ALGO2 – Largeur arborescente

François Schwarzentruber

April 26, 2022

1 Fixed-parameter tractable

Définition 1 (paramétrisation) Une *paramétrisation* est une fonction[1] κ qui à toute instance associe un entier.

Définition 2 (problème paramétrée) Un *problème paramétrée* a la forme suivante :

- entrée : une instance x ;
- paramétrisation : κ ;
- sortie : oui si x est une instance positive, non sinon.

Exemple 1 pSAT

- entrée : une formule propositionnelle φ
- paramétrisation : nombre de variables dans φ
- sortie : oui si φ est satisfaisable, non sinon.

Voici un algorithme : parcourir toutes les valuations et tester si φ est vraie. Sa complexité est $2^k poly(n)$ où n est la taille de φ et k est le nombre de variables dans φ.

Définition 3 (fpt-algorithme) Un algorithme est un fpt-algorithme par rapport à κ s'il existe une fonction calculable f et un polynôme *poly* tel que pour tout entrée x, $A(x)$ s'exécute en temps au plus $f(\kappa(x)) \times poly(|x|)$.

Définition 4 (fixed-parameter tractable) Un problème de paramétrisation κ s'il est décidé par un fpt-algorithme par rapport à κ.

2 Motivation

2.1 Sur les arbres = facile

Beaucoup de problèmes sont faciles sur des arbres avec programmation dynamique

Définition 5 **Ensemble indépendant maximum**

entrée : graphe $G = (V, E)$ non orienté
sortie : un ensemble $S \subseteq V$ de sommets deux-à-deux non adjacents.

Proposition 6 Ensemble indépendant maximum restreint aux arbres est dans P.

Démonstration. Programmation dynamique
$I(u)$:= taille maximale d'un ensemble indépendant dans le sous-arbre enraciné en u.
Le but est de calculer $I(r)$
Soit un ensemble indépendant contient u ou alors ne contient pas u. Ainsi on a :

$$I(u) = \max(1 + \sum_{w \text{ petit-enfant de } u} I(w), \sum_{w \text{ enfant de } u} I(w))$$

[1] Dans [FG06] p. 4, on la suppose calculable en temps polynomial.

```
fonction calculerI(u)
    si u est une feuille alors
        u.I := 1
    sinon
        pour w enfant de u faire
            calculerI(w)
        u.I = max(1 + ∑_{w petit-enfant de u} w.I, ∑_{w enfant de u} w.I)
```

■

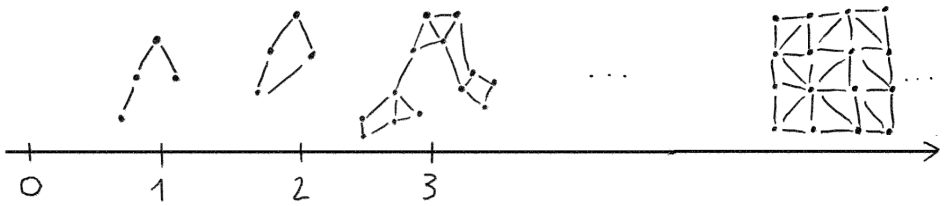
2.2 Paramétrisation qui mesure l'arbritude

En pratique, beaucoup de graphes ressemblent à des arbres.

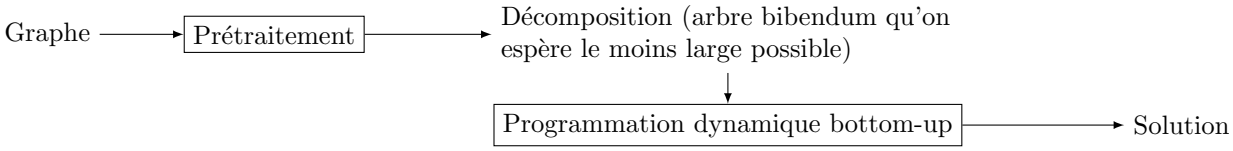
Exemple 2

- Graphes des programmes ont une tree-width ≤ 6 généralement
- Graphe de contraintes (CSP)
- series-parallel graph (ils sont de tree-width 2!)
- Heuristique pour TSP

Quid d'une paramétrisation qui mesure de combien un graphe ressemble à un arbre ?



2.3 Principe pour un algorithme efficace



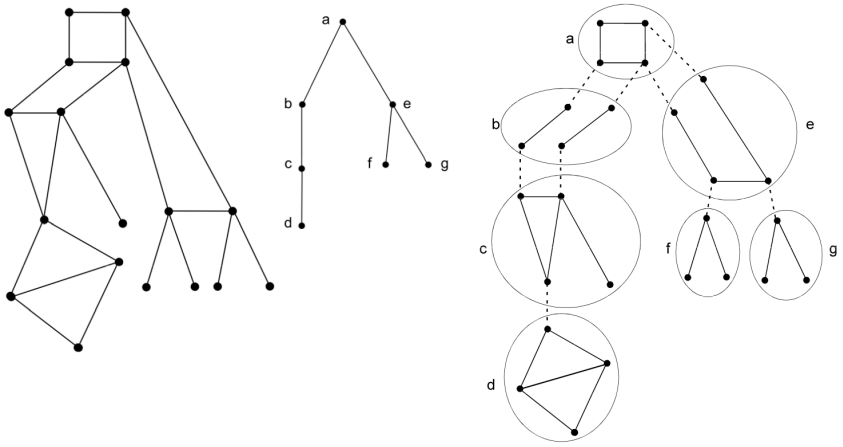
3 Définitions

Mettre un graphe dans un arbre bibendum.

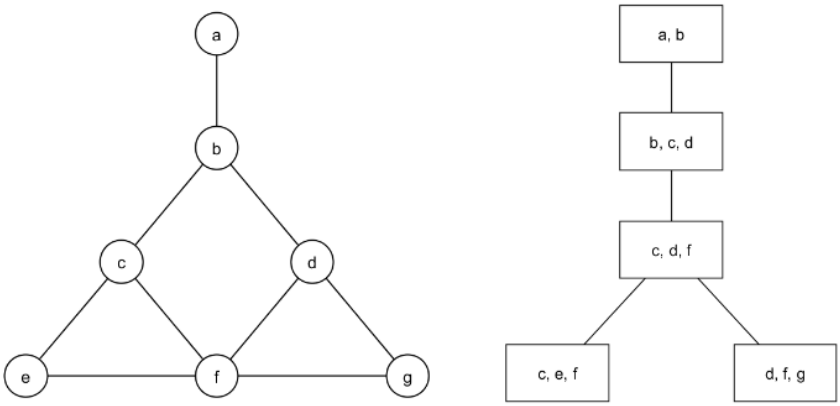
Définition 7 (décomposition d'un graphe) Soit $G = (V, E)$ un graphe non orienté. Une *décomposition* A de G est un arbre où :

1. les nœuds de A sont (étiquetés par) des *sacs*, i.e. des sous-ensembles de sommets de G ;
2. pour tout sommet x dans G, l'ensemble des nœuds dont les sacs contiennent x forment un sous-arbre connexe de A ;
3. toute arête $\{x, y\}$ de G est incluse dans au moins un sac.

Exemple 8



Exemple 3



Notation 9 (sac en un nœud) Étant donné un nœud t d'une décomposition, on note $\partial_t \subseteq V$ le sac du nœud t.

Notation 10 (ensemble de sommets d'un sous-arbre) Étant donné un nœud t d'une décomposition, l'ensemble de sommets du sous-arbre enraciné en t est $G_t = \cup_{u \text{ descendant de } t} \partial_u$.

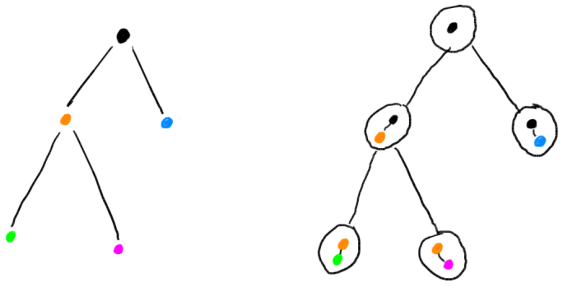
Définition 11 (largeur) La largeur d'une décomposition est le cardinal du plus gros sac moins 1.

Définition 12 (largeur arborescente − tree-width) La tree-width de G, notée $tw(G)$, est la largeur d'une décomposition la moins large.

4 Classes de graphes

Proposition 13 Soit G un graphe connexe. G est un arbre ssi $tw(G) = 1$.

DÉMONSTRATION. Si G est un arbre, alors on peut considérer la décomposition suivante qui est de largeur 1.



Réciproquement, considérons une décomposition de G de largeur 1. TODO: en exo

■

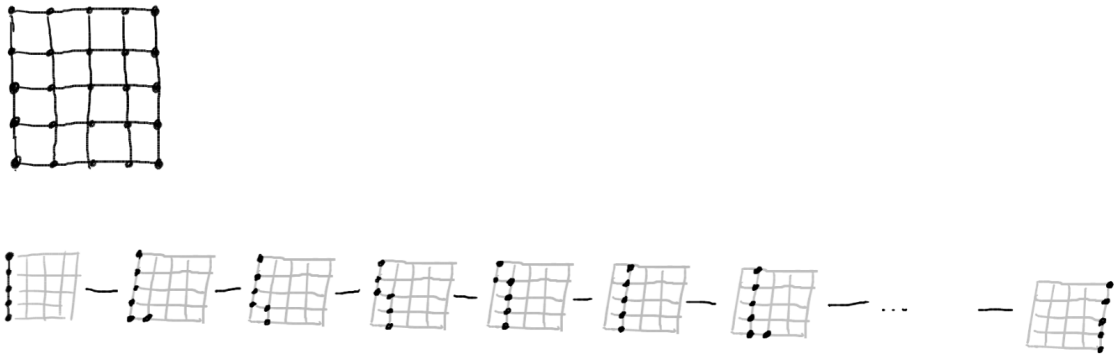
Proposition 14 La largeur arborescente d'un cycle est 2.

Démonstration.

■

Proposition 15 La largeur arborescente d'une grille $n \times n$ est n.

Démonstration.



■

Proposition 16 La largeur arborescente du graphe complet K_n est $n-1$.

Proposition 17 Si $tw(G) = k$, alors tout sous-graphe H de G vérifie $tw(H) \leq k$.

5 Taille d'une décomposition

Définition 18 Une décomposition d'un graphe G est petite si pour tout nœud x, y, si $x \neq y$ alors $\partial_x \not\subseteq \partial_y$.

Proposition 19 Une petite décomposition d'un graphe $G = (V, E)$ possède au plus $|V|$ nœuds.

Démonstration. Par induction sur $|V|$ (sachant qu'une feuille contient au moins un sommet qui n'est dans aucun autre sac). ■

Proposition 20 Tout graphe G admet une petite décomposition de largeur $tw(G)$. De plus, toute décomposition peut être transformée en temps linéaire en petite décomposition de même largeur.

Démonstration. Contracter les arcs (x, y) de la arbre-décomposition pour lesquelles $\partial_x \subseteq \partial_y$ ou $\partial_y \subseteq \partial_x$. Garder le sac le plus grand de ∂_x et ∂_y. ■

Proposition 21 Dans tout graphe G, il existe un sommet de degré $\leq tw(G)$.

Démonstration. Soit $G = (V, E)$ un graphe et considérons une décomposition petite de largeur $tw(G)$. Si G a moins de $tw(G)+1$ sommets, alors tous les sommets de degré $\leq tw(G)$ et donc c'est gagné. Sinon, G a plus de $tw(G)+2$ sommets. Mais alors la décomposition possède au moins deux noeuds. Soit f une feuille et p son parent. Il y a un sommet $s \in \partial_f \setminus \partial_p$. Alors ∂_f est le sac qui contient s. Ainsi, toutes les arêtes partant de s sont présentes dans ce sac. Or $|\partial_f \setminus \{s\} \leq tw(G)$. Et donc s est de degré au plus $tw(G)$. ■

6 Algorithmes pour décomposition

Définition 22 (problème de décision du calcul de la largeur arborescente) **tree-width**
entrée : un graphe G, un entier k
sortie : oui, si la tree-width de G est k, non sinon.

Théorème 23 (admis) Le calcul de la tree-width est NP-complet.

Théorème 24 (de Bodlaender, admis) Il existe un algorithme qui prend en entrée un graphe G qui retourne une décomposition optimale de largeur $k = tw(G)$ en temps $O(|G| \times 2^{poly(k)})$.

Théorème 25 (démontré en annexe) Il existe un algorithme prenant en entrée un graphe G qui retourne une décomposition de largeur $\leq 4tw(G) + 1$ en temps $O(|G| \times 2^{O(tw(G))})$.

7 Programmation dynamique

Proposition 26 Le problème Ensemble indépendant admet un algorithme de complexité $O(|G| \times f(tw(G)))$.

Démonstration. D'abord on calcule une petite décomposition arborescente de G. Ensuite, on applique la programmation dynamique bottom-up. Étant donné un nœud x dans l'arbre, et un sous-ensemble $A \subseteq \partial_x$, A indépendant sur ∂_x, on calcule $MIS(x, A)$ la taille d'un plus grand ensemble indépendant I dans G_x avec $I \cap \partial_x = A$.
On a

$$MIS(x, A) = |A| + \sum_{y \text{ enfant de } x} \max_{A' \subseteq \partial_y | A' \text{ indépendant sur } \partial_y \text{ et } A' \cap \partial_x = A \cap \partial_y} MIS(y, A') - |A \cap \partial_y|.$$

On calcule récursivement $MIS(x, \bullet)$.

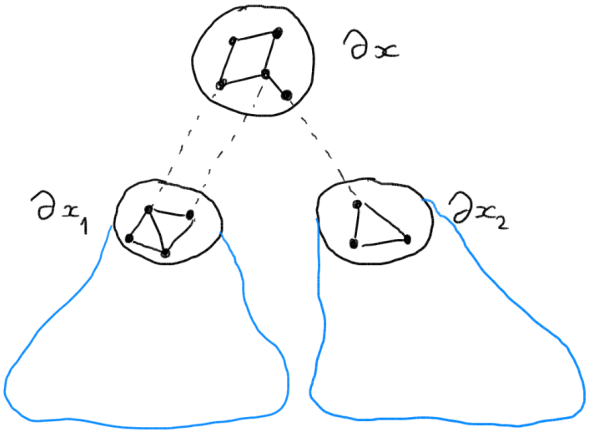
```
fonction calculerMIS(x)
    pour y enfant de u faire
        calculerMIS(y)
    pour A ⊆ ∂_x faire
        x.MIS[A] := ...
```

La complexité est de $T(x) = 4^k k^2 |\text{enfants de } x| + \sum_{y \text{ enfant de } x} T(y)$. Le 4^k vient de $2^k 2^k$: le premier 2^k de la boucle pour tout A, et le deuxième 2^k de la boucle pour A'. Le k^2 vient du fait qu'il faut parcourir les A indépendant sur ∂_x. Pour A', pas besoin de vérifier car c'est un dictionnaire, on parcourt les clefs. Ainsi on a $T(racine) = 4^k k^2 |V|$.
■

Proposition 27 Le problème de 3-coloration admet un algorithme de complexité $O(|G| \times f(tw(G)))$.

Démonstration. D'abord on calcule une petite décomposition arborescente de G. Ensuite, on applique la programmation dynamique bottom-up. Étant donné un nœud x dans l'arbre, on calcule $extCol(x)$ l'ensemble des colorations licites des sommets dans ∂_x qui peuvent être étendue en une coloration du sous-graphe G_x.

1. La relation de récurrence est, en notant $x_1, \ldots, x_k$ les fils de x :
   $extCol(x) = \{f \text{ coloration de } \partial_x \mid f \text{ coincide avec une coloration de } extCol(x_i) \text{ pour tout } i\}$



2. Le graphe G est coloriable ssi $extCol(racine)$ est non vide.

Exercice 28 Evaluer la complexité de l'algorithme de 3-coloration.

Exercice 29 Écrire le pseudo-code de l'algorithme qui prend en entrée une décomposition d'un graphe G et décide si le graphe est 3-coloriable.

Exercice 30 Implémenter l'algorithme de 3-coloration ci-dessus.

■

8 Théorème de Courcelle

Caractérisation logique d'une classe où l'approche programmation dynamique sur une décomposition fonctionne

Définition 31 (syntaxe de MSOL) La syntaxe de la *logique monadique du second ordre* sur les graphes est définie par la grammaire suivante :

$$\varphi \ ::= \ u{=}v \ \mid \ u{\in}X \ \mid \ uRv \ \mid \ \neg\varphi \ \mid \ \varphi \vee \varphi \ \mid \ \forall x\varphi \ \mid \ \forall X\varphi$$

Exemple 4 (ensemble indépendant en MSOL) $ind(X) := \forall u, \forall v, (u \in X \wedge v \in X) \rightarrow \neg uRv$

Exemple 5 (3-coloration en MSOL) $\exists A \exists B \exists C \forall u, u$ dans exactement un A, B, ou $C \wedge ind(A) \wedge ind(B) \wedge ind(C)$.

Exemple 6 (cycle hamiltonien en MSOL) $\exists C \subseteq Espanning(C) \wedge \forall u, deg2(u, C)$.

Définition 32 **Problème de Courcelle**

entrée : une formule φ de MSOL, une décomposition de G de largeur k
sortie : oui, si $G \models \varphi$, non sinon.

Définition 33 **Problème d'optimisation de Courcelle**

entrée : une formule $\varphi(X)$ de MSOL, une décomposition de G de largeur k
sortie : A de cardinal max/min tel que $G, [X := A] \models \varphi$, non sinon.

Théorème 34 (de Courcelle, admis) Le problème de Courcelle admet un algorithme en temps $O(|G| \times f(|\varphi|, k))$. Le problème d'optimisation de Courcelle admet un algorithme en temps $O(|G| \times f(|\varphi|, k))$.

Démonstration. (idée) Il y a un nombre fini de sous-graphes de taille k. On fait programmation dynamique bottom-up sur l'arbre. ■

Notes bibliographiques

Ce cours est adapté du chapitre 11 de [FG06]. Le théorème de Courcelle reste vraie pour MSOL sur les *hypergraphes*, où les variables du premier (deuxième) ordre peuvent dénoter aussi des (ensembles d') arêtes. L'application de programmation dynamique pour ensemble indépendant dans les arbres est présenté dans [DPV08].

References

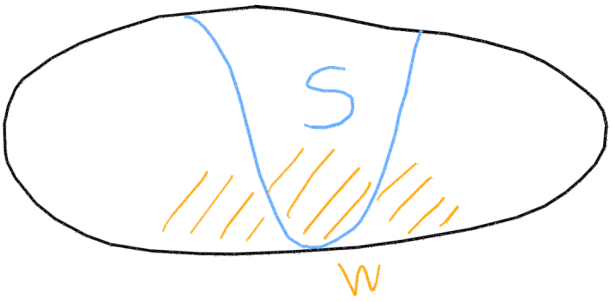
[DPV08] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. *Algorithms*. McGraw-Hill, 2008.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

A Algorithme d'approximation pour décomposition

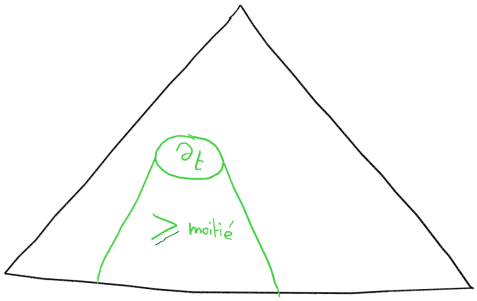
A.1 Séparation balancée

Définition 35 (W-séparateur balancé) Soit $G = (V, E)$ un graphe et $W \subseteq V$. Un W-séparateur balancé est un sous-ensemble $S \subseteq V$ tel que toute composante connexe de $G \setminus S$ contienne moins de la moitié des éléments de W.

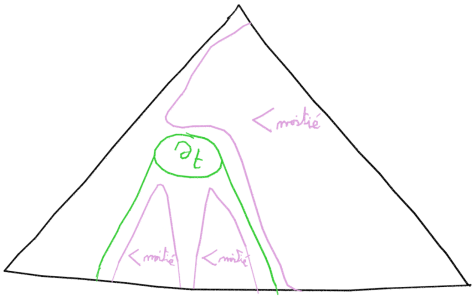


Proposition 36 Soit $G = (V, E)$ un graphe de largeur arborescente au plus k et $W \subseteq V$. Alors il existe un W-séparateur de G de cardinal au plus $k + 1$.

Démonstration. Considérons une décomposition T. On considère un nœud t tel que $B(T_t)$, i.e. l'union des sacs des nœuds du sous-arbre issu en t, contienne plus de la moitié des éléments de W. De plus, on suppose T_t de hauteur minimale (i.e. au plus profond dans l'arbre T).



On pose $S = \partial_t$ (i.e. le séparateur est le sac du nœud t).
Maintenant on pose :
$C_0 = B(T \setminus T_t) \setminus S$ (les sommets sont dans des sacs pas sous t, et pas dans S) ;
$C_i = B(T_{u_i}) \setminus S$ où u_i est le i-ème fils de t (les sommets qui sont sous u_i mais pas dans S).



Fait 37 C_0 contient strictement moins de la moitié de W.

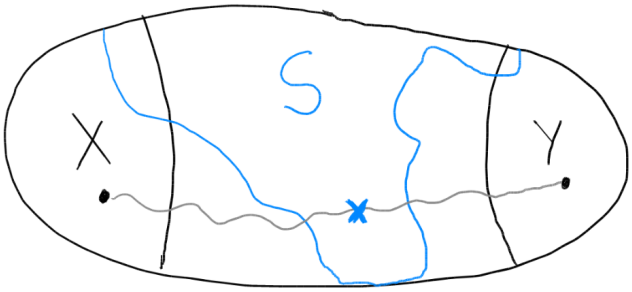
Démonstration. On a $|B(T_t) \cap W| \geq |W|/2$. On a $B(T \setminus T_t) \cap B(T_t) \subseteq \partial_t$. Ainsi C_0 et $B(T_t)$ sont disjoints. ■

Fait 38 C_i contient strictement moins de la moitié de W.

Démonstration. Car T_t est de hauteur minimale. ■
Chaque composante de $G \setminus S$ est dans un des C_i (exercice).
Ainsi chaque composante de $G \setminus S$ contient moins de la moitié des éléments de W. ■

A.2 Séparation

Définition 39 (séparateur) Soit $G = (V, E)$ un graphe. Soit $S, X, Y \subseteq V$. On dit que S sépare X de Y si tout chemin d'un sommet de X à un sommet de Y contient un sommet de S.



Exemple 40 TODO: à faire

Remarque 41 Tout sur-ensemble de X sépare X de Y. Tout sur-ensemble de Y sépare X de Y.

Proposition 42 Le problème suivant admet un algorithme en temps $O(poly(k, G))$:

Séparateur petit

entrée : un graphe $G = (V, E)$, deux ensembles $X, Y \subseteq V$, un entier k
sortie : un ensemble $S \subseteq V$ de cardinal $\leq k$ qui sépare X de Y, s'il en existe un, 'non' sinon.

Démonstration. Par réduction aux flots !
Cela provient du théorème Menger :

$$\max_{P \text{ ensemble de chemins disjoints de } X \text{ à } Y} |P| = \min_{S \text{ qui sépare } X \text{ et } Y} |S|.$$

Par chemins disjoints, on entend des chemins tels qu'il n'y ait pas de sommets intermédiaires communs. Pour P un ensemble de chemins disjoints deux à deux, $|P|$ est le nombre de chemins.

$\boxed{\leq}$ Soit S minimal qui sépare X de Y. Soit P. Montrons que $|P| \leq |S|$. Chaque chemin de P vient intersecter un sommet différent de S. Donc $|P| \leq |S|$.

$\boxed{\geq}$ Montrons $|S| \leq |P|$ si P et S sont optimums. Attention, ce n'est pas toujours le cas : on peut mal choisir les chemins (typiquement un exemple avec un chemin qui fait un zizag dans S énorme...). Pour montrer cela, on va montrer qu'il s'agit du théorème max flot/min cut dans un certain réseau de flots. Le réseau de flots est constitué d'une source, d'une copie de X, de deux copies de G, G et G^+, et d'une copie de Y, et d'un puits. On connecte la source avec capacité infinie à tous les sommets de X, et tous les sommets de Y au puits avec capacité infinie. Puis on a des arcs de capacité 1. Les sommets de X sont connectés à leur successeur dans G. Les sommets précédesseur de Y dans G^+ sont connectés à leur successeur dans Y. Un sommet u est relié juste à u^+. On relie u^+ à v si (u, v) est une arête.

Un flot f dans ce réseau correspond à un ensemble de chemins disjoints. A quoi correspond une coupe minimale ?

Si on a une coupe minimale $C = (S, T)$, elle est de capacité finie, et on peut montrer que les seuls arcs qui vont de la s-partie à la t-partie sont des arcs du type (u, u^+). Par l'absurde, supposons que l'on ait $u^+ \to v$ qui soit séparé, i.e. $u^+ \in S$ et $v \in T$. Dans le graphe résiduel du flot correspondant, ça veut dire que u^+ est accessible mais pas v. Or ce n'est pas possible car il y a de l'eau qui va de u à u^+. Mais alors u^+ n'est pas accessible dans le graphe résiduel (car on peut annuler l'eau de u à u^+ ce qui fait un arc u^+u dans l'autre sens).

On pose S l'ensemble des sommets u tel que (u, u^+) soit un arc séparé par la coupe minimale. Supprimer ces sommets sépare X et Y.

Si P et S sont optimum, On a donc $|P| \geq |f| = |C| \geq |S|$.

Voici donc l'algorithme :

1. construire le réseau de flots
2. appeler l'algorithme Ford-Fulkerson
3. soit S, l'ensemble des sommets u avec (u, u^+) qui est séparé par la coupe minimal.

4. si $|S| \leq k$, alors retourner S sinon retourner "non".

■

Corollaire 43 Le problème suivant admet un algorithme en temps $O(poly(k, G))$:

Séparateur petit'

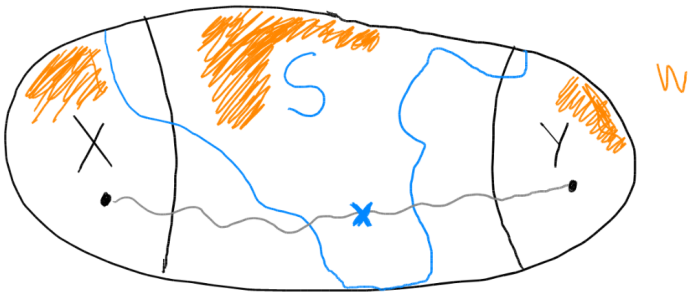
entrée : un graphe $G = (V, E)$, deux ensembles $X, Y, Z \subseteq V$, un entier k
sortie : un ensemble $S \subseteq V \setminus (X \cup Y)$ de cardinal $\leq k$ et $Z \subseteq S$, qui sépare X de Y, s'il en existe un, 'non' sinon.

DÉMONSTRATION. S'il y a un arc de X à Y, on répond non. Soit $S(X)$ les successeurs de X dans $V \setminus X$ (pareil pour $S(Y)$). On considère un nouveau graphe G' qui est le graphe G induit à $V' := V \setminus (X \cup Y)$. On utilise l'algorithme précédent sur l'instance $G', S(X) \cup Z, S(Y) \cup Z, k$. ■

A.3 Séparation faiblement balancé

Définition 44 (W-séparateur faiblement balancé) Soit $G = (V, E)$ un graphe et $W \subseteq V$. Un W-séparateur faiblement balancé est un $S \subseteq V$ tel qu'il existe $X, Y \subseteq W$ avec :

1. $W = X \sqcup (S \cap W) \sqcup Y$;
2. S sépare X de Y
3. $0 < |X| \leq 2|W|/3$, $0 < |Y| \leq 2|W|/3$.



Exemple 45 TODO: un jour

Proposition 46 Il y a un algorithme en temps $O(3^{3k} poly(k, G))$ pour le problème suivant :

Séparateur faiblement balancé petit

entrée : un graphe $G = (V, E)$, un ensemble $W \subseteq V$ avec $|W| = 3k + 1$, un entier k
sortie : un ensemble $S \subseteq V$ W-séparateur faiblement balancé de cardinal $\leq k + 1$, s'il existe un, 'non' sinon.

DÉMONSTRATION. On utilise l'algo qui calcule un séparateur comme sous-routine.

pour $X \subseteq W$ de cardinal au plus $2k$ **faire**
  pour $Y \subseteq W$ de cardinal au plus $2k$, Y disjoint de X **faire**
    $S :=$ algo pour séparateur petit' avec X, Y et $Z := W \setminus (X \cup Y)$ et $k + 1$.
    si S n'est pas "non" **alors renvoyer** S
renvoyer "non"

3^{3k+1} est une borne supérieure du nombre de (X, Y). En effet, X et Y sont disjoints et donc pour chaque élément de W, on indique 0, 1, ou 2 pour dire qu'il est dans X, dans Y ou dans aucun des deux. D'où la complexité.

■

Proposition 47 Soit $G = (V, E)$ avec $tw(G) = k$. Soit $W \subseteq V$ avec $|W| \leq 3k + 1$. Alors il y a un W-séparateur faiblement balancé de cardinal au plus $k + 1$.

DÉMONSTRATION. D'après la proposition 33, il y a un W-séparateur S balancé de cardinal $\leq k + 1$. Soit $C_1, \ldots, C_m$ les composantes connexes de $G \setminus S$, qui ont une intersection non vide avec W. Posons $W_i = C_i \cap W$. On a $W \setminus S = \bigsqcup_{i=1}^{m} W_i$. On a $|W_i| \leq |W|/2$.

Fait 48 $m \geq 2$.

Démonstration. Par l'absurde. $m \leq 1$ impliquerait $|W \setminus S| \leq |W|/2$. Mais aussi $W \setminus S \leq |W| - |S| = 3k+1-(k-1) = 2k+2 > |W|/2$. ■

Sans perte de généralité, supposons que $|W_1| \geq |W_2| \geq \ldots |W_m|$. Soit i minimum tel que $\sum_{j=1}^{i} W_i \geq \frac{|W \setminus S|}{3}$ (existe car l'union (disjointe) des W_i est $W \setminus S$).

On pose $X = \bigsqcup_{j=1}^{i} W_j$ et $Y = \bigsqcup_{j=i+1}^{m} W_j$.

Fait 49 $|X| > 0$.

Fait 50 $|Y| > 0$.

Fait 51 $|X| \leq 2/3|W|$.

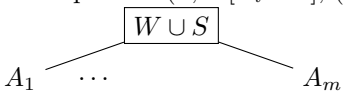
Fait 52 $|Y| \leq 2/3|W|$.

■

A.4 Algorithme calculant une décomposition

entrée : k un entier, $G = (V, E)$ un graphe, W un sous-ensemble de sommets avec $|W| \leq 3k+1$
sortie : une décomposition arborescente de G de largeur $\leq 4k+1$ où le sac à la racine inclut W, ou échec s'il n'y a pas de telle décomposition

fonction décomposition(k, G, [$W = \emptyset$])
  si $|V| \leq 4k+2$ **alors**
    renvoyer une arbre-racine où le sac contient tout V
  sinon
    W := un sur-ensemble de W de $3k+1$ éléments
    S := un W-séparateur balancée faible (s'il en a pas échec)
    $C_1, \ldots, C_m$:= les composantes connexes de $G \setminus S$
    pour $i := 1$ à m **faire**
      A_i := décomposition(k, $G[C_i \cup S]$, $(W \cap C_i) \cup S$)
    renvoyer



Proposition 53 L'algorithme termine et sa spécification est bien respectée.

Démonstration. **Terminaison.** La taille des graphes diminue strictement au cours des appels.

Fait 54 $|C_i \cup S| < |V|$.

Démonstration. Il y a une partition $W = X \sqcup (S \cap W) \sqcup Y$. Ainsi, C_i touche X ou Y mais pas les deux. Donc il y a forcément un élément de W qui n'est pas dans $C_i \cup S$. Ainsi, $C_i \cup S$ ne peut pas être V tout entier. ■

Correction.

Fait 55 $|(W \cap C_i) \cup S| \leq 3k+1$.

Démonstration. $|(W \cap C_i) \cup S| \leq |W \cap C_i| + |S| \leq 2/3|W| + |W| = |W|$. ■

Proposition 56 décomposition(k, G, W) retourne une décomposition arborescente de largeur $\leq 4k+1$.

Démonstration. Un sommet qui est à la fois dans $C_i \cup S$ et dans $C_j \cup S$ pour $i \neq j$ est dans S et donc dans le sac de la racine. Pour les arêtes : tout va bien.

Et pour la largeur : $|W \cup S| \leq 3k+1+k+1 \leq 4k+2$. D'où une largeur d'au plus $4k+1$. ■

Proposition 57 Si $tw(G) \leq k$, alors décomposition(k, G, W) réussit.

Démonstration. Par induction sur le nombre de sommets dans G. Par la proposition 40, si $tw(G) \leq k$, on montre qu'il existe un W-séparateur balancé faible (et donc pas d'échec !).

■ ■