}$ et $M' \setminus M = \{b\}$
4/ $M \setminus \{a\} = M' \setminus \{b\}$

D'après la *Propriété 3*, on note qu'un élément M et son image M' sont de même taille. Il paraît donc évident d'effectuer la recherche de cette image de M, avec $t = |M|$, aux seuls éléments de taille t. C'est pourquoi on définit $\mathcal{M}^t$ la restriction de $\mathcal{M}$ aux éléments de taille t.

Définition 3

On note $G_t = (\mathcal{M}^t, E_t, \sigma_t)$ la restriction de G à un sous-graphe composé des sommets de $\mathcal{M}$ de taille t et dont les arêtes coïncident avec les arêtes de E.

Propriété 4

Soit $G = (\mathcal{M}, E, \sigma)$. L'ensemble des sous-graphes $G_1, G_2, ..., G_{|J|}$ réalise une partition disjointe du graphe G.

Preuve de la Propriété 4

∘ Montrons que les ensembles $V_1, V_2, ..., V_{|J|}$ réalisent une partition de V.

Comme $V = \mathcal{M}$, $V_1 = \mathcal{M}^1$, ..., $V_{|J|} = \mathcal{M}^{|J|}$. $\mathcal{M}^t$ représente la restriction de la collection $\mathcal{M}$ aux éléments de taille t, il est donc clair qu'un élément ne peut pas appartenir à deux ensembles $\mathcal{M}^t$ différents. De plus, $\mathcal{M} = \bigcup_i \mathcal{M}^i$.

◦ Montrons que les ensembles E_1, E_2, ..., $E_{|J|}$ réalisent une partition de E.
Les arcs de E représentent la relation $\varphi_{a,b}$ qui existe entre deux ensembles et ne peut donc ainsi relier que des éléments de même taille (*cf. Propriété 3*), c'est-à-dire appartenant à un même ensemble $\mathcal{M}^t$.

◦ Qu'en est-il pour σ_t ?
Il est évident que σ_t n'est autre qu'une restriction des étiquettes des arcs de E_t.

$\mathcal{M}^t$ réalise une partition de V et E_t réalise une partition de E donc $G_t = (\mathcal{M}^t, E_t, \sigma_t)$ réalise une partition de $G = (\mathcal{M}, E, \sigma)$.

□

Corollaire 1

Toute composante connexe du graphe $G = (\mathcal{M}, V, \sigma)$ est constitué d'ensembles de même cardinalité.

Preuve de la Propriété 1

La preuve est immédiate d'après la *Propriété 4* (les partitions sont disjointes).

Propriété 5

Soit $G = (\mathcal{M}, E, \sigma)$ un φ-graphe. Soient $e = (M, M')$ et $e' = (M, M'')$ deux arcs de E avec $\sigma(e) = (i, j)$ et $\sigma(e') = (k, l)$, alors :

***1/** si $i = k$ alors (M', M'') appartient à E avec pour étiquette $\sigma(M', M'') = (j, l)$*

***2/** si $j = l$ alors (M', M'') appartient à E avec pour étiquette $\sigma(M', M'') = (k, i)$*

Autrement dit, lorsque les arcs (M, M') et (M, M'') sont présents dans le graphe et coïncident sur leur première étiquette ou sur la seconde, alors on a obligatoirement l'arc (M', M''). Si la première étiquette est commune, $\sigma(M', M'')$ est définie par les secondes étiquettes des deux arcs (M, M') et (M, M''). Si c'est la seconde étiquette qui est commune alors $\sigma(M', M'')$ est définie par les premières étiquettes de ces deux arcs.

Preuve de la Propriété 5

On peut représenter les deux arcs de E de la façon suivante :

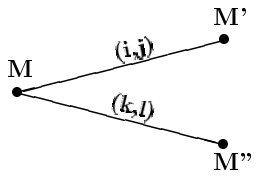


FIG. 2 – Complémentarité du φ-graphe

On a les arcs : - $e = (M, M')$ avec $\sigma(e) = (i, j)$
- $e' = (M', M'')$ avec $\sigma(e') = (k, l)$

Ainsi : $M \setminus M' = i$ et $M \setminus M'' = k$
$M' \setminus M = j$ et $M'' \setminus M = l$

- cas **1/** : $i = k$

On est dans la configuration où M' est l'image de M par $\varphi_{i,j}$ et M'' est l'image de M par $\varphi_{k,l}$ ainsi que l'image de M' par $\varphi_{i,l}$.

Si on appelle X la partie commune à M, M' et M'', on peut alors écrire :
- $M = X \cup \{i\}$
- $M' = X \cup \{j\}$
- $M'' = X \cup \{l\}$

Ainsi on retrouve facilement : $M' \setminus M'' = j$ et $M'' \setminus M' = l$. Autrement dit M'' est l'image de M' par $\varphi_{j,l}$ et donc l'arc $e'' = (M', M'')$ d'étiquette $\sigma(e'') = (j, l)$ appartient à E.

- cas **2/** : $j = l$

On a M' l'image de M par $\varphi_{i,j}$ et M'' l'image de M par $\varphi_{k,l}$ ainsi que l'image de M' par $\varphi_{k,j}$. Comme $j = l$, un seul élément de M' et M'' diffère d'avec M.

On sait que : - $M \setminus M' = i$
- $M \setminus M'' = k$

De plus : - $M' \setminus M = j$
- $M'' \setminus M = l = j$

Donc on en déduit que : $i \in M''$ et $k \in M'$.

Ainsi : - $M'' \setminus M' = i$
- $M' \setminus M'' = k$.

Donc M'' est l'image de M' par $\varphi_{k,i}$ donc l'arc (M', M'') d'étiquette $\sigma(M', M'') = (k, i)$ appartient à E.

□

Exemple :

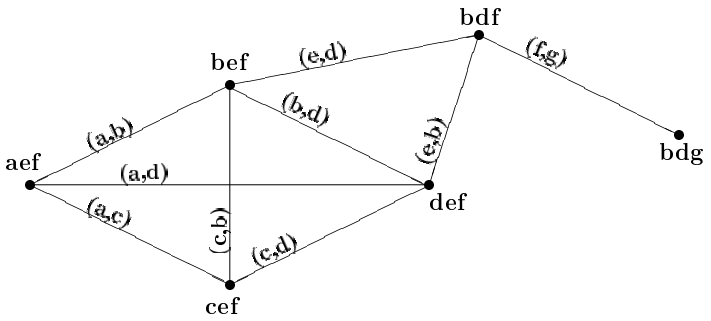
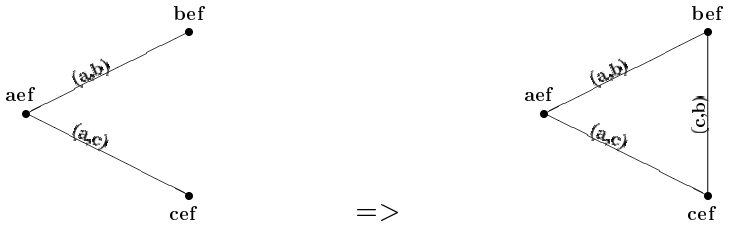


FIG. 3 – Exemple de φ-graphe

On retrouve bien les propriétés énoncées prédédemment :

- **1/** :



- **2/** :

FIG. 4 – Complémentarité du φ-graphe

On sait que si deux arcs (M, M') et (M, M'') sont présents avec $i = k$ alors, d'après la *Propriété 5*, l'arc (M', M'') est présent aussi. M, M' etM'' forment donc une clique dans G. On va donc maintenant s'intéresser à cette notion.

Définition 4

Soit $G = (\mathcal{M}, E, \sigma)$ un φ-graphe. Soit $\mathcal{C} = \{S_1, ..., S_n\}$ une collection de sommets connexes dans G.
*On dit que C forme une φ-**clique** si et seulement si $|S_1 \cap ... \cap S_n| \geq |S_1| - 1$ et $\nexists S_{n+1}$ tel que $|S_1 \cap ... \cap S_n \cap S_{n+1}| = |S_1| - 1$.*

Exemple :

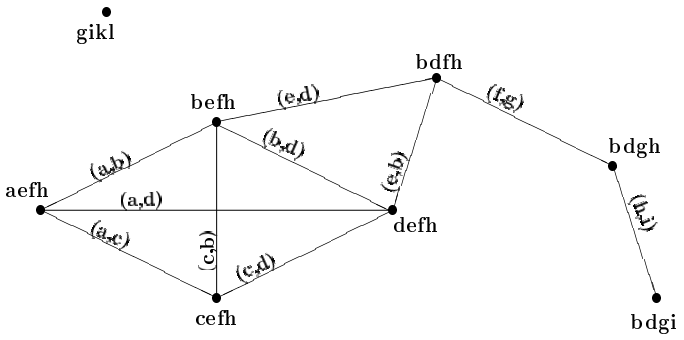


FIG. 5 – Exemple de φ-graphe

Le sous-graphe composé des éléments {aefh,befh,cefh,defh} représente une φ-clique. Il en est de même pour le sous-graphe {bdfh,bdgh}.

Sur ce graphe on trouve 6 φ-cliques :

1 - {aefh,befh,cefh,defh} $\Rightarrow |\bigcap_{i=1}^{n} s_i| = |\text{efh}| \geq t-1$
2 - {befh,bdfh} $\Rightarrow |\bigcap_{i=1}^{n} s_i| = |\text{bfh}| \geq t-1$
3 - {defh,bdfh} $\Rightarrow |\bigcap_{i=1}^{n} s_i| = |\text{dfh}| \geq t-1$
4 - {bdfh,bdgh} $\Rightarrow |\bigcap_{i=1}^{n} s_i| = |\text{bdh}| \geq t-1$
5 - {bdgh,bdgi} $\Rightarrow |\bigcap_{i=1}^{n} s_i| = |\text{bdg}| \geq t-1$
6 - {gikl} $\Rightarrow |\bigcap_{i=1}^{n} s_i| = |\text{gikl}| \geq t-1$

Une φ-clique $\mathcal{C} = \{S_1, ..., S_n\}$ est alors définie par un couple $\mathtt{C} = (\mathtt{inter}, \mathtt{union})$ tel que :

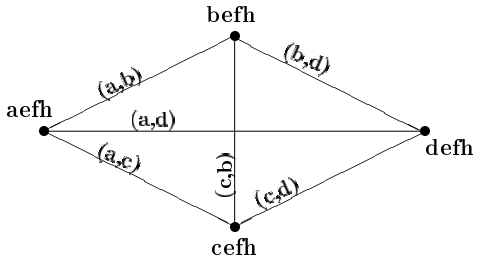
$$\begin{cases} \mathtt{C.inter} = S_1 \cap ... \cap S_n \\ \mathtt{C.union} = (S_1 \cup ... \cup S_n) \setminus \mathtt{C.inter} \end{cases}$$

Il est à noter que, par définition, une φ-clique est maximale.

`C.inter` représente la partie commune aux éléments de la clique et `C.union` les objets qui diffèrent. Les éléments de `C` sont donc de la forme $\mathtt{C.inter} \cup \{a\}$ avec $a \in \mathtt{C.union}$. On en déduit que la taille de l'ensemble `C.union` représente la taille de la φ-clique.

Exemple :

Prenons la φ-clique $(efh, abcd)$ du graphe précédent :



`C.inter` = efh et `C.union` = abcd

FIG. 6 – Exemple de φ-clique

Remarque : $\forall M \in \varphi\text{-clique},\ \forall (a,b) \in \mathtt{C.union}^2\ alors\ \varphi_{a,b}(M) \in \varphi\text{-clique}.$

Soit C un ensemble vérifiant la *Définition 4*. D'après la *Propriété 3*, pour tout couple (M, M') de C, l'arête $e = (M, M')$ d'étiquette $\sigma(e) = (M \setminus M', M' \setminus M)$ appartient à E. Donc l'ensemble C, appelé φ-clique, forme bien une clique dans le φ-graphe.

Propriété 6

Deux φ-cliques différentes ne peuvent pas avoir la même intersection.

Preuve de la Propriété 6

Supposons qu'il existe deux φ-cliques C_1 et C_2 telles que : $\texttt{C}_1\texttt{.inter} = \texttt{C}_2\texttt{.inter}$ et $C_1 \neq C_2$. $C_1 \neq C_2$ implique qu'il existe un $M = \texttt{C}_2\texttt{.inter} \cup \{a\}$ appartenant à C_2 tel que $a \in \texttt{C}_2\texttt{.union}$ et $a \notin \texttt{C}_1\texttt{.union}$. Comme $\texttt{C}_1\texttt{.inter} = \texttt{C}_2\texttt{.inter}$ alors M pourrait être rajouté à C_1 ce qui contredit la notion de maximalité d'une φ-clique. Donc C_1 n'est pas une φ-clique.

□

Propriété 7

Soit $G = (\mathcal{M}, E, \sigma)$ un φ-graphe. Alors il existe un recouvrement $\mathcal{R} = \{C_1, ..., C_n\}$ en φ-cliques de G tel que tout sommet M de G appartient à au plus $|M|$ φ-cliques et tel que $\mathcal{R}$ réalise une partition des arêtes de G.

Preuve de la Propriété 7

∘ On considère un élément M du φ-graphe G. Chacune des φ-cliques auxquelles M peut appartenir a pour $\texttt{C.inter}$ $M \setminus \{x\}$. Comme $|M| = t$, il est clair qu'il y a t façons de choisir cet objet x. Donc tout sommet de G appartient à au plus t φ-cliques. Pour le cas des sommets isolés, on note que chaque sommet appartient à au moins une φ-clique composée de lui-même.

∘ Soit un arc (M, M') appartenant à une φ-clique C_1 et à une φ-clique C_2. Si $(M, M') \in C_1$ alors $\texttt{C}_1\texttt{.inter} = M \cap M'$. De même, si $(M, M') \in C_2$ alors $\texttt{C}_2\texttt{.inter} = M \cap M'$. Or d'après le *Lemme 6*, deux φ-cliques ne peuvent pas avoir la même intersection, donc l'arc (M, M') n'existe pas. Ainsi une arête ne peut appartenir qu'à au plus une φ-clique et donc $\mathcal{C}$ réalise une partition des arêtes de G.

∘ Soient M et M' deux sommets du φ-graphe G, et $e = (M, M')$ l'arête qui les relie d'étiquette $\sigma(e) = (x, y)$. Alors $M \cap M' = M \setminus \{x\} = M' \setminus \{y\}$ donc il existe une φ-clique contenant M et M' et contenant ainsi cette arête. Donc une arête appartient à au moins une φ-clique.

Comme chaque sommet appartient à au moins une φ-clique, ainsi que chaque arête, $\mathcal{R}$ réalise donc un recouvrement en φ-cliques de G.

□

Exemple :

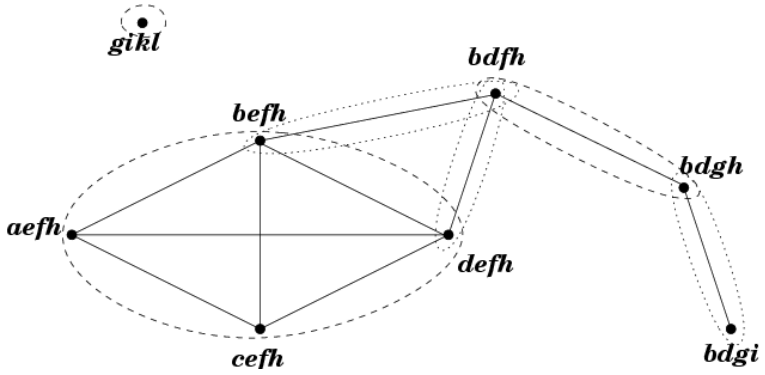


FIG. 7 – Exemple de recouvrement par des φ-cliques

On obtient un recouvrement de ce φ-graphe par 6 φ-cliques : $\mathcal{R} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$.

$$avec \begin{cases} C_1 = (efh, abcd) \\ C_2 = (bfh, ed) \\ C_3 = (dfh, eb) \\ C_4 = (bdh, fg) \\ C_5 = (bdg, hi) \\ C_6 = (gikl, \emptyset) \end{cases}$$

Propriété 8

Soit $G = (\mathcal{M}, E, \sigma)$ un φ-graphe. Un recouvrement $\mathcal{R}$ en φ-cliques de G est une représentation de G.

Preuve de la Propriété 8

Si on calcule un recouvrement $\mathcal{R}$ en φ-cliques d'un graphe G, on obtiendra alors un ensemble de φ-cliques. Telles qu'elles sont définies, on dispose des informations `C.inter` et `C.union`. Puisque `C.inter` représente la partie commune aux éléments de la φ-clique et `C.union` les objets qui diffèrent, on peut facilement construire le graphe G en recomposant chaque sommet de la manière suivante : $\forall x \in$ `C.union` : $x \cup$ `C.inter`. Ensuite il reste à ajouter les arcs entre tous les sommets d'une même φ-clique.
Donc G se déduit facilement de $\mathcal{R}$.

□

On propose maintenant un algorithme qui construit l'ensemble des sommets du φ-graphe (c'est-à-dire la famille $\mathcal{M}$) à partir de l'ensemble des φ-cliques recouvrant le φ-graphe.

Algorithme 2: `CalculFamille(`$\mathcal{C}$`)`

Données : Une collection d'ensembles $\mathcal{C}$ correspondant aux φ-cliques
Résultat : Une collection $\mathcal{M}$ sur J correspondant aux sommets du φ-graphe
début
  $\mathcal{M} \leftarrow \emptyset$
1 **pour chaque** $C = (\texttt{inter}, \texttt{union}) \in \mathcal{C}$ **faire**
2 **pour chaque** $a \in \texttt{union}$ **faire**
3 $\mathcal{M} \leftarrow \mathcal{M} \cup (\texttt{C.inter} \cup \{a\})$
  retourner $\mathcal{M}$
fin

A partir de cet algorithme, un ensemble M, correspondant à un sommet du φ-graphe, peut être retrouvé $|M|$ fois, c'est-à-dire le nombre d'arêtes sortantes. On doit faire cette opération pour tout M, donc pour la somme de toute arête sortante pour chacun des sommets, ce qui correspond dans la matrice à $\sum_{a \in J} \sum_{b \in J} d_{\mathcal{M}}(a, b)$. On arrive à une complexité en $\mathcal{O}(|J| \times ||\mathcal{M}||)$, avec $\mathcal{O}(|J| \times ||\mathcal{M}||) \leq \mathcal{O}(|J|^2 \times |\mathcal{M}|)$. L'opération d'union (*ligne* ***3***) a une complexité de $\mathcal{O}(|J|)$.

Donc dans le pire des cas, cet algorithme a une complexité en temps en $\mathcal{O}(|J|^3 \times |\mathcal{M}|)$.

On propose maintenant un algorithme permettant de récupérer l'ensemble des arêtes du graphe et leurs étiquettes à partir de l'ensemble des φ-cliques recouvrant le φ-graphe.

Algorithme 3: `CalculArêtes(`$\mathcal{C}$`)`

Données : Une collection d'ensembles $\mathcal{C}$ correspondant aux φ-cliques
Résultat : Un couple (E, σ) correspondant aux arêtes du φ-graphe et à leur étiquette
début
  $E \leftarrow \emptyset$
  $\sigma \leftarrow \emptyset$
1 **pour chaque** $C = (\texttt{inter}, \texttt{union}) \in \mathcal{C}$ **faire**
2 **pour chaque** $(a, b) \in \texttt{union}$ **faire**
  $e = ((\texttt{C.inter} \cup \{a\}), (\texttt{C.inter} \cup \{b\}))$
3 $E \leftarrow E + e$
  $\sigma(e) = (a, b)$
  retourner (E, σ)
fin

Le coût des boucles (*lignes* ***1*** *et* ***2***) est égal au nombre d'arêtes, c'est-à-dire en $\mathcal{O}(|J| \times ||\mathcal{M}||)$. L'opération de la *ligne* ***3*** s'exécute en $\mathcal{O}(|J|)$. Donc la complexité totale de cet algorithme est de $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$.

Remarque : si seule la liste σ des étiquettes nous intéresse alors on a $\mathcal{O}(|J| \times ||\mathcal{M}||)$, sachant que cette liste est suffisante pour mettre à jour la *matrice des distances*.

4 Stratégies de résolution

4.1 Algorithme général de résolution du problème *Distance*

On propose ici un algorithme de résolution du problème *Distance*. On utilise l'algorithme présenté précédemment `InitDistance`.

Algorithme 4: `Distance(`$\mathcal{M}$`)`

Données : Une collection d'ensembles $\mathcal{M}$ définie sur J
Résultat : La matrice $d_{\mathcal{M}} = J \times J$ des distances mise à jour
début
1 $d_{\mathcal{M}}$ = `InitDistance`($\mathcal{M}$)
2 G = `CalculGraphe`($\mathcal{M}$)
3 **pour chaque** $\sigma(e) = (a, b) \in G$ **faire**
$d_{\mathcal{M}}(a, b)$ - -
fin

Principe : à partir de la collection $\mathcal{M}$, on souhaite calculer le φ-graphe correspondant.

On peut remarquer qu'à la *ligne* **3** de l'algorithme, seules les étiquettes du graphe sont parcourues. Donc la fonction `CalculGraphe` pourrait se contenter de retourner uniquement la liste des étiquettes.

On rappelle que la complexité de la fonction `InitMatrice` (*ligne* **1**) est en $\mathcal{O}(|J| \times ||\mathcal{M}||)$. La boucle (*ligne* **3**) s'exécute en $\mathcal{O}(|J| \times ||\mathcal{M}||)$. Reste à connaître la complexité de la fonction `CalculGraphe` (*ligne* **1**).

4.2 Type de données abstrait

On utilisera un T.D.A. (Type de Données Abstraits) ***Map*** similaire à l'objet `Map` du langage Java. Cette structure associe une clé à chaque valeur. Ici les clés sont les ensembles de la collection. Les valeurs associées aux clés dépendent de l'algorithme.

Ce T.D.A. doit disposer des opérations suivantes :
- `new()` : crée un objet ***Map*** et retourne un objet vide.
- `get(e)` : retourne la valeur associée à la clé `e` si elle existe et `Nil` sinon.
- `put(e,value)` : insère l'ensemble `e` dans le ***Map*** et lui associe `value`.

La complexité en temps de ces opérations dépend de la structure de données utilisée pour implémenter le T.D.A. ***Map***. On propose une implémentation qui prend en compte le type des clés, c'est-à-dire des ensembles. L'arbre lexicographique permet de représenter les ensembles par les branches.

La structure de données *arbre lexicographique* permet de proposer des algorithmes pour les opérations d'insertion et de recherche admettant une complexité indépendante de la taille de la collection.

Arbre lexicographique

Définition 5

Soit $\mathcal{F}$ une collection de sous-ensembles de X sur un ensemble J muni d'un ordre $<_J$.
On associe à $\mathcal{F}$ un unique arbre $T(\mathcal{F})$ lexicographique, tel que :

1/ chaque arête de l'arbre est étiquetée par un élément de J,
2/ à chaque nœud notifié de l'arbre correspond un ensemble de $\mathcal{F}$,
3/ à chaque ensemble de $\mathcal{F}$ correspond un chemin unique dans l'arbre tel que cet ensemble corresponde à l'union des étiquettes de ce chemin,
4/ l'ordre des arêtes d'un chemin coïncide avec l'ordre $<_J$,
5/ l'ordre des arêtes sortantes d'un nœud coïncide avec l'ordre $<_J$.

Exemple : Soit $J = \{a, b, c, d, e\}$ et $\mathcal{F} = \{abc, bcde, ad, cde, cd, e\}$

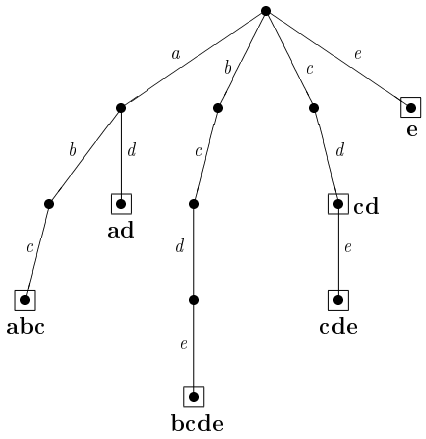


FIG. 8 – Arbre lexicographique

Les nœuds encadrés correspondent aux éléments de $\mathcal{F}$.

Un booléen peut être associé à chaque nœud de l'arbre pour indiquer si le nœud est notifié (c'est-à-dire correspond à un ensemble de la collection) ou non. Le type de champ utilisé pour stocker la valeur associée n'est pas précisé et dépend de l'algorithme utilisé. Seuls les nœuds marqués auront ce champ non nul.

Remarque : le nombre maximum de nœuds de l'arbre $T(\mathcal{F})$ est donné par $||\mathcal{F}||$.

Remarque : une telle structure ne permet pas de stocker les mots du dictionnaire puisque ceux-ci peuvent contenir des occurences multiples de lettres et ne respectent pas l'ordre $<_J$.

Il y a deux façons d'implémenter la liste des fils d'un nœud de l'arbre lexicographique :
- avec une structure `Liste` : chaque élément de la liste correspond à une arête de l'arbre et contient donc le label de l'arête et un pointeur vers le fils,
- avec une structure `Tableau` : le tableau est indicé par les labels possibles et chaque entrée contient soit `Nil` soit un pointeur vers le fils.

La complexité des opérations `put` et `get` dépend alors de l'implémentation choisie.

Proposition 1

Si l'arbre lexicographique est implémenté avec une structure `Liste` *:*
- l'opération `put(M,value)` *a une complexité en temps en* $\mathcal{O}(|J|)$ *;*
- l'opération `get(M)` *a une complexité en temps en* $\mathcal{O}(|J|)$*.*

Les temps d'accès sont dûs au fait qu'un élément n'apparaît qu'une seule fois dans un ensemble. Le coût de création d'un nœud est en temps constant.

Proposition 2

Si l'arbre lexicographique est implémenté avec une structure `Tableau` *:*
- l'opération `put(M,value)` *a une complexité en temps en* $\mathcal{O}(|M| \times |J|)$ *;*
- l'opération `get(M)` *a une complexité en temps en* $\mathcal{O}(|M|)$*.*

Les temps d'accès viennent de l'accès direct aux fils pour un attribut donné. La création d'un nœud de l'arbre coûte $\mathcal{O}(|J|)$ puisque l'on doit créer et initialiser des tableaux de taille $|J|$.

4.3 Stratégie par contrôle de sommets

L'approche par contrôle de sommets est similaire à celle présentée dans [MN].

On dispose de $\mathcal{M}$ et on souhaite obtenir un graphe étiqueté $G = (\mathcal{M}, E, \sigma)$. L'idée est de tester, pour tout sommet $M \in \mathcal{M}$, toutes les arêtes "sortantes" possibles, et ne garder que les valides. C'est-à-dire que pour un M donné, on teste pour tout couple (a, b) s'il existe un $M' \in \mathcal{M}$ tel que $M = \varphi_{a,b}(M')$.

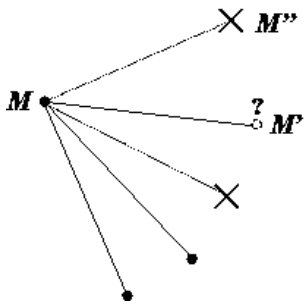


FIG. 9 – Contrôle de sommets

Sur cette figure 9, on travaille sur le sommet M. On teste tous les couples possibles, et on remarque, par exemple, que M'' n'est pas une image de M, et on se pose la question pour M'.

Pour chaque ensemble M de $\mathcal{M}$ et pour chaque couple (a,b) de J^2, on vérifie si $\varphi_{a,b}(M)$ appartient à $\mathcal{M}$. Si c'est le cas alors on ajoute l'arc $e = (M, M')$ dans le φ-graphe avec pour étiquette $\sigma(e) = (a, b)$. Pour stocker la collection $\mathcal{M}$ on utilise le T.D.A. ***Map***.

Algorithme 5: `CalculGraphe(`$\mathcal{M}$`)` : méthode de contrôle de sommets

Données : Une collection d'ensembles $\mathcal{M}$ définie sur J
Résultat : Un graphe étiqueté $G = (\mathcal{M}, E, \sigma)$ correspondant au φ-graphe
début
  $V \leftarrow \mathcal{M}$
  $E \leftarrow \emptyset$
  // Initialisation de la structure ***Map***
  `Map` $\mathcal{T}$ = `new()`
  pour chaque $M \in \mathcal{M}$ **faire**
    $\mathcal{T}$`.put(`$\mathcal{M}$`,"present")`
1 **pour chaque** $M \in \mathcal{M}$ **faire**
2 **pour chaque** *couple* $(a, b) \in J^2$ **faire**
3 **si** $\mathcal{T}$`.get(`$\varphi_{a,b}(M)$`)` $\neq Nil$ *et* $M \neq \varphi_{a,b}(M)$ **alors**
       $e = (M, M')$
4 $E \leftarrow E + e$
       $\sigma(e) = (a, b)$
  retourner $G = (V, E, \sigma)$
fin

Proposition 3
L'algorithme par contrôle de sommets a une complexité en temps en $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$

Preuve
On suppose que l'on utilise l'implémentation de l'arbre lexicographe par la structure `Tableau`. Ainsi l'initialisation de cette structure se fait en $\mathcal{O}(\sum_{M\in\mathcal{M}} |M| \times |J|) = \mathcal{O}(|J| \times ||\mathcal{M}||)$. La boucle de la *ligne* ***1*** exécute $|\mathcal{M}|$ itérations tandis que la boucle de la *ligne* ***2*** en exécute $|J|^2$. A la *ligne* ***3***, le calcul de $\varphi_{a,b}(M)$, comme le test $\varphi_{a,b}(M) \in \mathcal{M}$, se fait en $\mathcal{O}(|M|)$. Ainsi la complexité totale est de $\mathcal{O}(\sum_{M\in\mathcal{M}} |J|^2 \times |M|) = \mathcal{O}(|J|^2 \times ||\mathcal{M}||)$. □

Remarque : cette implémentation peut être optimisée en choisissant a dans M et b dans $J \setminus M$. La complexité dans le pire des cas n'est toutefois pas remise en cause.

Remarque : l'arc (M, M') étant parcouru deux fois, il apparaîtra deux fois dans la liste des arêtes E. En conséquence, il faudra doubler l'initialisation de la matrice des distances dans l'algorithme général.

4.4 Stratégie par contrôle d'arêtes

On dispose d'un graphe étiqueté $G = (\mathcal{M}, E, \sigma)$. L'idée est de tester, pour tout couple d'éléments (M, M') de $\mathcal{M}$, si ils sont reliés par une arête ou non, c'est-à-dire si $M' = \varphi_{a,b}(M)$. Sur la figure 10, on travaille sur chaque couple possible. On note que M''' est l'image de M, ce qui n'est pas le cas pour M''.

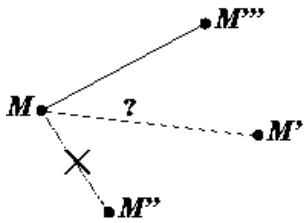


FIG. 10 – Contrôle d'arêtes

Cette nouvelle stratégie s'appuie alors sur une propriété forte des arcs du φ-graphe, la *Propriété 3*. Pour chaque couple d'éléments (M, M') de $\mathcal{M}$, on vérifie si il existe a et b tels que $M = \varphi_{a,b}(M')$. D'après la *Propriété 3*, les items a et b existent si et seulement si M et M' sont de même cardinalité et si ils partagent $|M| - 1$ éléments. Dans ce cas, on ajoute l'arc $e = (M, M')$ dans le φ-graphe avec pour étiquette $\sigma(e) = (a, b)$.

Algorithme 6: `CalculGraphe(`$\mathcal{M}$`)` : méthode de contrôle d'arêtes

Données : Une collection d'ensembles $\mathcal{M}$ définie sur J
Résultat : Un graphe étiqueté $G = (\mathcal{M}, E, \sigma)$ correspondant au φ-graphe
début
  $E \leftarrow \emptyset$
1 **pour chaque** $(M, M') \in \mathcal{M}^2$ **faire**
2 **si** $M \setminus M' = \{a\}$ *et* $M' \setminus M = \{b\}$ *et* $|M| = |M'|$ **alors**
3 $E \leftarrow E + (M, M')$
   $\sigma(M, M') = (a, b)$
  retourner $G = (\mathcal{M}, E, \sigma)$
fin

Proposition 4
L'algorithme par contrôle d'arêtes a une complexité en temps en $\mathcal{O}(|\mathcal{M}| \times ||\mathcal{M}||)$.

Preuve
La boucle de la *ligne* **1** exécute $|\mathcal{M}|^2$ itérations. Tous les tests de la *ligne* **2** peuvent être réalisés en $\mathcal{O}(|M|)$ si les ensembles M et M' sont stockés triés selon l'ordre $<_J$. Ainsi, la complexité globale est de $\mathcal{O}(|\mathcal{M}| \times ||\mathcal{M}||)$.

□

4.5 Stratégie par calcul des φ-cliques

La méthode étudiée ici consiste à construire un recouvrement à partir d'une collection $\mathcal{M}$ sur un ensemble J. En effet, d'après la *Propriété 8*, pour une collection $\mathcal{M}$ d'ensembles sur J, le recouvrement en φ-cliques de $G = (\mathcal{M}, E)$ est une représentation φ-graphe, donc revient à calculer ce dernier. L'idée de l'algorithme suivant est donc de faire abstraction du φ-graphe G. On se pose alors la question de savoir à quelle(s) φ-clique(s) chaque élément de l'ensemble $\mathcal{M}$ appartient.

D'après la *Propriété 3*, on sait que pour tout M d'une φ-clique C, avec $t = |M|$, il existe un objet x de M, tel que $M \setminus x =$ `C.inter`. Lorsqu'un élément est inséré dans le φ-graphe, chaque sous-ensemble de taille $t-1$ qui le compose est susceptible d'être l'`inter` d'une φ-clique. Il est évident que cette φ-clique peut être déjà formée ou apparaître plus tard. On est donc obligé de créer les t φ-cliques de taille 1 auxquelles chacun des sommets peut appartenir, si celui-ci ne fait pas déjà partie d'une φ-clique existante.

Algorithme 7: `CalculGraphe(`$\mathcal{M}$`)` : méthode des φ-cliques

Données : Une collection d'ensembles $\mathcal{M}$ définie sur J
Résultat : Un graphe étiqueté $G = (\mathcal{M}, E, \sigma)$ correspondant au φ-graphe
début
    `Map` $\mathcal{T}$ = `new()`
1   **pour chaque** $M \in \mathcal{M}$ **faire**
2   **pour chaque** $x \in M$ **faire**
            $C = M \setminus \{x\}$
3   `Union` = $\mathcal{T}$.`get`(C)
            si `Union` $\neq Nil$ **alors**
                `Union` = `Union` $\cup \{x\}$
                $\mathcal{T}$.`put`(C,`Union`)
            sinon
4               $\mathcal{T}$.`put`(C,$\{x\}$)
5   (E, σ) = `CalculAretes`($\mathcal{C}$)
    retourner $G = (\mathcal{M}, E, \sigma)$
fin

Proposition 5

L'algorithme admet une complexité en temps en $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$.

Preuve

On suppose que la structure ***Map*** est implémentée en utilisant un arbre lexicographique avec des objets de type `List`. La boucle de la *ligne* ***1*** exécute $|\mathcal{M}|$ itérations, tandis que la boucle de la *ligne* ***2*** exécute $|M|$ itérations. L'affectation de la *ligne* ***3*** se fait en $\mathcal{O}(|J|)$. L'insertion de la *ligne* ***4*** se fait également en $\mathcal{O}(|J|)$. Ainsi cette boucle de l'algorithme s'effectue en $\mathcal{O}(\sum_{M \in \mathcal{M}} |M| \times |J|) = \mathcal{O}(|J| \times ||\mathcal{M}||)$.La fonction `CalculAretes` (*ligne* ***5***) a une complexité temporelle de $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$.

Donc la complexité globale de cet algorithme est $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$.

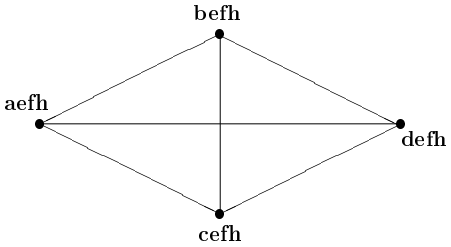
□

Exposons maintenant un exemple d'exécution de cet algorithme.

4.6 Exemple d'exécution

Pour mieux illustrer cet exemple on représentera le φ-graphe, mais celui-ci n'a plus lieu d'être.

Partons du φ-graphe suivant :



et le recouvrement qui lui est associé :

i	1	2	3	4	5	6	7	8	9	10	11	12	13
`Ci.inter`	aef	aeh	afh	efh	bef	beh	bfh	cef	ceh	cfh	def	deh	dfh
`Ci.union`	h	f	e	abcd	h	f	e	h	f	e	h	f	e

Dans la table ci-avant, i représente le numéro de la φ-clique, `Ci.inter` la partie commune aux éléments de la φ-clique C_i, et `Ci.union` les éléments qui diffèrent.

Remarquons que les φ-cliques C_1, C_2, C_3 et C_4 sont les quatre φ-cliques auxquelles peut appartenir l'élément $aefh$. C_4, C_5, C_6 et C_7 pour l'élément $befh$; C_4, C_8, C_9 et C_{10} pour l'élément $cefh$; et C_4, C_{11}, C_{12} et C_{13} pour l'élément $defh$.

Le φ-graphe ci-dessus est la représentation de la φ-clique C_4 puisque `C4.inter` représente les objets communs aux éléments $aefh$, $befh$, $cefh$ et $defh$.

Si l'on souhaitait insérer un élément dans le φ-graphe, on doit traiter trois cas différents pour mettre à jour le recouvrement :
- le sommet ajouté n'appartient à aucune φ-clique existante,
- le sommet vient s'insérer dans une φ-clique de taille 1,
- le sommet vient s'insérer dans une φ-clique de taille supérieure à 1.

• **Insertion d'un sommet n'appartenant à aucune φ-clique existante** :

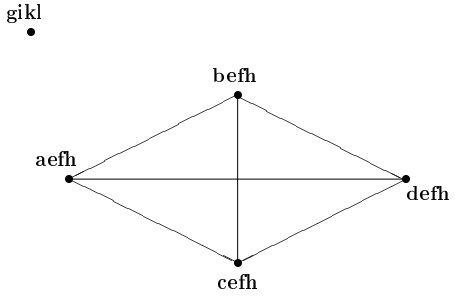
C'est le cas où le sommet ne contient aucun des `Ci.inter`. On doit alors ajouter quatre nouvelles φ-cliques dans la table précédemment définie.

On souhaite insérer l'élément $M = gikl$. Ni gik, ni gil, ni gkl, ni ikl ne définit le $\texttt{C}_\texttt{i}\texttt{.inter}$ d'une φ-clique existante.

On crée quatre nouvelles entrées dans le tableau telles que : $\forall x \ \ \texttt{C}_\texttt{i}\texttt{.inter} = M \setminus \{x\}$ et $\texttt{C}_\texttt{i}\texttt{.union} = \{x\}$:

i	14	15	16	17
$\texttt{C}_\texttt{i}\texttt{.inter}$	gik	gil	gkl	ikl
$\texttt{C}_\texttt{i}\texttt{.union}$	l	k	i	g

Et si on le représente par le φ-graphe associé, on obtient un sommet isolé :

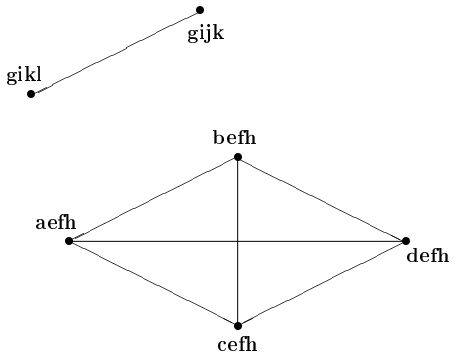


• **Insertion d'un sommet dans une φ-clique de taille** 1 :

Maintenant on insère l'élément $M = gijk$. On doit tester l'appartenance des sous-ensembles $M \setminus \{x\}$ (gij, gik, gjk, ijk) à une φ-clique existante. On trouve que gik est égal à $\texttt{C}_{14}\texttt{.inter}$, donc on doit mettre à jour cette φ-clique en ajoutant $M \setminus \{gik\} = j$ à $\texttt{C}_{14}\texttt{.union}$ et on crée les trois autres φ-cliques :

i	14	18	19	20
$\texttt{C}_\texttt{i}\texttt{.inter}$	gik	gij	gjk	ijk
$\texttt{C}_\texttt{i}\texttt{.union}$	lj	k	i	g

Et sur le φ-graphe on ajoute l'arc (gikl,gijk) :



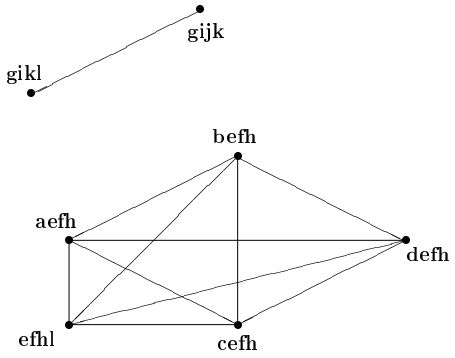
• **Insertion d'un sommet dans une φ-clique de taille supérieure à** 1 :

Enfin on souhaite insérer $M = efhl$ dans ce φ-graphe. Encore une fois on teste l'égalité entre chacun des sous-ensembles $M \setminus \{x\}$, $\forall x$, avec les $\texttt{C}_\texttt{i}\texttt{.inter}$ existants.

$efh =$ `C4.inter`, donc M appartient à la φ-clique C_4, et on doit mettre à jour le `C4.union` avec $M \setminus efh = l$:

i	4	21	22	23
`Ci.inter`	efh	efl	ehl	fhl
`Ci.union`	abcdl	h	f	e

Sur le φ-graphe on doit rajouter toutes les arêtes de $efhl$ vers les autres éléments de la φ-clique C_4 :



4.7 Utilisation de la mémoire

Premièrement il est évident que la matrice des distances doit être stockée en mémoire et requiert un espace mémoire en $\mathcal{O}(|J|^2)$.

Concernant la stratégie de *contrôle de sommets*, un arbre lexicographique implémenté avec des objets de type `Tableau` est utilisé. Le nombre de nœuds est clairement borné par $||\mathcal{M}||$ et puisque chaque nœud requiert un tableau de taille $|J|$, cette stratégie requiert un espace mémoire en $\mathcal{O}(|J| \times ||\mathcal{M}||)$.

Pour la stratégie de *contrôle d'arêtes*, aucun arbre lexicographique n'est utilisé. Toutefois la collection $\mathcal{M}$ doit être présente en mémoire, donc cette stratégie requiert un espace mémoire en $\mathcal{O}(||\mathcal{M}||)$.

Enfin, pour la stratégie de calcul des φ-cliques, l'arbre lexicographique est utilisé avec des objets de type `Liste`. Une telle implémentation requiert un espace mémoire en $\mathcal{O}(||\mathcal{M}||)$. Mais $\mathcal{M}$ n'est pas la collection stockée dans l'arbre. En effet, pour chaque $M \in \mathcal{M}$, on stocke ses $|M|$ sous-ensembles de taille $|M| - 1$. Et puisque $|M|$ est borné par $|J|$ on conclut que cette stratégie nécessite un espace mémoire en $\mathcal{O}(|J| \times ||\mathcal{M}||)$.

En conclusion, on peut dire que, peu importe la stratégie utilisée, il faut au moins $\mathcal{O}(||\mathcal{M}||)$ espace mémoire. Mais on peut noter, d'après la *Propriété 3*, que tous les ensembles de $\mathcal{M}$ n'ont pas besoin d'être présents en mémoire en même temps. En effet, si $M = \varphi_{a,b}(M')$, alors M et M' sont de même taille. L'idée est alors de faire une partition de $\mathcal{M}$ par rapport à la taille des ensembles. Ainsi seuls les ensembles de même taille doivent être en mémoire au même moment. Les calculs réalisés pour ces ensembles sont complètement indépendants des calculs sur les ensembles de taille différente. On remarque que cette partition de $\mathcal{M}$ peut

se faire avec une complexité temporelle en $\mathcal{O}(||\mathcal{M}||)$, et un unique parcours de la collection. Ainsi le partitionnement ne change en rien la complexité globale des différentes stratégies.

Une autre remarque est que, puisque les calculs par taille sont complètement indépendants, on peut facilement implémenter une version répartie de ces stratégies. Ceci peut éventuellement accélérer les calculs. On aura alors des sous-graphes du φ-graphe indépendants les uns des autres, puisque toute composante connexe ne relie que des sommets correspondant à des ensembles de même cardinalité. En effet, en partitionnant la collection $\mathcal{M}$ par les ensembles $\mathcal{M}^t$, correspondant à la restriction de $\mathcal{M}$ aux éléments de taille t, on peut espèrer ainsi optimiser les calculs. Pour ce faire, on doit initialiser t *matrices de distances*, telles que $d^t_{\mathcal{M}}(a,b) = |\mathcal{M}^t_{a\bar{b}}| + |\mathcal{M}^t_{\bar{a}b}|$. Les différents algorithmes prennent alors en paramètre l'ensemble $\mathcal{M}^{\,t}$ au lieu de la collection $\mathcal{M}$ complète. Une fois les calculs réalisés on met à jour la matrice $d^t_{\mathcal{M}}$. Enfin il reste à additionner les t matrices $d^t_{\mathcal{M}}$ pour obtenir la *matrice des distances* $d_{\mathcal{M}}$.

4.8 Complexité du problème *Distance*

On a vu précédemment que le complexité de l'algorithme général (cf. section **4.1**) dépendait de la complexité de l'algorithme `CalculGraphe`, dont on a présenté trois implémentations possibles.

- **Méthode *Contrôle de sommets* :**

La complexité en temps de la méthode par contrôle de sommets admet une complexité en $\mathcal{O}(|J|^2 \times ||\mathcal{M}||)$. Ce qui donne la même complexité pour l'algorithme général.

- **Méthode *Contrôle d'arêtes* :**

La complexité temporelle de l'implémentation par contrôle d'arêtes est de $\mathcal{O}(|\mathcal{M}| \times ||\mathcal{M}||)$. Donc l'algorithme général a une complexité en $\mathcal{O}((|J| + |\mathcal{M}|) \times ||\mathcal{M}||)$.

- **Méthode *Calcul des φ-cliques* :**

On rappelle que seule la liste des étiquettes est nécessaire pour mettre à jour la matrice des distances. On peut donc remplacer la fonction `CalculAretes` à la *ligne* ***5*** de l'algorithme *Calcul des φ-cliques*, par une fonction `CalculEtiquettes` :

Algorithme 8: `CalculEtiquettes`$(\mathcal{C})$

Données : Une collection d'ensembles $\mathcal{C}$ correspondant aux φ-cliques
Résultat : Un ensemble E correspondant aux étiquettes des arêtes du φ-graphe
début
  $E \leftarrow \emptyset$
  pour chaque $C = (\texttt{inter}, \texttt{union}) \in \mathcal{C}$ **faire**
    pour chaque $(a,b) \in \texttt{union}$ **faire**
      $E \leftarrow E + (a,b)$
  retourner (E, E)
fin

La complexité de `CalculEtiquettes` est en $\mathcal{O}(|J| \times ||\mathcal{M}||)$.

Donc, si dans l'algorithme général on utilise cette fonction `CalculEtiquettes` pour retourner les étiquettes, au lieu du graphe G, la complexité totale est alors de $\mathcal{O}(|J| \times ||\mathcal{M}||)$.

Voici un récapitulatif des différentes complexités obtenues pour résoudre le problème *Distance* :

Algorithme	Complexité temporelle	Complexité spatiale
Contrôle de sommets	$\mathcal{O}(\|J\|^2 \times \|\|\mathcal{M}\|\|)$	$\mathcal{O}(\|J\| \times \|\|\mathcal{M}\|\|)$
Contrôle d'arêtes	$\mathcal{O}((\|J\| + \|\mathcal{M}\|) \times \|\|\mathcal{M}\|\|)$	$\mathcal{O}(\|\|\mathcal{M}\|\|)$
Calcul des φ-cliques	$\mathcal{O}(\|J\| \times \|\|\mathcal{M}\|\|)$	$\mathcal{O}(\|J\| \times \|\|\mathcal{M}\|\|)$

FIG. 11 – Tableau récapitulatif des complexités

On en déduit que l'algorithme le plus efficace en temps est celui par la méthode de *calcul des φ-cliques* dans tous les cas. Cet algorithme atteint la complexité minimale évoquée à la section **2.3**. Donc, si les mises à jour se font par décrémentation de 1, *calcul des φ-cliques* est optimal.

5 Sommets jumeaux

Les sommets jumeaux d'un graphe sont des sommets ayant un comportement symétrique vis à vis de leur voisinage. Quel est le lien entre jumeaux d'un graphe et attributs clones ?

Rappelons d'abord que a et b sont clones si et seulement si $d_{\mathcal{M}}(a, b) = 0$

5.1 Définitions

Soit $G = (V, E)$ un graphe non orienté.

Définition 6

*Le **voisinage** ou **voisinage ouvert** d'un sommet s est noté $\mathcal{V}(s)$.*
*Le **voisinage clos** est noté $\mathcal{V}[s] = \mathcal{V}(s) \cup \{x\}$.*

Définition 7

*Deux sommets a et b sont dits **jumeaux** s'ils ont les mêmes voisins. S'ils sont adjacents on dit que ce sont de **vrais jumeaux** (caractérisé par $\mathcal{V}[a] = \mathcal{V}[b]$) ; sinon on dit que ce sont de **faux jumeaux** (caractérisé par $\mathcal{V}(a) = \mathcal{V}(b)$).*

On s'intéresse ici plus particulièrement à la notion de vrais jumeaux.

On note que cette relation *jumeaux* est une relation d'équivalence :

⋉ *réflexive* : pour tout $x \in V$: $\mathcal{V}[x] = \mathcal{V}[x]$. Un sommet a les mêmes voisins que lui-même.

⋉ *symétrique* : pour tout $x, y \in V$: si $\mathcal{V}[x] = \mathcal{V}[y]$ alors $\mathcal{V}[y] = \mathcal{V}[x]$. Si y a les mêmes voisins que x, alors x a les mêmes voisins que y.

⋉ *transitive* : pour tout $x, y, z \in V$: si $\mathcal{V}[x] = \mathcal{V}[y]$ et $\mathcal{V}[y] = \mathcal{V}[z]$ alors $\mathcal{V}[x] = \mathcal{V}[z]$. Si x et y ont les mêmes voisins et si y et z ont aussi les mêmes voisins, alors x et z ont les mêmes voisins, donc sont jumeaux.

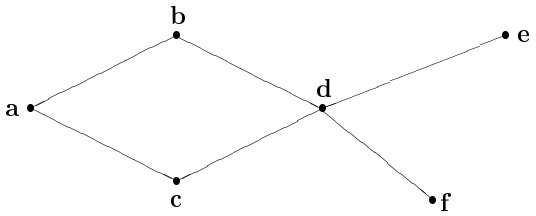
5.2 Lien entre *Jumeaux* et *Clones*

On définit le problème suivant :

Problème 5 : Recherche de sommets jumeaux
Entrée : un graphe $G = (V, E)$
Sortie : une partition $\mathcal{P}$ des sommets de G en classes de jumeaux

Exemple :

G :



$$\mathcal{P} = \{(e, f), (b, c), (a), (d)\}$$

Dans [Pau98], est proposé un algorithme de calcul des sommets jumeaux, avec une complexité en temps en $\mathcal{O}(n + m)$, où n est le nombre de sommets, et m est le nombre d'arêtes du graphe G. De même, dans [dM03], on trouve un algorithme de recherche de jumeaux dans un graphe, avec une complexité temporelle en $\mathcal{O}(n + m)$.

On rappelle le problème de recherche des clones :

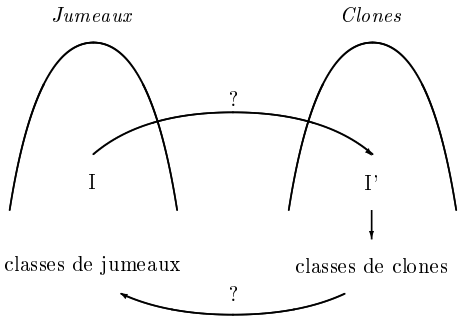
Problème 1 : Détection des classes de clones
Entrée : $\mathcal{F}$ une famille d'ensembles
Sortie : $C_1, ..., C_k$ les classes de clones de $\mathcal{F}$

Proposition 6

Le problème Recherche de sommets jumeaux et le problème Calcul des classes de clones sont équivalents.

La preuve est donnée par les deux transformations suivantes :

5.2.1 Transformation #1



Lemme 1

Soit un graphe $G = (V, E)$. a, b jumeaux dans $G \iff a, b$ clones dans E.

Preuve du Lemme 1

a et b sont jumeaux dans $G = (V, E)$.

$\Leftrightarrow \mathcal{V}[a] = \mathcal{V}[b]$

$\Leftrightarrow \forall x \in \mathcal{V}[a] \ : \ (a, x) \in E$ et $(b, x) \in E$

$\Leftrightarrow \forall (x, y) \in E$ on a $\varphi_{a,b}(\{x, y\}) \in E$

$\Leftrightarrow$ a et b sont donc clones dans E.

□

Corollaire 2

Dans les familles d'ensembles à 2 éléments, les attributs clones peuvent être détectés en temps linéaire.

Preuve du Corollaire 2

Il suffit de construire le graphe $G = (V, \mathcal{F})$ et d'appliquer l'algorithme de [Pau98].

□

Corollaire 3

a, b faux jumeaux dans $G \iff a, b$ clones dans E et $(a, b) \notin E$.

Preuve du Corollaire 3

Ceci vient du fait que les vrais jumeaux sont reliés par une arête.

□

Corollaire 4

Un graphe est sans jumeau si et seulement si il n'y a pas de clone dans les arêtes.

Preuve de la transformation # 1

- Comment passer de I à I' ?

I correspond à une instance du problème *Jumeaux*, c'est-à-dire un graphe $G = (V, E)$, quant à I', c'est une instance du problème *Clones*, donc une famille $\mathcal{F}$ d'ensembles. On insère dans $\mathcal{F}$ les ensembles MM' tels que $e = (M, M') \in E$ pour obtenir ainsi I'. Ensuite on calcule les clones sur cette collection.

- Comment retrouver les sommets jumeaux à partir des classes de clones ?

D'après le *Lemme 1*, à partir des classes de clones, on retrouve exactement les classes de sommets jumeaux.

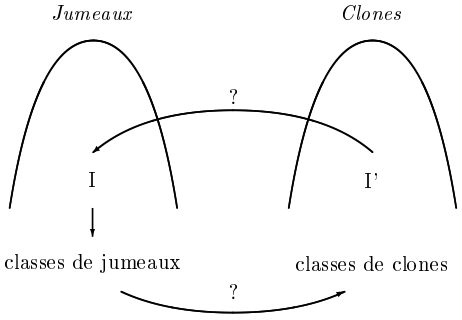
□

Exemple :

Sur le graphe précédent, si on construit I' on obtient : $\mathcal{F} = \{ab, ac, bd, cd, de, df\}$. L'algorithme de recherche des clones donne les classes suivantes : $\mathcal{C} = \{\{a\}, \{b, c\}, \{d\}, \{e, f\}\}$. Et

il est clair que sur ce graphe on retrouve bien b et c jumeaux ainsi que e et f.

5.2.2 Transformation #2



On définit $Bip^t_{a,b} = (J \setminus \{a,b\}, (\mathcal{M}_{a\bar{b}} \cup \mathcal{M}_{\bar{a}b}) \cap \mathcal{M}^t, \in)$

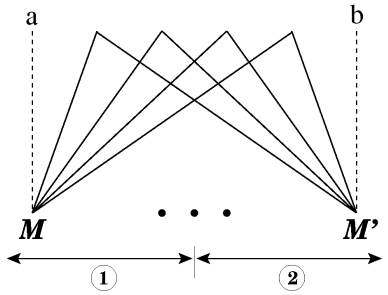


FIG. 12 – Graphe biparti

① correspond aux éléments tels que $a \in M$ et $b \notin M$
② correspond aux éléments tels que $a \notin M'$ et $b \in M'$

On ne peut pas avoir d'éléments M'' tels $b \in M''$ et $a \notin M''$ avec $\mathcal{V}(M') = \mathcal{V}(M'')$. En effet, dans un tel cas, on a M' et M'' qui ont les mêmes voisins et contiennent b tous les deux, or ce n'est pas possible puisque la collection $\mathcal{M}$ ne contient pas de doublon.

On en déduit, pour les mêmes raisons, que sur ce biparti les classes de jumeaux ne peuvent être que de taille 1 ou de taille 2.

Donc M et M' sont jumeaux si et seulement si $M = \varphi_{a,b}(M')$.

Lemme 2

Soit $\mathcal{M}$ une famille d'ensembles. $\forall M, M' \in \mathcal{M}$ avec $M \neq M'$, $M = \varphi_{a,b}(M') \iff M$ et M' sont jumeaux dans $Bip^t_{a,b}$.

Preuve du Lemme 2

$\circ \Longrightarrow$:

$M = \varphi_{a,b}(M')$ et $M \neq M'$

$\Rightarrow M \setminus M' = \{a\}$ et $M' \setminus M = \{b\}$

$\Rightarrow M \setminus \{a\} = M' \setminus \{b\} = inter$

$\Rightarrow$ dans $Bip^t_{a,b}$: $\mathcal{V}(M) = inter$ et $\mathcal{V}(M') = inter$

donc M et M' sont jumeaux dans $Bip^t_{a,b}$

$\circ \Longleftarrow$:

$M, M' \in (\mathcal{M}_{a\bar{b}} \cup \mathcal{M}_{\bar{a}b}) \cap \mathcal{M}^t$. Puisqu'ils sont jumeaux, on a $\mathcal{V}(M) = \mathcal{V}(M')$ dans $J \setminus \{a, b\}$. De plus, $|\mathcal{V}(M)| = t - 1$ car $M \in (\mathcal{M}_{a\bar{b}} \cup \mathcal{M}_{\bar{a}b}) \cap \mathcal{M}^t$ et donc soit $a \in M$ soit $b \in M$. Il en est de même pour M'. Comme $M \neq M'$ on a deux cas :

$1/\ a \in M \Rightarrow b \in M' \Rightarrow M = \varphi_{a,b}(M')$

$2/\ b \in M \Rightarrow a \in M' \Rightarrow M = \varphi_{a,b}(M')$

□

Corollaire 5

a et b ne sont pas clones dans $\mathcal{M} \Longleftrightarrow \exists t$ et $M \in Bip^t_{a,b}$ et M n'admet pas de jumeau dans $Bip^t_{a,b}$.

Autrement dit, dans $Bip^t_{a,b}$, toutes les classes de jumeaux sont de taille 2 et il n'existe pas de M sans jumeau.

Preuve du Lemme 5

$\circ \Longrightarrow$:

Si a et b ne sont pas clones alors il existe un M dans $\mathcal{M}$ tel que $\varphi_{a,b}(M)$ n'est pas dans $\mathcal{M}$. Donc M n'admet pas de jumeaux dans $Bip^{|M|}_{a,b}$ (d'après le *Lemme 2*).

$\circ \Longleftarrow$:

Soit $M \in Bip^t_{a,b}$ un élément qui n'a pas de jumeau dans $Bip^t_{a,b}$. alors, d'après le *Lemme 2*, il n'existe pas de M' tel que $M = \varphi_{a,b}(M')$. Donc a et b ne sont pas clones.

□

Preuve de la transformation # 2

- Comment passer de I' à I ?

On produit $\forall a, b \in J$ et $\forall t$ les graphes bipartis $Bip^t_{a,b}$. Puis on détecte les classes de jumeaux sur chacun de ces graphes.

- Comment retrouver les classes de clones à partir de l'ensemble des sommets jumeaux ?

On travaille par taille, c'est-à-dire que les éléments inférieurs du graphe biparti sont tous de même cardinalité. D'après le *Corollaire 5*, a et b sont clones si et seulement si, pour tout t, ils sont jumeaux dans $(J \setminus \{a, b\}, (\mathcal{M}_{a\bar{b}} \cup \mathcal{M}_{\bar{a}b}) \cap \mathcal{M}^t, \in)$.

□

6 Conclusion

Nous avons présenté la notion de similitude (et sa notion duale de distance) entre deux attributs et posé le problème du calcul de cette mesure.

Dans un premier temps, sous l'hypothèse que seules des opérations d'incrémentation (ou décrémentation) unitaire étaient autorisées, nous avons montré que ce problème ne pouvait pas être résolue en moins que $\mathcal{O}(|J| \times ||\mathcal{M}||)$.

Une modélisation du problème par φ-graphe a été proposée, nous permettant de ramener ce problème à celui du calcul du φ-graphe associé (plus précisemment au calcul des étiquettes du φ-graphe).

A partir de là, plusieurs stratégies de calcul de ce φ-graphe ont été présentées. La meilleure d'entre elles admet une complexité en $\mathcal{O}(|J| \times ||\mathcal{M}||)$. Elle est donc optimale sous l'hypothèse de l'incrémentation unitaire.

Enfin, nous avons montré le lien qui existe entre la notion de sommets jumeaux d'un graphe et celle d'attributs clones.

Nous pourrions alors nous interroger sur l'existence d'un algorithme efficace de calcul des classes de clones. La première idée étant de ne pas avoir à calculer la matrice des distances. La seconde résidant dans le fait que les algorithmes présentés ne permettent que des décrémentations unitaires, pourrait-on alors optimiser par une décrémentation par paquets ?

On a montré que a et b sont jumeaux dans un graphe si et seulement si a et b sont clones dans les arêtes. On pourrait alors généraliser la notion de jumeaux dans les hypergraphes avec une algorithmique efficace associée. Ceci permettrait de proposer une nouvelle façon de réduire les hypergraphes. Actuellement, à ma connaissance, il n'y en a que deux : suppression de sommets pendants (un sommet n'appartient qu'à une hyperarête) et suppression d'hyperarêtes incluses (pour obtenir un hypergraphe simple).

Références

[dM03] Fabien de Montgolfier. *Décomposition modulaire des graphes - Théorie, extensions et algorithmes.* PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc -, 2003.

[GMS00] Michel Goosens, Franck MittelBach, and Alexander Samarin. *LaTeX Companion.* CampusPress, 2000.

[Lan01] Jean-Marc Lanlignel. *Autour de la décomposition en coupes.* PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc -, 2001.

[LRT00] Jean-Marc Lanlignel, Olivier Raynaud, and Eric Thierry. Pruning graphs with digital search trees. application to distance hereditary graphs. In *Proceedings 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2000)*, volume 1770 of *Lecture Notes in Computer Science*, pages 529–541. Springer-Verlag, 2000.

[MN] Raoul Medina and Lhouari Nourine. Clone items : a pre-processing information for knowledge discovery. *Soumis.*

[Pau98] Christophe Paul. *Parcours en Largeur Lexicographique : un Algorithme de Partitionnement - Application aux Graphes et Généralisations.* PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc -, 1998.