Isomorphisme de graphes

LEFÈVRE Jonas

avril 2009

Table des matières

1 Introduction

La question est de savoir si deux graphes sont “les mêmes”. Est-ce qu'il existe un renommage des sommets qui permet de passer de l'un à l'autre ? C'est un problème fondamental puisque c'est la décidabilité d'une égalité sur les graphes qui est en jeu. Par ailleurs ce problème a des applications plus pratiques assez nombreuses : en réseau, en chimie, en biologie,en bases de données,...

Ce problème est en général NP-complet mais dans certains cas il est plus facile, nous verrons donc d'abord des résultats sur sa complexité. Ensuite on étudiera comment le résoudre effectivement pour les cas “simples” et même un peu pour les cas plus durs. Finalement, on s'interressera à la notion de plongement qui est une variante du problème de l'isomorphisme.

Il est acquis qu'on peut facilement tester le fait qu'ils soient non-isomorphes en testant les nombres d'arrêtes, de sommets, les degrés... On supposera donc que les graphes considérés n'ont pu être distingués par ces tests préléminaires. Il est possible de discuter de la pertinence de tels tests au début d'algorithme linéaires ou presque.

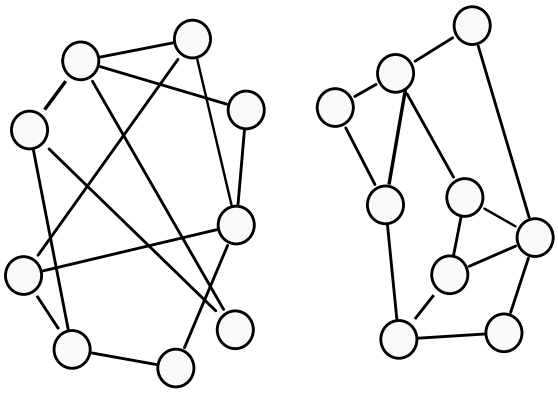

FIG. 1 – Deux graphes isomorphes

JEU : Dessiner l'isomorphisme entre les deux graphes de la Figure 1

2 Définitions

Définition 1 *Un arbre est un graphe connexe sans cycle. On dit que l'arbre est enraciné si on distingue un sommet appellé la racine. On oriente alors naturellement un arbre enraciné à partir de sa racine. On définit alors le niveau d'un noeud par :*

- *le noeud le plus éloigné de la racine est de niveau 0*
- *le niveau du père d'un noeud est un de plus que le niveau de son fils*

Cela caractérise la distance à la racine (qui est de niveau maximal).

Définition 2 *Soit $G_1 = (E_1, V_1)$ et $G_2 = (E_2, V_2)$ deux graphes. Un isomorphisme de graphe de G_1 dans G_2 est une bijection ϕ de E_1 dans E_2 qui vérifie :*

$$(i, j) \in V_1 \Leftrightarrow (\phi(i), \phi(j)) \in E_2$$

Définition 3 *On définit un cographe par réccurence :*

- *un sommet est un cographe.*
- *si $G_1 = (E_1, V_1)$ et $G_2 = (E_2, V_2)$ sont deux cographes, $H = (E_1 \cup E_2, V_1 \cup V_2)$ est un cographe.*
- *si $G_1 = (E_1, V_1)$ et $G_2 = (E_2, V_2)$ sont deux cographes, $H = (E_1 \cup E_2, V_1 \cup V_2 \cup V_1 \times V_2 \cup V_2 \times V_1)$ est un cographe.*

Je renvoie le lecteur à l'excellent cours de Graphes d'Eric Thierry pour les propriétés des cographes. Je les utiliserais dans la suite sans plus de références.

Définition 4 *Soit $G_1 = (E_1, V_1)$ et $G_2 = (E_2, V_2)$ deux graphes. Un plongement de G_1 dans $G_2 =$ est une injection ϕ de E_1 dans E_2 telle que pour toute arête $(i, j) \in V_1$ il existe un chemin disjoint de $\phi(i)$ à $\phi(j)$.*

3 Complexité

3.1 Cas général

Théorème 1 *Décider de l'existence d'un isomorphisme entre deux graphes G_1 et G_2 est dans NP.*

Démonstration On peut vérifier en temps linéaire en le nombre d'arrête qu'une fonction est bien un isomorphisme entre deux graphes. Le problème est donc dans NP.

Il n'existe, à ce jour, pas de résultat sur la NP-complétude de ce problème. Cependant certaines variantes ou restrictions sont soit plus facile, soit plus dures. Ce problème est cependant très riche en résultats de complexité ; au cours du premier semestre nous avons ainsi démontré qu'il était dans AM et dans zero-knowledge (c'est-à-dire que l'on peut prouver que l'on connait la solution sans l'exhiber).

3.2 Cas particuliers "faciles"

Théorème 2 *Décider de l'existence d'un isomorphisme entre deux arbres enracinés G_1 et G_2 est dans P.*

Démonstration On verra plus loin un algorithme polynomial qui résoud ce problème. Il démontrera donc ce théorème.

Théorème 3 *Décider de l'existence d'un isomorphisme entre deux arbres G_1 et G_2 est dans P.*

Démonstration Étant donné le théorème précédent, il suffit d'enraciner arbitrairement G_1 (en choisissant un sommet dont le degré est le moins réalisé dans le graphe par exemple).

On teste ensuite l'isomorphisme en enracinant G_2 successivement sur tout les sommets de même degré.

Au pire, on multiplie la complexité par le nombre de sommets, on reste donc polynomial.

Théorème 4 *Décider de l'existence d'un isomorphisme entre deux cographes G_1 et G_2 est dans P.*

Démonstration On construit les arbres canoniques associés à G_1 et G_2 en tant que cographes. Ces arbres sont enracinés.

On applique alors l'algorithme pour ces deux arbres et on déduit trivialement d'un isomorphisme entre les co-arbres un isomorphisme entre les cographes.

3.3 Cas particulier “durs”

Théorème 5 *Décider de l'existence d'un isomorphisme entre graphes dirigés sans cycle (DAG) G_1 et G_2 est exactement aussi dur que pour deux graphes généraux.*

Démonstration Le problème est clairement dans NP.

On réduit à partir de l'isomorphisme de graphe.

On considère la transformation suivante :

pour toute arête $e = (i, j) \in E$,on la supprime et on construit un nouveau sommet e et les deux arcs orienté (i, e) et (j, e).

Sur la figure 2, on peut voir un graphe qui a subit cette transformation, les anciens sommets apparraissent en noir et les nouveaux en blanc. Cette

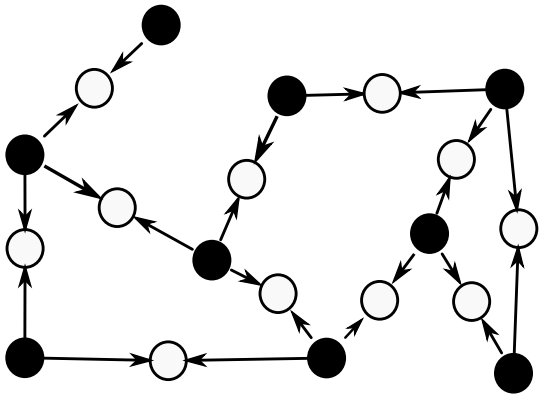

FIG. 2 – Graphe transformé

transformation est polynomiale et transforme un graphe non orienté en un grahe orienté acyclique.

On effectue cette transformation sur les deux graphes. On regarde si les deux DAG sont isomorphes. On a donc une transformation polynomiale d'un problème en l'autre. On en déduit l'équivalence en terme de difficulté des deux problèmes (l'autre sens étant évident).

4 Algorithmes

4.1 Pour les arbres enracinés, en temps polynomial

Comme annoncé précédemment nous allons voir ici un algorithme qui résoud le problème de l'isomorphisme d'arbres enracinés T_1 et T_2.

Isomorphisme(T_1,T_2) =

Initialisation :

- $\forall i \in E_1$, $T_1[i] \leftarrow 0$ et $\forall i \in E_2$, $T_2[i] \leftarrow 0$
- On classe les sommets par niveau (d'après la définition de la section 1).

Boucle au niveau i : On suppose que tout les sommets des niveaux $i-1$ a été numéroté, et qu'on a les listes L_1 (resp. L_2) des sommets de T_1 (resp. T_2) du niveau $i-1$ classés par ordre croissant suivant la numérotation.

1. Pour tout sommet s de T_1 de niveau i et qui n'est pas une feuille, on construit l'ulpe suivant $t_s = (i_{v_0}, \ldots, i_{v_k})$ où les $(v_l)_{l \leq k}$ sont les voisins de s dans le niveau $i-1$, i_w est le numéro du sommet w et on a $i_{v_0} \leq \ldots \leq i_{v_k}$.
2. On pose alors $S_1 = \{t_s | s \text{ sommet de niveau i dans } T_1\}$.
3. Faire de même avec T_2.
4. On trie S_x en S'_x pour $x = 1, 2$.

Fin de Boucle : Si $S'_1 \neq S'_2$ alors on rejete sinon on numerote les sommets du niveau i par leur place dans S'_x et on repasse dans la boucle.

Conclusion : Si l'algorithme ne rejete pas les racines (*i.e.*niveau maximal) alors on a un isomorphisme entre T_1 et T_2 et il est donné par la numérotation par niveau.

L'étude de complexité est assez simple.

Les phases d'initialisation, de fin de boucle et de conclusion s'effectuent en temps linéaire.

La boucle prend $O(\sum_{v \text{ de niveau } i} deg(v) \log deg(v))$ pour la construction de l'ulpe et $O(n_i \log n_i)$, si il y a n_i sommets de niveau i, pour trier la liste. Une boucle prend donc au total $O(n_i \log n_i)$. Chaque sommet ne passant qu'une fois dans la boucle, on a une complexité totale pour l'algorithme qui est en $O(n \log n)$ où n est le nombre de sommets. On peut optimiser cet algorithme pour atteindre le linéaire, en utilisant un tri linéaire.

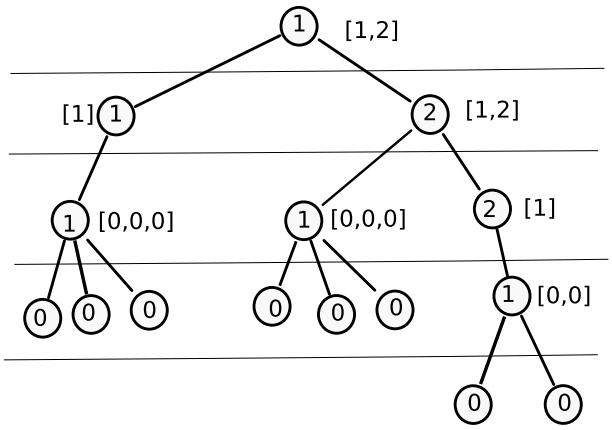

FIG. 3 – Arbre où a été exécuté l'algorithme

4.2 Pour les graphes en général, en temps exponentiel

Bien que classé difficile (*i.e.* il n'est *a priori* pas dans P), il existe des algorithmes pour tester si deux graphes sont isomorphes. L'algorithme le plus efficace connu actuellement a une complexité en $O(e^{\sqrt{cn.log(n)}})$, il est du à Luks et Zemlyachenko. L'algorithme suivant est en $O(n!)$ et grosso modo il revient à tester toutes les possibilités.

Isomorphisme(G_1,G_2) =

Tests : On teste avant toute chose que les deux graphes on :
- le même nombre de sommets
- le même nombre d'arrêtes
- le même nombre de sommets de degré d pour tout d

On rejete si ce n'est pas le cas.

Initialisation :
- On pose M=$\{\}$ qui sera le graphe de l'isomorphisme (c'est-à-dire l'ensemble des couples $(v, w) \in V_1 \times V_2$ tels que w soit l'image de v).
- On pose $S_1 = V_1$ et $S_2 = V_2$ les ensembles des sommets non traités et $T = \{\}$ les sommets traités de G_1.

Boucle :

1. On choisit $v \in S$ et $w \in S_2$ tel que : $\forall u \in T, (u, v) \in E_1 \Leftrightarrow (w, u') \in E_2$ où on a$(u, u') \in M$

2. Si ceci est possible, on retire v de S_1 et w de S_2 et on ajoute v à T ; et on recommence la boucle. Si l'appel suivant est rejeté, on recommence la boucle en interdisant le choix de v.
3. Sinon on rejete.

Conclusion : Si S_1 est vide alors on a M qui est un isomorphisme de G_1 dans G_2.

Théorème 6 *Étant donnée l'existence d'un isomorphisme entre deux graphes G_1 et G_2, calculer un tel isomorphisme est dans P*

Démonstration On considère G_1 et G_2 deux graphes que l'on sait isomorphes.

Construction_Isom(G_1,G_2) =

Boucle : Pour tout sommet s des deux graphes

1. On construit un pseudo-arbre de racine s, il est organisé par niveau, c'est-à-dire que l'on regroupe par niveau les sommets.
2. On peut alors associer à tout sommet v un quadruplet $q_s(v) = (i, n_i, n_{i-1}, n_{i+1})$ où :
   - i est le niveau où v se trouve.
   - n_{i-1} est le nombre de voisins de v au niveau $i-1$.
   - n_i est le nombre de voisins de v au niveau i.
   - n_{i+1} est le nombre de voisins de v au niveau $i+1$.
3. On associe à chaque sommet v l'ensemble trié des quadruplets $\{q_v(s)\}_{v \in V}$

Fin :On identifie les éléments qui ont le même ensemble de quadruplets, et on construit ainsi l'isomorphisme entre G_1 et G_2.

Cette construction repose sur le fait que l'isomorphisme respecte le voisinage et les distances, donc les niveaux.

Chaque boucle prend un temps en $O(n^2)$, on voit alors qu'on peut borner la complexité totale par $O(n^3)$. L'algorithme est bien polynomial.

SOLUTION DU JEU : Essayer cet algorithme sur la Figure 1

5 Plongement et sous-graphe

5.1 Plongement

Le problème du plongement est donc de trouver une manière de voir un graphe G_2 comme une subdivision du graphe G_1. Ce problème a des applications directes dans les problèmes de routages sur des réseaux.

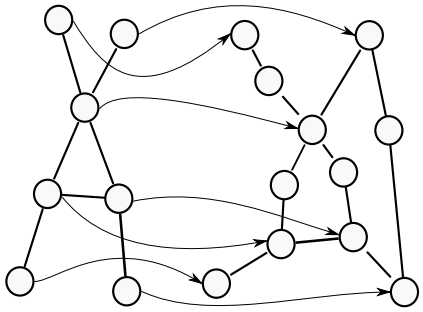

FIG. 4 – Plongement

Théorème 7 *Décider de l'existence d'un plongement d'un graphe G_1 dans un graphe G_2 aussi dur que le problème d'isomorphisme de graphe.*

Démonstration Le problème est clairement au moins aussi dur que l'isomorphisme de graphe : si il existe un plongement de G_1 dans G_2 et qu'ils ont le même nombre de sommet alors ils sont isomorphes.

Il semblerait qu'il soit NP-complet mais je n'ai pu réussir à trouver une preuve satisfaisante.

5.2 Sous-graphe isomorphe

L'isomorphisme de sous-graphe est une restriction du problème précédent puisqu'ici on ne s'autorise plus la subdivision des arrêtes.

Théorème 8 *Décider de l'existence d'un sous-graphe d'un graphe G_2 tel qu'il soit isomorphisme à un graphe G_1 est NP-complet.*

Démonstration Le problème est encore une fois clairement dans NP.

On peut réduire ce problème à partir du problème de clique_max : On teste pour chaque graphe complet si il est isomorphe à un sous-graphe de notre graphe. On pourrait ainsi déterminer la taille de la clique maximum.

5.3 Isomorphisme de sous-arbre

Théorème 9 *Décider de l'existence d'un sous-arbre de T_2 tel qu'il soit isomorphe à un arbre T_1 est dans P.*

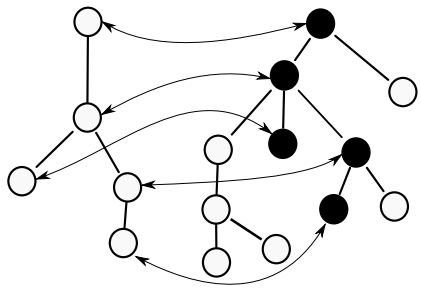

FIG. 5 – Isomorphisme de sous-arbre

(non-)Démonstration Je ne démontrerai pas ici ce résultat mais voici quelques idées qui permettent de comprendre c'où vient ce résultat. On peut se contenter de travailler sur deux arbres enracinés et et de rechercher un sous-arbre de T_2 isomorphe à T_1 enraciné à la racine de T_2. En testant les différentes racines possibles pour T_2 (tout sommet dont le degré est supérieur ou égale à celui de la racine de T_1 peut convenir) on peut se ramener au cas général non enraciné.

6 Conclusion

Le problème de l'isomorphisme de graphe est donc un problème centrale de l'informatique. De part sa nature, il intérresse dans beaucoup de domaines (et même en dehors de l'informatique). Il pose aussi la question de reconnaitre que deux graphes sont en fait les mêmes. Dans le cas général c'est un problème difficile à résoudre mais il semble appartenir à une classe intermédiaire entre P et NP (qui a été nommée GI en son honneur). Il possède de nombreuses variantes qui sont ou plus dures ou plus simples ou aussi difficiles, cela est une preuve de sa richesse. De manière générale on arrive à bien travailler avec ce problème qu'en se ramenant à des arbres, pour le reste on est à peine capable de faire mieux que l'énumération exhaustive des bijections entre les sommets. Bien que les résultats soient déjà nombreux sur le sujet (je n'en ai donné ici qu'un bref aperçu), il reste donc encore beaucoup à faire.

Références

[1] The design and analysis of computer algorithms, livre de Aho, Hopcroft et Ullman, Addison-Wesley (1974).

[2] Algorithms on trees and graphes, livre de G. Valiente, Springer (2002).

[3] Graphes, cours d'E. Thierry (2009)