Largeur de clique (clique-width)

Xavier Pujol

Contents

1 Définition

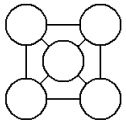
La largeur de clique, ou clique-width en anglais, est un paramètre qu'on définit pour tout graphe G non orienté et non vide. Il est égal au nombre minimum d'étiquettes nécessaires pour construire le graphe avec la méthode de construction suivante.

1.1 Méthode de construction

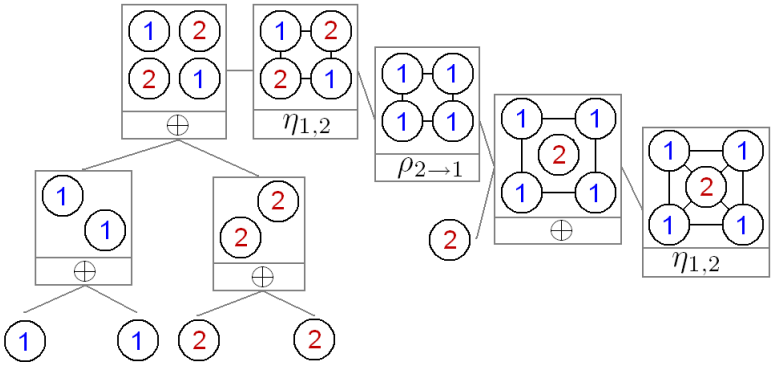
Dans ce paragraphe, on considère des graphes "étiquetés", c'est à dire que chacun de leurs sommets porte une étiquette $e \in \mathbb{N}$. La méthode permet de construire des graphes, à partir de points isolés, en appliquant les opérations suivantes :

- Union disjointe $\oplus$ (opération binaire) : deux graphes sont placés côte à côte sans ajouter d'arête. Les étiquettes des sommets ne sont pas modifiées.
- Renommage $\rho_{i\to j}$ (opération unaire, i et j deux étiquettes distinctes) : dans un graphe, remplace toutes les étiquettes i par j.
- Insertion d'arêtes $\eta_{i,j}$ (opération unaire, i et j deux étiquettes distinctes) : relie tous les sommets d'étiquette i à tous les sommets d'étiquette j. Le graphe n'étant pas orienté, l'ordre de i et j n'a pas d'importance.

Voici un exemple avec le graphe G suivant.



La construction de ce graphe peut être représentée sous forme d'arbre.



1.2 Définition

La largeur de clique d'un graphe G, notée $\text{cwd}(G)$ est le nombre minimal d'étiquettes nécessaire pour construire ce graphe. Ce nombre inclut les étiquettes qui apparaîssent lors des opérations $\rho_{i\to j}$ même si l'étiquette j n'est présente sur aucun sommet de base. Les étiquettes qui apparaîssent dans le résultat ne nous intéressent pas ici. On peut d'ailleurs changer toutes les étiquettes en 1 sans augmenter la largeur de clique. Pour tout graphe $G = (V, E)$, $\text{cwd}(G)$ est bien définie et $\text{cwd}(G) \leq |V|$. Voici la méthode de construction générale :

- On note $V = \{v_1, \ldots, v_n\}$, et chaque sommet v_i prend l'étiquette i.
- Pour chaque arête $E = (v_i, v_j)$, insérer une opération $\eta_{i,j}$.

Revenons à l'exemple précédent. L'arbre de construction montre que

$$\text{cwd}(G) \leq 2$$

Comme l'opération $\eta_{i,j}$ de création d'arêtes ne s'applique que pour $i \neq j$, aucune arête ne peut être créée en utilisant une seule étiquette. Ainsi, un graphe a une largeur de clique 1 si et seulement si il n'a aucune arête. Par conséquent, sur l'exemple

$$\text{cwd}(G) = 2$$

1.3 Rapport avec les cographes

On a $\text{cwd}(G) \leq 2$ si et seulement si G est un cographe (l'exemple précédent en est un).

($\Leftarrow$) : à partir de l'arbre de construction du cographe G, on construit un nouvel arbre en remplaçant l'opération parallèle $P(G_1, G_2)$ par $G_1 \oplus G_2$ et l'opération série $S(G_1, G_2)$ par $\eta_{1,2}(\rho_{2\to 1}(G_1) \oplus \rho_{1\to 2}(G_2))$. Toutes les feuilles sont étiquetées par 1. Il y a bien deux étiquettes utilisées seulement.

($\Rightarrow$) : par induction, pour A utilisant les étiquettes 1 et 2, on construit trois cographes : G pour le graphe en entier, G_1 et G_2 pour les sous-graphes des sommets étiquetés par

1 et 2. Pour $\oplus$, on met en parallèle les deux instances de G, G_1 et G_2. Pour $\eta_{1,2}$, G est la mise en série de G_1et G_2, qui sont inchangés. Pour $\rho_{1\to 2}$, on prend G inchangé, $G_2 = G$ et G_1 vide (l'inverse pour $\rho_{2\to 1}$).

On peut donc décider si $\text{cwd}(G) \leq 2$ en temps polynomial. Plus généralement, à k fixé, on peut déterminer si $\text{cwd}(G) \leq k$ en temps polynomial en la taille du graphe, mais ce temps est exponentiel en k.

1.4 Rapport avec la largeur arborescente (tree-width)

Etant donné que les cliques K_n sont des cographes, on a $\text{cwd}(K_n) = 2$ pour tout $n \geq 2$. Or la largeur arborescente d'une clique K_n est $\text{twd}(K_n) = n$. Il existe donc des graphes de largeur de clique bornée et de largeur arborescente arbitrairement grande.

L'inverse est faux. Il a été démontré dans l'article [4] que pour un graphe G non orienté on a :

$$\text{cwd}(G) \leq 2^{2\text{twd}(G)+1} + 1$$

2 Algorithmes efficaces pour les graphes de largeur de clique fixée

A largeur de clique k fixée, une certaine catégorie de problèmes sur les graphes peuvent être résolus en temps linéaire. Il s'agit des problèmes de décision exprimables dans MSOL, logique monadique du second ordre, et de certains problèmes d'optimisation liés. C'est le sujet de l'article [1].

2.1 MSOL (Monadic Second Order Logic)

La logique du second ordre permet, en plus des possibilités de la logique du premier ordre, de quantifier sur des relations d'arité quelconque ou des fonctions. La logique monadique se restreint aux quantifications sur des ensembles d'éléments du domaine (ou des relations d'ordre 1, ce qui est équivalent, puisqu'une relation d'arité k peut être considérée comme un ensemble de $k-$uplets). Lorsqu'on travaille sur un graphe, le domaine des variables du premier ordre est l'ensemble des sommets et on définit la relation d'adjacente E entre deux sommets. Pour des graphes étiquetés par des nombres dans $[1,k]$, on définit en plus les ensembles $U_1,\ldots,U_k$, où U_i est l'ensemble des noeuds d'étiquette i.

Voici par exemple une formule du second ordre sur un graphe exprimant sa connexité :

$$\forall X,Y,[(\exists a \in X) \wedge (\exists a \in Y) \wedge (\forall a, a \in X \vee a \in Y)] \Rightarrow (\exists x \in X, y \in Y, E(x,y))$$

Une formule peut contenir des variables libres. En associant un entier à chaque élément du domaine (ici les sommets d'un graphe), on poser un problème d'optimisation : quel est le meilleur assignement satisfaisant la formule. Lorsque la fonction d'optimisation est linéaire, à coefficients entiers, le problème d'optimisation appartient à la classe LinEMSOL et on sait résoudre le problème en temps linéaire.

2.2 Théorème de Feferman-Vaught pour MSOL

Ce théorème est utilisé dans la partie principale de l'algorithme. On considère un graphe $G = G_1 \oplus G_2$ (union disjointe) et une formule du second ordre $\theta(X_1, \ldots, X_n)$ sur ce graphe. On note $\text{sat}(G, \theta)$ l'ensemble des solutions de θ, une solution étant un n-uplet $(x_1, \ldots, x_n)$ d'ensembles de sommets. Le théorème de Feferman-Vaught indique qu'il existe des formules $\varphi_1, \ldots, \varphi_m$ sur G_1 et $\psi_1, \ldots, \psi_m$ sur G_2(toutes d'arité n) telles que :

$$\text{sat}(G, \theta) = \bigcup_{i=1}^{m} \text{sat}(G_1, \varphi_i) \boxtimes \text{sat}(G_2, \psi_i)$$

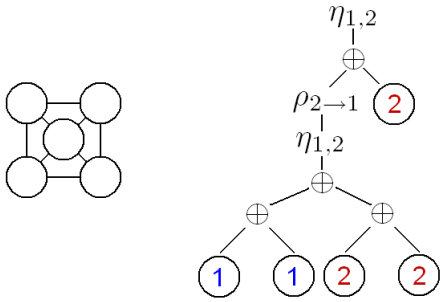
L'opérateur $\boxtimes$ est défini ainsi :

$$A \boxtimes B = \{(y_1 \cup z_1, \ldots, y_n \cup z_n); y \in A, z \in B\}$$

Cela signifie que les solutions $(x_1, \ldots, x_n)$ de θ sont construites en prenant deux solutions de φ_i et ψ_i pour un i quelconque, notées $y = (y_1, \ldots, y_n)$ et $z = (z_1, \ldots, z_n)$. La $k-$ième variable de la solution de θ, notée x_k, est alors l'union des sommets de y_k et de z_k.

2.3 Algorithme

Ce paragraphe présente l'algorithme linéaire sur un exemple. On considère le problème d'optimisation CLIQUE-MAX ("trouver la plus grande clique dans un graphe G"), qu'on cherche à résoudre sur l'exemple de la présentation reproduit ici avec son arbre de construction.



L'algorithme descend récursivement dans l'arbre de construction. A chaque niveau, on cherche à optimiser une ou plusieurs formules MSOL. Initialement, on part de l'arbre complet et il n'y a qu'une seule formule, celle du problème. Voici la formule (en l'occurence, elle possède une seule variable libre) :

$$\theta(X) : \forall u, v, (v \in X \wedge u \in X \wedge u \neq v) \Rightarrow E(u, v)$$

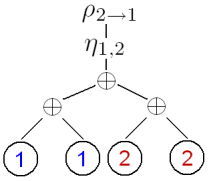
L'opérateur à la racine de l'arbre est unaire ($\eta_{1,2}$). On applique maintenant l'algorithme au graphe correspondant au sous-arbre. Sur ce graphe, on considère la formule

$$\theta'(X) : \forall u, v, (v \in X \wedge u \in X \wedge u \neq v) \Rightarrow (E(u, v) \vee (U_1(u) \wedge U_2(v)) \vee (U_2(u) \wedge U_1(v)))$$

Les prédicats U_i indiquent si un sommet est étiqueté par i. $\theta'(X)$ est vérifiée sur le nouveau graphe si et seulement si $\theta(X)$ est vérifiée sur le graphe de départ. Dans le cas général, l'opération pour trouver θ' consiste à remplacer toutes les occurences de $E(u,v)$ comme dans l'exemple en prenant U_i et U_j au lieu de U_1et U_2. S'il y avait plusieurs formules, on répéterait la transformation sur chaque formule.

L'opérateur à la racine du sous-arbre est maintenant l'opérateur binaire $\oplus$. C'est le cas qui nécessite l'utilisation du théorème de Feferman-Vaught. Je n'ai pas étudié la preuve du théorème, je donne donc les formules qui marchent de façon *ad hoc*. Il y en a une seule par sous-arbre

- Sur le gros sous-arbre :



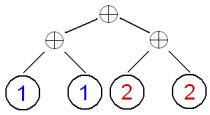
  $\varphi = \theta = \forall u, v, (v \in X \wedge u \in X \wedge u \neq v) \Rightarrow E(u,v)$

- Sur la feuille isolée : $\psi = \theta =$ VRAI

S'il y avait plusieurs formules à la racine, il faudrait appliquer le théorème à chacune et ajouter toutes les formules obtenues aux sous-arbres.

Sur le sous-arbre le plus gros, l'opérateur à la racine est $\rho_{2\to 1}$. On se contente de reprendre la formule pour le sous-arbre en remplaçant les U_1 par $U_1 \vee U_2$, et les U_2 par FAUX. Cela ne change rien ici. On continue de la même façon pour $\eta_{1,2}$.

Pour l'arbre



avec une formule égale à θ', on prend comme dans le cas précédent les formules $\varphi(X) = \theta(X)$ et $\psi(X) = \theta(X)$ pour les sous-arbres.

Pour l'arbre

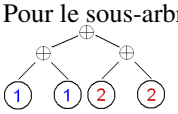
on crée deux formules pour chaque feuille :

- $\varphi_1(X) = \theta(X) =$ VRAI
- $\varphi_2(X) = \neg\exists x \in X =$ est_vide(X)
- $\psi_1 = \varphi_2$
- $\psi_2 = \varphi_1$

On remonte alors pour calculer l'optimum de chaque formule.

- Pour les feuilles, on teste tous les cas pour chaque formule. Ici, l'optimum est $f(x) = |X| = 1$ pour φ_1 et ψ_2, et 0 pour φ_2 et ψ_1.
- Pour une union disjointe $G = G_1 \oplus G_2$, une solution de la formule au sommet de l'arbre est appartient à l'un des ensembles $\text{sat}(G_1, \varphi_i) \boxtimes \text{sat}(G_2, \psi_i)$, $i \in [1, m]$. L'optimum est donc le max des optima sur i. D'après la définition du produit $\boxtimes$ et parce que la fonction à optimiser est linéaire, l'optimum pour un i fixé est la somme des optima de φ_i sur G_1 et de ψ_i sur G_2.

  Pour un sous-arbre 

, il y a par exemple deux i possibles, $i = 1$ ou $i = 2$. Pour $i = 1$, $\text{opt}(\varphi_1) + \text{opt}(\psi_1) = 1 + 0 = 1$. Pour $i = 2$, $\text{opt}(\varphi_2) + \text{opt}(\psi_2) = 0 + 1 = 1$. L'optimum est donc de taille $\max(1, 1) = 1$.
  Pour le sous-arbre
  

  on trouve opt $= 1 + 1 = 2$. Il est intéressant de noter qu'à ce niveau, l'optimum ne correspond pas à l'optimum de la formule initiale (la plus grande clique de ce graphe est de taille 1, il ne contient aucune arête).
- Pour les opérations $\eta_{i,j}$ et $\rho_{i \to j}$, on reprend tel quel l'optimum du niveau inférieur.
- Dans notre exemple, on trouve à la racine un optimum égal à 3, ce qui est bien la taille de la plus grande clique du graphe.

3 NP-complétude et inapproximabilité

Deux articles [2][3] apportent les deux résultats suivants :

- Déterminer à partir de G et k si $\text{cwd}(G) \leq k$ est NP-complet. Par contre, on a déjà vu que si k est fixé, le problème est polynomial.
- Si $P \neq NP$, alors pour tout pour tout $\varepsilon \in]0, 1[$, il n'y a pas d'algorithme polynomial calculant $\text{cwd}(G)$ avec la garantie d'une erreur inférieure à n^{ε}, n nombre de sommets du graphe (on rappelle que $\text{cwd}(G) \leq n$).

Sans entrer dans les détails, voici l'organisation de la preuve.

- Le point de départ est l'existence des mêmes résultats pour un autre paramètre, path-width (noté pwd), semblable à la tree-width, mais avec la contrainte que l'arbre de la décomposition soit une chaîne.
- Dans le premier article, un lemme prouve que le résultat est vrai pour tout paramètre α tel que $|\alpha - \text{pwd}|$ soit borné par une constante.

- Ce premier article introduit un paramètre intermédiaire, "sequential clique-width", notée cwd_1, défini comme la largeur de clique avec une restriction sur l'opération $\oplus$: l'un des deux graphes de l'opérateur doit être un point. Le but de cet article est d'établir le résultat pour cwd_1. Ceci est fait à l'aide du lemme précédent, mais pas directement avec $\alpha = \text{cwd}_1$.
  A partir d'un graphe G, on construit un graphe $G' = f(G)$ et il est établit que

  $$\text{pwd}(G) \leq \text{cwd}_1(G') \leq \text{pwd}(G) + 4$$

  Le lemme est appliqué avec $\alpha(G) = \text{cwd}_1(f(G))$. Cela prouve la difficulté et l'inapproximabilité du paramètre cwd_1 pour les graphe de la forme $f(G)$, donc *a fortiori* pour les graphes en général.

- Le second article borne l'écart entre cwd et cwd_1, là encore sur $G' = f(G)$.

  $$\text{cwd}(G') \leq \text{cwd}_1(G') \leq \text{cwd}(G') + 18$$

  En combinant les deux résultats, on obtient une borne sur $|\text{cwd}(G') - \text{pwd}(G)|$, d'où la NP-complétude et l'inapproximabilité.

Remarque : le problème $\text{cwd}(G) = k$ est aussi NP-difficile. En effet, pour déterminer si $\text{cwd}(G) \leq k$, il suffit de tester si $\text{cwd}(G) = i$ pour tout $i \leq k$, ce qui une réduction polynomiale car on sait que $\text{cwd}(G) \leq \#\text{sommets}(G)$.

References

[1] B. Courcelle, J. Makowski and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width, Theory Comput. Syst. 33 (2000).

[2] Michael R. Fellows, Frances A. Rosamond, Udi Rotics and Stefan Szeider. Proving NP-hardness for clique-width I: non-approximability of sequential clique-width. Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 80, 2005.

[3] Michael R. Fellows, Frances A. Rosamond, Udi Rotics and Stefan Szeider. Proving NP-hardness for clique-width II: non-approximability of clique-width. Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 81, 2005.

[4] B. Courcelle and S. Olariu. *Upper bounds to the clique width of graphs*. Discrete Applied Mathematics, 101(1-3):77-114, 2000.