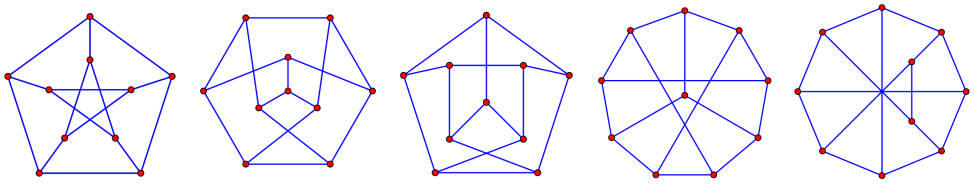
# Algorithmique et Programmation

## Projet : Heuristique de Weisfeiler-Lehman pour l'isomorphisme de graphe

Ecole normale supérieure
Département d'informatique
td-algo@di.ens.fr

2011-2012

Deux graphes sont isomorphes s'il est possible de renomer les sommets du premier de telle manière qu'on obtienne le deuxième. Plus formellement, deux graphes non-orientés $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont isomorphes s'il existe une bijection $\rho : V_1 \to V_2$ telle que $\{x, y\} \in E_1$ équivaut à $\{\rho(x), \rho(y)\} \in E_2$. Par exemple, les 5 graphes suivants sont tous isomorphes :



On ne connaît pas d'algorithme polynomial permettant de déterminer si deux graphes sont isomorphes. Il existe un algorithme *sous-exponentiel*, en $\mathcal{O}\left(e^{\sqrt{n \log n}}\right)$ [1]. Cet algorithme est difficile à implémenter efficacement, en partie parce qu'il repose de façon cruciale sur la théorie des groupes de permutations.

Cependant, on dispose de plusieurs techniques (on dit des "heuristiques") qui permettent de résoudre la plupart des instances du problème efficacement, et qui sont en pratique bien plus efficace que l'algorithme sous-exponentiel sauf dans des cas très particuliers. Il est en fait relativement difficile de fabriquer des graphes qui mettent simultanément toutes ces techniques à la peine (cf. [2]).

L'ensemble des méthodes qu'on désigne collectivement sous le nom d'heuristique de Weisfeiler-Lehman comptent parmi les plus simples de ces techniques. Elles ont été inventées dans les années 1970 (et apparement même dans les années cinquante en URSS), mais leur temps d'exécution peut être exponentiel dans le pire des cas. Il n'en démeure pas moins qu'elles terminent en temps polynomial la plupart du temps.

On rappelle qu'étant donné un sous-ensemble $H \subseteq V$, alors le sous-graphe induit par $H$ est le graphe dont les sommets sont $H$ et les arêtes sont les arêtes de $G = (V, E)$ qui relient ensemble des sommets de $H$ (c'est en quelque sorte la "restriction de $G$ à $H$"). On note $\Gamma(x)$ l'ensemble des sommets adjacents à $x$, et on note $\overline{X}$ le complémentaire (dans $V$ la plupart du temps) d'un ensemble. Le degré d'un noeud $x$ est la cardinalité de $\Gamma(x)$.

# 1 Idée de base et intuition

**Backtracking.** L'idée de base est de spécifier $\rho$ progressivement. Une fois qu'on a construit un isomorphisme $\rho : H_1 \to H_2$ entre les deux sous-graphes $G_1(H_1)$ et $G_2(H_2)$, on cherche à l'étendre à des sous-graphes plus grand (d'un sommet, typiquement). Si cela s'avère impossible, c'est que le choix de $\rho$ était mauvais. Si tous les choix de $\rho$ sont mauvais, c'est que $G_1$ et $G_2$ ne sont pas isomorphes.

En gros, à chaque étape, on fixe l'image de $\rho$ sur un nouveau point $x \in V_1$, et on essaye de vérifier s'il n'y a pas de contradiction. En cas de contradiction, on revient sur un/des choix précédent(s). C'est donc une méthode de *backtracking*. Une contradiction peut se présenter de deux manières :

1. Si $\rho(u)$ et $\rho(v)$ sont définis et que $\{u, v\} \in E_1$, alors $\{\rho(u), \rho(v)\} \notin E_2$ est contradictoire.
2. Si $\rho(u)$ et $\rho(v)$ sont définis et que $\{\rho(u), \rho(v)\} \in E_2$, alors $\{u, v\} \notin E_1$ est contradictoire.

Si on a trouvé une contradiction, c'est que le choix qu'on a fait en définissant $\rho(x)$ était mauvais, et il faut en essayer une autre. Si *toutes* les manières de définir $\rho(x)$ aboutissent à une contradiction, alors c'est qu'on a fait une erreur auparavant, en choisissant l'image d'un point précédent.

On a tout intérêt à détecter les contradictions le plus tôt possible. En même temps, si on choisit arbitrairement la valeur de $\rho(x)$, les seules contradictions qui peuvent provenir de ce choix-là mettent en jeu des arêtes qui touchent $x$ ou $\rho(x)$.

**Élagage de l'arbre de recherche.** Sous cette forme de base, l'algorithme pourrait tester toutes les permutations possibles de l'ensemble des sommets, ce qui donnerait une complexité en $\mathcal{O}(n!)$. Une des observations cruciale pour améliorer l'algorithme de base consiste à dire que deux graphes isomorphes ont nécessairement le même nombre de sommets.

En particulier, si $G_1$ et $G_2$ sont isomorphes, et si un isomorphisme envoie $x \in V_1$ sur $y \in V_2$, alors les voisinages de $x$ et de $y$ sont des graphes isomorphes. Un voisinage (de rayon $r$) d'un sommet $x \in V_1$ est formé de l'ensemble $V_r(x)$ des sommets qui sont accessibles depuis $x$ par un chemin de longueur au plus $r$, ainsi que de l'ensemble des arêtes de $E_1$ qui relient les sommets du voisinage entre eux. Ainsi, on peut affirmer que :

$$x \in V_1 \text{ est envoyé sur } y \in V_2 \implies \forall r \geq 0,\ V_r(x) \text{ et } V_r(y) \text{ sont isomorphes.}$$

Il y a plusieurs manières d'exploiter cette observation. La plus simple, et pas la moins efficace consiste à dire que deux graphes isomorphes ont nécessairement le même nombre de sommets. Il ne sert donc à rien de poser $\rho(x) = y$ si $x$ et $y$ ont des voisinages de tailles différentes pour un certain rayon. Les tailles des voisinages peuvent être calculées efficacement avec un parcours en profondeur.

# 2 Méthode de Weifeiler-Lehman

Dans cette section, on va voir une "bonne" manière (élégante, plutôt efficace) de mettre ces idées en oeuvre. On va colorier les sommets, en faisant en sorte que les couleurs soient préservées par isomorphisme. Par exemple, l'idée qu'on a exprimée dans le paragraphe précédent consiste à dire qu'on va colorier les sommets par les tailles de leurs voisinages.

## 2.1 Coloriages

On considère donc un ensemble de couleurs $\Omega$, et pour une couleur $\ell \in \Omega$, on note $V[\ell]$ l'ensemble des sommets dans $V$ qui sont de cette couleur. Une procédure de coloriage $f : V \to \Omega$ est correcte si elle donne la même couleur à $x$ dans $G$ et à $\rho(x)$ dans $\rho(G)$, pour tout isomorphisme $\rho$.

**Certificats de non-isomorphisme.** Si le schéma d'étiquetage colorie $G_1$ et $G_2$ de manière incompatible (les tailles des ensembles de la même couleur ne correspondent pas, le nombre de couleurs utilisé est différent, etc.), alors les deux graphes ne sont prouvablement *pas* isomorphes, et les deux coloriages en sont un certificat qui "prouve" qu'ils ne sont pas isomorphes, et qui "explique pourquoi".

**Coloriages Stables.** Étant donné une fonction de coloriage, on peut en augmenter la puissance quasi-gratuitement (en terme de prise de tête en tout cas). Si chaque sommet $x$ est (correctement) colorié $f(x)$, alors le coloriage :

$$\begin{aligned} f_2 : V &\to \Omega \times \mathbb{N}^{\Omega} \\ x &\mapsto \Big( f(x), \{\!\{ f(y) \mid y \text{ est adjacent à } x \}\!\} \Big) \end{aligned}$$

est correct par construction. La notation $\{\!\{\dots\}\!\}$ dénote un multi-ensemble, c'est-à-dire un ensemble avec multiplicité (on compte combien de fois chaque élément est présent), donc $\{\!\{x, x\}\!\} \neq \{\!\{x\}\!\}$. Un multi-ensemble d'éléments de $\Omega$ est donc une fonction de $\Omega$ dans $\mathbb{N}$.

Le coloriage donné par $f_2$ est plus fin que celui de $f$, dans le sens que $f_2(x) = f_2(y)$ implique $f(x) = f(y)$, mais que l'implication n'est pas forcément vraie dans l'autre sens. On peut définir de manière analogue $f_3, f_4, \dots$ en appliquant la même procédure de façon itérative.

On peut sans perte de généralité supposer que l'ensemble $\Omega$ des couleurs est $\{1, \dots, n\}$ (car il ne peut y avoir plus de $n$ couleurs différentes sur les $n$ sommets du graphe). De la même manière, on peut prendre le graphe colorié par $f_2, f_3, \dots$, trier les couleurs apposées (qu'on peut supposer ordonnées d'une manière ou d'une autre) et les numéroter, et de la sorte les étiquettes sont toujours des entiers dans $\{1, \dots, n\}$, même après un nombre arbitraire d'étapes de coloriage. On aura intérêt à garder la table de correspondance entre les numéros et les "vraies" couleurs. On va supposer dans la suite qu'on utilise cette "renumératotation" à chaque étape, et que les couleurs sont donc toujours de petits entiers.

Nous affirmons qu'au bout d'un certain nombre d'itérations (au pire $n$), alors le coloriage se stabilise, dans le sens que $f_i$ et $f_{i+1}$ colorient de la même façon. Il est alors inutile de continuer.

**Démarrage du coloriage.** Il est conseillé de poser que $f_0$ donne la même couleur à tout le monde. Alors, $f_1$ va automatiquement colorier les sommets en fonction de leur degré, $f_2$ en fonction de leurs voisinages de taille 2, etc.

### 2.2 Backtracking coloré

Étant donné deux graphes, On va donc tenter de construire l'isomorphisme $\rho$ en entrelaçant des étapes de coloriage et des étapes de backtracking, comme suggéré par le pseudo-code suivant. En gros, on utilise le coloriage pour éviter de faire des choix "à l'aveugle" pendant le backtracking. En fait, il est connu que l'utilisation du coloriage permet d'éviter tout backtracking sur la majorité des graphes.

```
1:  function WEISFEILER-LEHMAN_1(G_1, G_2, ρ)
2:      Si ρ est complètement spécifié, return ρ
3:      Colorier les deux graphes de façon stable
4:      Si les coloriages sont incompatibles, return false
5:      if il existe u ∈ V_1 et v ∈ V_2 dont la couleur est unique then
6:          let ρ' = ρ ∪ (u ↦ v, pour chaque paire dont la couleur est unique) in
7:          Tester contradictions. Le cas échéant, return false
8:          return WEISFEILER-LEHMAN_1(G_1, G_2, ρ')
9:      else
10:         choisir un sommet u dans G_1
11:         for all v ∈ V_2, de la même couleur que u do
12:             let ρ' ← ρ ∪ (u ↦ v) in
13:             Tester contradictions. Le cas échéant, return false
14:             Calculer WEISFEILER-LEHMAN_1(G_1, G_2, ρ'). Si le résultat n'est pas false, le retourner
15:         end for
16:         return false
17:     end if
18: end function
```

Je ne suis pas absolument certain que les tests ligne 7 et 13 soient indispensables, mais je n'en suis pas sûr. On peut envisager plusieurs manières de choisir le sommet $u$ ligne 10. Le plus naturel semble être de choisir un sommet qui porte la couleur la moins fréquente (pour limiter le nombre de sous-cas à tester). Une solution alternative pourrait être de choisir le sommet $x$ tel que :

$$\max_{\ell \in \Omega} \min \left\{ |\Gamma(x) \cap V[\ell]|, \left|\overline{\Gamma(x)} \cap V[\ell]\right| \right\}$$

soit maximal. L'intuition est qu'on essaye alors de casser le plus gros "morceau".

## 3 Travail demandé

Vous devez écrire un programme qui teste si deux graphes sont isomorphes. Vu que l'algorithme peut ne pas avoir un comportement polynomial, on attachera une certaine importance à la qualité et à l'efficacité de l'implémentation (l'idée, c'est qu'il faut que ça aille vite). On la testera peut-être sur des graphes "durs" (par exemple, des graphes dont tous les noeuds ont le même degré).

Votre programme doit lire deux graphes (dont les sommets sont numérotés de 0 à $n-1$) donnés dans deux fichier, au format suivant : la première ligne contient l'entier $n$. Ensuite, la $i$-ème ligne décrit les voisins du sommet numéro $(i-2)$. Chaque ligne comporte d'abord le nombre de voisins, puis un espace, puis les numéros des voisins, séparés par des espaces.

Votre programme devra répondre "oui" ou "non", et si oui, afficher la correspondance entre les numéros des sommets des deux graphes. Vous avez tout intérêt à tester que l'isomorphisme retourné est bel et bien un isomorphisme.

## Références


[1] László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. ACM.

[2] Scott Fortin. The graph isomorphism problem. Technical report, University of Alberta, 1996. disponible en ligne.