Parti pris → seulement graphes orientés → pas un bon choix.

925

Graphes : représentations et algorithmes

Les graphes permettent de modéliser de nombreux problèmes de logistique comme la gestion de tâches, l'installation de réseaux, la distribution ...

I Définitions générales

I.0. Déf $\vec{G}=(V,\vec{E})$ est un graphe orienté ssi $\begin{cases} V \text{ est un ensemble fini non vide} \\ \vec{E} \subset V^2_{\neq} = \{(x,y) \in V^2 \mid x \neq y\} \end{cases}$.

On appellera sommet les éléments de V et arcs ceux de $\vec{E}$.

On appelle ordre du graphe $\# V$, et taille du graphe $\# \vec{E}$. Cf fig 1

Rq Le terme graphe désigne aussi des graphes non orientés que l'on définit dans la section II.2.

Soit $\vec{G}=(V,\vec{E})$ un graphe orienté. $n = \# V$, $m = \# \vec{E}$.

1) Graphe partiel et graphe induit

I.1 Déf $G'=(V',\vec{E}')$ est un graphe partiel de $\vec{G}$ ssi $V'=V$, $\vec{E}' \subset \vec{E}$

— induit — ssi $V' \subset V$ $E' = \{(u,v) \in V'^2 \mid (u,v) \in \vec{E}\}$

I.2 Rq On obtient un graphe partiel en retirant des arcs. G admet 2^m graphes partiels.

On — induit — sommets et les arcs qui n'ont alors plus de sens. G admet 2^n graphes induits. Cf fig 2

2) Degrés, chemins, forte connexité

Cf fig 3.

I.3 Déf Soit $(u,v) \in V^2$. u est successeur de v ssi $(v,u) \in \vec{E}$

- u est prédécesseur de v ssi $(u,v) \in \vec{E}$
- le degré sortant de u, noté $deg^+(u)$ est son nombre de successeurs
- — entrant — $deg^-(u)$ — prédécesseurs

I.4 Rq $\sum_{u \in V} deg^+(u) = \sum_{u \in V} deg^-(u) = m$

I.5 Déf Soit $(s,t) \in V^2$. Soit $(a_i = (u_i, v_i))_{i \in [1,l]} \in E^l$ où $l \geq 1$.

- $(a_i)_{i \in [1,l]}$ est un chemin de s à t ssi $u_0 = s$, $\forall i \in [1,l]$ $u_i = v_{i-1}$, $v_l = t$
- l s'appelle alors la longueur du chemin de s à t
- si $s = t$ on parle de circuit

I.6 Déf Soit $(s,t) \in V^2$. s et t sont fortement connectés dans G ssi il existe un chemin de s à t et un chemin de t à s dans G.

I.7 Pté La forte connexion dans G est une relation d'équivalence sur V.

I.8 Déf On appelle composantes fortement connexes (CFC) les classes d'équivalences pour cette relation.

Un graphe est dit fortement connexe s'il n'a qu'une seule CFC.

3) Pondération

I.9 Déf p est une pondération (ici entière) de $\vec{G}$ ssi $p \in \mathcal{F}(\vec{E}, \mathbb{Z})$

Muni de p on dira que G est pondéré.

I.10 Déf Si $(a_i)_{i \in [1,l]}$ est un chemin dans G, son coût est $\sum_{i=1}^{l} p(a_i)$.

⚠ Même en passant au min on n'a pas une distance car les coûts d'un chemin de s à t et d'un chemin de t à s sont a priori ≠.

II Représentations de graphes

II.0 Il s'agit ici de coder de manière effective l'objet mathématique qu'est le graphe orienté. De choix de cet encodage dépendront les complexités des algorithmes d'où son importance.

1) Représentations matricielles

(On identifie V à $(v_i)_{i \in [1,n]}$ et $\vec{E}$ à $(a_k)_{k \in [1,m]}$)

II.1 Déf La matrice d'adjacence de $\vec{G}$ est $(M_{ij})_{(i,j) \in [1,n]^2}$ où $M_{i,j} = \begin{cases} 1 \text{ si } (v_i, v_j) \in V \\ 0 \text{ sinon.} \end{cases}$

pas obligé d'en parler (car aucun algo présenté n'utilise son importance)

II.2 Déf La matrice d'incidence de $\vec{G}$ est $(N_{i,k})_{(i,k) \in [1,n] \times [1,m]}$ où $N_{i,k} = \begin{cases} -1 \text{ si } \exists v \in V, (v_i, v) \in \vec{E} \\ 1 \text{ si } \exists v \in V (v, v_i) \in \vec{E} \\ 0 \text{ sinon} \end{cases}$

colonnes : arêtes

II.3 Pté Pour $i \in [1,n]$ $deg^-(v_i) - deg^+(v_i) = \sum_{j=1}^{n} M_{i,j} = \sum_{k=1}^{m} N_{i,k}$

Cf fig 4. + pbd la démo

2) Représentation avec des listes d'adjacences

II.4 Déf Pour $i \in [1,n]$ on note $L_S(i) = \{j \in [1,n] \mid v_j \text{ est successeur de } v_i\}$

$L_P(i) = \{$ — | — prédécesseur — $\}$

(change si le graphe est orienté ou non)

Alors $TS = (L_S(i))_{i \in [1,n]}$ est le tableau des listes de successeurs

$TP = (L_P(i))_{i \in [1,n]}$ est le tableau des listes de prédécesseurs

II.5 Rq : Dans certains algorithmes, on implémente la liste d'adjacence avec un arbre binaire de recherche ou un tas pour réduire la complexité.

3) Complexités comparées

II.6

	matrice d'adjacence	matrice d'incidence	liste de successeurs	liste de prédécesseurs
espace mémoire	$O(n^2)$	$O(n \times m)$	$O(n+m)$	$O(n+m)$
v_i est-il successeur de v_j ? / v_j — prédécesseur de v_i	$O(1)$	$O(m)$	$O(1)$ / $O(m)$	$O(m)$ / $O(1)$
a est-elle incidente en i	—	$O(1)$	—	—
parcourir les successeurs de v_i ? / parcourir les préd de v_j ?	$O(n)$	$O(m)$	$O(deg^+(v_i))$ / $O(m)$	$O(m)$ / $O(deg^-(v_j))$

?

III Parcours de graphes orientés

1) Parcours en général

III.0 Un parcours fournit un ordre de visite des sommets du graphe reflétant certaines de ses propriétés structurelles. Les algorithmes de parcours sont à la base de nombreux algorithmes tels que Dijkstra, Bellman-Ford, Prim..

III.1 Déf Soit $V^0 \subset V$ un ensemble de sommets de $\vec{G}$. La bordure de V^0 est l'ens. des successeurs de sommets de V^0 qui ne sont pas dans V^0
Aut. dit $B(V^0) = \{y \in V \setminus V^0 \mid \exists x \in V^0, (x,y) \in \vec{E}\}$

III.2 Déf Un parcours de $\vec{G}$ est une liste $(s_i)_{i \in [1..n]}$ des sommets de G tel que $\forall i \in [1..n-1]$ $B(\{s_1, \dots s_i\}) = \emptyset$ ou $s_{i+1} \in B(\{s_1 \dots s_i\})$

III.3 On construit le parcours en ajoutant successivement les sommets dans une liste L
Pour gérer B la bordure de la liste courante il nous faut une structure de donnée munie des opérations
- créer
- est-vide
- ajouter
- extraire

Ci-contre on désigne par bordure ce type abstrait. On verra dans la suite qu'on peut utiliser des files, piles, files de priorité selon des clés ajoutées aux sommets...

```
PARCOURS_GÉNÉRIQUE (G)
L = créer_liste      // parcours
B = créer_type_bordure
Tant que L ≠ V
    Choisir s ∈ V \ L   (*)
    B.ajouter(s)
    Tant que non(B.est_vide())
        x = B.extraire      (▲)
        Pour y ∈ succ(x)
            Si y n'a pas été visité
            dans B ajouter(y)
        L.ajouter(x)   (□)
retourner L
```

Rq En ★ la bordure est vide.

III.4 Pté La complexité de PARCOURS_GÉN. est $O(n \times (C_{ajouter} + m \times C_{extraire}))$

2) Parcours en largeur Cf fig 5.

III.4 On obtient un parcours en largeur en choisissant d'implémenter le type bordure par une file (on visite d'abord les successeurs du premier entré dans B)

III.5 Rq Si v est le premier sommet visité et si w est à distance k de v alors tous les sommets à distance inférieure à k de v sont visités avant W. Autrement dit on parcourt les sommets par distance à la source croissante ce qui permet de calculer dans une arborescence la distance à la source (en nombre d'arcs).

III.6 Rq Dans le cas d'un graphe infini, le parcours en largeur garantit que l'on visitera un jour chaque sommet. Cela sert notamment à déterminiser une MT.

3) Parcours en profondeur Cf fig 5.

III.7 On obtient un parcours en profondeur en choisissant d'implémenter le type bordure par une pile (on visite d'abord les successeurs du dernier entré dans B)

III.8 Déf Considérant une exécution d'un parcours en profondeur de $\vec{G}$ et pour $s \in V$, on définit
→ pré(s) = la date de prévisite de s
= le moment où s a été dépilé de la bordure (▲)
→ post(s) = la date de post visite de s
= le moment où tous les successeurs de s ont été visités (□)

III.9 Déf Un tri topologique des sommets de G est une liste ordonnée $(v_i)_{i \in [1..n]}$ des sommets telle que $(v_i, v_j) \in \vec{E} \Rightarrow i < j$.

III.10 Pté Trier les sommets selon des valeurs de post-visite décroissantes fournit un tri topologique, pourvu que le graphe soit acyclique.

III.11 Pté G présente un circuit ssi il existe $x \in V$ et $y \in succ(x)$ tel que pré(y) ≤ pré(x) et post(y) ≥ post(x),
"x est descendant de y dans G"

↳ On peut donc utiliser un parcours en profondeur pour la détection de circuit

III.12 Pour calculer les CFC de G on commence par calculer L = PARCOURS_PROF(G), on calcule ensuite G^t le transposé de G défini par $(V, \{(v,u) \mid (u,v) \in \vec{E}\})$
On applique alors le parcours en profondeur à G^t en modifiant la ligne (*) pour choisir s selon l'ordre défini par L, et repérer à ce moment là qu'on a une nouvelle CFC

IV Chemins dans un graphe

A) Plus courts chemin

IV.0 Calculer des itinéraires revient à calculer des plus courts chemins dans un graphe orienté pondéré connexe, ceci n'a un sens que lorsque le graphe ne présente pas de circuit de poids négatif.

1) Algorithme de Bellman-Ford

Entrée : G graphe orienté connexe
p pondération de G
s un sommet de G

Sortie : les longueurs des plus courts chemins de s à chacun des sommets s'il n'y a pas de cycle ≤ 0, FAUX sinon

```
BELLMAN-FORD (G, p, s)     (IV.1)
d = tableau indexé par V initialisé à +∞ ; d[s] = 0
Pour i allant de 0 à |S|-1
    Pour chaque (u,v) ∈ E
        si d[v] > d[u] + p(u,v)
        alors d[v] = d[u] + p(u,v)
Pour chaque (u,v) ∈ E [si d(v) > d(u) + p(u,v) alors retourner FAUX
retourner d
```

IV.2 Pté Si G est donné codé par une liste d'adjacence, l'algorithme de B.F termine en $O(n \times m)$.

2) Algorithme de Dijkstra

Entrée : G graphe orienté connexe
p pondération ≥ 0
s sommet de G

Sortie : d tableau des longueurs des plus courts chemins à s.

DIJKSTRA (G, p, s) (IV.3)

```
d = tableau indexé par V initialisé à +∞ ; d[s] = 0
F = crée_file_de_priorité_min
Pour tout u ∈ V
    [F.ajouter(u, d[u])
Tant que non F.est_vide
    u = F.extraire_min
    pour chaque v ∈ succ(u)
        [si d[v] > d[u] + p(u,v)
        [alors d(v) = d[u] + p(u,v)
renvoyer d
```

pas la vraie complexité !

IV.4) Complexité : - avec une liste : $O(n^2)$ et une liste d'adjacence
- avec un tas min : $O(m \log n)$ pour avoir $succ = O(1)$

3) Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall est un algorithme de programmation dynamique permettant de calculer la distance min entre tout couple de sommets d'un graphe sans cycles de poids négatif.

Entrée : G, w graphe pondéré sans cycle négatif

Sortie : Tableau T $n \times n$ tel que $T[i,j] = d(i,j)$ si les sommets sont $1, \dots, n$

FLOYD-WARSHALL (G, w) (IV.5)

```
D := w
Pour k = 1 à n faire
    D^(k) = (d_ij^(k))
    Pour i,j ∈ n faire
        d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
Retourner D^(n)
```

IV.6 Complexité en $O(n^3)$

B) Chemin Hamiltonien et Voyageur de Commerce

1) Chemin Hamiltonien : Un chemin hamiltonien est un chemin passant une et une seule fois par chaque sommet.

Déf : HAM(G) est le problème de décision associé à l'existence d'un chemin Hamiltonien dans G.

Théorème : HAM est NP-complet. ← DEV I

2) Voyageur de commerce. Le problème du voyageur de commerce (PVC) est de trouver dans un graphe complet le cycle Hamiltonien de poids min.

Pté : Si w vérifie l'inégalité Δ, il existe un algo d'approx. 2 en tps pol. utilisant PRIM.

V Problèmes de réseaux

1) Problème et modélisation

V.0 Le problème est d'installer un réseau électrique alimentant un ensemble de villes à moindre coût, connaissant les axes où il est possible d'installer des lignes et à quel coût.

V.1 On le modélise par un graphe non orienté, pondéré où les sommets sont les villes, les arcs sont les axes d'installation possibles, et leur poids le coût de réalisation de la ligne. On le suppose connexe quitte à travailler composante par composante.

2) Notion d'arbre couvrant dans un graphe non orienté

V.2 Déf $\vec{G}$ est non orienté si $\forall (u,v) \in V$, $(u,v) \in \vec{E} \Rightarrow (v,u) \in \vec{E}$.
On le note alors $G = (V, E)$ où $E = \{\{u,v\} \mid (u,v) \in \vec{E}\}$ $\begin{cases} m = \# E \text{ plus } \# \vec{E} \\ n = \# V \end{cases}$

le mettre en déf (au début)

V.3 Voir un graphe non orienté comme cas particulier de graphe orienté permet de réutiliser les définitions mais la terminologie change :

arc → arête	sommets fortement connectés → sommets connectés
chemin → chaîne	composantes fortement connexes → composantes connexes
circuit → cycle	graphe fortement connexe → graphe connexe.

V.4 Déf G est un arbre ⇔ G est connexe acyclique (ie sans cycle)
V.5 Pté ⇔ G connexe minimal ⇔ G est acyclique maximal
⇔ G connexe et $\#E = \#V - 1$ ⇔ G acyclique et $\#E = \#V - 1$

V.6 Déf · Un arbre couvrant de G est un graphe partiel de G qui est un arbre.
· Si G est pondéré, un arbre couvrant de poids minimal (ACPM) est un arbre couvrant dont la somme des poids de ses arêtes est minimale.

V.7 Rq Il existe toujours un ACPM, mais il n'y a pas unicité a priori.

3) Résolution du problème et algorithmes

V.8 Une solution au problème serait donc un ACPM du graphe considéré. On présente ici deux algorithmes de calcul d'ACPM (entrée G connexe pondéré, sortie un ACPM) qui sont en fait deux instanciations d'un algorithme plus général. ~ DEV II.

Prim $(G = (V,E), p)$

```
A = crée liste ; VA = crée liste
choisir s∈V ; VA.ajouter(s)
tant que VA.taille < n
    [choisir (x,y) ∈ E de coût min tel que {x ∈ VA, y ∉ VA}
    [VA.ajouter(y) ; L.ajouter((x,y))
retourner A.
```

Kruskal $(G = (V,E), p)$

```
L = liste des arêtes triées selon p croissant
A = crée_liste , i = 0
tant que (V, A) n'est pas connexe
    [i <- i+1
    [si e_i connecte 2 composantes connexes de (V,A)
    [alors A.ajouter(e_i)
retourner A.
```

↳ en $O(n \log(n) + m \log(n))$ pour une implémentation fig 1 : $\vec{G} = (V, \vec{E})$

$V = \{a, b, c, d, e, f\}$

$E = \{(a,b), (a,c), (b,d), (c,d), (b,c), (c,b), (e,d), (c,a)\}$

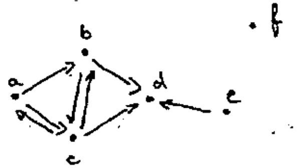
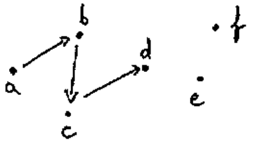
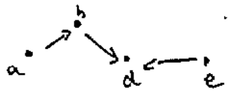


fig 2 : $\vec{G}_1$:



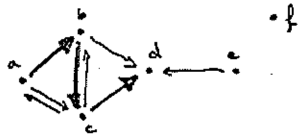
graphe partiel de $\vec{G}$

$\vec{G}_2$



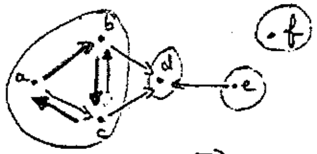
graphe induit de $\vec{G}$

fig 3



— chemin de a à d dans $\vec{G}$

$deg^+(a) = 2 \quad deg^-(d) = 3$



— circuit de $\vec{G}$

— composantes fortement connexes de $\vec{G}$

fig 4 : Pour $\vec{G}$

$$M = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Ts :
- → {b, c}
- → {c, d}
- → {a, b, d}
- → ∅
- → {d}
- → ∅

Chaque colonne ≡ une arête

−1 = départ

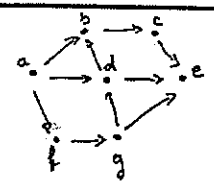
+1 = arrivé

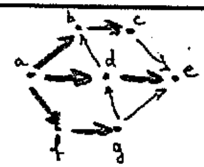
$$N = \begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 & 1 & -1 & 0 & -1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Tp :
- → {c}
- → {a, c}
- → {a, b}
- → {b, c, e}
- → ∅
- → ∅

fig 5

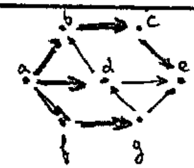
$\vec{G}_3$:





L = [a, b, d, f, c, e, g]

parcours en largeur



L = [a, b, c, e, d, f, g]

parcours en profondeur

Bibliographie

- Cormen
- Carton
- COGIS - ROBERT "Théorie des graphes".

CYCLES ET COCYCLES

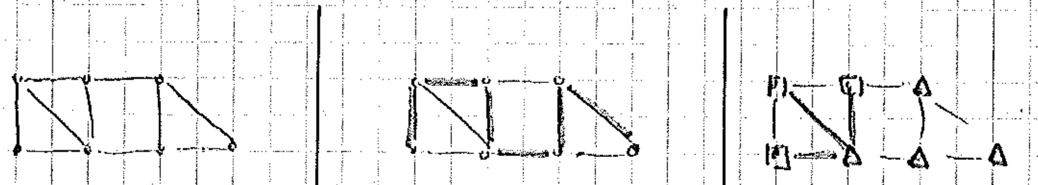
Soit $G=(V,E)$ un graphe non orienté connexe $\# V = n$ $\# E = m$

- G est un <u>arbre</u> $\Leftrightarrow$ G est connexe et acyclique $\Leftrightarrow$ G connexe et $m=n-1$
$\Leftrightarrow$ G acyclique et $m=n-1$
$\Leftrightarrow$ G est connexe minimal (ie tt retrait d'arête lui fait perdre sa connexité)
$\Leftrightarrow$ G est acyclique maximal (ie tt ajout d'arête introduit un cycle)

- Un <u>arbre couvrant de G</u> est un sous-graphe de G qui est un arbre.

↳ Rq : En tant que sous-graphe ses sommets sont exactemt ceux de G
Ce qui importe c'est quelles arêtes de G il sélectionne.
Dans la suite je dirai "A est un arbre couvrant de G" pour (V,A) est un arbre couvrant

- Une <u>coupe</u> de G est une bipartition de V.
Le <u>cocycle</u> associé à la coupe (V_1,V_2) est $\{(u,v)\in E \mid u\in V_1$ et $v\in V_2$ ou vice versa$\}$.



G
A arbre courant de G

(X, Y)
□ △
bipartition
Co cocycle associé

On peut mettre les déf. et le premier lemme dans Le plan

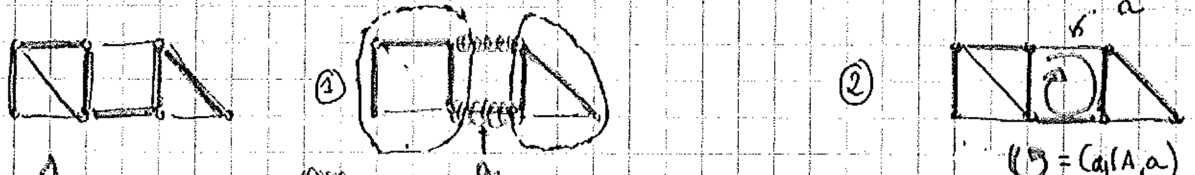
Lemme de l'échange d'arbre

A = {ens d'arêtes}
(A couvrant ⇒ ses sommets sont exactement ceux de G)

Soit A un arbre couvrant de G. (A^c désigne $E \setminus A$)

① Pour $a\in A$, il existe un unique cocycle noté $Co(a,A)$ tel que $Co(a,A)\cap A = \{a\}$
De plus pour tout $b\in Co(a,A)$, $b\neq a$, $A\cup\{b\}\setminus\{a\}$ est encore un arbre couvrant

② Pour $a\in A^c$, il existe un unique cycle noté $Cy(a,A)$ tel que $Cy(a,A)\cap A^c = \{a\}$.
De plus pour tout $b\in Cy(a,A)$, $A\cup\{a\}\setminus\{b\}$ est encore un arbre couvrant



G A
$Co(A,a)$
a
$= Cy(A,a)$

Preuve :

① Soit $a=(u,v)$ une arête de A. En tant qu'arbre A (ou plutôt (V,A)) est connexe minimal, donc a est un isthme c-à-d que $A\setminus\{a\}$ est séparé en 2 composantes connexes : celle de u et celle de v, ceci fournit naturellemt une coupe (U,V) dont le cocycle associé contient a (U = sommets reliés à u ds $A\setminus\{a\}$, idem pour V). De plus ce cocycle ne contient aucune autre arête de A, car elle serait dans $A\setminus\{a\}$ et nierait la déf. de composante connexe. <u>D'où l'existence</u>
Si le cocycle associé à la bipartition (X,Y) convient on a par ex $u\in X$ et $v\in Y$. Si $s\in U$ il existe dans $A\setminus\{a\}$ un chemin le reliant à u donc $s\notin Y$ sinon ce chemin aurait une arête dans le cocycle IMPOSSIBLE.
Donc néc $s\in X$. D'où $U\subset X$, symétriquement $V\subset Y$ donc $X=U$, $V=Y$. <u>D'où l'unicité</u>

Pour $b\in Co(a,A)$ b joint U et V les 2 comp. connexes de $A\setminus\{a\}$. Ainsi $A\setminus\{a\}\cup\{b\}$ est connexe et a $n-1$ arêtes → c'est l'arbre couvrant

② Soit $a=(u,v)\notin A$. Il existe un unique chemin de u à v dans A, en le concaténant à a on obtient un cycle dont la seule arête de A^c est a. Il répond aux pb et est unique car cette construc est nécessaire. Ce cycle est le seul de $A\cup\{a\}$ donc pour $b\in Cy(a,A)$ $A\cup\{a\}\setminus\{b\}$ est de nouveau acyclique, il a $n-1$ arêtes → c'est bien un arbre couvrant. □

ARBRES COUVRANTS DE POIDS MINIMAL (ACPM)

cf Cogis et Robert
Théorie des graphes, au delà des ponts de Königsberg
éd. Vuibert
p106 - 118.

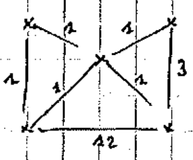
Soit $G=(V,E)$ un graphe non orienté connexe $\#V = n$ $\#E = m$.
Soit p une pondération des arêtes de G ie une f° de E dans $\mathbb{N}$.

Pour A un arbre couvrant de G (A désigne seulem^t les arêtes, le vrai arbre est (V,A))
son poids sera $p(A) = \sum_{a \in A} p(a)$.

Un ACPM est un arbre couvrant dont le poids est min. parmi les arbres couvrants.

• (B,R) est un approximant de G $\Leftrightarrow$ $\begin{cases} B \subset E,\ R \subset E \text{ et } B \cap R = \emptyset \\ \text{il existe } A \text{ un ACPM tq } B \subset A \text{ et } R \subset A^c = E \setminus A. \end{cases}$
je dirais alors que A justifie l'approximant.

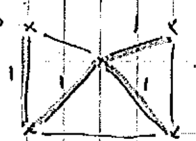
⚠ Tout arbre couvrant de poids min ne couvrira pas nécessairement B.
De m̂ tt ACPM n'exclura pas nséc R.



— B
— R
(B,R) est un approximant car :

A
ACPM
$p(A) = 4$

mais



$\tilde{A}$
ACPM
$p(\tilde{A}) = 4$

• Si (B,R) est un approximant de G et $a \in E$
 · a est blanche $\Leftrightarrow$ $a \notin B$, $a \notin R$
 · a est bleue $\Leftrightarrow$ $a \in B$
 · a est rouge $\Leftrightarrow$ $a \in R$

• $(\emptyset,\emptyset)$ est toujours un app.

• $\left.\begin{array}{l}(B,R) \text{ approximant} \\ \#B = n-1\end{array}\right\} \Rightarrow B$ est un AC.

• $\left.\begin{array}{l}(B,R) \text{ approximant} \\ \#R = m-(n-1)\end{array}\right\} \Rightarrow E \setminus R$ est un AC

Rq Aucun cycle ne peut être tt bleu
Aucun cocycle ne peut ê tt rouge.

L'idée est de construire un ACPM en faisant grossir un approximant à partir de $(\emptyset,\emptyset)$, ce qui veut dire qu'on va colorier les arêtes en bleu ou en rouge selon qu'on les sélectionne ou qu'on les rejette. On donne donc deux règles qui permettent un coloriage valide.

Pté ① Si $\begin{cases} (B,R) \text{ est un approximant} \\ Co \text{ un cocycle } \subset B^c \\ a \in Co_{\setminus R} \text{ minimale} \end{cases}$
alors $(B \cup \{a\}, R)$ est un approximant

RÈGLE BLEUE
ie du COCYCLE

Choisir Co un cocycle rouge et blanc
Choisir $a \in Co$ min parmi les arêtes blanc
La colorier en bleu

② Si $\begin{cases} (B,R) \text{ est un approximant} \\ C \text{ un cycle } \subset R^c \\ a \in C_{\setminus B} \text{ maximal} \end{cases}$
alors $(B, R \cup \{a\})$ est un approximant

RÈGLE ROUGE
ie du CYCLE

Choisir C un cycle bleu et blanc
Choisir $a \in C$ max parmi les arêtes blanches
La colorier en rouge.

Preuve ① Notons (X,Y) la coupe associée au cocycle considéré Co et
Soit A un ACPM justifiant l'approximant (B,R) (ie un ACPM tq $B \subset A$ et $R \subset A^c$).

→ Co ne peut être complètement rouge (sinon A, qui ne prend pas d'arêtes rouges, ne relierait pas X et Y, or A couvrant connexe.
→ Prendre a une arête blanche minimale dans Co a bien un sens.
→ A, qui n'a que des arêtes bleues et blanches, en a nécessairement une dans Co, or Co n'a que des arêtes rouges et blanches, donc A a néc une arête blanche e dans Co. Par minimalité de A, $cout(a) = cout(e)$.
Ainsi, grâce au lemme de l'échange, $A \cup \{a\} \setminus \{e\}$ est encore un ACPM et justifie l'approximant $(B \cup \{a\}, R)$

② Soit A un ACPM justifiant l'approximant (B,R)

- C ne peut être entièrement bleu (sinon A qui couvre les bleus ne serait pas acyclique).
- Choisir a arête blanche maximale dans C a bien un sens
- A ne pouvant recouvrir C, A^c a nécessairement une arête dans C, or les arêtes de A^c sont rouges ou blanches et celles de C bleues ou blanches, on en déduit que A^c a une arête blanche e dans C. La minimalité du poids de A assure la maximalité du poids de A^c. Donc coût (a) = coût(e) et grâce au lemme d'échange on en déduit $A_{\setminus\{a\}} \cup \{e\}$ est encore un ACPM donc $(B, R \cup \{a\})$ est encore un approximant. □

On a donc l'algo générique suivant :

rq c'est un algorithme non déterministe, on ne sait pas si on applique RB ou RR, et ni à quel cycle ou cocycle.

ACPM : entrée : un graphe connexe non orienté G, p une valuation des arêtes
sortie : un ensemble d'arêtes définissant un ACPM de G.

```
ACPM (G=(V,E), p)
  init. ttes les arêtes sont blanches
  tant qu'il y a des arêtes blanches
     appliquer la règle bleue ou la règle rouge
  renvoyer les arêtes bleues.
```

Pté ACPM termine et fournit un arbre couvrant de poids min.

① A chaque étape (B,R) où B = {arêtes bleues}, R = {arêtes rouges} est un approximant
En effet $(\emptyset,\emptyset)$ est un app. et les règles bleu ou rouge préservent cet invariant.

② A chaque étape on peut bien appliquer l'une ou l'autre des règles (s'il reste une arête blanche)
En effet soit a une arête blanche, et soit A un ACPM justifiant l'approximant (B,R).
→ soit $a \in A$. On considère alors $Co(a,A)$, l'unique cocycle de G tq $Co(a,A) \cap A = \{a\}$. A part a qui est blanche, ce cocycle n'a que des arêtes de A^c donc rouges ou blanches. On peut lui appliquer RB.
→ soit $a \notin A$. On considère alors $Cy(a,A)$, l'unique cycle de G tq $Cy(a,A) \cap A^c = \{a\}$. A part a qui est blanche, ce cycle n'a que des arêtes de A donc bleues ou blanches. On peut lui appliquer RR.

③ L'algo termine car à chaque étape on colorie une arête et qu'il n'y en a qu'un nombre fini

④ Quand la boucle tant que termine (B,R) est un approximant d'après 1 et $B \cup R = E$. Il existe un ACPM A justifiant (B,R) donc $B \subset A$ et $R \subset A^c$ donc $A \subset R^c = E \setminus B = B$ d'où $A = B$, en renvoyant les arêtes bleues on fournit bien un ACPM. □

```
Kruskal (G p)
  L = liste des arêtes triées par p croissant
  A = liste vide
  i = 0
  tant que (V,A) n'est pas connexe
     i ← i+1
     si ei connecte 2 composantes connexes de (V,A)
     alors A.ajouter(ei)
  retourner A.
```

Pté L'algorithme de Kruskal ci-contre est une application particulière de ACPM, cela assure sa terminaison et sa correction.

Preuve Considérons bleues les arêtes de A, rouges celles considérées dans le tant que mais non sélectionnées dans A, blanches celles non encore considérées ie les e_j pour $j > i$.

Cas du si : On suppose que A a deux comp connexes A_1 et A_2, et que e les relie. Quitte à faire grossir A_1 et A_2 on considère un coupe (X,Y) telle que $A_1 \subset X$ et $A_2 \subset Y$, ainsi $e \in Co$ le cocycle associé à (X,Y). Une autre arête de ce cocycle ne peut être bleue (dif de comp. connexe de A), si elle est de poids $\leq p(e)$, alors on l'a déjà considérée elle est rouge. Donc e est l'arête blanche de poids min de ce cocycle, la sélectionner c'est appliquer RB

Cas du sinon : Si $e = \{u,v\}$, u et v sont dans la même composante connexe de A, ie reliés par un chemin γ de A. Le cycle formé de γ et e est alors tt bleu sauf e, e est bien l'arête blanche de poids maximal du cycle, la passer (ie la rejeter, la colorier en rouge) c'est bien appliquer RR.

```
Prim (G, p)
A = créer_liste
VA = créer_liste
Choisir s ∈ V
VA.ajouter(s)
tant que VA ≠ V
    choisir (x,y) ∈ E tq x ∈ VA  de coût min
                         y ∉ VA
    VA.ajouter(y)
    A.ajouter(x,y)
retourner A
```

```
Prim_plus (G, p)
A = créer_liste()
B = créer_file_de_priorité_min()
Π = créer_tableau indexé par V, initialisé à NULL
d = ———————————————————————— +∞
Choisir s ∈ V ; d[s] = 0 ; B.ajouter(s, d(s)).
Tant que non B.est vide()
    x = B.extraire()
    pour tout y ∈ succ(x) non fermé
        si d[y] > p(x,y)
        alors Π[y] = x
              d[y] = p(x,y)
              B.ajouter(y, d[y])
    A.ajouter(Π(x), x)
    fermer x.
retourner A.
```

ou Beauquier Berstelle Chrétienne
(plus compact !)

Autres
Dvpts possibles :

- voyageur de commerce (Prim)
- Kruskal / Prim
- Preuve de Dijstra
- Floyd-Warshall → comment / Pourquoi ça fonctionne ?   (plus dvpts possibles ?)
  (Algèbre ...)
  cf. 2014/2015

Développement : NP-complétude du problème de l'existence d'un chemin Hamiltonien dans un graphe

→ Prérequis : le problème 3-SAT est NP-complet.

Définition : Un chemin dans un graphe G orienté est dit Hamiltonien si il passe une et une seule fois par chaque sommet.

Définition : On note HAM(G) le problème de décision associé à l'existence d'un chemin Hamiltonien dans le graphe G.

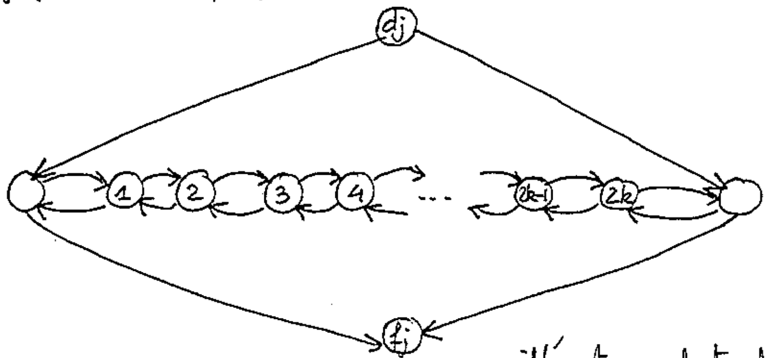
Théorème : HAM est NP-complet

- Etape 1 : HAM est dans NP.
  En effet, étant donné un chemin (une suite de sommets $s_1, \dots, s_n$), on peut vérifier en temps polynomial si c'est un chemin Hamiltonien puisqu'il suffit de vérifier que $s_i \neq s_j$ si $i \neq j$.

- Etape 2 : HAM est NP-difficile.
  Pour montrer cela, on va réduire le problème 3-SAT à HAM.
  Soit $\varphi = C_1 \wedge \dots \wedge C_k$ une instance de 3-SAT. On va construire le graphe G_φ tel que φ satisfiable $\Leftrightarrow$ G_φ possède un chemin Hamiltonien.

  - On note m le nombre de variables apparaissant dans φ.
  - On suppose que chaque variable apparait au plus une fois dans chaque clause. Ceci n'est pas contraignant puisque $x \vee x \equiv x$ et si x et $\bar{x}$ apparaissent dans une clause C_i alors cette clause est automatiquement satisfaite.
  - On note les variables x_j pour $j = 1$ à m.

- Dans G_φ, on va associer à chaque variable x_j ($j \in \{1, \dots, m\}$) un sous-graphe appelé gadget associé à x_j qui est représenté par le graphe suivant qui possède $2k+4$ sommets.

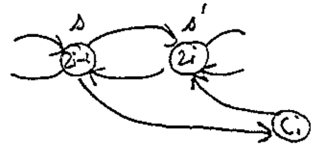


Le graphe G_φ est obtenu à partir des différents gadgets de la manière suivante :

- $\forall j \in \{1, \dots, m-1\}$, on crée une arête entre f_j et d_{j+1}
- On ajoute k nouveaux sommets $C_1, \dots, C_k$ associés à chacune des clauses et ajouter des arêtes dans les cas suivants :

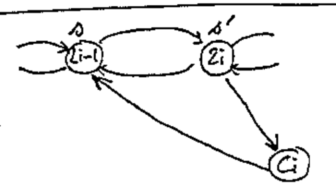
Si la variable v_j apparait (positivement ou négativement) dans la clause C_i, on note respectivement s et s' les sommets numérotés $2i-1$ et $2i$ dans le gadget associé à v_j et on ajoute soit

- L'arête $\overrightarrow{sC_i}$ et $\overrightarrow{C_is'}$ si v_j apparait positivement dans la clause C_i



$\boxed{v_j \in C_i}$

- les arêtes $\overrightarrow{s'C_i}$ et $\overrightarrow{C_is}$ si v_j apparait négativement dans C_i



$\boxed{\overline{v_j} \in C_i}$

→ Le graphe G_φ ainsi obtenu possède $k + m(2k+4)$ sommets et peut donc être construit en temps polynomial à partir de φ.

Propriété : φ est satisfiable ssi G_φ possède un chemin Hamiltonien
ssi G_φ possède un chemin Hamiltonien de d_1 à f_m.

- Un chemin Hamiltonien dans G_φ va nécessairement de d_1 à f_m puisque le degré entrant de d_1 est nul et le degré sortant de f_m est nul.

Supposons φ satisfiable. Il existe une valuation ν telle que $\nu(\varphi)=1$
En particulier pour chaque clause C_i, il existe une variable $v_{j(i)}$ telle que soit : • $\nu(v_{j(i)})=1$ et $v_{j(i)}$ apparait positivement dans C_i.
• $\nu(v_{j(i)})=0$ ——— négativement ———.

On construit un chemin Hamiltonien de d_1 à f_m en parcourant chaque gadget associé à v_j
- de gauche à droite : si $j=j(i)$ et v_j apparait positivement dans C_i et ~~puis~~ en faisant un "détour" par C_i (ie en empruntant les arêtes $\overrightarrow{sC_i}$ et $\overrightarrow{C_is'}$).
- de droite à gauche : ——— négativement ——— ——— (——— $\overrightarrow{s'C_i}$ et $\overrightarrow{C_is}$).

- On vérifie facilement que l'on ~~vérifie~~ parcourt une fois les sommets de chaque gadget et que chaque sommet C_i est rencontré car la variable $v_{j(i)}$ est telle que $\nu(v_{j(i)})=1$ et $v_{i(j)}$ apparait positivement dans C_i.
ou ——— 0 ——— négativement ———.

Réciproquement, Si ~~on a~~ il existe un chemin Hamiltonien allant de d_1 à f_m alors nécessairement :
- chaque gadget est parcouru de gauche à droite
ou de droite à gauche et si un sommet C_i est visité juste après s (resp s') alors le chemin emprunte ~~et~~ juste après l'arête $\overrightarrow{C_is'}$ (resp $\overrightarrow{C_is}$)

sinon le sommet s' (resp s) n'est jamais rencontré par le chemin.

On construit alors une valuation qui satisfait φ :
- si le gadget associé à v_j est parcouru de gauche à droite :
On pose $\nu(v_j)=1$
- ——— droite à gauche :
On pose $\nu(v_j)=0$

- Pour chaque clause C_i, il existe un gadget associé à v_j tel que C_i est parcouru lors du parcours des gadgets associés à v_j.
Si ce gadget est parcouru de gauche à droite et que le chemin fait un détour par C_i, c'est que v_j apparait ~~néga~~ positivement dans C_i et on a posé $\nu(v_j)=1$ donc $\nu(C_i)=1$
Sinon v_j apparait négativement dans C_i et on a posé $\nu(v_j)=0$ donc $\nu(\overline{v_j})=1$ puis $\nu(C_i)=1$

On a bien fabriqué une valuation qui satisfait φ.

- Comme 3-SAT est NP-~~complet~~ dur et qu'on l'a réduit en temps polynomial à HAM, HAM est également NP-difficile

Conclusion : On a bien prouvé que HAM est NP-complet.

Ref : Carton (page 193)