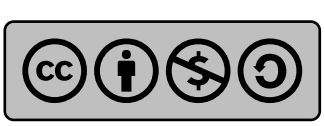
Algorithmes de Communications dans les Réseaux

Cyril Gavoille

LaBRI
Laboratoire Bordelais de Recherche
en Informatique, Université de Bordeaux

gavoille@labri.fr

13 décembre 2019
– 88 pages –



Ce document est le support d'un cours donné aux étudiants de Master Informatique (Recherche) à l'Université de Bordeaux 1 et à l'école d'ingénieur ENSEIRB depuis 2000. Il nécessite moins de 20h de cours pour être totalement présenté. Les chapitres 3 et 4 (routage compact et structures de données distribuées) peuvent former un cours indépendant.

Table des matières

CHAPITRE 1

Modèles de communication

Sommaire

Mots clés et notions abordées dans ce chapitre :

- store-and-forward, circuit-switching, wormhole
- deadlock, k-port, half/full-duplex
- communications globales (diffusion, échange total ...)

Un complément d'information sur ce chapitre peut être trouvé dans [Rum94].

1.1 Généralités

Par « réseau d'interconnexion » on entend principalement :

- un réseau de communication d'une machines parallèles à mémoire distribuées (architecture plutôt régulière : grille, hypercube, cycle, ...) ; ou bien
- un réseau d'ordinateurs inter-connectés (architecture quelconque, irrégulière).

Modélisation du réseau d'interconnexion par un graphe : $G = (V, E)$.

V (pour *vertex*) = sommets, nœuds, processeurs, routeurs, ...

E (pour *edge*) = arêtes, liens, canaux, ...

L'objectif général est d'envoyer des messages entre les sommets de G.

Notons que les graphes servent à modéliser d'autres types de réseaux, dans lequel des problèmes de communication peuvent intervenir :

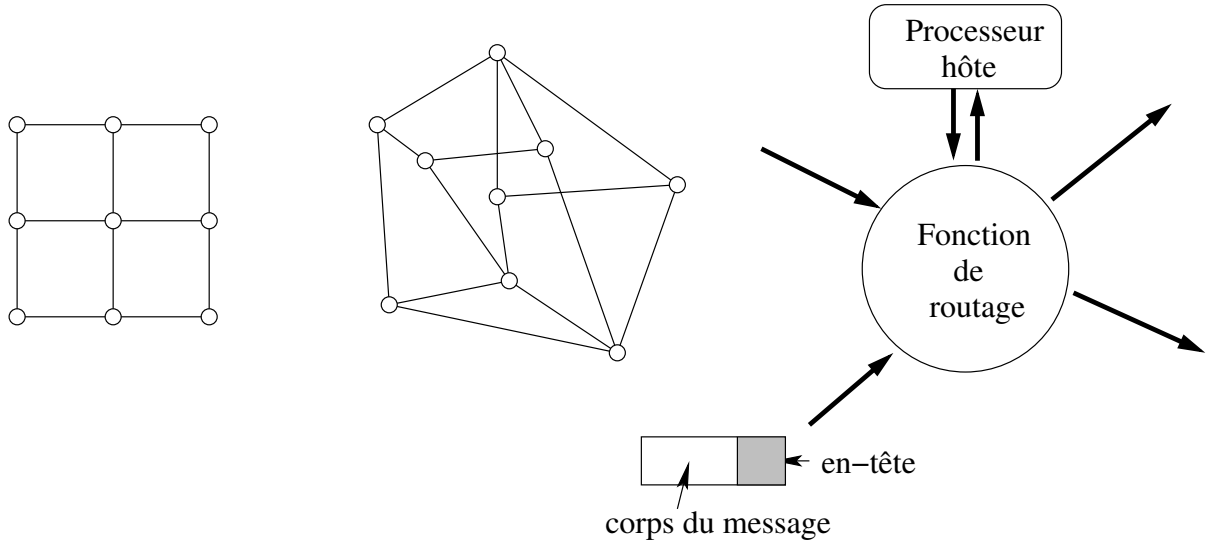


FIGURE 1.1 – Machine parallèle, réseau quelconque et un routeur.

- les réseaux sociaux (entités et relations entre les individus) ;
- le graphe du web (fichiers html et liens hyper-texte).
- réseaux radio, réseaux ad-hoc, etc.

Type de liens : bidirectionel ○——○ ou ○⇄○ (on parle de graphe orienté symétrique ou non orienté) ou unidirectionnel ○——➤○ (graphe orienté, orientation, arc).

Hypothèse pour un réseau d'interconnexion et pour le cours : G est un graphe connexe, simple (sans multi-arêtes), sans boucle, non orienté.

1.2 Modes de commutation

Seulement les principaux modes.

Commutation de circuits (*circuit-switching*)

Principe : c'est le modèle du téléphone. On réserve une suite de liens par l'envoie d'un en-tête contenant la destination, puis à la réception d'un accusé de réception, la source transmet le message en un coup.

⇒ On peut bloquer beaucoup de liens pour un message court.

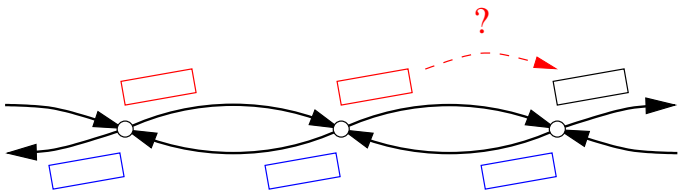
Commutation de messages (*store-and-forward*)

Principe : le message avance pas à pas vers la destination. Un seul nœud à la fois est utilisé comme ressource.

⇒ Besoin de stocker le message dans chaque routeur, besoin de mémoire (conflit de type nœud).

Commutation ou routage *wormhole*

Principe : découpage du message en message élémentaire (ou *flit* pour *flow control digit*) qui peut être stocké complètement dans le registre d'un canal (chaque lien a un *buffer*). Seul le 1er *flit* (la tête) contient l'en-tête. Les autres *flits* suivent à la queue-leu-leu, le dernier *flit* libérant les liens au fur et à mesure. Le message avance que si le prochain registre de canal est libre. Notons que deux messages peuvent se croiser sur la même arête s'ils ne vont pas dans la même direction.



⇒ Pas de copie de message (le message est stocké sur les liens). Le message peut être plus court que la longueur de la route. Impossible de couper un chemin : problème d'inter-blocage (*dead-lock*, conflit de type lien) lié à la fonction de routage.

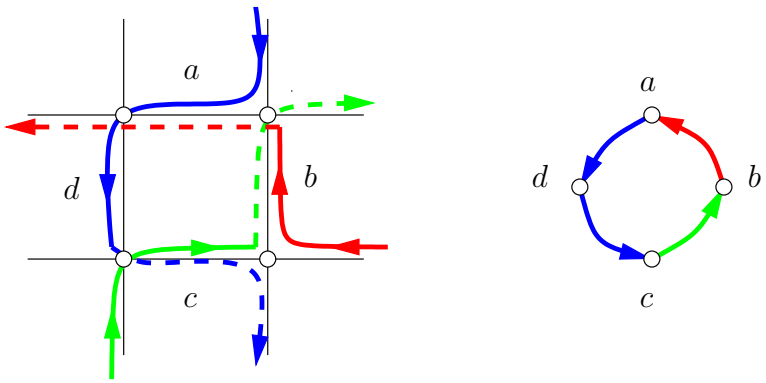


FIGURE 1.2 – Exemple d'inter-blocage en commutation *wormhole*.

On peut définir le *graphe de dépendance* des arêtes de la fonction de routage R, noté $\vec{D}(R)$. Formellement, c'est un graphe orienté où les sommets représentent les arêtes de G,

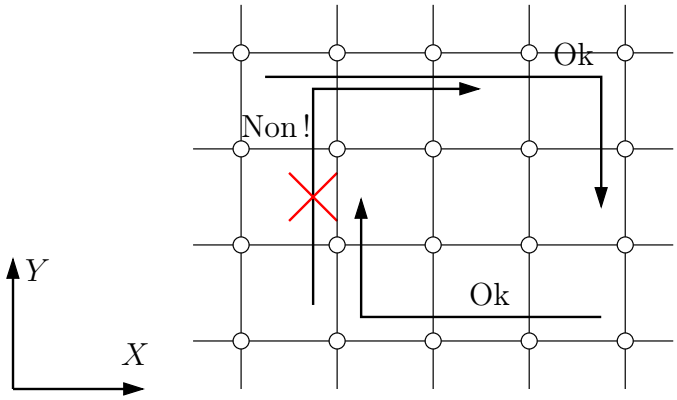


FIGURE 1.3 – Abscence d'inter-blocage pour le routage XY dans la grille.

et il existe un arcs (a, b) dans $\vec{D}(R)$ si la fonction de routage induit une route où les arêtes a et b se suivent (partage une extrêmité).

Dans l'exemple de la figure 1.2, on obtient un cycle orienté à quatre sommets (orienté dans le même sens).

Il n'est pas difficile de voir que $\vec{D}(R)$ est acyclique si et seulement si R est sans inter-blocage. En fait, si $\vec{D}(R)$ possède un cycle, alors il PEUT avoir un inter-blocage. Il faut que certaines communications aient lieux simultanément. Alors que si $\vec{D}(R)$ est acyclique le routage est garanti sans inter-blocage.

Le routage XY ne possède pas d'inter-blocage car il manque des virages (ceux de type Y suivi de X).

1.3 Modélisation du temps

La modélisation du temps que prend une communication permet la comparaison des différents modèles de commutation, mais aussi d'évaluer les différents algorithmes de communication en comparant leur complexité. Ce que l'on présente ici est une modélisation possible. Il y en a d'autres.

Temps constant

C'est le modèle le plus simple.

$$T_{\text{voisin à voisin}} = 1$$

Le temps ne dépend pas de la longueur du message, on peut donc concaténer les messages sans sur-coût.

Temps linéaire

$$T_{\text{voisin à voisin}} = \beta + L\tau$$

où β = temps d'initialisation (*start-up*), L = longueur du message (en bits), τ = temps de propagation d'un bit ($1/\tau$ = bande passante).

Temps linéaire à distance $d > 1$

• pour le *circuit-switching* et *wormhole* :

$$T_{\text{nœud à distance } d} = \alpha + d\delta + L\tau$$

où δ = temps de commutation des routeur (copie d'un *flit* d'un lien vers un son suivant). Pour $d = 1$, $\beta = \alpha + \delta$.

• pour le *store-and-forward* :

$$T_{\text{nœud à distance } d} = d(\beta + L\tau)$$

C'est d fois plus long qu'en *wormhole* !

Autres modélisations du temps

Il existe aussi le temps constant à distance d. Il s'agit du modèle « téléphone » (ou *circuit-switching*) où le temps représente ici le nombre d'appels. Il existe aussi des modèles de type « radio » où l'on peut diffuser en temps constant à une distance r donnée.

Optimisation du temps

Deux optimisations (au moins) sont possibles : le pipeline et la méthode des chemins disjoints.

• En commutation *store-and-forward* temps linéaire, il devient avantageux d'utiliser l'effet pipeline. Principe : on découpe le message M en p petits messages de taille $\ell = L/p$ bits (on suppose L divisible par p). On transmet les messages à la queue-leu-leu comme pour simuler le routage *wormhole*. On cherche à optimiser ℓ en fonction de la distance d, L, β et τ.

$$T_{x\to y} = d(\beta + \ell\tau) + (p-1)(\beta + \ell\tau) = (d + L/\ell - 1)(\beta + \ell\tau)$$

On souhaite trouver ℓ qui minimise $T_{x\to y}$? Équation du second degré

$$\Rightarrow \quad \ell_{\min} = \sqrt{\frac{L\beta}{(d-1)\tau}} \quad \text{et} \quad T^{\min}_{x\to y} = \left(\sqrt{L\tau} + \sqrt{(d-1)\beta}\right)^2$$

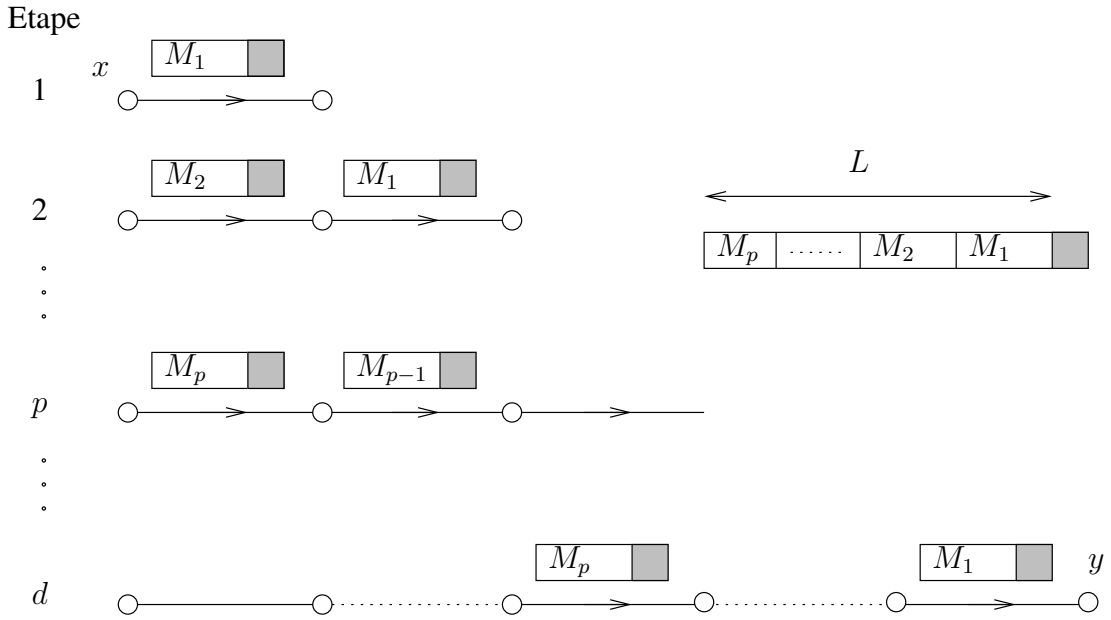


FIGURE 1.4 – Pipeline pour la commutation store-and-forward.

Ainsi quand L est grand (devant d et β), $T^{\min}_{x\to y} = L\tau + o(L)$ ce qui masque les termes relatifs à la distance. Comparable au routage *wormhole*.

Remarque : pour être plus juste il faudrait tenir compte de l'en-tête ajouté à chaque nouveau message M_i.

• Un second type d'optimisation est possible : router les messages suivant des routes disjointes (si c'est possible) au lieu d'un chemin, ceci afin de d'augmenter le parallélisme. Si on a m chemins arêtes-disjoints de longueur au plus d' supérieur ou égal à d, on peut découper le message initial de longueur L en p' blocs et router chaque bloc de longueur L/p' (supposée de longueur entière pour simplifier) indépendamment sur chacun des chemins. Les temps de transmission deviennent (pour le mode *wormhole* et *store-and-forward* respectivement) :

$$T_{\text{nœud à distance } d} = \alpha + d'\delta + \frac{L}{p'}\tau \quad \text{et} \quad T_{\text{nœud à distance } d} = d'\left(\beta + \frac{L/p'}{\tau}\right)$$

De plus, dans le mode *store-and-forward*, on peut pipe-liner indépendamment les p' blocs sur chacun des chemins pour obtenir un temps de $\left(\sqrt{(L/p')\tau} + \sqrt{(d'-1)\beta}\right)^2$.

1.4 Contraintes de communication

Il y a deux types contraintes : nœud ou lien.

Contrainte lien : *half/full-duplex*

Un lien entre x et y est *full-duplex* si x peut émettre vers y et recevoir de y en même temps. Si ce lien ne peut qu'émettre ou recevoir on parle de lien *half-duplex* (c'est le mode du Talki-Walki).

Contrainte nœud : modèle *k*-port

Si un nœud ne peut communiquer (émettre et/ou recevoir) que sur 1 seul lien à la fois, on parle du modèle 1-port. Plus généralement, si le nœud ne peut communiquer que sur au plus k de ses liens, on parle du modèle k-port. Et s'il peut communiquer simultanément sur tous ses liens, on parle du modèle ∞-port ou Δ-port (cette dernière notation étant relative au degré maximum du graphe souvent noté Δ.)

Notez qu'en *half-duplex*, un nœud x peut, dans le même temps, émettre vers y et recevoir de $z \neq y$ si le nœud x est dans le modèle k-port avec $k > 1$.

1.5 Schémas de communication usuels

Point-à-point Un sommet source envoie un message vers une unique destination. Voir le chapitre 3 consacré au Routage.

Diffusion (*broadcasting* ou *one-to-all*) Un sommet transmet son message vers tous les autres.

Concentration (*gathering* ou *all-to-one*) Un sommet récupère les messages de tous les autres sommets.

Échange total (*gossiping* ou *all-to-all*) Tous les sommets diffusent leur message.

Il existe d'autres schémas. Par exemple la distribution (*scattering*) : un sommet diffuse une information personnalisée à chacun des sommets. Il existe aussi la multi-distribution (*multicastering*) où la distribution s'effectue vers certain sommets seulement.

Bibliographie

[Rum94] J. d. Rumeur, *Communications dans les réseaux de processeurs*, Études et Recherches en Informatique (ERI), MASSON, 1994.

CHAPITRE 2

Communications globales

Sommaire

Mots clés et notions abordées dans ce chapitre :

- temps de diffusion, échange total, modèle téléphone
- arbre, cycle, clique, hypercube, graphes minimaux
- résultats de complexité et algorithmes d'approximation

Un complément d'information sur ce chapitre peut être trouvé dans [Rum94].

2.1 Généralités

Dans un premier temps nous nous intéressons au temps d'exécution d'un schéma de communication en mode *store-and-forward* temps constant (modèle le plus simple), puis nous parlerons du mode *circuit-switching* temps constant en dernière partie. Plusieurs paramètres permettent de mesurer ce temps : le nombre d'étapes de communications, le temps total de communication, le nombre total de messages échangés, le nombre de bits échangés, ...

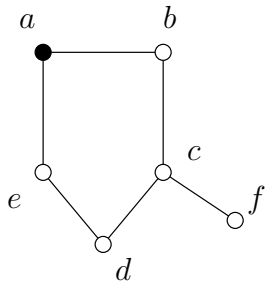
Lorsque la complexité choisie est le « nombre d'étapes » on supposera que chaque sommet peut exécuter, à une étape i donnée, une certaine communication élémentaire. Autrement dit plusieurs communications peuvent être réalisées en parallèles, et le réseau est dit *synchrone*. Si, au contraire, il n'y a aucune horloge globale (*asynchrone*), la complexité en temps la plus adaptée est le nombre de messages échangés, puisque dans le pire des cas chaque communication élémentaire peut être faite à des instants différents.

Dans tout ce chapitre nous nous placerons dans le cas synchrone. Ce modèle est le plus adapté pour analyser une machine parallèle dont la dimension physique est faible, il le sera moins pour analyser un réseau distribué de grande dimension. Lorsqu'on souhaite analyser le parallélisme (ou le degré de parallélisme) d'un algorithme voir d'un problème, le modèle synchrone est très pertinent.

Un algorithme réalisant un schéma de communication se présente donc sous la forme d'une suite d'étapes, chaque étape étant un ensemble de communications pouvant être faites en parallèles.

Étape	Faire en parallèle
1	$x \to y$, $w \leftrightarrow z$, ...
2	$y \to z$, ...
...	

Exemple 1. On considère le graphe suivant et la diffusion d'un message depuis a dans le modèle *store-and-forward*, 1-port et *half-duplex*.



Algo1

1. $a \to b$
2. $a \to e$, $b \to c$
3. $c \to d$
4. $c \to f$

Algo2

1. $a \to b$
2. $a \to e$, $b \to c$
3. $c \to f$, $e \to d$

FIGURE 2.1 – Diffusion à partir de a.

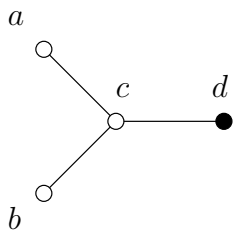
L'algo2 est optimal en nombre d'étapes puisque tout algorithme de diffusion nécessitera 3 étapes (au moins) pour informer f depuis a.

Exemple 2. concentration dans le graphe suivant vers le sommet d dans le modèle *store-and-forward*, 2-port et *half-duplex*.

Notez qu'en mode 1-port la concentration aurait nécessité 1 étape supplémentaire. En effet, après la 1ère étape de tout algorithme de concentration en mode 1-port, soit a soit b n'aura pas pu communiquer son message. Il lui faudra donc encore deux étapes pour atteindre d.

Remarque : en mode *store-and-forward*, tout algorithme de concentration peut être déduit d'un algorithme de diffusion et inversement) avec le même nombre d'étapes ! et ce quelle que soit la contrainte (*half-*, *full-duplex*, k-port, ...). Il suffit d'inverser le sens de chaque communication élémentaire.

Exemple 3. échange total dans le cycle à 4 sommets dans le modèle *store-and-forward*,

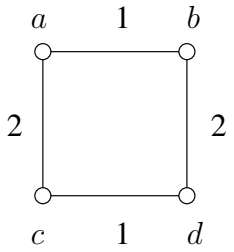


Algo.

1. $a \to c$, $b \to c$
2. $c \to d$

FIGURE 2.2 – Concentration vers d.

1-port et *full-duplex*.



Algo.

1. $a \leftrightarrow b$, $c \leftrightarrow d$
2. $a \leftrightarrow c$, $b \leftrightarrow d$

FIGURE 2.3 – Échange total.

Notez qu'en *half-duplex* (toujours 1-port) l'échange total nécessite a priori 4 étapes. Cependant en *half-duplex* 2-port 3 étapes suffisent :

1. $a \to b$, $b \to d$, $d \to c$, $c \to a$
2. $a \to b$, $b \to d$, $d \to c$, $c \to a$
3. $a \to b$, $b \to d$, $d \to c$, $c \to a$

Dans des exemples plus complexes, il peut devenir très vite difficile de vérifier qu'à la fin d'une succession d'étapes données le schéma de communications globales (ici un échange total) ait été correctement effectué. Pour cela on peut utiliser le formalisme suivant. Soit $M(x, i)$ l'ensemble des messages connus par x après la i^{e} étape de l'algorithme. Dans ce cas, à l'étape i une communication $x \to y$ se traduira par la relation : $M(y, i) = M(y, i-1) \cup M(x, i-1)$. Nous avons par exemple pour a (il faut faire la même chose sur tous les nœuds) : $M(a, 0) = \{a\}$, $M(a, 1) = M(a, 0) \cup M(c, 0) = \{a, c\}$, $M(a, 2) = M(a, 1) \cup M(c, 1) = \{a, c, d\}$, puis $M(a, 3) = M(a, 2) \cup M(c, 2) = \{a, b, c, d\}$.

2.2 Diffusion : *store-and-forward* et temps constant

Hypothèse dans toute la section : *store-and-forward* et temps constant. On s'intéresse donc au nombre d'étapes.

Étant donné un graphe $G = (V, E)$ et un sommet $x \in V$, on note $b_M(x, G)$ le nombre minimum d'étapes pour réaliser une diffusion dans G depuis x dans le modèle $M \in \{H_k, F_k\}$ où $k = 1, 2, \ldots, \infty$ et où H_k est une notation pour le modèle *half-duplex* k-port, et F_k signifie *full-duplex* k-port. On note également,

$$b_M(G) = \max_{x \in V(G)} b_M(x, G) \ .$$

Bien évidemment, le modèle H_k étant plus contraint que le modèle F_k, on a $b_{F_k}(x, G) \leqslant b_{H_k}(x, G)$. En fait on a :

Proposition 1 *Pour tout G, tout sommet x et tout entier k, $b_{F_k}(x, G) = b_{H_k}(x, G)$.*

Preuve. Considérons un algorithme A réalisant une diffusion d'un message M depuis x dans le modèle F_k et ayant un nombre d'étapes $b_{F_k}(x, G)$. Considérons une étape quelconque A et une communication du type $u \leftrightarrow v$. Avant cette communication les sommets impliqués (u ou v) connaissent ou ne connaissent pas M. Si u et v ne connaissent pas M, alors la communication $u \leftrightarrow v$ peut être supprimée sans modifier comportement de A, et sans augmenter son nombre d'étapes.

Si u (resp. v) est informé, alors on peut transformer $u \leftrightarrow v$ en communication du type $u \rightarrow v$ (resp. $u \rightarrow v$). De proche en proche, le nouvel algorithme ainsi obtenu fonctionne en *half-duplex* et simule, avec le même nombre d'étapes, l'algorithme A. □

Dans la suite nous noterons plus simplement $b_k(\cdot)$ au lieu de $b_{F_k}(\cdot)$ ou $b_{H_k}(\cdot)$.

On note $d_G(x, y)$ la distance entre x et y dans G, c'est-à-dire le nombre minimum d'arête de G d'un chemin connectant x à y. Le diamètre d'un graphe est la plus grande distance entre deux sommets du graphe. Dit autrement, si D est le diamètre de $G = (V, E)$, $D = \max_{x,y \in V} d_G(x, y)$.

Proposition 2 *Soit G un graphe avec trois sommets r, x, y tels que $d_G(r, x) = d_G(r, y) = t$. Alors $b_1(G) \geqslant t + 1$.*

Preuve. Il faut supposer qu'au temps t, x et y ont été informé. Dans ce cas on forme le chemin $P = x_t, x_{t-1}, \ldots, x_1, x_0$ de $x = x_t$ à $r = x_0$ tel que x_{i-1} est un voisin de x_i par lequel x_i a été informé. Par induction x_i a été informé au temps i, for $i \in \{0, \ldots, t\}$. De même pour y on forme un chemin $Q = y_D, \ldots, y_0$ de y à r avec la même construction. Les chemins P et Q s'intersectent en $w = x_{i-1} = y_{i-1}$ ($w = r$ est possible). Par hypothèse, w

ne peut avoir informé x_i et y_i au même temps (1-port) : contradiction. □

Exercice. Construire un exemple de graphe de diamètre D ayant t sommet à distance D d'un sommet r avec $b_1(G) \geqslant D-1+t$. En déduire qu'il existe des graphes à n sommets et de diamètre constant (indépendant de n) tels que $b_1(G)$ soit une fonction de n non constante. Solution : prendre un chemin d'extrémité r et de longueur $D-1$ au bout duquel sont connectés t sommets de degré 1. Il faut $D-1$ étapes pour informer l'autre extrémité du chemin, puis t étapes pour informer $x_1, \ldots, x_t$.

Exercice. Montrer que la généralisation de la proposition précédente à la contrainte k-port n'est pas possible en construisant un graphe de diamètre D, ayant $k+1$ sommets $r, x_1, \ldots x_k$ tels que $d(r, x_i) = d(r, x_{i+1}) = D$ pour tout $i < k$, et pourtant avec $b_k(G) = D$. Solution : prendre arbre de degré trois composé d'un chemin d'extrémités r et x_1, de longueur $D-1$, et d'où chaque sommet interne à distance $i > D/2$ de r par un chemin de longueur $D-i-1$, il faut $k < D/2$.

Proposition 3 *Soit C_n le cycle à n sommets. Alors $b_1(C_n) = \lceil n/2 \rceil$.*

Preuve. Le diamètre de C_n vaut $\lfloor n/2 \rfloor$. L'algorithme naturel montre que : 1) si $n = 2p$, $b_1(C_{2p}) \leqslant p$ ce qui est optimal à cause du diamètre ; 2) si $n = 2p+1$, $b_1(C_{2p+1}) \leqslant p+1$ ce qui est 1 de plus que le diamètre, ce qui est aussi optimal d'après la proposition 2. □

On remarque qu'en fait $b_1(x, C_n) = \lceil n/2 \rceil$ pour tout sommet x de C_n.

Proposition 4 *Soit T un arbre enraciné en r, de profondeur h et de degré maximum Δ. Alors $b_1(r, T) \leqslant (\Delta - 1)h + 1$.*

Preuve. On construit un algorithme glouton et on raisonne sur les sommets à distance i de r : à distance 1, le dernier sommet est informé au temps Δ ; à distance 2, le dernier sommet est informé au temps $\Delta + (\Delta - 1)$; ... à distance i, le dernier sommet est informé au temps $\Delta+(i-1)(\Delta-1)$. Donc $b_1(r, T) \leqslant \Delta+(\Delta-1)(h-1) = (\Delta-1)h+1$. □

Remarquons que la borne ci-dessus est atteinte pour un arbre Δ-aire complet (la racine à Δ fils). En effet, il existe un fils qui sera informé au temps Δ (et pas avant). Au niveau 1 le dernier fils informé le sera au temps $t(x) \geqslant \Delta$. Au niveau 2, il existe un fils x qui sera informé au temps $t(y) \geqslant t(x) + \Delta - 1$. Au total, il faut $\Delta + (h-1)(\Delta-1)$ pour informer toutes les feuilles.

Exercice. Pour tout arbre T, on définit $\rho(T)$ son rayon, c'est-à-dire la hauteur minimum de T minimisée sur toutes les racines possibles de T. Montrer que $b_1(T) \leqslant \Delta\rho(T)$.

Solution : Soit x un sommet de T tel que $b_1(x, T) = b_1(T)$. On enracine T en r qui est de profondeur $\rho(T)$. Si $x = r$, alors $b_1(T) = b_1(r, T) \leqslant (\Delta - 1)\rho(T) + 1$ donc au plus

$\rho\Delta$ ($\rho(T) \geqslant 1$). Supposons $x \neq r$. Le temps de diffusion depuis x est au plus le temps nécessaire à x pour informé r plus le temps nécessaire r pour diffuser dans tout T, excepté dans la branche contenant x. D'où $b_1(T) \leqslant d(x,r) + b_1(r,T) - 1$. Comme $d(x,r) \leqslant \rho(T)$ et $b_1(r,T) \leqslant (\Delta-1)\rho(T)+1$, on a $b_1(T) \leqslant \Delta\rho$.

Proposition 5 *Soit G un graphe à n sommets. Alors $b_1(G) \geqslant \log_2 n$.*

Preuve. Soit $N(t)$ le nombre maximum de sommets que l'on peut informer en t étapes. $N(0) = 1$, $N(1) = 2$, ... $N(t) \leqslant 2N(t-1)$. Donc $N(t) \leqslant 2^t$. Il est possible de diffuser en temps $b_1(G)$ dans G donc $n \leqslant N(b_1(G))$, ce qui implique $b_1(G) \geqslant \log_2 n$. □

En fait la même preuve indique que $b_1(x,G) \geqslant \log_2 n$ pour tout sommet x de G. Notons que cela donne une borne optimale pour C_6.

Exercice. Montrer que pour tout graphe à n sommets, $b_k(G) \geqslant \log_{k+1} n$.

Problème : construire un arbre B_t de racine r_t et à 2^t sommets tel que $b_1(r_t, B_t) = t$?

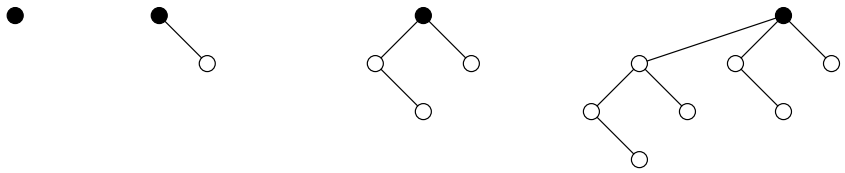


FIGURE 2.4 – Arbres binomiaux : B_0, B_1, B_2 et B_3.

De manière récursive, on a B_t qui est l'union de deux B_{t-1} dont les racines sont connectés par une arête. On choisi la nouvelle racine r_t comme l'une des deux anciennes racines r_{t-1}. En effet, au temps 1, r_t envoie son information à la racine r_{t-1} non informé. Après 1 étape, les deux anciennes racines sont informées. Ce qui montre la récurrence $b_1(r_t, B_t) \leqslant 1 + b_1(r_{t-1}, B_{t-1}) \leqslant 2^t$. La proposition 5 permet de conclure que $b_1(r_t, B_t) = t$.

Une autre façon de construire B_t est d'ajouter une feuille à tout sommet de B_{t-1}, la racine restant la même.

Les arbres B_t de racine r_t sont appelés arbres *binomiaux*. Leur nom vient du fait que le nombre de sommet à distance i de r_t est exactement le binomial $\binom{t}{i}$, le i-ème coefficient du développement de $(x+y)^t$. Avec $x = 1$ et $y = 1$ on a d'ailleurs $2^t = (1+1)^t = \sum_{i=0}^{t} \binom{t}{i}$.

Lorsque l'on considère la contrainte k-port, on peut facilement généraliser les arbres binomiaux aux arbres que l'on pourrait appeler $(k+1)$-nomiaux.

Soit H_t l'hypercube de dimension t. Il s'agit d'un graphe dont l'ensemble des sommets sont les mots binaires de longueur t. Deux sommets sont adjacents si et seulement si leur mot diffère d'un bit exactement (cf. figure 2.5).

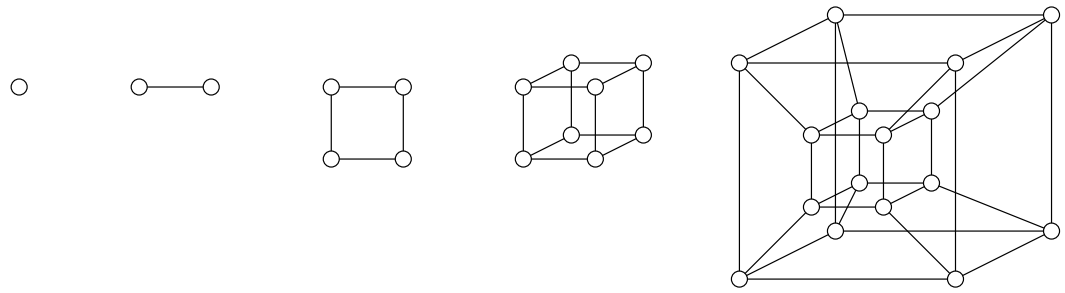


FIGURE 2.5 – Hypercube de dimensions 0, 1, 2, 3 et 4.

Il est a remarquer que l'hypercube H_t est un réseau réalisant un bon compromis entre le degré maximum et le diamètre (tout le monde est proche sans pour autant avoir beaucoup d'arêtes). Ainsi en dimension 4, H_4 qui possède $2^4 = 16$ sommets a un degré de $t = 4$ qui est aussi égale à son diamètre. Pour comparaison, une grille à $n = 16 = 4 \times 4$ sommets à un degré maximum aussi de 4 mais un diamètre plus grand, ici de $6 = (4-1) \times 2$.

Le diamètre d'une machine parallèle peut être un paramètre important si l'on réalise qu'un processeur cadencé, disons, à 1 GHz exécute 10^9 instructions/s, c'est-à-dire que la durée d'une instruction est de 10^{-9}s = 1 ns (1 milliardième de seconde). Et que par conséquent, dans ce laps de temps, une information ne peut parcourir que 300 000 km/s $\times 10^{-9}$ s $= 3 \times 10^8$ m/s $\times 10^{-9}$ s $= 0,3$ m $= 30$ cm seulement ! Vu sous cet angle, les communications peuvent être un frein aux calculs temporellement dépendants. Il faut donc les optimiser ! En fait, dans le cuivre les signaux électromagnétiques se déplacent seulement à 2/3 de la vitesse de la lumière – la borne tombe à 20 cm, et les électrons à 10 000 km/s seulement !

Proposition 6 *Pour tout entier $t \geqslant 0$, $b_1(H_t) = t$.*

Preuve. Soit $x \in \{0,1\}^t$ un sommet de H_t. On va prouver que $b_1(x, H_t) \leqslant t$ (par la proposition 5 on sait déjà que $b_1(H_t) \geqslant t$). Pour cela il suffit de montrer que H_t possède un arbre binomial B_t de racine x. Par induction. Soit $t > 0$, et posons $x = bw$ avec $b \in \{0,1\}$ et $w \in \{0,1\}^{t-1}$. L'ensemble des sommets $S_0 = \{0u \mid u \in \{0,1\}^{t-1}\}$ et $S_1 = \{1u \mid u \in \{0,1\}^{t-1}\}$ induisent dans H_t un sous-hypercube de dimension $t-1$. Ces ensembles sont disjoints. Par induction, on peut enraciner dans S_0 un arbre B_{t-1} en $0w$, et dans S_1 un arbre B_{t-1} enraciné en $1w$. L'arête entre $0w$ et $1w$ complète la construction de B_t, arbre que l'on peut enraciner en x puisque $x = 0w$ ou $x = 1w$. □

Un graphe G à n sommets est « minimal pour la diffusion » si $b_1(G) = \lceil \log_2 n \rceil$. Montrer que tout sommet x d'un graphe minimal pour la diffusion à $n = 2^t$ sommets doit satisfaire $\deg(x) \geqslant t$. En effet, soit x un sommet de degré $d < t$. Considérons une diffusion à partir de x. Pour informer 2^t sommets, x doit diffuser à chaque étape. Il ne peut le faire si $d < t$. Contradiction.

Par conséquent, H_t est un graphe minimal pour la diffusion, puisque tout sommet de H_t est de degré t. Il existe des graphes différents de H_t qui sont minimaux pour la diffusion (voir figure pour un exemple avec $n = 8$.)

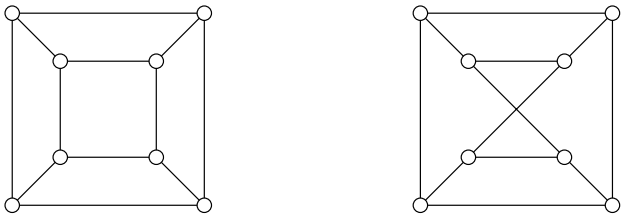


FIGURE 2.6 – Hypercube et Twisted hypercube de dimension 2.

Soit $B(n)$ le plus petit nombre d'arêtes d'un graphe minimal pour la diffusion ayant n sommets. D'après ce qui a été dit précédemment, $B(2^t) = t \cdot 2^{t-1}$ (1/2 somme des degrés de H_t). Les valeurs de $B(n)$ ne sont connues que pour certaines valeurs de n (typiquement lorsque $n = 2^t$ ou $n = 2^t \pm 1$). En fait, il a été démontré (Grigni, Peleg [GP91]) que $B(n) = \Theta(L(n-1) \cdot n)$ où $L(x)$ représente de nombre de bits de poids forts consécutifs dans l'écriture binaire de x. Par exemple, si $x = 11101011000$, alors $L(x) = 3$.

Le résultat a été étendu à $B_m(n)$, le nombre minimum d'arêtes d'un graphe à n sommets pour effectuer une diffusion en temps optimal plus m unités de temps [APR10] (donc $B(n) = B_0(n)$).

Proposition 7 *Soit K_n le graphe complet à n sommets, c'est-à-dire le graphe possédant toutes les arêtes possibles. Alors $b_1(K_n) = \lceil \log_2 n \rceil$.*

Preuve. Remarque : le résultat est évident pour $n = 2^t$, puisque K_n contient H_t. On fait une induction similaire à celle utilisée pour H_t. Supposons que $n \geqslant 2$ (pour $n < 2$ c'est trivialement vraie). Soit x un sommet quelconque de K_n. On coupe K_n en deux sous-graphes complets de tailles strictement plus petites : S_x qui contient x et de cardinalité $\lceil n/2 \rceil$ (qui est $< n$ pour $n \geqslant 2$) et $S_{\bar{x}}$ de cardinalité $n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$. Soit $\bar{x}$ un sommet quelconque de $S_{\bar{x}}$. Comme x et $\bar{x}$ sont adjacents, il suffit à la première étape de faire : $x \to \bar{x}$, puis d'exécuter en x et $\bar{x}$ une diffusion de temps respectifs $b_1(S_x)$ et $b_1(S_{\bar{x}})$. Par induction,

$$b_1(x, K_n) \leqslant 1 + \max\left\{b_1(\bar{x}, K_{\lceil n/2 \rceil}), b_1(x, K_{\lfloor n/2 \rfloor})\right\} \leqslant 1 + \lceil \log_2 \lceil n/2 \rceil \rceil \ .$$

Il suffit alors de vérifier que $1 + \lceil \log_2 \lceil n/2 \rceil \rceil = \lceil \log_2 n \rceil$ pour tout $n \geqslant 2$. (La preuve donnée dans l'exercice suivant pour le cas général est presque plus convaincantes)

Si $n = 2p$, alors

$$1 + \lceil \log_2 \lceil n/2 \rceil \rceil = 1 + \lceil \log_2 p \rceil = \lceil \log_2 (2p) \rceil = \lceil \log_2 n \rceil \ .$$

Si $n = 2p+1$, alors $\lceil \log_2 n \rceil = \lceil \log_2(2p+1) \rceil = \lceil \log_2(2p+2) \rceil$ pour tout $p \geqslant 1$ (que l'on vérifie pour les deux cas : $2p+2 = 2^k$ et $2p+2 \neq 2^k$ avec k maximum), et donc

$$1 + \lceil \log_2 \lceil n/2 \rceil \rceil \;=\; 1 + \lceil \log_2(p+1) \rceil \;=\; \lceil \log_2(2p+2) \rceil \;=\; \lceil \log_2 n \rceil \;.$$

□

Exercice. Montrer que pour la roue R_n (graphe composée d'un cycle à $n-1$ sommets et d'un sommet central connecté à tout ceux du cycle), on a $b_1(R_n) = \Theta(\sqrt{n}\,)$. (Lemme 4.1 du papier de Kortsarz,Peleg [KP95]). Généraliser à tout k en montrant que $b_k(R_n) = \Theta(\sqrt{n/k}\,)$.

Solution : Soit c le sommet de central, et soit x_0 un sommet tel que $b_k(x_0, R_n) = b_k(R_n)$. On a $b_k(c, R_n) \leqslant 1 + b_k(R_n)$, car pour diffuser depuis c, il suffit de faire $c \to x_0$ puis d'appliquer un algorithme optimal de diffusion depuis c en temps $b_k(c, R_n)$. D'un autre coté, on a toujours $b_k(R_n) \leqslant b_k(c, R_n)$, il découle que $b_k(R_n) \leqslant b_k(c, R_n) \leqslant b_1(R_n) + 1$. Par conséquent il suffit de montrer que $b_k(c, R_n) = \Theta(\sqrt{n}\,)$.

Borne sup : On propose un schéma qui fait (on commence par décrire une solution pour $k = 1$) : 1) c informe $t \geqslant 1$ sommets uniformément répartis sur le cycle, et puis 2) chaque sommet du cycle ainsi informé diffuse en parallèle sur son segment d'au plus $\frac{1}{2}\lceil (n-1)/t \rceil$ sommets (on peut faire un peu mieux avec des segments de longueur différentes). Cela montre que pour tout $t \geqslant 1$, $b_1(c, R_n) \leqslant t + \frac{1}{2}\lceil (n-1)/t \rceil - 1 \leqslant t + \frac{1}{2}(n-1)/t < \sqrt{2n} = O(\sqrt{n}\,)$ (le -1 provient du fait qu'un sommet du segment est déjà informé). Pour la généralisation, c informe d'abord $O(\sqrt{kn}\,)$ voisins en temps $O(\sqrt{kn}/k) = O(\sqrt{n/k}\,)$. Ensuite, les segments de taille au plus $O(n/\sqrt{kn}\,) = O(\sqrt{n/k}\,)$ sont informés en temps $O(\sqrt{n/k}\,)$ comme dans un chemin (on peut faire 2 fois plus vite si $k \geqslant 2$).

Borne inférieure : Soit t le nombre d'étapes d'un algorithme A de diffusion depuis c, et soit I l'ensemble des sommets qui ont été informé directement par c avec l'algorithme A. Formellement, $I = \{x \mid c \to x \in A\}$. Évidemment $t \geqslant |I|$. Il existe alors un sommet y du cycle tel que $d(y, I) \geqslant \frac{1}{2}(n-1)/|I|$. Pour informer y, il faut donc un temps d'au moins $d(y, I)$ étapes. Le nombre d'étapes pour A est donc $t \geqslant \max\{|I|, \frac{1}{2}(n-1)/|I|\}$, ce qui implique $t \geqslant \sqrt{(n-1)/2} = \Omega(\sqrt{n}\,)$. Pour la généralisation, on remarque qu'on a maintenant $|I| \leqslant tk$, les autres arguments ne changeant pas. Il reste $t \geqslant (n-1)/(2tk)$, ce qui implique $t \geqslant \sqrt{(n-1)/(2k)} = \Omega(\sqrt{n/k}\,)$.

Exercice. Montrer que pour tout $k > 0$, $b_k(K_n) = \lceil \log_{k+1} n \rceil$.

Solution : On partage K_n en cliques $C_1, \ldots, C_p$ de taille $q = \lceil n/(k+1) \rceil$, sauf pour la clique C_p qui peut avoir moins de q sommets. Le sommet x, celui tel que $b_k(x, K_n) = b_k(K_n)$, diffuse son message en choisissant un sommet pour chacune des cliques $C_2, \ldots, C_p$, en supposant que $x \in C_1$. Le nombre de cliques est $p = \lceil n/q \rceil \leqslant k+1$, car

$$\left\lceil \frac{n}{q} \right\rceil = \left\lceil \frac{n}{\lceil \frac{n}{k+1} \rceil} \right\rceil \;\leqslant\; \left\lceil \frac{n}{\frac{n}{k+1}} \right\rceil \;=\; \lceil k+1 \rceil \;=\; k+1$$

Donc informer les $p-1 \leqslant k$ cliques $C_2, \ldots, C_p$ peut être fait par x en une seule étape. Après cette étape, toutes les cliques ont un sommet informé. Il reste à diffuser en parallèle dans des cliques de tailles au plus $q = \lceil n/(k+1) \rceil$. ...

Il faut montrer que $1 + \lceil \log_{k+1} \lceil n/(k+1) \rceil \rceil = \lceil \log_{k+1} n \rceil$. Il faut poser $n = a \cdot (k+1)^p$ avec $p \in \mathbb{N}$ et $a \in \mathbb{R}$ tel que $1 \leqslant a < k+1$.

Si $a = 1$, c'est-à-dire $n = (k+1)^p$, alors $1 + \lceil \log_{k+1} \lceil n/(k+1) \rceil \rceil = p = \lceil \log_{k+1} n \rceil$. Donc supposons $1 < a < k+1$. Observons de suite que cela implique $\lceil \log_{k+1} a \rceil = 1$. D'une part on a

$$\lceil \log_{k+1} n \rceil \;=\; \lceil \log_{k+1}(k+1)^p + \log_{k+1} a \rceil \;=\; p+1 \;.$$

D'un autre coté, $\lceil n/(k+1) \rceil = \lceil a \cdot (k+1)^{p-1} \rceil$. Donc,

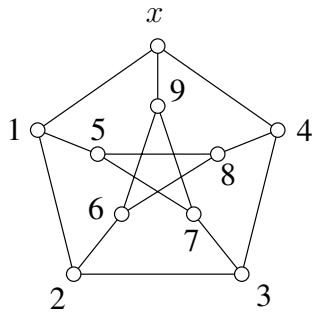
$$\begin{array}{rcl} a \cdot (k+1)^{p-1} \;\leqslant\; \lceil n/(k+1) \rceil & < & a \cdot (k+1)^{p-1} + 1 \\ \Leftrightarrow \quad \lceil \log_{k+1} a \rceil + p - 1 \;\leqslant\; \lceil \log_{k+1} \lceil n/(k+1) \rceil \rceil & < & \lceil \log_{k+1} a \rceil + p - 1 + 1 \\ \Leftrightarrow \quad p \;\leqslant\; \lceil \log_{k+1} \lceil n/(k+1) \rceil \rceil & < & p+1 \end{array}$$

Donc $\lceil \log_{k+1} \lceil n/(k+1) \rceil \rceil = p$. Par conséquent, on a établit que pour tout entier n,

$$1 + \lceil \log_{k+1} \lceil n/(k+1) \rceil \rceil \;=\; p+1 \;=\; \lceil \log_{k+1} n \rceil \;.$$

Exercice. Montrer que $b_\infty(G) = D$ pour tout graphe de diamètre D. Solution : on utilise l'algorithme glouton.

Exercice. Soit P le graphe de Petersen représenté ci-dessous. Montrer que $b_1(x, P) = 4$; $b_2(x, P) = 3$; $b_3(x, P) = 2$.



2.3 Échange total : *store-and-forward* et temps constant

Étant donné un graphe $G = (V, E)$ on note $g_M(G)$ le nombre minimum d'étapes pour réaliser un échange total dans G dans le modèle $M \in \{H_k, F_k\}$ où $k = 1, 2, \ldots, \infty$ et où H_k est une notation pour le modèle *half-duplex* k-port, et F_k signifie *full-duplex* k-port.

Exercice. Montrer que pour tout G et tout $k \geqslant 1$, $b_k(G) \leqslant g_{F_k}(G) \leqslant g_{H_k}(G) \leqslant 2b_k(G)$.

Proposition 8 *Pour tout entier $t \geqslant 0$, $g_{F_1}(H_t) = t$.*

Preuve. On coupe H_t en deux sous-hypercubes disjoints de dimensions $t-1$. On remarque que si l'on sait réaliser un échange total dans H_{t-1} en temps, disons τ, alors pour obtenir un échange total dans H_t il suffit d'effectuer l'échange *full-duplex* suivant : $0w \leftrightarrow 1w$ pour tout $w \in \{0,1\}^{t-1}$. Par induction on vérifie qu'on a bien un échange total dans H_t et que le temps est $\tau + 1 = t$. □

Exercice. Montrer que pour tout $t > 0$, $g_{H_1}(H_t) > g_{F_1}(H_t)$. Solution : ?

Problème de ouvert : $g_{H_1}(H_t)$?
Il est connu que $g_{H_1}(H_t) \geqslant g_{H_1}(K_n) \geqslant 2 + \log_\rho(n/2) \sim 1.44t$ où $\rho = (1+\sqrt{5})/2 \approx 1.618...$ est le nombre d'or. Ce résultat se démontre en analysant la norme d'une matrice $(m_{i,j})$ indiquant, à un instant donné si le sommet i a reçu l'information du sommet j.

Krumme [Kru92] a montré que $g_{H_1}(H_t) \leqslant 1.88t$. Donc $1.44t \leqslant g_{H_1}(H_t) \leqslant 1.88t$. De manière générale, il a été montrer que $g_{H_k}(K_n) \geqslant \log_{\lambda(k)} n$ où $\lambda(k) = (k + \sqrt{k^2+4})/2$, $k \geqslant 1$.

2.4 Diffusion dans le modèle *circuit-switching* et temps constant

C'est le modèle du téléphone (disons téléphone filère). On suppose qu'un appel prend un temps 1 (temps constant) quelle que soit la distance entre la source et la destination. Les sommets intermédiaires ne participent pas à la communication, ils ne peuvent recopier le message transitant sur leur chemin. (On suppose donc qu'il n'y a pas d'écoute téléphonique !) Des appels peuvent être réalisés en parallèle seulement si les chemins n'ont pas de sommet en commun (on parle de chemins « sommet-disjoints »).

Étant donné un graphe $G = (V, E)$ et un sommet $x \in V$, on note $c_k(x, G)$ — c pour *call* — le nombre minimum d'étapes pour réaliser une diffusion dans G depuis x dans le modèle k-port, où en une étape une communication peut avoir lieu entre x et au plus à un ensemble de k sommets, à condition que la communication puisse s'effectuer le long de chemins sommets-disjoints. Comme précédemment on note,

$$c_k(G) = \max_{x \in V(G)} c_k(x, G)$$

en remarquant que, pour la diffusion, il n'y a pas de différence suivant la contrainte *half-duplex* ou *full-duplex* (cela ne sert à rien de recevoir un message d'un sommet que l'on informe, donc supposé non-informé).

Remarque :

- On a $c_k(x, G) \leqslant b_k(x, G)$, et aussi $c_1(G) \geqslant \log_2 |V(G)|$.
- Si l'on considère une variante c'_1 du temps de diffusion c_1 en autorisant des appels parallèles selon des chemins arêtes-disjoints au lieu de sommets-disjoints (deux chemins peuvent donc se croiser), alors pour tout graphe G à n sommets, $c'_1(G) = \lceil \log_2 n \rceil$ (Farley [Far80]), ce qui rend le problème moins intéressant.

Proposition 9 *Soit T un arbre de n sommets et de degré maximum Δ. Alors $c_1(T) \leqslant \Delta \lfloor \log_2 n \rfloor$.*

Évidemment, pour tout sommet r de T, $c_1(r, T) \leqslant (\Delta - 1)h + 1$, où h est la hauteur de r. On peut se convaincre qu'il est possible d'aller plus vite que cette borne, surtout quand h est grand devant $\log_2 n$, en considérant un chemin. (cf. figure 2.7).

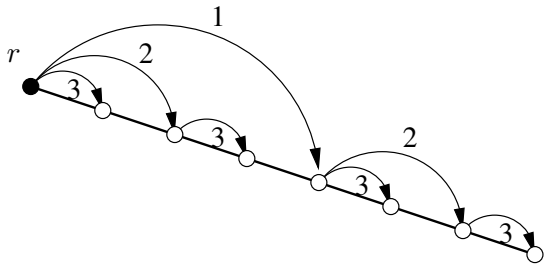


FIGURE 2.7 – Diffusion en 3 étapes dans un chemin à 8 sommets (pourquoi le nombre d'étapes est optimal ?).

Preuve. Montrons que $c_1(r, T) \leqslant \Delta \lfloor \log_2 n \rfloor$ pour toute racine r de T. On suppose donc que T est enraciné en r, et on note T_x le sous-arbre de T composé de tous les descendants de x (x compris).

La stratégie va consister à envoyer le message de la source à un sommet « central ». Puis à faire une récurrence comme on le ferait pour une grille : on coupe en deux sous-grilles et on informe la 2e partie. Malheureusement, quand on coupe un arbre en deux, cela peut donner plein de petits bouts ... d'où le facteur Δ devant le $\log_2 n$.

Un sommet x de T est un *centroïde* si $|T_x| \geqslant n/2$ et $|T_u| < n/2$ pour tout descendant u de x dans T. En parcourant les sommets de T depuis r, il est facile de voir que tout arbre T possède un centroïde. En effet, si le sommet x courant est tel que $|T_x| \geqslant n/2$ (initialement $x = r$ on a $T_x = T$ donc $|T_x| = n$) et de plus n'est pas un centroïde alors c'est que x possède un fils u tel que $|T_u| \geqslant n/2$. (On remarque que ce fils est unique sinon T aurait au moins $n + 1$ sommets). On choisit alors u comme nouveau sommet courant en posant $x = u$. La taille de T_x diminuant à chaque pas (tout en restant $\geqslant n/2$), l'algorithme converge vers un centroïde prouvant son existence.

(En fait on peut montrer aussi, dans le cas d'un arbre enraciné, que le centroïde est unique. En effet si x et y sont deux centroïde de T alors on ne peut pas avoir x descendant de y ou le contraire. Donc il existe un sommet z qui est le plus petit ancêtre commun à x et à y, sommet qui n'est ni dans T_x ni dans T_y. Cela implique que l'arbre contient au moins $|T_x|+|T_y|+1 > n$ sommet, une contradiction. Ce n'est pas valable dans le cas non-enraciné, puisqu'il peut avoir une arête entre x et y sans le sommet z entre !)

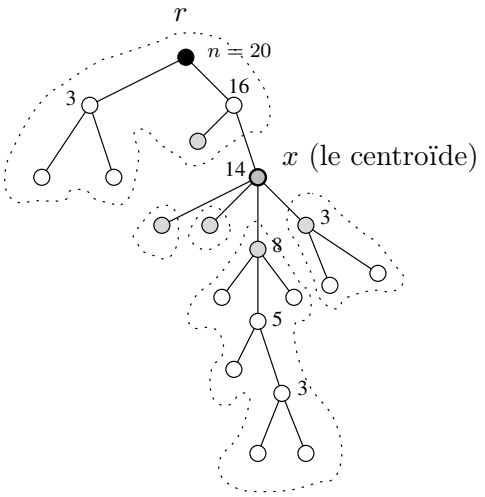


FIGURE 2.8 – Exemple d'arbre pour la diffusion en $\Delta \lfloor \log_2 n \rfloor$.

Le protocole consiste donc à faire $r \to x$, puis à faire $x \to f_1, \ldots, x \to f_k$ où f_i est le i^{e} fils de x, $i \leqslant \Delta - 1$. (En fait x peut très bien transmettre le message à n'importe quel sommet de T_{f_i}). Si $x = r$, alors on diffuse simplement aux fils de r, qui peuvent être au nombre de Δ. Ensuite reste à diffuser récursivement et en parallèle dans les arbres $T_{f_1}, \ldots, T_{f_k}$ et aussi dans l'arbre $T \setminus T_x$.

Soit $c(n)$ le nombre d'étapes suivit par le protocole. Par construction, $|T_x| \geqslant n/2$, donc $|T \setminus T_x| \leqslant n/2$. Il suit que tous les arbres dans lesquels il reste à diffuser en parallèle sont de taille $\leqslant n/2$. On a donc, en remarquant que $\lfloor \lfloor n/2 \rfloor /2 \rfloor = \lfloor n/4 \rfloor$ (enlever 2 fois 1 bit ou 2 bits d'un coup à l'écriture binaire de n fait $\lfloor n/4 \rfloor$ dans les deux cas)

$$c(n) \;\leqslant\; \Delta + c\left(\lfloor n/2 \rfloor\right) \;\leqslant\; 2\Delta + c(\lfloor n/4 \rfloor) \;\leqslant\; \ldots \;\leqslant\; i\Delta + c\left(\left\lfloor n/2^i \right\rfloor\right)$$

avec $c(1) = 0$. Notons que $2^{\lfloor \log_2 n \rfloor} > 2^{(\log_2 n)-1} = n/2$. Donc $n/2^{\lfloor \log_2 n \rfloor} < 2$, d'où $\left\lfloor n/2^{\lfloor \log_2 n \rfloor} \right\rfloor \leqslant 1$. Donc pour tout $i \geqslant \lfloor \log_2 n \rfloor$, $c(n) \leqslant i\Delta$. D'où $c(n) \leqslant \Delta \lfloor \log_2 n \rfloor$ en prenant $i = \lfloor \log_2 n \rfloor$. □

Dans la preuve précédente, remarquons que x peut transmettre à ses sous-arbres $T_{f_1}, \ldots, T_{f_k}$ dans un ordre quelconque. Par exemple il peut le faire dans l'ordre décroissant de $|T_{f_i}|$, f_i pouvant alors commencer sa diffusion dans T_{f_i} immédiatement après. Notons

$A_1, A_2, \ldots, A_\delta$ les arbres de la forêt $T \setminus \{x\}$, donc les arbres obtenus en supprimant x de T. Notons que $\delta \leqslant \Delta$. On suppose que ces arbres sont numérotés de telle sorte que $|A_1| \geqslant \cdots \geqslant |A_\delta|$. Si $r \neq x$, un des sous-arbres A_i doit correspondre à $T \setminus T_x$. On le note A_t. Par commodité, on pose $t = 0$ si $r = x$. Pour tout $i \in \{1, \ldots, \delta\}$, $i \neq t$, on note x_i un sommet quelconque de A_i.

Le nouveau protocole considéré est le suivant (bien sûr, si $t = 0$, c'est-à-dire $r = x$, seule les dernières étapes sont réalisées) :

étape	communication
1	$r \to x_1$, puis x_1 diffuse dans A_1
...	...
$t-1$	$r \to x_{t-1}$, puis x_{t-1} diffuse dans A_{t-1}
t	$r \to x$, puis r diffuse dans A_t
$t+1$	$x \to x_{t+1}$, puis x_{t+1} diffuse dans A_{t+1}
...	...
δ	$x \to x_\delta$, puis x_δ diffuse dans A_δ

En gros, dès que x_i est informé, il diffuse dans son sous-arbre A_i. La racine r (si $\neq x$) diffuse dans son sous-arbre A_t juste après l'étape t.

D'après ce protocole, tous les sommets du sous-arbre :

- A_1 sont informés après l'étape $1 + c(|A_1|)$
- A_2 sont informés après l'étape $2 + c(|A_2|)$
- ...
- A_t sont informés après l'étape $t + c(|A_t|)$
- ...
- A_δ sont informés après l'étape $\delta + c(|A_\delta|)$

Donc tous les sommets de T sont informés après un temps :

$$\max_{1 \leqslant i \leqslant \delta} \{i + c(|A_i|)\}$$

Observons que $|A_1| \leqslant n/2$ et que pour tout $i > 1$, $|A_i| \leqslant n/i$. Autrement l'équation de récurrence sur $c(n)$ se modifie légèrement en :

$$c(n) \ \leqslant \ \max_{2 \leqslant i \leqslant \Delta} \{i + c(n/i)\} \ .$$

(L'indice i commence à 2 car pour A_1 on a le terme $1 + c(|A_1|) \leqslant 1 + c(n/2) < 2 + c(n/2)$

ce qui est la même contrainte que pour A_2.) En développant $c(n/i)$ on obtient :

$$\begin{aligned} c(n) &\leqslant \max_{2\leqslant i_1\leqslant\Delta}\left\{i_1+\max_{2\leqslant i_2\leqslant\Delta}\{i_2+c(n/(i_1i_2))\}\right\} \\ &\leqslant \max_{2\leqslant i_1,i_2\leqslant\Delta}\{i_1+i_2+c(n/(i_1i_2))\} \\ &\leqslant \ldots \\ &\leqslant \max_{2\leqslant i_1,\ldots,i_s\leqslant\Delta}\left\{\sum_{j=1}^{s} i_j + c\left(n/\prod_{j=1}^{s} i_j\right)\right\} . \end{aligned}$$

Supposons que s est le plus petit entier tel que $\prod_{j=1}^{s} i_j \geqslant n$. En notant $P=\prod_{j=1}^{s} i_j$, on a alors que $c(n/P)=0$. Pour simplifier, supposons que les entiers i_j sont rangés dans le même ordre que leur indices : $i_1\leqslant\cdots\leqslant i_s$. On a donc :

$$c(n) \leqslant \max_{2\leqslant i_1\leqslant\cdots\leqslant i_s\leqslant\Delta}\sum_{j=1}^{s} i_j .$$

[A FINIR] ...

Exercice. Extension au modèle k-port. Pour tout $k>0$, $c_k(T)\leqslant\Delta\lfloor\log_{k+1} n\rfloor$. Solution : ?

2.5 Complexité des communications globales

Instance : un graphe G et un entier B.
Question : $b_1(G)\leqslant B$?

Le problème ci-dessus est NP-complet (Farley [Far79]). Il reste NP-complet même si l'on restreint G à être planaire et de degré borné par 3 ou plus, ou si le graphe a un degré borné par 3 ou plus et a un diamètre logarithmique, ou encore si l'on fixe B à 3 ou plus (Jakoby, Reischuk, Schindelhauer [JRS98]). Le problème général ne peut être approximé à un facteur moins que $3-\varepsilon$ pour tout $\varepsilon>0$, sauf si P = NP (Elkin,Kortsarz [EK02]).

Meilleurs algorithmes polynomiaux d'approximation connus pour le calcul de $b_1(G)$:

- $b_1(G)+O(\sqrt{n}\,)$ (Kortsarz, Peleg [KP95]),
- $O(b_1(G)\cdot\log n/\log\log n)$ (Elkin,Kortsarz [EK03]), et
- $2\Delta b_1(G)$, si Δ est de degré maximum du graphe (Ravi [Rav94]).

Par une méthode du type diviser-pour-reigner (ou par programmation dynamique), on peut montrer que le calcul de $b_1(G)$ est linéaire si G est un arbre (Slater, Cockayne, Hedetniemi [SCH81], Proskurowski [Pro81] et Scheuerman, Wu [SW84]).

Instance : un graphe G et un entier C.
Question : $c_1(G)\leqslant C$?

Le problème ci-dessus est NP-complet [FHM+94].

Meilleurs algorithmes polynomiaux d'approximation connus pour le calcul de $c_1(G)$:

- $O(c_1(G) \cdot \log n / \log\log n)$ (Kortsarz, Peleg [KP95]), et
- $O(c_1(G) \cdot \log n / \log c_1(G))$ (Fraigniaud [Fra01]).

Pour les arbres, la complexité du calcul de $c_1(G)$ n'est pas connue. On connaît simplement un algorithme polynomial garantissant un nombre d'appels $\leqslant 3c_1(G)$ (Fraigniaud [Fra01]. Pour la preuve voir [Fra00, th.3 p.9]).

Problème ouvert : approximation constant en temps polynomial pour le calcul de $O(b_1(G))$ et de $O(c_1(G))$ pour G quelconque ?

Rappels sur P vs. NP. On appelle **P** la classe des problèmes que l'on peut résoudre en temps Polynômial sur une machine de Turing, c'est-à-dire la classe dont il existe un algorithme dont le temps d'exécution est au plus n^k où k est une constante et où n est la taille de l'entrée (exprimée en bits ou en mots mémoire). La classe **NP** est l'ensemble des problèmes que l'on peut résoudre en temps Polynômial sur une machine de Turing Non déterministe. Clairement P $\subseteq$ NP, certains problèmes de NP (les problèmes NP-complets) n'ont pas encore été résolu en temps polynômial. D'où la question à 1 million $: est-ce que P = NP ?

Maintenant un problème $\Pi \in$ NP-complet si tout problème de NP peut se réduire polynomialement à Π. Exemple de réduction : le problème de concentration se réduit (en temps linéraires même) au problème de la diffusion. Le problème 3-SAT est un problème NP-complet important car beaucoup de problèmes de NP ont une réduction simple à lui. De manière générale tous les problèmes de NP se réduisent polynomilement à tout problème NP-complet. 3-SAT est le problème de la satisfiabilité d'une expression booléenne B exprimée sous forme de facteurs de 3 variables (conjonction de disjonctions). Par exemple : $B = (x \vee y \vee \neg z) \wedge (u \vee v \vee z) \wedge (\neg x \vee z \vee \neg v)$. Ici n est le nombre de facteurs (ou clauses), le nombre totale de variables dans l'expression étant bien évidemment $\leqslant 3n$.

Lorsqu'un problème est d'optimisation est NP-complet, on cherche des algorithmes d'approximation, c'est-à-dire une heuristique dont le résultat est proche de la solution mais qui prend un temps polynômial.

Bibliographie

[APR10] A. Averbuch, I. Peeri, and Y. Roditty, *Improved upper bound on m-time-relaxed k-broadcasting communication networks*, 2010.

[EK02] M. Elkin and G. Kortsarz, *Combinatorial logarithmic approximation algorithm for directed telephone broadcast problem*, in 34th Annual ACM Sympo-

sium on Theory of Computing (STOC), ACM Press, May 2002, pp. 438–447. DOI : 10.1145/509907.509972.

[EK03] M. Elkin and G. Kortsarz, *Sublogarithmic approximation for telephone multicast : Path out of jungle*, in 14th Symposium on Discrete Algorithms (SODA), ACM-SIAM, January 2003, pp. 76–85.

[Far79] A. M. Farley, *Minimal broadcast networks*, Networks, 9 (1979), pp. 313–332. DOI : 10.1002/net.3230090404.

[Far80] A. M. Farley, *Minimum-time broadcast networks*, Networks, 10 (1980), pp. 59–70. DOI : 10.1002/net.3230100106.

[FHM+94] R. Feldmann, J. Hromkovič, S. Madhavapeddy, B. Monien, and P. Mysliwietz, *Optimal algorithms for dissemination of information in generalized communication modes*, Discrete Applied Mathematics, 53 (1994), pp. 55–78. DOI : 10.1016/0166-218X(94)90179-1.

[Fra00] P. Fraigniaud, *Approximation algorithms for collective communications with limited link and node-contention*, Research Report LRI-1264, LRI, Université Paris-Sud, 91405 Orsay cedex, France, 2000.

[Fra01] P. Fraigniaud, *Approximation algorithms for minimum-time broadcast under the vertex-disjoint paths mode*, in 9th Annual European Symposium on Algorithms (ESA), vol. 2161 of Lecture Notes in Computer Science, Springer, August 2001, pp. 440–451. DOI : 10.1007/3-540-44676-1_37.

[GP91] M. Grigni and D. Peleg, *Thight bounds on minimum broadcast networks*, SIAM Journal on Discrete Mathematics, 4 (1991), pp. 207–222. DOI : 10.1137/0404021.

[JRS98] A. Jakoby, R. Reischuk, and C. Schindelhauer, *The complexity of broadcasting in planar and decomposable graphs*, Discrete Applied Mathematics, 83 (1998), pp. 179–206. DOI : 10.1016/S0166-218X(97)00110-8.

[KP95] G. Kortsarz and D. Peleg, *Approximation algorithms for minimum time broadcast*, SIAM Journal on Discrete Mathematics, 8 (1995), pp. 401–427. DOI : 10.1137/S0895480193245923.

[Kru92] D. W. Krumme, *Fast gossiping for the hypercube*, SIAM Journal on Computing, 21 (1992), pp. 365–380. DOI : 10.1137/0221026.

[Pro81] A. Proskurowski, *Minimum broadcast trees*, IEEE Transactions on Computers, C-30 (1981), pp. 363–366. DOI : 10.1109/TC.1981.1675796.

[Rav94] R. Ravi, *Rapid rumor ramification : approximating the minimum broadcast time*, in 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, November 1994, pp. 202–213. DOI : 10.1109/SFCS.1994.365693.

[Rum94] J. d. Rumeur, *Communications dans les réseaux de processeurs*, Études et Recherches en Informatique (ERI), MASSON, 1994.

[SCH81] P. J. Slater, E. J. Cockayne, and S. T. Hedetniemi, *Information dissemination in trees*, SIAM Journal on Computing, 10 (1981), pp. 692–701. doi : 10.1137/0210052.

[SW84] P. Scheuermann and G. Wu, *Heuristic algorithms for broadcasting in point-to-point computer networks*, IEEE Transactions on Computers, C-33 (1984), pp. 804–811. doi : 10.1109/TC.1984.1676496.

CHAPITRE 3

Communications point-à-point : introduction au routage compact

Sommaire

Mots clés et notions abordées dans ce chapitre :

- fonction de routage, fonction de routage simple
- complexité de Kolmogorov, algorithme glouton et *hitting set*
- tables de routage, routage par intervalles, routage dans les arbres
- ensembles k-dominants, graphes planaires et de genre g
- graphes planaires extérieurs, graphes p-plan
- dilatation, étirement, techniques générales

Un complément d'information sur certaines notions abordées dans ce chapitre peut être trouvé dans [Pel00].

3.1 Généralités, modèles et définitions

On s'intéresse au graphe des routeurs, les processeurs (ou réseaux locaux) étant connectés aux routeurs par un lien spécial. Pour communiquer avec ses (routeurs) voisins, le routeur dispose de liens adressables par des *numéros de ports*. Ces ports, que l'on peut voir comme des registres ou un espace mémoire particulier, peuvent être lus ou écrits. Pour

simplifier, mais ceci n'est pas obligatoire, on supposera que le même port peut être lus et écrit. Les numéros de ports peuvent être vus comme une numérotation locale des extrémités (ou brins) d'arête du graphe des routeurs. En général, le numéro de port en x vers le routeur voisin y est indépendant du numéro de port en y vers le routeur x.

On appelle *réseau* tout couple (G, port) où G est un graphe étiqueté connexe et où $\text{port}(\cdot,\cdot)$ est une fonction entière non nulle telle que $\text{port}(x,y) \neq \text{port}(x,z)$ pour tous sommets distincts x, y, z de G avec x voisins de y et z. En général, $V(G) = \{1, \ldots, |V(G)|\}$, c'est-à-dire les noms sont vus comme des entiers consécutifs. Mais comme on le verra plus tard dans la section 3.5 il peut arriver que les noms des sommets (on parle parfois d'étiquettes ou d'adresses) soient pris dans un ensemble plus grand $\{1, \ldots, M\}$ avec $M > |V(G)|$. Le numéro de port 0 est réservé au lien spécial entre le routeur et le processeur (ou réseau local) associé au routeur. La lecture ou l'écriture sur ce port particulier sera interprété comme si le routeur était la source ou la destination d'un message. Bien que non nécessaire, on supposera dans la suite que toute fonction de port vérifie que $\text{port}(x,y) \in \{1, \ldots, \deg(x)\}$.

Définition 1 (fonction de routage) *Une* fonction de routage R *sur un réseau* (G, port) *est une fonction telle que, pour tous* $u, v \in V(G)$*, ils existent une suite* $(x_0, \ldots, x_k)$ *de sommets, une suite* $(h_0, \ldots, h_k)$ *d'en-têtes, deux suites* $(q_0, \ldots, q_k)$ *et* $(p_0, \ldots, p_k)$ *de numéros de ports, telles que, pour tout* $i \in \{0, \ldots, k-1\}$ *:*

- $R(x_i, h_i, q_i) = (h_{i+1}, p_i)$ *;*
- $(x_i, x_{i+1}) \in E(G)$ *;*
- $p_i = \text{port}(x_i, x_{i+1})$ *;*
- $q_{i+1} = \text{port}(x_{i+1}, x_i)$ *;*
- $x_0 = u$*,* $x_k = h_0 = v$*, et* $q_0 = p_k = 0$*.*

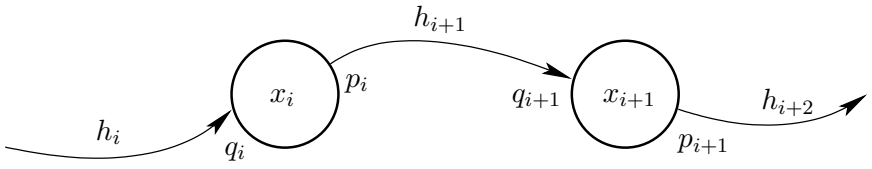


FIGURE 3.1 – Un modèle de fonction de routage.

Le routage d'un message de u vers v est donc obtenu en calculant $R(u, v, 0) = (h, p)$, puis en transmettant sur le port p de u le message avec l'en-tête h. Lorsque le numéro de port calculé par la fonction de routage devient 0, c'est que le message est arrrivé à destination.

Par extension, on dira que R est une fonction de routage sur le graphe G s'il existe un réseau (G', port), où G' est un graphe isomorphe à G (donc éventuellement avec les sommets renumérotés), sur lequel R est une fonction de routage.

Définition 2 (fonction de routage simple) *Une fonction de routage R est* simple *s'il existe une fonction P appelée* fonction de port*, telle que $R(x, h, q) = (h, P(x, h))$ pour tout sommet x, en-tête h et port q.*

Dit autrement, une fonction de routage simple ne dépend que du sommet courant et de l'en-tête, pas du port d'entrée, et ne peut modifier l'en-tête h qui vaut donc le nom de la destination pendant toute le transit du message. Cela implique que les routes induites par une fonction de routage simple ne peuvent contenir de boucle, sans quoi la destination ne reçoit jamais le message qui lui est destiné. Ceci est en contradiction avec la définition d'une fonction de routage qui implique qu'il existe une chaîne entre toutes les paires de sommets du graphe. Un moyen rapide de résumé ce discours est de dire qu'une fonction de routage simple ne génère que des chemins simples dans G.

Dans l'exemple 3.2, il est nécessaire de pouvoir modifier l'en-tête. C'est parfois très utile. Par exemple, x_1 ne sait pas router vers $x_5, \ldots, x_{50}$ mais sait que x_2 ou x_3 savent. Dans ces conditions il route vers x_2. C'est une stratégie très utilisée pour répartir l'information de routage sur les sommets.

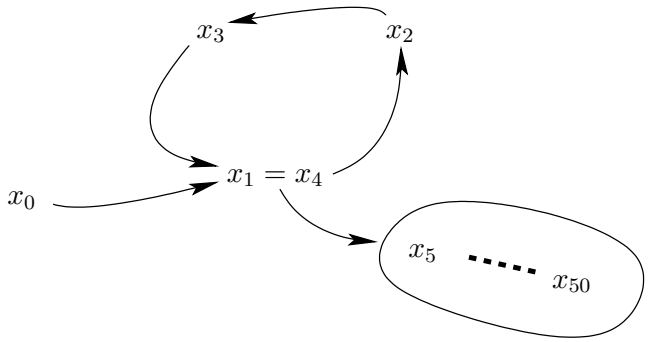


FIGURE 3.2 – Une fonction de routage non-simple.

Une fonction de routage R est dîte de *plus courts chemins* si la longueur de la route induite par R entre toute paire de sommets est la plus petite possible (ou de coût minimum s'il les arêtes du graphe sont valuées). Notez qu'une fonction de routage (non simple) qui induit des boucles dans le graphe ne peut être de plus courts chemins.

On verra en fin de chapitre des fonctions de routage qui utilise des fonctions de routage nécessitant la modification des en-têtes. Dans la première partie cependant, toutes les fonctions de routage seront simples.

3.2 Objectifs et exemples

L'objectif est de construire pour chaque graphe une fonction de routage « efficace ». La méthode ou l'algorithme de construction est appelé parfois *schéma de routage* qui ne doit

pas être confondu avec la fonction de routage.

On peut donner le sens suivant au qualificatif « efficace » :

1. les routes sont les plus courtes possibles
2. le nombre total de routes utilisant un lien (ou un sommet) est aussi faible que possible (pour limiter les attaques par exemple)
3. la décision de routage est rapide
4. abscence de *dead-lock*
5. le routage utilise localement peu de ressources : l'algorithme est concis (tables compactes)
6. la constructions de la fonction de routage est rapide
7. le schéma de routage s'applique à tous les graphes (universel)
8. le nombre de changement d'en-tête dans le transit d'un message est le plus petit possible (toujours 0 signifie fonction de routage simple)
9. ...

Tous ces critères sont bien souvent antinomiques. On verra qu'il est par exemple difficile, voir impossible, de garantir que les routes soient des plus courts chemins tout en imposant un algorithme de routage concis dans tous les routeurs et tous les graphes.

La plupart de temps on souhaite parler de la fonction de routage implémentée dans un routeur x. Si R est une fonction de routage sur G et x un sommet de G, alors la fonction de routage en x est la restriction de R en x, notée R_x, qui vaut donc $R_x(h, q) = R(x, h, q)$ pour tout en-tête h et port q.

Définition 3 (taille mémoire) *La* taille mémoire *d'une fonction de routage R_x est la longueur du plus petit programme qui implémente R_x.*

La taille mémoire n'est définie que pour les fonctions calculables. La taille mémoire de R_x ne doit pas être confondue avec la complexité en espace de R_x qui définit la quantité de mémoire nécessaire pour calculer $R_x(h, q)$. Ici il s'agit de la taille de la description de la fonction R_x : combien de bits sont nécessaires pour définir R_x ? et pas combien de mémoire est nécessaire pour évaluer R_x sur l'entrée (h, q).

La taille mémoire d'une fonction de routage est proche d'un concept mathématique très riche : la Complexité de Kolmogorov. Cette notion à pour objectif de définir la *quantité d'information* d'un objet donné. Plus formellement, la complexité de Kolmogorov d'une chaîne binaire S (ou de l'entier en base deux qu'il représente), notée $K(S)$, est la longueur en bits du plus petit programme qui renvoie S et qui s'arrête. Cette définition dépend du langage de programmation considéré (C, FORTRAN, LISP, ...). Cependant il n'est pas nécessaire de le préciser car il est toujours possible de traduire un programme d'un langage vers un autre, et cette traduction est un programme de longueur ne dépendant que des

langages, et pas de l'objet S. Si bien, qu'à une constante additive près tous les plus petits programmes sont de même longueur ! et $K(S)$ est une valeur inhérente à l'objet S.

On peut démontrer que le calcul de $K(\cdot)$ est indécidable, c'est-à-dire qu'il n'existe pas d'algorithme qui, pour toute entrée S, calcule $K(S)$.

Exercice. Pourquoi K n'est pas calculable par un algorithme ? Supposons qu'il existe un algorithme calculant $K(S)$ pour tout S. Cela signifie qu'il existe un programme de taille constante (indépendante de S) qui, pour toute entrée S, calcule $K(S)$ et qui s'arrête. Considérons le programme `P` suivant. Pour simplifier on suppose que K prend comme paramètre un entier qui est ensuite converti en chaîne binaire en utilisant sa représentation binaire.

```
int P(int n){
int s;
for(s=0;s<(1<<n);s++)
   if (K(s)>=n) return s;
return -1;
}
```

À propos du programme `P`, qui comporte `K` comme sous-programme, on peut faire les deux remarques suivantes :

- au total `P` (qui comprend le sous-programme `K`) est de taille constante, disons K_0 bits, indépendante du paramètre n, `K` étant de taille constante par hypothèse.
- `P` s'arrête et ne renvoie jamais -1. En effet, il existe toujours un entier $s \in [0, 2^n[$ tel que $K(s) \geqslant n$ (et donc comme `P` liste les 2^n entiers de l'intervalle $[0, 2^n[$, il s'arrête avant 2^n). En effet, le nombre de programmes de longueur $k < n$ qui s'arrêtent sur des valeurs différentes est au plus $\sum_{k=0}^{n-1} 2^k = 2^n - 1$. Donc il ne peut avoir qu'au plus $2^n - 1$ entiers s tel que $K(s) < n$, et donc parmi $[0, 2^n[$ il en existe un de complexité de Kolmogorov au moins n.

Soit $s_0 = \texttt{P}(K_0 + 1)$. Le programme `P` permet donc d'afficher l'entier $s_0 \geqslant 0$ et il s'arrête. Donc, $K(s_0) \leqslant K_0$. D'un autre coté si le programme s'est arrêté sur une valeur $\neq -1$, c'est à cause du test `if`, qui indique que $K(s_0) \geqslant n = K_0 + 1$: contradiction. Donc il n'y a pas d'algorithme qui calcule la fonction K.

Dans la pratique, on se contente de borner supérieurement ou inférieurement la complexité d'une chaîne binaire pour un ensemble de chaînes binaires, et pas d'une chaîne binaire particulière. Par exemple, on souhaitera dire que parmi les chaînes binaires de longueur $4n$ comprenant n bits à 1, il en existe au moins une, S_0, telle que $K(S_0) \geqslant \log_2(256/27) \cdot n \approx 3.24n$ bits, et que pour toute chaîne binaire S de cette famille, $K(S) \leqslant \log_2(256/27) \cdot n$.

Exercice. Pourquoi $3.24n$? On peut montrer que pour tout $c > 1$, et tout entier n assez grand,

$$\binom{cn}{n} \approx \left(\frac{c^c}{(c-1)^{c-1}}\right)^n .$$

En particulier, $\log_2 \binom{4n}{n} \approx \log_2(4^4/3^3)n = \log_2(256/27)n = (8 - 3\log_2 3)n \approx 3.24n$.

Comme nous nous intéressons aux fonctions de routages sur les graphes finis, disons à n sommets, l'ensemble des fonctions de routage est dénombrables. On peut donc établir une bijection entre les fonctions de routage et $\{0,1\}^*$. Autrement dit, la taille mémoire d'une fonction de routage R_x est exactement $K(S(R_x))$ où $S(R_x)$ est une chaîne binaire représentante de R_x. Le représentant $S(R_x)$ pourrait être par exemple, le code binaire d'un programme implémentant R_x.

Exemple 1 : Hypercube de dimension d, H_d. Ce graphe possède $n = 2^d$ sommets, numérotés par les chaînes binaires de longueur d : allant de 0...0 à 1...1. Posons le port de l'arête $\{x, y\}$ comme port(x, y) = numéro de bit où x et y différent, ces numéros allant de 1 à d. On peut écrire la fonction de routage R_x comme suit :

```
int R(int y){
if(y==x) return 0;
return MSB(x^y);
}
```

où MSB(w) est la fonction qui renvoie le numéro de bit de poids fort du mot binaire w, les bits étant numérotés de d (=bit de poids le plus fort) à 1 (=bit de poids le plus faible), ou bien 0 si $w = 0$. Notons que si $w > 0$, alors MSB$(w) = \lfloor \log_2 w \rfloor + 1$.

On peut montrer que pour certaines familles infinies de graphes, il existe des fonctions de routage de plus courts chemins de taille mémoire constante, indépendante du nombre de sommets.

Exercice. Construire un tel exemple ?

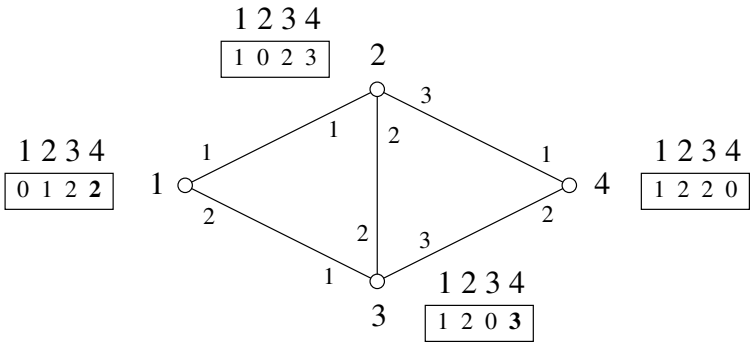
Solution : $K_{2,n-2}$ ... [A FINIR]

3.3 Les tables de routage

Cette technique générale correspond à la « tabulation » d'une fonction de routage simple. Pour chaque paire (source,destination), on stocke le numéro de port de sortie. Au passage, cela montre si besoin était, que l'ensemble des fonctions de routage simple est dénombrables, et que la taille mémoire de ces fonctions est donc correctement définie.

Fonction de routage du sommet $x = 1$:

```
int R(int y){
  int T[]={1,1,2,2}; // T[x]=n'est pas mis à 0 mais à 1 pour
  if(y==1) return 0; //      avoir exactement d valeurs possibles
  return T[y-1];     // en C les indices commencent à 0
}
```



Avantages :

- Toute fonction de routage simple peut être codée, en particulier toute fonction de plus courts chemins ;
- Le temps de calcul de la fonction est constant.

Inconvénient :

- Taille mémoire : $n \cdot \lceil \log_2 d \rceil + \lceil \log_2 n \rceil + O(1) = O(n \log d)$ bits pour un sommet de degré d. En effet, il y a n entrées dans la table, chaque entrée étant un entier dans $[1, d]$ codable sur $\lceil \log_2 d \rceil$ bits (on écrit simplement $d - 1$ en binaire). On a besoin de stocker x, sur $\lceil \log_2 n \rceil$, et la taille du programme, hormis la table, a une taille constante $O(1)$.

Taille linéaire des tables de routage, situation d'Internet (Alcatel) ...

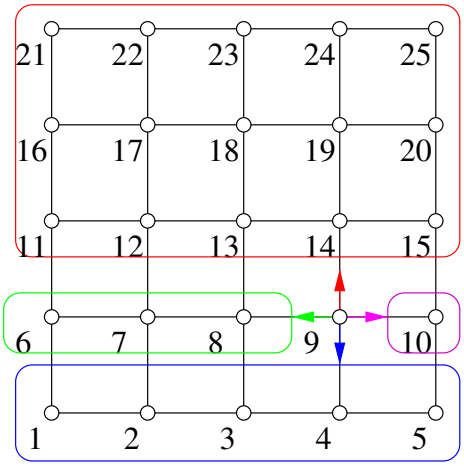
3.4 Routage par intervalles

Grosso modo c'est une technique qui revient à inverser la table de routage. Les tables listent pour chaque destination possible le port de sortie, alors qu'ici on liste pour chaque port les destinations qui l'utilisent.

Intuition : routage de type « hôtel ». Les destinations utilisant la même direction sont regroupées en 1 ou plusieurs intervalles d'entiers consécutifs.

Cette technique a réellement été implémentée dans les années 1990, dans les routeurs des Transputers C400 [DGJP93][FP94][Inm91][MT90][MTW93]. Ces routeurs, qui se voulaient universels pour assembler des architectures quelconques, avaient un certain nombres de registres qui servaient à coder les différents intervalles.

Exemple 1 : grille 2D

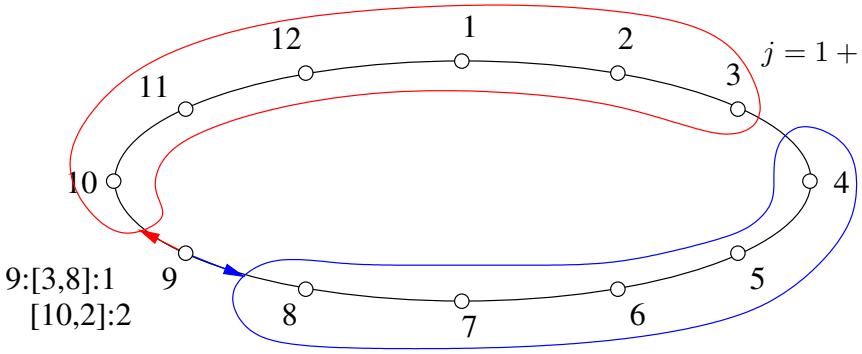


9:[1,5]:1
[6,8]:2
[10,10]:3
[15,25]:4

On peut facilement généraliser aux grilles de dimension d. Le degré d'un sommet est au plus $2d$, et il y a toujours un intervalle par lien. Donc la taille mémoire est $d \cdot (2\lceil \log_2 n \rceil + \lceil \log_2 d \rceil) + O(1) = O(d \log n)$ bits par sommet (le terme $O(1)$ est pour la taille du programme de routage proprement dit). On remarque que l'hypercube est une grille de dimension $\log n$.

On étend la notion d'intervalle $[a, b]$ avec $a \leqslant b$, à celle d'intervalle « modulo n » (ou cyclique) en posant $[b, a] = [b, n] \cup [1, a]$ si $a \leqslant b$.

Exemple 2 : cycle



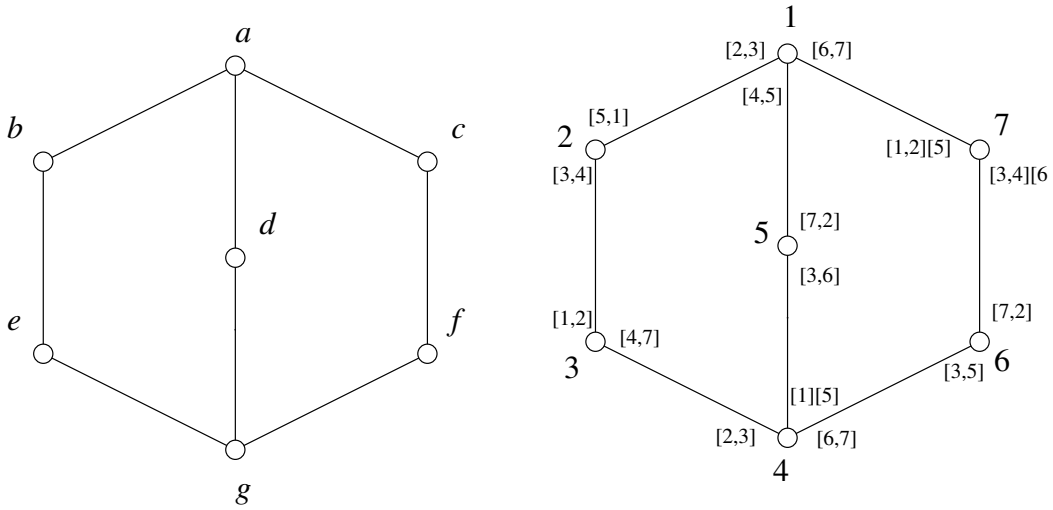
$j = 1 + ((i + \lfloor n/2 \rfloor - 1) \bmod n)$

i :$]j, i[$: 1
$]i, j]$: 2

Exemple 3 : un graphe non régulier

Dans cet exemple, on remarque :

- Le routage est de plus courts chemins
- Il y a deux intervalles sur certain lien (ex : 7)



- Tout routage par intervalles de plus courts chemins sur ce graphe nécessite 2 intervalles sur au moins un lien (analyse de cas).

Taille mémoire : S'il y a k intervalles par lien pour un sommet de degré d, alors $O(kd \log n)$ bits suffisent pour ce sommet. Donc si $k = O(1)$ comme dans les grilles, l'hypercube ou le cycle, alors la taille mémoire devient de $O(d \log n)$, qui est à comparer à la complexité $O(n \log d)$ pour les tables.

Les questions naturelles sont donc : que vaut k pour un graphe donné ? est-il borné pour tout graphe ?

Théorème 1 (Santoro-Kahtib [SK85]) *Tout arbre possède un routage par intervalles avec un intervalle par arc.*

Preuve. La numérotation des sommets est selon un parcours en profondeur d'abord (DFS). L'affectation des intervalles : un intervalle par fils, et aussi un pour le père, en remarquant que l'ensemble des descendants forment un intervalle dans la numérotation en profondeur. Ainsi, pour chaque fils, leurs descendants forment un intervalle, et aussi un intervalle pour le père car $[1, n] \setminus [a, b] = 1$ intervalle (modulo n). □

Corollaire 1 *Tout graphe (connexe) possède un routage par intervalles avec un intervalle par arc.*

Preuve. Il suffit de considérer un arbre couvrant (cf. figure 3.3) □

Remarque : cela n'est pas forcément un routage de plus courts chemins.

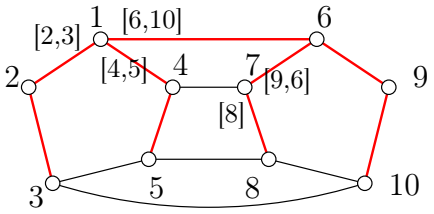


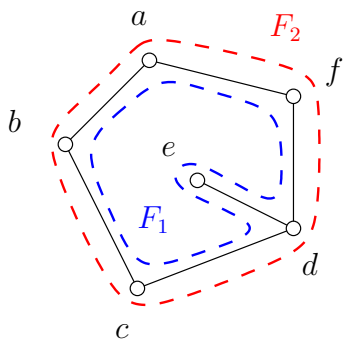
FIGURE 3.3 – Routage selon les arêtes d'un arbre couvrant.

Un graphe *planaire* est un graphe que l'on peut plonger sur le plan, c'est-à-dire que l'on peut dessiner sans croisement d'arêtes. Les *faces* d'un plongement sont les parties connexes de $\mathbb{R}^2$ que l'on obtient en découpant le long des arêtes.

Un *plan* d'un plongement donné (sur le plan $\mathbb{R}^2$ ou sur toute autre surface) est :

- soit le sous-graphe induit par le bord d'une face,
- soit un sous-graphe d'au plus trois sommets.

Attention, le bord d'une face n'induit pas forcément un cycle simple du graphe, comme le cas de la face F_1 dans l'exemple 3.4.



Bord de F_1 : (a, b, c, d, e, d, f, a)
Bord de F_2 : (a, b, c, d, f, a)

FIGURE 3.4 – Faces et bords.

Définition 4 (graphe *p*-plan) *Un graphe* p-plan *est un graphe* G *que l'on peut plonger dans* $\mathbb{R}^2$ *tel que les sommets de* G *se partitionnent en au plus* p *plans disjoints.*

Théorème 2 (Frederickson-Janardan [FJ88]) *Tout graphe* p*-plan possède un routage par intervalles de plus courts chemins avec au plus* $3p/2$ *intervalles par arc.*

Remarques :

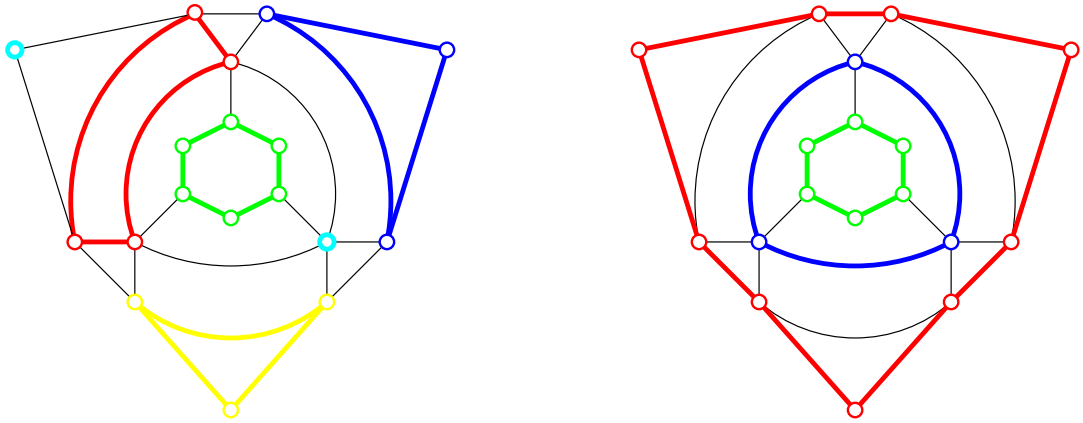


FIGURE 3.5 – Exemple d'un graphe 3-plan.

- Les arbres sont 1-plan, ils n'ont qu'une face ! Les graphes qui peuvent être dessinés de sorte que tous les sommets appartiennent à une seule face sont appelés graphes *planaires extérieurs* (*outerplanar* en anglais). Ils sont donc 1-plan, et inversement, tout graphe 1-plan possède bien évidemment un dessin où tous les sommets se trouvent sur le bord de la même face.
- Pour $p = 1$, le théorème 2 nous dit qu'il faut $\leqslant \lfloor 3 \times 1/2 \rfloor = 1$ intervalle. On retrouve donc le résultat précédant (théorème 1) pour les arbres.
- Il est NP-complet de savoir si G est p-plan, mais il existe des algorithmes d'approximations à un facteur constant près. On peut aussi montrer que la largeur arborescente des graphes p-plan est $O(\sqrt{p}\,)$.

Preuve.

Numérotation des sommets : les plans $P_1, \ldots, P_p$ sont disjoints et partitionnent les sommets de G. Successivement, chaque plan P_i (pris dans un ordre quelconque) est numéroté en suivant le bord de sa face si P_i est le bord d'une face (ordre de la première visite, un sommet pouvant apparaître plusieurs fois). Sinon l'ordre de parcours des sommets des plans d'au plus trois sommets n'a pas d'importance. On remarque que dans le cas d'un arbre, le parcours du bord de la face extérieure correspond à un parcours en profondeur d'abord.

Affectation des intervalles : pour chaque sommet x, on associe un arbre couvrant de plus courts chemins enraciné en x, noté T_x. Pour chaque arête $\{x, u\}$ de G, on note $I(x, u)$ l'ensemble de sommets descendants de u (u compris) dans l'arbre T_x. L'ensemble $I(x, u)$ est représenté par l'union d'un certain nombre d'intervalles.

Il est clair que le routage ainsi défini est un routage par intervalles de plus courts chemins puisque le routage de x vers y s'effectue sur l'arête de T_x du chemin (et donc du

plus court) de x à y. Il est cependant est faux de dire que la route de x à y est dans T_x. (Pourquoi ?)

Il reste à montrer que, pour toute arête $\{x, u\}$, l'ensemble $I(x, u)$ est composé de l'union d'au plus $3p/2$ intervalles.

On appelle *segment* est un ensemble consécutif de sommets pris autour du bord de la face d'un plan si celui-ci est le bord d'une face, sinon un sous-ensemble quelconque de sommets d'un plan. Un segment est composé d'1 ou 2 intervalles selon que le point de départ de la numérotation du plan se situe dans le segment ou pas.

Proposition 10 *Pour tout $i \in \{1, \ldots, p\}$, $I(x, u) \cap P_i$ est un segment.*

Preuve. Supposons que P_i a au moins quatre sommets, le lemme étant clairement vrai sinon. P_i est alors le bord d'une face, disons F_i. Sans perte de généralité, considérons une suite de quatres sommets (a, w, b, z) pris dans un ordre cyclique autour de P_i, tels que $a, b \in I(x, u)$ et $z \notin I(x, u)$. Il suffit de montrer que $w \in I(x, u)$.

On note S le segment contenant a, w, b, et $C = T_x[u, a] \cup T_x[u, b]$, où $T_x[\alpha, \beta]$ représente le chemin du sommet α au sommet β dans l'arbre T_x. Soit R la région du plan délimitée par la courbe fermée $S \cup C$ ($R = \varnothing$ est possible), et soit $\bar{R} = R \cup S \cup C$ la région ainsi que son bord.

Par l'absurde, supposons que $w \notin I(x, u)$. Donc $w \in I(x, u_w)$ pour un certain voisin $u_w \neq u$ de x. De même comme $z \notin I(x, u)$, $z \in I(x, u_z)$ pour un voisin $u_z \neq u$ de x. Notons que $u_w = u_z$ est possible, de même que $u \in S$.

Montrons que $T_x[u_w, w]$ ou $T_x[u_z, z]$ intersecte C (voir la figure 3.6).

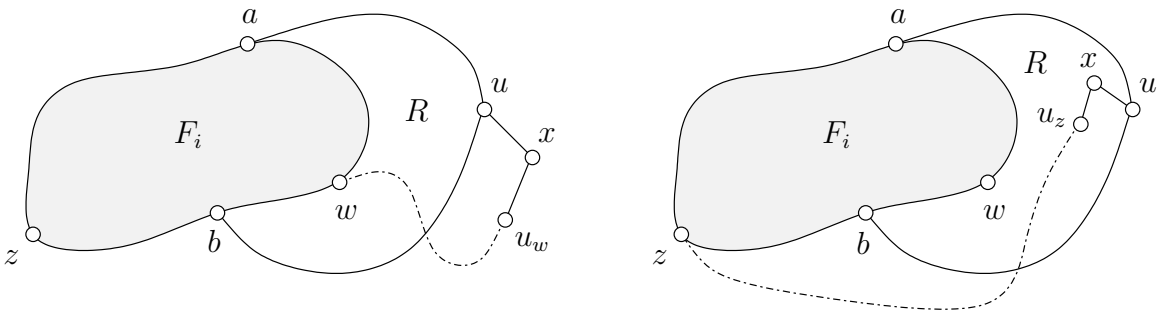


FIGURE 3.6 – Deux cas possibles : $x \in \bar{R}$ ou $x \notin \bar{R}$.

Si $x \notin \bar{R}$, alors $u_w \notin \bar{R}$. Comme $w \in \bar{R}$ et que F_i ne contient pas d'arête, $T_x[u_w, w]$ intersecte C. Si $x \in \bar{R}$, alors $u_z \in \bar{R}$. Comme $z \notin \bar{R}$ et que F_i ne contient pas d'arête, $T_x[u_z, z]$ intersecte C.

Comme l'un des deux chemins intersecte C, il en découle une contradiction (laquelle ?) : donc $w \in I(x, u)$, ce qui prouve la proposition 10.

Commentaire. La contradiction n'est pas à cause de la planarité ou du fait que tous les chemins soient des plus courts chemins, mais tout simplement parce que ce sont des chemins d'un même arbre, T_x, qui ne peut donc contenir de cycle. □

De la proposition précédente, on déduit que $I(x,u)$ est composé d'au plus $2p$ intervalles, puisque :

$$I(x,u) \;=\; I(x,u) \cap \bigcup_{i=1}^{p} P_i \;=\; \bigcup_{i=1}^{p} I(x,u) \cap P_i$$

et que $I(x,u) \cap P_i$ est composé d'au plus deux intervalles. On va voir que c'est un peu moins.

Observons que comme $I(x,u) \cap P_i$ est formé d'un segment, alors $P_i \setminus (I(x,u) \cap P_i)$, son complémentaire dans P_i, l'est aussi. Si l'un est formé de deux intervalles, l'autre est formé d'un seul, le point de départ de la numérotation n'appartenant qu'à un seul des deux segments. Étant donnée l'arête $\{x,u\}$, il est ainsi possible de partitionner P_i en au plus trois intervalles non cycliques : $P_i = A_i \cup B_i \cup C_i$ (voir la figure 3.7). Le segment $I(x,u) \cap P_i$ est donc composé d'1 ou 2 intervalles non cycliques pris dans l'ensemble $\{A_i, B_i, C_i\}$. Globalement $I(x,u)$ est composé d'un certain nombre d'intervalles pris parmi les $t = 3p$ intervalles de $\bigcup_{i=1}^{p} \{A_i, B_i, C_i\}$.

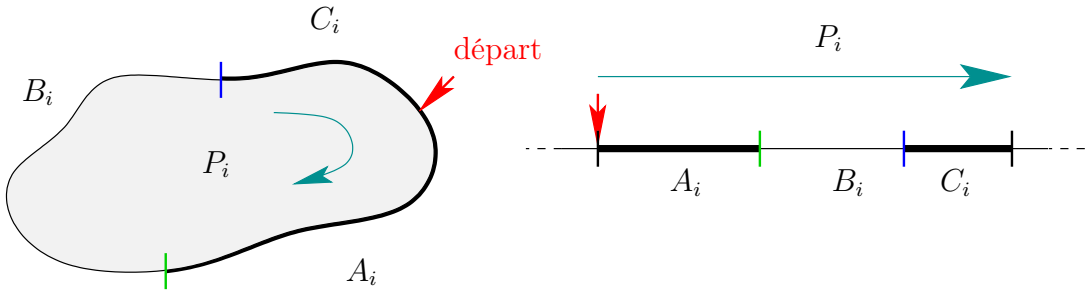


FIGURE 3.7 – Exemple de segment (en gras) dans P_i, et les trois intervalles A_i, B_i, C_i associés.

Soit k le nombre minimum d'intervalles pour représenter $I(x,u)$. Si $k > t/2$, alors dans $I(x,u)$ il doit exister deux intervalles consécutifs : A_i et B_i, B_i et C_i, ou C_i et A_{i+1}, pour un certain i. Et donc ces deux intervalles pourraient être regroupés pour former un seul intervalle (éventuellement cyclique), diminuant ainsi d'1 le nombre d'intervalles de $I(x,u)$: contradiction, k était minimum. Donc $k \leqslant t/2 = 3p/2$ ce qui termine la preuve. □

On remarque que dans la preuve précédante, le routage s'effectue selon une collection d'arbres qui auraient pu être arbitraires, le nombre d'intervalles étant majoré par $3p/2$ indépendamment du choix des arbres. En particulier, si le graphe est valué (coût sur les arêtes), alors la technique fonctionnera avec des arbres de coût minimum.

On observe que $p = \Omega(n)$ est possible, dans le cas d'une grille par exemple. Pour une grille $(2k) \times (2k)$ par exemple, il faut au moins $1 + (k-1)^2$ plans pour couvrir tous les sommets, soit $n/4$ plans environ pour une grille carrée.

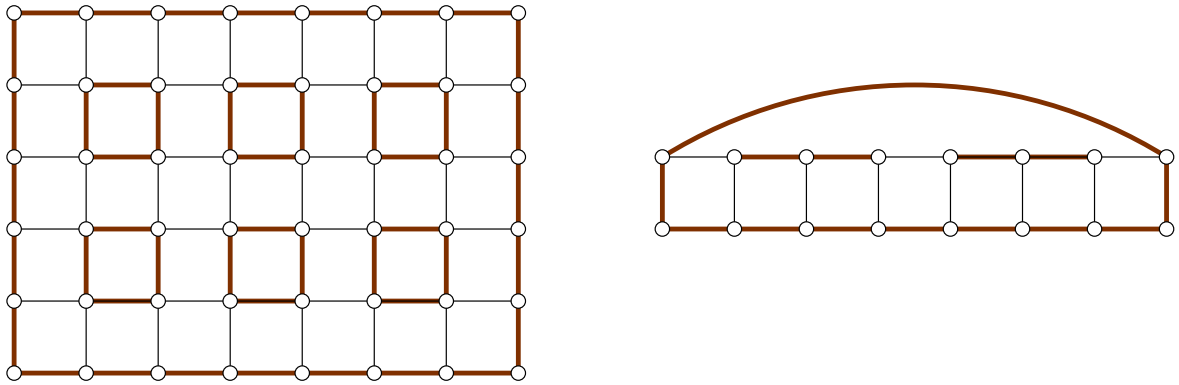


FIGURE 3.8 – Graphes planaires $\Omega(n)$-plan.

On remarque aussi que tout graphe (planaire ou pas) possède au plus $\lceil n/3 \rceil$ plans, en utilisant les plans à trois sommets, le dernier plan ayant n mod 3 sommets.

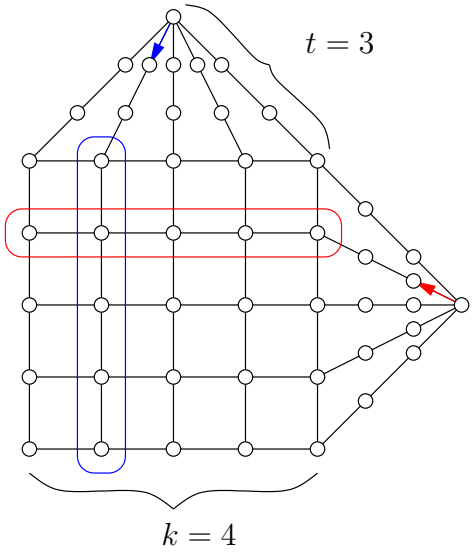
Exercice. Construire un graphe planaire p-plan avec p minimum et seulement $3p$ sommets ? Solution : on peut construire le graphe à partir de p triangles indépendants, puis d'ajouter des arêtes jusqu'à obtenir un graphe planaire triangulés. Le nombre de plan est au moins p puisque dans ce graphe un plan ne peut comprendre qu'au plus trois sommets.

Problème ouvert : Est-ce que tout graphe planaire possède un routage par intervalles de plus courts chemins avec $O(\sqrt{n})$ intervalles ? et plus généralement avec $O(\sqrt{p})$ intervalles pour un graphe planaire p-plan ? On sait simplement qu'il existe des graphes planaires où tout routage de plus courts chemins nécessite au moins $0.38\sqrt{n}$ intervalles [GP96a]. On sait aussi que si l'on fixe les routes à suivre un système d'arbres couvrants de plus courts chemins, alors pour le graphe $K_{2,n-2}$, $\Omega(n)$ intervalles (et même $\Omega(n)$ bits d'information) par sommet sont nécessaires [GH99].

Borne inférieure sur le nombre d'intervalles ? matrices de contraintes ?

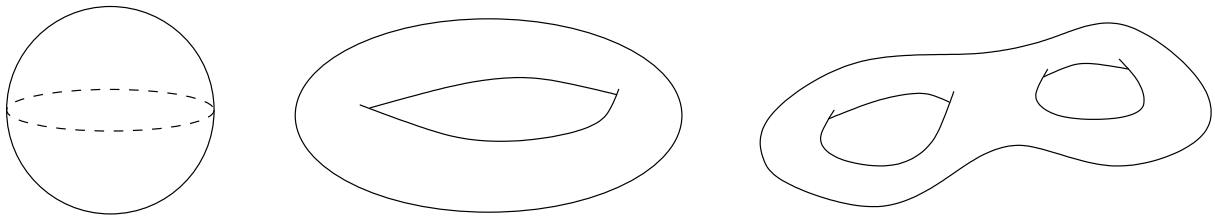
Une surface orientable de genre g, g entier $\geqslant 0$, est une sphère avec g trous. La sphère est de genre 0, le tore de genre 1, le double-tore de genre 2, etc. Un graphe est planaire si et seulement s'il peut être dessiné sur une sphère (de genre 0 donc).

Les maillages d'objets 3D (résultant de l'application de la technique des éléments finis) sont typiquement des graphes de genre « faible » (carosserie de voiture, mais surtout terrain, ...). Toute surface S (compact sans bord) est homémorphe (à une déformation continue près – une tasse de café est homémorphe à un beignet) à une sphère percée d'un certains nombre de trous (plus ou moins faible). Et tout graphe possède un genre, le genre



Il faut $3t > k + t$
ici $3 \cdot 3 > 4 + 3$

FIGURE 3.9 – Graphe avec $\Omega(\sqrt{n}\,)$ intervalles.



sphère $g = 0$ tore $g = 1$ double tore $g = 2$

minimum d'une surface sur laquelle il peut être plongé, c'est-à-dire dessiné sans croissement d'arêtes.

Définition 5 (p-plan de genre g) *Un graphe* p-plan de genre g *est un graphe* G *que l'on peut plonger sur une surface orientable de genre* g *tel que les sommets de* G *se partitionnent en au plus* p *plans disjoints.*

Le graphe dessiné à droite sur la figure 3.8 est un graphe 1-plan de genre 1.

Théorème 3 ([FJ88]) *Tout graphe* p*-plan de genre* g *possède un de routage par intervalles de plus courts chemins avec au plus* $3p/2 + g$ *intervalles par arc.*

Preuve. Non donnée dans le cours. □

Il existe cependant des graphes planaires nécessitant $p+g = \Omega(n)$ pour toutes valeurs de p et g, ce qui rend inefficace l'utilisation du théorème 3. C'est le cas de la grille $(2k) \times (2k)$. Intuitivement, une anse supplémentaire permet de fusionner au plus deux faces. A part la face extérieure, il n'existe pas de paire de faces permettant de couvrir plus de 8 sommets. En genre g il faut au moins $1+(k-1)^2-2g$ plans. Ainsi, pour $g = (k-1)^2/2 = (\sqrt{n}/2)^2/2 = n/8$, il faut plus de $(k-1)^2 - 2g = k^2/2 = n/8$ plans.

On remarque également que si G a une couverture par p plans qui sont tous des bords de faces, alors on peut optimiser le nombre d'intervalles en utilisant le théorème 3. On connecte tous les plans (en faite les faces) en ajoutant une anse pour chaque nouveau plans. Il faut $g = p-1$ anses pour avoir un seul plan de genre g connectant tous les anciens plans. Le nombre d'intervalles est alors $\leqslant \lfloor 3/2 + g \rfloor = p$. Malheureusement, la technique échoue si les plans ne sont pas initialement des bords de face.

Définition 6 (ensemble *k*-dominant) *Soit G un graphe, $S \subseteq V(G)$, et k un entier $\geqslant 0$. S est un* ensemble k-dominant *pour G si $d_G(x, S) \leqslant k$, c'est-à-dire si tout sommet est à distance au plus k d'un des sommets de S. On note $\gamma_k(G)$ la cardinalité du plus petit ensemble k-dominant de G.*

Remarques :

- $\gamma_0(G) = |V(G)|$ pour tout G.
- Il est NP-difficile de calculer $\gamma_1(G)$.
- $\gamma_k(G) = 1$ pour tout $k \geqslant$ diamètre de G.

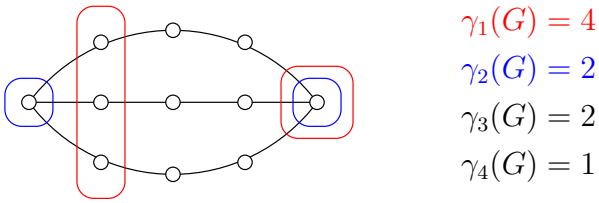


FIGURE 3.10 – Exemples d'ensembles k-dominants pour $k = 1, 2, 3, 4$.

Définition 7 (dilatation) *La* dilatation *d'une fonction de routage R sur un graphe G est la longueur de la plus longue route induite par R sur G.*

Remarques :

- La dilatation $\geqslant$ diamètre de G
- La dilatation d'une fonction de routage simple est $\leqslant |V(G)| - 1$

Théorème 4 *Pour tout entier $k \geqslant 0$, tout graphe (connexe) G de diamètre D possède une fonction de routage par intervalles de dilatation $D+k$ et avec au plus $\frac{1}{2}\gamma_k(G)+1$ intervalles par arc. De plus la longueur de la route entre x et y est au plus $d_G(x, y) + 2k$.*

Preuve. Soit $\{x_1, \ldots, x_t\}$ un ensemble k-dominant pour G avec $t = \gamma_k(G)$. Soit T_i un arbre de racine x_i couvrant G et de plus courts chemins. On définit des régions R_i comme l'ensemble des sommets y « plus près » de x_i que de n'importe quelle autre région (il faut fixer une règle uniforme pour casser les égalités, comme par exemple prendre le plus petit indice i si un sommet est à égale distance entre plusieurs x_i). On pose $\hat{T}_i = T_i \cap R_i$.

On a les propriétés suivantes : 1) chaque R_i induit un sous-graphe connexe de G (suppose connexe bien sûr). 2) La famille $\{R_i\}$ est une partition des sommets de G. 3) La profondeur de chaque arbre $\hat{T}_i$ est au plus k.

[A FINIR] □

Remarques :

- Si $k = D$, alors on retrouve le résultat des arbres puisque $\gamma_D(G) = 1$ pour tout graphe G.
- La valeur $k = \lceil D/2 \rceil$ est particulièrement intéressante (prendre une étoile et montrer la différence entre $k < D/2$ et $k = \lceil D/2 \rceil$ pour les arbres).
- Il est conjecturé que $\gamma_k(G) \leqslant 4$ si $k \geqslant D/2$ et G est planaire.
- En particulier, pour tout graphe G à n sommets, $\gamma_k(G) \leqslant \sqrt{n \ln n} + 1$ si $k \geqslant D/2$. On peut le montrer à l'aide de l'algorithme ci-dessous qui sera démontré plus tard dans le cours.

Algorithme glouton :
Soit $x \mapsto B(x)$ la boule de rayon $\lceil D/2 \rceil$ centrée en x.

1. S'il existe un x_0 tel que $|B(x_0)| < \sqrt{n \ln n} + 1$, alors renvoyer $B(x_0)$, sinon
2. $P = \varnothing$ et soit $F = \{B(x) \mid x \in V(G)\}$. On note $F(x) = \{B \in F \mid x \in B\}$
3. Tant que $F \neq \varnothing$ faire :
   - choisir un x tel que $|F(x)|$ est maximum
   - $P = P \cup \{x\}$, $F = F \setminus F(x)$
4. Renvoyer P.

Notons que l'étape 1 est valide car il faut remarquer que $B(x) \cap B(y) \neq \varnothing$ pour toute paire x, y de sommets. Dit autrement, toute boule intersecte toutes les autres. La preuve qu'après le point 3, $|P| \leqslant \sqrt{n \ln n} + 1$ sera donnée plus loin dans le cours dans ce chapitre, en section 3.6.

3.5 Routage dans les arbres

Expliquer la nuance entre fonction de routage et schéma de routage. Un schéma de routage est la donnée d'une fonction de routage et de son implémentation. On parle donc d'un schéma de routage utilisant des tables de taille M, plutôt qu'une fonction de routage.

Le théorème suivant nous indique qu'on peut toujours faire en sorte d'avoir des tables de routage de $O(\sqrt{n})$ bits pour les arbres à n sommets. Cette borne est à rapprocher de la taille donnée par la technique du routage par intervalles : $O(d \log n)$ bits pour un sommet de degré d, ce qui est $\Omega(\sqrt{n}\,)$ lorsque $d = \Omega(\sqrt{n}/\log n)$.

Théorème 5 (Gavoille [Gav00][EGP03]) *Tout arbre à n sommets possède un schéma de routage de plus courts chemins avec des tables de $O(\sqrt{n}\,)$ bits. De plus, il existe un arbre et un sommet où toute tout schéma de routage de plus courts chemins nécessite des tables de $\Omega(\sqrt{n}\,)$ bits.*

On rappelle que $f(n) = \Omega(g(n))$ signifie qu'il existe une constante c et un entier n_0 tels que pour tout $n \geqslant n_0$, $f(n) \geqslant c \cdot g(n)$.

Preuve. Tout d'abord, dire que la fonction R_x est implémentable en K bits, signifie, qu'il existe un programme P qui implémente R_x et dont la longueur est K bits. En fait, cette longueur peut dépendre du langage considérer (C ou FORTRAN ou LISP ... ou une machine de Turing Universelle) mais il est facile de voir que tout ces langages sont équivalents à une constante additive près qui ne dépend que du langage (et non du programme) : cette constante est la longueur du traducteur d'un langage vers un autre.

Il est malheureusement impossible de décrire explicitement un contre exemple de graphe qui nécessite beaucoup « d'informations de routage », comme pour le routage par intervalles. Par exemple, on ne peut pas écrire une chaîne binaire S de 1000 bits avec $K(S) \geqslant 999$, bien que l'on peut facilement prouver que la moitié des chaînes binaires de 1000 bits ne peuvent être compressées d'un seul bit.

Borne inférieure : $\Omega(\sqrt{n}\,)$ bits sont nécessaires.

Cette preuve est basée sur la longueur de programme (c'est-à-dire la complexité de Kolmogorov).

Soit T un arbre à n sommets enraciné en x. On définit la *signature* de T comme la suite d'entiers $S(T) = (n_1, \ldots, n_d)$, où n_i représente le nombre de descendants du $i$ème fils de x avec $n_1 \leqslant \cdots \leqslant n_d$.

On observe que l'on a : $d = \deg(x)$, $\sum_{i=1}^{d} n_i = n - 1$, et que pour toute suite d'entiers $S = (n_1, \ldots, n_d)$ telle que $1 \leqslant n_1 \leqslant \cdots \leqslant n_d$ et $\sum_{i=1}^{d} n_i = n - 1$, il existe un arbre T_S à n sommets (par exemple de hauteur 2) tel que $S(T_S) = S$.

Soit R une fonction de routage de plus courts chemins sur T, et R_x la fonction de routage en x. Soit M la taille du programme implémentant R_x. Considérons le programme P suivant :

$P =$ « pour $y = 1$ à n sauf x, trier les réponses identiques $R_x(y, 0)$ »

On observe que P affiche la signature de T et s'arrête. La taille de ce programme est $|P| \leqslant M + 2\log n + O(1)$. Soit $K(S(T))$ la taille du plus petit programme qui affiche $S(T)$ et qui s'arrête. Par définition de P, $|P| \geqslant K(S(T))$. Donc $M \geqslant K(S(T)) - O(\log n)$.

Soit $f(N)$ le nombre de suites d'entiers $(n_1, \ldots, n_d)$ telles que $1 \leqslant n_1 \leqslant \cdots \leqslant n_d$ et $\sum_{i=1}^{d} n_i = N$ (une telle suite s'appelle une partition de l'entier N). Par exemple $f(4) = 5$, car :

- $4 = 1 + 1 + 1 + 1$
- $4 = 2 + 1 + 1$
- $4 = 2 + 2$
- $4 = 3 + 1$
- $4 = 4$

Il existe une partition S_0 de $N = n - 1$ telle que la $K(S_0) \geqslant \log_2 f(N)$. En effet, il existe 2^t programmes différents de longueurs t bits. Donc $f(N) \leqslant 2^t$ ce qui implique $t \geqslant \log_2 f(N)$.

En particulier, il existe un arbre T_0 à n sommets de signature S_0 telle que $K(S(T_0)) = K(S_0) \geqslant \log_2 f(n-1)$. On a donc $M \geqslant \log_2 f(n-1) - O(\log n)$.

Par une formule de Ramanujan (1911) on a que $f(N) \approx \frac{1}{4\pi N\sqrt{3N}} e^{\sqrt{\pi N/3}} < e^{\sqrt{2N}}$, et donc que $\log_2 f(n-1) = \Omega(\sqrt{n}\,)$. En conclusion, $M = \Omega(\sqrt{n}\,)$.

Borne supérieure : $O(\sqrt{n}\,)$ bits suffisent.

Soit T un arbre à n sommet. Pour des raisons pratique, on considère que T est enraciné en un sommet quelconque. Considérons un sommet interne x ayant d fils (donc de degré $d+1$). Pour la racine, le schéma est similaire.

Numérotation des sommets : la racine a le numéro 1, puis les autres sommets sont numérotés selon un parcours en profondeur d'abord avec priorité selon le poids croissant des fils (le poids est la taille du sous-arbre issu du sommet considéré).

Ports : le port vers le père est le 1, et le port vers fils ayant le i-ième numéro le plus petit est $i + 1$ (si x était la racine, le numéro aurait été seulement i).

Table du sommet x : $(x, n_1, \ldots, n_d)$

Algorithme de routage de x vers y : (fonction de routage simple $R_x(y)$)

1. Si $y = x$ alors $R_x(y) = 0$.
2. Si $y \notin]x, x + \sum_{i=1}^{d} n_i]$, alors $R_x(y) = 1$.
3. Sinon, $R_x(y) = p + 1$ (ou p si x est racine) tel que $y \in]x + \sum_{i=1}^{p-1} n_i, x + \sum_{i=1}^{p} n_i]$.

La validité du routage est évidente, les cas 2 correspondant au cas où y n'est pas un descendant de x. Si c'est un descendant, on vérifie que p est alors unique et que le p-ème fils de x contient effectivement le sommet numéroté y.

Il reste à montrer maintenant que l'on peut coder la table de x en $O(\sqrt{n}\log n)$ bits. Comme x se code en $O(\log n)$ bits, il reste à coder efficacement $(n_1, \ldots, n_d)$.

En fait on va montrer un codage « simple » utilisant $O(\sqrt{n}\log n)$ bits bien qu'un codage théorique mais peu efficace de $O(\sqrt{n})$ bits/sommet existe. Notre codage « simple » permet un calcul de la fonction de routage en temps $O(\sqrt{n}\,)$, alors que le codage théorique est a priori en $e^{O(\sqrt{n})}$, exponentiellement plus lent. En adaptant notre codage, on peut facilement diminuer la complexité à $O(\log n)$.

La suite $(n_1, \ldots, n_d)$ est codée par « répétition », c'est-à-dire par des couples $(a_1, r_1), \ldots, (a_k, r_k)$ tels que :

$$n_1, n_2, \ldots, n_d \;=\; \overbrace{a_1, a_1, \ldots, a_1}^{r_1}, \overbrace{a_2, \ldots, a_2}^{r_2}, \ldots, \overbrace{a_k \ldots, a_k}^{r_k} \;\simeq\; (a_1, r_1), (a_2, r_2), \cdots, (r_k, a_k)$$

Comme $a_{i+1} \neq a_i$ et $a_1 \geqslant 1$, on a $a_i \geqslant i$ pour tout i. On a ainsi :

$$n - 1 \;\geqslant\; \sum_{i=1}^{k} a_i \;\geqslant\; \sum_{i=1}^{k} i \;=\; \frac{1}{2}k(k+1) \;>\; \frac{1}{2}k^2$$

ce qui implique $k \leqslant \sqrt{2(n-1)} < \sqrt{2n}$. La suite $(n_1, \ldots, n_d)$ se code ainsi par deux suites, $(a_i)_i$ et $(r_i)_i$, de moins de $\sqrt{2n}$ entiers, donc en tout sur $O(\sqrt{n}\log n)$ bits. Ceci complète la preuve. □

En fait le routage défini au théorème 5 n'est finalement pas très efficace, quoiqu'on puisse toujours, pour chaque sommet, faire le minimum (en taille mémoire) entre le routage par intervalles (si le degré est faible) et $O(\sqrt{n}\,)$ bits. Le schéma suivant par contre est très utile.

Théorème 6 (Fraigniaud-Gavoille [FG01]) *Tout arbre à n nœuds possède un schéma de routage de plus courts chemins basé sur des adresses et des tables de $O(\log n)$ bits.*

La constante dans le $O(\log n)$ est donc importante ! On va voir dans la preuve que les tables et les adresses sont codées sur $3.78\log n$ bits, alors que pour démontrer la borne inférieure du théorème 5 il était crucial que les adresses soient sur $\log n$ bits, c'est-à-dire des entiers compris entre 1 et n.

Preuve. L'idée est de numéroter très astucieusement les sommets dans un espace plus grand que $[1, n]$, ceci afin d'éviter la borne inférieure du théorème 5. La table de routage d'un sommet va être composée uniquement de son adresse. C'est un routage à base d'adresses similaire au routage de l'hypercube (exemple d'introduction). Il n'y a pas de table de routage à proprement parlé.

Poids et identifiants des sommets. D'abord on choisit un sommet quelconque comme racine, puis on associe à chaque sommet u son *poids*, noté $w(u)$, qui vaut le nombre de

descendants de u, u compris. On lui associe également un identifiant $\mathrm{id}(u) \in [1, n]$ obtenu selon un parcours en profondeur avec priorité selon le poids décroissant de ses fils. Dit autrement, le fils de numéro $\mathrm{id}(u) + 1$ est le plus lourd des fils de u. Si plusieurs ont le même poids. On notera $w_1(u)$ le poids du fils le plus lourd de u, s'il existe. Sinon, on pose $w_1(u) = 0$ ... ou toute autre valeur.

Numérotation des ports. On va supposer que les numéros de ports des arêtes incidentes au sommet u qui n'est pas la racine, vont de 0 à $\deg(u) - 1$, au lieu de 1 à $\deg(u)$ par convention. Ceci va beaucoup simplifier la description du schéma. Le numéro de port menant vers le père de u est 0, si bien que les numéros de ports menant vers les f fils de u seront dans $[1, f]$. Les numéros de ports sont associés aux fils selon le parcours utilisé pour les identifiants. Plus précisément, on affectera le port $i \in [1, f]$ au fils v_i de u si $\mathrm{id}(v_i)$ est la ie plus grande valeurs parmi celles des fils de u.

Le code -1 sera utilisé comme port local (à la place du 0 habituel). Pour retrouver la convention habituelle des numéros de ports, il suffira de faire $+1$ à tous les numéros de ports renvoyer par l'algorithme de routage donné ci-après (sauf pour la racine).

Chemin compressé. On définit $\mathrm{path}(u)$ comme la suite des numéros de port du chemin allant de la racine à u. On définit $\mathrm{cpath}(u)$ (pour *compressed path*) comme la suite obtenue à partir de $\mathrm{path}(u)$ en supprimant les 1.

Par exemple : ...

Adresse du sommet u. $\mathrm{label}(u) = (\mathrm{id}(u), w(u), w_1(u), \mathrm{cpath}(u))$ où $w(u)$ est le poids du sommet u et $w_1(u)$ le poids du fils lourds de u.

Algorithme de routage de x vers y : $\mathrm{label}(x) \to \mathrm{label}(y)$

1. Si $\mathrm{id}(y) = \mathrm{id}(x)$, alors renvoyer -1.
2. Si $\mathrm{id}(y) \notin [\mathrm{id}(x), \mathrm{id}(x) + w(x)[$, alors renvoyer 0.
3. Si $\mathrm{id}(y) \in]\mathrm{id}(x), \mathrm{id}(x) + w_1(x)]$, alors renvoyer 1.
4. Sinon, renvoyer $\mathrm{cpath}(y)[|\mathrm{cpath}(x)| + 1]$.

La validité des trois premiers cas est évidente : on commence par tester si $x = y$, puis si y n'est pas un descendant de x, et enfin s'il est un descendant du fils lourds de x.

Pour le dernier cas, y est un descendant de x. Donc $\mathrm{path}(x)$ vu comme mot est un préfixe de $\mathrm{path}(y)$, ce qui implique que $\mathrm{cpath}(x)$ est un préfixe de $\mathrm{cpath}(y)$, la suppression des « 1 » n'affectant pas l'ordre des valeurs de path(). Comme y n'est pas non plus un descendant du fils lourd de x, le numéro de port menant vers y est la première valeur de $\mathrm{cpath}(y)$ qui n'est pas dans $\mathrm{cpath}(x)$.

Taille des adresses. Les trois premiers champs de label(u) nécessitent au plus $3\lceil \log n \rceil$ bits. Il reste à montrer que cpath(u) $= (p_1, \ldots, p_k)$ se code sur $O(\log n)$ bits. Montrons d'abord que

$$\prod_{i=1}^{k} p_i \;\leqslant\; \frac{n}{w(u)} \;\leqslant\; n \;.$$

Cette équation se prouve en effectuant une marche de la racine jusqu'à u en suivant les ports de path(u) tout en analysant le poids des sommets rencontrés (voir figure 3.5).

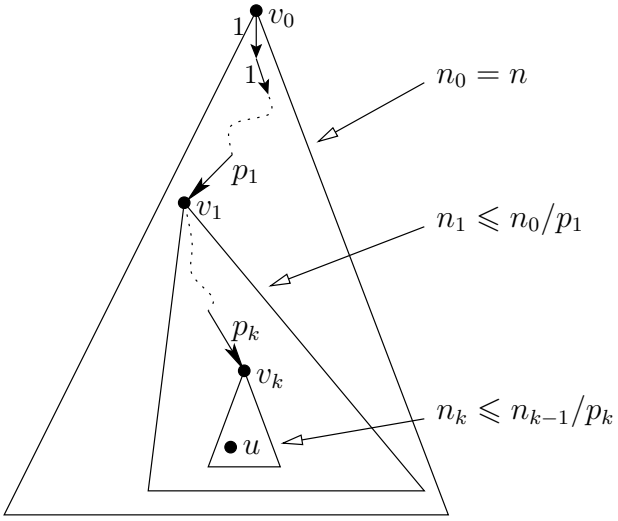


FIGURE 3.11 – Illustration de la preuve du théorème 6.

De la racine, disons le sommet v_0, après avoir traversé un certain nombre de ports numéroté 1 (éventuellent aucun), on utilise le port p_1 pour atteindre un sommet, disons v_1. Soit n_1 le poids de v_1. Il est clair que $n_1 \leqslant n/p_1$, car dans un arbre de poids au plus n, le poids de chacun des p_1 fils les plus lourds ne peut être $> n/p_1$. Plus généralement, une fois arrivé au sommet v_{i-1} de poids n_{i-1}, après avoir traversé un certain nombre de ports numérotés 1, on utilise le port p_i pour atteindre un sommet v_i de poids $n_i \leqslant n_{i-1}/p_i$. Ce processus s'arrête lorsqu'on utilise le port p_k, donc avec l'inégalité $n_k \leqslant n_{k-1}/p_k$.

En multipliant toutes ces inégalités entre elles (cela est valide car elles concernent des entiers positifs), on obtient (en posant $n_0 = n$ le poids de v_0)

$$\begin{aligned} \prod_{i=0}^{k} n_i \;&\leqslant\; \prod_{i=1}^{k} n_{i-1}/p_i \\ \Leftrightarrow \quad \prod_{i=1}^{k} p_i \;&\leqslant\; n_0/n_k \;\leqslant\; n \;. \end{aligned}$$

Pour coder cpath(u) on concatène l'écriture binaire de chaque $p_i - 2$ (chaîne S_1), puis on double les bits (chaîne S_2) de façon à pouvoir extraire chaque valeur de la concaténation.

Plus précisément, on met un 1 dans S_2 chaque fois que l'on commence une nouvelle valeur dans S_1. S_2 revient à coder les virgules de cpath(u) que l'on perd avec la concaténation. Dans l'exemple suivant, cela donne :

$$\begin{array}{rcccccccl} \text{cpath}(u) & = & (\ 7, & 2, & 9, & 5, & 14\ ) & & \\ & & 5 & 0 & 7 & 3 & 12 & & (p_i - 2) \\ S_1 & = & 101 & 0 & 111 & 11 & 1100 & & (\text{bin}(p_i - 2)) \\ S_2 & = & 100 & 1 & 100 & 10 & 1000 & & \end{array}$$

Par exemple, pour extraire $p_3 = \text{cpath}(u)[3]$ avec le codage (S_1, S_2), il suffit de repérer la position des 3e et 4e bits à 1 dans S_2, puis d'extraire les bits dans S_1 entre ces positions là, et d'ajouter 2 pour retrouver p_3. On peut montrer que cela peut être fait avec un nombre constants d'instructions arithmétiques de bases sur les entiers de $O(\log n)$ bits.

En ce qui concerne la longueur de ce codage, on rappelle que la longueur de l'écriture binaire de x vaut $\lceil \log(x+1) \rceil$ si $x > 0$ et 1 si $x = 0$. Pour tout entier $p \geqslant 2$, on note k_p le nombre de fois où la valeur p apparaît dans cpath(u). Pour pouvoir analyser simplement la longueur du codage, on va séparer les valeurs $p_i < 6$ et celles $\geqslant 6$. (On pourrait prendre une valeur > 6 comme seuil, mais l'analyse devient plus complexe.) Soit Q l'ensemble des valeurs $p_i \geqslant 6$. Le produit des p_i s'écrit donc :

$$\prod_{i=1}^{k} p_i \ = \ 2^{k_2} \cdot 3^{k_3} \cdot 4^{k_4} \cdot 5^{k_5} \cdot \prod_{q \in Q} q \ .$$

Dans le codage de la chaîne S_1 remarquons que lorsque $p_i \in \{2, 3\}$, on utilise 1 bit ($p_i - 2 = 0$ ou 1). Pour $p_i \in \{4, 5\}$, on utilise 2 bits (mot "10" ou "11"). Pour $q \in Q$, on utilise $\lceil \log(q - 2 + 1) \rceil$ bits. D'où :

$$\begin{aligned} |S_1| &= k_2 + k_3 + 2k_4 + 2k_5 + \sum_{q \in Q} \lceil \log(q-1) \rceil \\ &< k_2 + k_3 + 2k_4 + 2k_5 + \left(\sum_{q \in Q} \log(q-1) \right) + |Q| \\ &< k_2 + k_3 + 2k_4 + 2k_5 + |Q| + \log \prod_{q \in Q} (q-1) \end{aligned}$$

Malheureusement, on ne peut pas dire grand chose de $\prod_{q \in Q}(q-1)$, sinon que c'est au plus $\left(\prod_{q \in Q} q\right) - |Q|$. On va donc simplement utiliser le fait que $\prod_{q \in Q}(q-1) < \left(\prod_{q \in Q} q\right)$. Maintenant, par définition de Q,

$$\begin{aligned} \log \prod_{q \in Q} q &= \left(\log \prod_{i=1}^{k} p_i \right) - k_2 - k_3 \log 3 - k_4 \log 4 - k_5 \log 5 \\ &< \left(\log \prod_{i=1}^{k} p_i \right) - k_2 - k_3 - 2k_4 - 2k_5 \ . \end{aligned}$$

car $\log 3 > 1$, et $\log 4, \log 5 \geqslant 2$.

D'autre part, $\prod_{i=1}^{k} p_i \leqslant n$, donc :

$$|S_1| \;<\; |Q| + \log\left(\prod_{i=1}^{k} p_i\right) \;<\; \log_6 n + \log n \;=\; \left(1 + \frac{1}{\log 6}\right)\log n \;\approx\; 1.38 \log n$$

car Q est un ensemble de valeurs $\geqslant 6$ dont le produit est $\leqslant n$, et donc $|Q| \leqslant \log_6 n$.

Notons que coder S_1 avec les k virgules (mais sans S_2) est une fausse mauvais idée. En effet, il est parfaitement possible que $k = |Q| \geqslant \log_6 n \approx 0.63 \log n$. Du coup, le codage de cpath(u) nécessiterait alors de $|S_1| + k \approx 1.38 \log n + 0.63 \log n \approx 2.01 \log n$ symboles sur assymptotiquement $\log 3 \approx 1.58$ bits. Cela fait un total de $3.17 \log n$ bits. C'est moins bon.

Ainsi cpath(u) se code sur moins de $2|S_1| \approx 2.77 \log n$ bits. Notons qu'il a été démontré dans [War93] que le nombre $Z(n)$ de suites d'entiers $\geqslant 2$ dont le produit est $\leqslant n$ est $Z(n) \sim \kappa^{-1} n^{\rho}$ avec $\kappa = -\rho \cdot \xi'(\rho)$ où ρ est l'unique solution réelle de l'équation $\xi(\rho) = 2$ et où ξ la fonction de Zeta de Riemann, $\xi(x) = \sum_{i \geqslant 1} i^{-x}$. On a $\rho \approx 1.7286$ et $\kappa = 3.1429$. On pourrait donc coder théoriquement, en listant et comptant toutes les suites $(p_1, \ldots, p_k)$ possibles, cpath(u) avec $\lceil \log Z(n) \rceil \approx 1.72 \log n$ bits. Cependant, extraction de cpath$(u)[i]$ en temps constant poserait problème avec ce type de codage exhaustif.

Au total cela fait donc $5.77 \log n$ bits pour label(u) en ajoutant le codage pour id(u), $w(u)$ et $w_1(u)$. On peut encore compacter l'étiquette label(u) en remarquant que $\prod_{i=1}^{k} p_i \leqslant n/w(u)$, où $w(u) = n_k$ est le poids du sommets u. Donc cpath(u) se code en fait avec au plus $2.77 \log (n/w(u))$ bits. Dans ces conditions, on a tout intérêts de coder les entiers $w(u)$ et $w_1(u) \leqslant w(u)$ chacun sur $\lceil \log w(u) \rceil$ bits. Il faut en plus un mot binaire de $O(\log \log n)$ bits décrivant la longueur du codage de $w(u)$ et $w_1(u)$. La longueur du codage de $w(u)$ n'est pas exactement $K = \lceil \log n \rceil$ bits, mais une certaine valeur ℓ de l'intervalles $[1, K]$ donc qui se code en $\lceil \log K \rceil = O(\log \log n)$ bits.

Au final, la longueur du codage est :

$$\lceil \log n \rceil + 2 \lceil \log w(u) \rceil + 2.77 \log (n/w(u)) + O(\log \log n) \;<\; 3.78 \log n \;.$$

Notons que la valeur exacte de la constant est $1 + 2(1 + 1/\log 6) = 3.7737....$ Ceci termine la preuve du théorème 6. □

Remarques : Thorup et Zwick [TZ01] ont montré qu'il est possible de remplacer le $3.78 \log n$ par $(1 + o(1)) \log n$, mais la preuve possède des incorrections.

Dans la pratique on souhaite superposer plusieurs schémas de routage. Typiquement, les schémas complexes ressemblent à ceci : si l'adresse a telle préfixe, alors utiliser le schéma 1 (par exemple router dans l'arbre T_1), sinon le schéma 2 (selon l'arbre T_2). Ici il faut donc que plusieurs routage d'arbre puissent coexister. Malheureusement, pour que le théorème 6 s'applique il est fondamental que les numéros de ports soit correctement permutés (port 1 pour le fils le plus lourd). Bien sûr, les ports (tout comme les adresses de sommets), ne peuvent être optimisés qu'une fois pour l'ensemble du graphe.

Le modèle *port-fixé* suppose donc que les numéros de tous les ports sont fixés autour de chaque sommet x et sont des entiers entre $\{1, \ldots, \deg(x)\}$, et ne peuvent pas changer lors de la construction de la table.

Théorème 7 (Fraigniaud-Gavoille [FG01]) *Tout arbre à n nœuds possède un schéma de routage, dans le modèle port-fixé, basé sur des adresses et des tables de $O(\log^2 n / \log\log n)$ bits. Et c'est la meilleure complexité possible.*

Preuve. On ne va pas prouver la borne inférieure qui est assez difficile. L'idée (pour la borne supérieure) est assez similaire au schéma précédent. On associe aux sommets un numéro selon un parcours DFS de l'arbre. Ensuite un sommet va stocker le routage pour ses t fils les plus lourds, à l'aide d'intervalles. La destination y stocke son path(y) en supprimant les ports correspondant à des fils de rang $\leqslant t$. L'algorithme de routage est a peu près le même sinon qu'on avait précédemment $t = 1$.

La taille des adresses est de $O(t \log n)$ pour les t intervalles et $O((\log_{t+1} n) \log n)$ pour le path compressé. Il n'est pas difficile de voir que si le port p est stocké dans ce chemin compressé, alors la taille de l'arbre diminue d'un au moins facteur $t + 1$. Donc le nombre de ports stockés sera d'au plus $O(\log_{t+1} n)$, et un port coûte au plus $O(\log n)$ bits.

La fonction $t \log n + (\log_{t+1} n) \log n$ est optimisée en choisissant $t = \Theta(\log n / \log\log n)$, ce qui donne des adresses de taille $O(\log^2 n / \log\log n)$ comme annoncé. □

3.6 Facteur d'étirement et techniques générales

On admettra le résultat suivant dont des preuves peuvent être consultées dans [GG01][GP96b][BHV99][FG97].

Théorème 8 (Incompressibilité des tables de routage) *Il existe un graphe à n sommets où tout schéma de routage de plus courts chemins utilisant des tables de routage d'au plus M bits et des adresses d'au plus A bits vérifie $M + A = \Omega(n)$. De plus cette borne peut se produire simultanément sur $\Theta(n)$ sommets du graphe.*

On rappelle que la technique des tables de routage standard utilise $O(n \log n)$ bits/-sommets avec des adresses de $\lceil \log n \rceil$ bits et permet de générer pour tout graphe des plus courts chemins.

À cause du théorème précédent, on est amené à relaxer les conditions s'il l'on désire des tables de routage sous-linéaires, c'est-à-dire en $o(n)$ bits par sommets. Ce qui est contraint ici, c'est-à-dire ce qui « coûte » en complexité mémoire, ce sont les plus courts chemins.

Définition 8 (facteur d'étirement) *Soit R une fonction de routage sur un graphe G. On note $d_R(x, y)$ la longueur de la routage de x à y induite par R. Le* facteur d'étirement *de R est le plus petit réel s tel que $d_R(x, y) \leqslant s \cdot d_G(x, y)$ pour tout $x, y \in V(G)$.*

Le facteur d'étirement de R est 1 si et seulement si R est une fonction de routage de plus courts chemins.

L'intérêt de cette définition, est qu'il existe un compromis entre l'étirement et la taille des tables de routage, les adresses sont toujours supposées être de taille polylogarithmique. Grosso modo ce compromis établit que la taille des tables est de $n^{\Theta(1/s)}$ pour tout étirement s, et c'est le meilleur compromis possible. Plus concrètement, la table suivante donne les meilleurs schémas de routage connus (la taille des tables est donnée à un facteur polylog près et k est un entier $\geqslant 2$) :

s	table
1	n
3	$\sqrt{n}$
7	$n^{1/3}$
...	...
$4k-5$	$n^{1/k}$

Le meilleur compromis possible est de $n^{1/k}$ pour la taille mémoire (toujours à un facteur polylog près) pour un étirement de $2k-1$, k entier $\geqslant 1$, si une certaine conjecture d'Erdös est vraie. Cette conjecture relie le nombre maximum d'arêtes que peut avoir un graphe de maille $2k+2$. Il est conjecturé qu'il existe des graphes avec $\Omega(n^{1+1/k})$ arêtes. Malheureusement, cela n'a été démontré que pour $k = 1, 2, 3, 5$. Autrement dit, seule les deux premières lignes du tableau (étirement 1 et 3) proposent un compromis optimal. Pour le reste, c'est un problème largement ouvert.

On va montrer un résultat un peu plus faible (étirement est 5 au lieu de 3), mais qui reste simple à démontrer.

Théorème 9 (étirement 5) *Pour tout graphe à n sommets, il existe un schéma de routage de facteur d'étirement au plus 5 avec des tables de routage de $O(\sqrt{n}\cdot\log^2 n/\sqrt{\log\log n})$ bits/sommet et des adresses de $O(\log^2 n/\log\log n)$ bits.*

Preuve. On associe à chaque sommet la boule des exactement t plus proches voisins, t étant un paramètre que l'on optimisera à la fin cependant tel que $0 < t \leqslant n/2$. Soit $x \mapsto B(x)$, pour tout sommet $x \in V$. Si plusieurs sommets sont à même distance et qu'on ne peut pas tous les inclure dans $B(x)$ sans dépasser le volume t imposé, on les départage suivant leur identité, par exemple en préférant la plus petite (on suppose donc que l'ensemble des sommets est muni d'une relation d'ordre total $\prec$). On a la propriété importante suivante :

Fait 1 (monotonicité) *Soit un plus court chemin de x à y passant par z. Si $y \in B(x)$, alors $y \in B(z)$.*

En effet, pour le prouver, considérons un sommet $u \in B(z)$ arbitraire. Si $d(z,u) > d(z,y)$ alors $y \in B(z)$. Supposons que $d(z,u) \leqslant d(z,y)$. Donc $d(x,u) \leqslant d(x,z) + d(z,u) \leqslant$

$d(x,z)+d(z,y) = d(x,y)$ car z est sur un plus court chemin entre x et y. Si $d(z,u) < d(z,y)$ ou bien si $u \prec y$, alors dans les deux cas $u \in B(x)$ en vertu de la règle appliquée en cas d'égalité. On en déduit que $B(z) \subseteq B(x)$. Si $y \notin B(z)$ et $y \in B(x)$ alors $|B(z)| < |B(x)|$ ce qui est absurde, toutes les boules sont de taille exactement t. Donc $y \in B(z)$ dans ce là aussi. Reste le cas où $d(z,u) = d(z,y)$ et $y \prec u$. Comme $u \in B(z)$, c'est donc que $y \in B(z)$ aussi en vertu de la règle appliquée en cas d'égalité. Dans tous les cas de figure, on a prouver que $y \in B(z)$.

On choisit un ensemble de sommets P qui vérifie :

1. $P \cap B(x) \neq \varnothing$ pour tout $x \in V$; et
2. $|P| \leqslant (n \ln n)/t + 1$.

Un tel ensemble (connu sous le nom *hitting set* an anglais) peut être calculé par l'algorithme glouton suivant qu'on a déjà rencontré à la section 3.4 à propos du calcul des ensembles k-dominants :

1. $P := \varnothing$ et soit $F = \{B(x) \mid x \in V(G)\}$. On note $F(x) := \{B \in F \mid x \in B\}$
2. Tant que $F \neq \varnothing$ faire :
   - choisir un $p \in V$ tel que $|F(p)|$ est maximum
   - $P := P \cup \{p\}$, $F := F \setminus F(p)$

Notons au passage que cet algorithme est très général et s'applique à n'importe quelle famille d'ensembles. On peut remarquer aussi que l'algorithme renvoie un ensemble P de taille proche de l'optimal si l'on considère un chemin dont les boules sont les intervalles de longueur $2t$. Il est clair que pour tout P on doit avoir $|P| \geqslant n/(2t)$.

Prouvons que $|P| \leqslant (n \ln n)/t + 1$. Soit f_i la cardinalité de F à la fin de la ie itération. On a $f_0 = n$. Montrons qu'il existe un $p \in V$ tel que $|F(p)| \geqslant f_i \cdot t/n$. En effet, on a

$$\sum_{v \in V} |F(v)| \;=\; \sum_{B \in F} |B| \;\geqslant\; f_i \cdot t \;.$$

Pour visualiser l'égalité précédente, on construit le graphe orienté suivant : pour toute $B(x) \in F$, et tout $y \in B(x)$, on met un arc (x,y). Il y a donc un arc entrant en y pour chaque boule $B \in F$ contenant y, et il y a un arc sortant de chaque x pour tous les $y \in B(x)$. (C'est un graphe orienté avec des boucles !) Les deux sommes comptent différemment le nombre d'arcs de ce graphe : une somme compte les flèches, l'autre les queues des arcs.

Donc, il existe un $p \in V$ au moins égale à la moyenne, soit $f_i \cdot t/n$.

On a donc :

$$\begin{aligned}
f_0 &= n \\
f_1 &\leqslant f_0 - f_0 \cdot t/n = f_0 \cdot (1 - t/n) \\
f_2 &\leqslant f_1 - f_1 \cdot t/n = f_1 \cdot (1 - t/n) = f_0 \cdot (1 - t/n)^2 \\
\ldots &\leqslant \ldots \\
f_i &\leqslant f_0 \cdot (1 - t/n)^i \quad \forall i \geqslant 0
\end{aligned}$$

L'algorithme s'arrête au premier i_0 tel que $f_{i_0} < 1$, et alors $|P| = i_0$. En admettant que pour tout $0 < \alpha \leqslant 1/2$ et $\beta > 0$, $(1-\alpha)^\beta < e^{-\alpha\beta}$ (car pour $n \geqslant 2$, $(1-1/n)^n < e^{-1}$), on a alors ($\alpha \leqslant 1/2$ est vérifié puisque $t \leqslant n/2$) :

$$\begin{aligned} (1-t/n)^{i_0} &< e^{-t\cdot i_0/n} \\ f_0 \cdot (1-t/n)^{i_0} &\leqslant f_0/e^{t\cdot i_0/n} \end{aligned}$$

et donc $f_0/e^{t\cdot i_0/n} < 1$ implique que $i_0 > (n \ln f_0)/t$. L'algorithme s'arrête en particulier à la fin de l'étape $i_0 = \lfloor (n \ln n)/t \rfloor + 1$.

On associe $x \mapsto p(x)$ = l'élément de P le plus proche de x (forcément $p(x) \in B(x)$), et soit T_p un arbre de plus courts chemins couvrant G enraciné en $p \in P$. On note $\lambda(y, T_p)$ l'étiquette de routage du sommet y dans l'arbre T_p. On a vu que cette étiquette est au plus sur $O(\log^2 n / \log\log n)$ bits.

Adresse d'un sommet : $\text{label}(u) = (u, p(u), \lambda(u, T_{p(u)}))$. La taille est donc de $O(\log^2 n / \log\log n)$ bits.

Algorithme de routage de x vers y :

1. Si $y \in B(x)$, alors router vers y sur une arête de plus court chemin.
2. Sinon, router vers y en utilisant $T_{p(y)}$.

Table d'un sommet x :

- son adresse $\text{label}(x)$;
- $B(x)$ et une table de routage pour tous les sommets de $B(x)$;
- $\lambda(x, T_p)$ pour tous les $p \in P$.

Au total, cela fait $O(t \log n + (n \ln n)/t \cdot \log^2 n / \log\log n)$ bits. En choisissant $t = \sqrt{n/\log\log n} \cdot \log n$, on obtient une taille de table de routage de complexité :

$$\sqrt{n/\log\log n} \cdot \log^2 n + (n \ln n)/(\sqrt{n/\log\log n} \cdot \log n) \cdot \log^2 n / \log\log n \log^2 n) = 2\sqrt{n} \cdot \log^2 n / \sqrt{\log\log n}$$

Étirement : Soit $d = d(x, y)$ la distance entre deux sommets quelconques. Si $y \in B(x)$, alors l'étirement est de 1 par la propriété de monotonicité des boules. Supposons que $y \notin B(x)$. $d(x, p(x)) = d(x, P) \leqslant d$ car $d(x, y) < d(x, P)$ est impossible (sinon un point plus proche que $p(x) \in B(x)$ ne serait pas dans $B(x)$: une contradiction.

Donc, par l'inégalité triangulaire, $d(p(x), y) \leqslant d(p(x), x) + d(x, y) \leqslant 2d$. Maintenant $d(y, p(y) \leqslant d(y, p(x))$ car $p(y)$ est le plus proche sommet de P de y. Donc $d(y, p(y)) \leqslant 2d$.

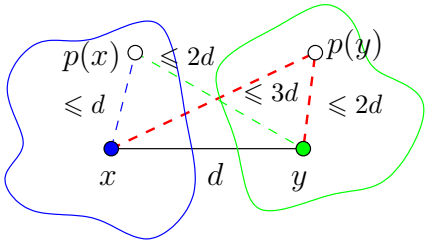


FIGURE 3.12 – Étirement $\leqslant 5$ entre x et y.

Au final, $d(x, p(y)) \leqslant d(x, y) + d(y, p(y)) \leqslant 3d$, et donc la route est de longueur au plus $d(x, p(y)) + d(p(y), y) \leqslant 3d + 2d = 5d$. L'étirement est donc $\leqslant 5$ dans tous les cas. En fait, on peut démontrer que la longueur de la route de x à y plus la longueur de la route de y à x ne dépasse pas 6 fois la distance $d(x, y)$. Autrement dit, en moyenne, l'étirement ne dépasse pas 3. □

Pour montrer que le cas décrit par la figure 3.12 (étirement 5) malheureusement arrive, il faut considérer un sommet x de degrés $> t$ ($t = 4$ dans la figure 3.13) et un sommet y voisin de x de degrés $t - 2$ avec un sommet $p(y)$ a distance 2 de y. De plus il y a deux chemins disjoints de x à $p(y)$, $y \notin B(x)$ et $B(y)$ comprend x et $p(y)$.

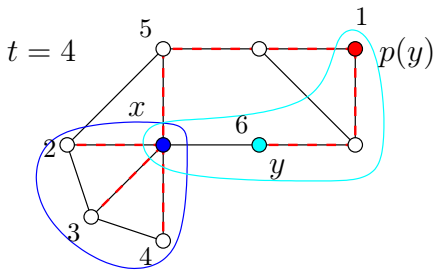


FIGURE 3.13 – Un exemple où l'étirement est de 5 pour la route de x à y.

La preuve précédente marche même si le graphe est pondéré (valeurs $\geqslant 0$ sur les arêtes), puisqu'au final seule l'inégalité triangulaire est utilisée pour borner l'étirement.

On remarque aussi que le schéma précédent à la particularité de générer une fonction de routage simple. On parle parfois de schéma de routage *loop-free* ou encore « direct ». Ce n'est pas le cas du schéma ayant un facteur d'étirement 3 et qui a une taille mémoire en $\sqrt{n}$. Il n'est pas difficile de voir que tout schéma de routage *loop-free* nécessite une taille mémoire en $\Omega(\sqrt{n})$ (si les adresses restent dans l'intervalles $[1, n]$), et ceci quel que soit l'étirement. En effet, tout schéma de routage *loop-free* dans les arbres doit être de plus courts chemins, et on a vu que dans ce cas des tables de $\Omega(\sqrt{n})$ bits étaient nécesaires dans le pire des cas (voir la section 3.5).

3.7 Variantes sur le routage compact

Il existe de nombreuses variantes pour le problème du routage compact. Par exemple, le routage dans les graphes pondérés et/ou orienté. On peut aussi se demander ce qu'il advient des résultats présentés précédemment si on impose label$(u) = u$, c'est-à-dire si, lorsqu'on construit la table de routage, on ne peut pas optimiser les adresses. C'est le concept de schéma de routage avec *indépendance des noms*.

Quelques résultats ? Indépendance des noms (dans les arbres) ?

Partitions éparses. S'il existe une partition vérifiant 1) 2) 3) alors on peut router avec X mémoire et Y étirement.

Bibliographie

[BHV99] H. Buhrman, J.-H. Hoepman, and P. M. Vitányi, *Space-efficient routing tables for almost all networks and the incompressibility method*, SIAM Journal on Computing, 28 (1999), pp. 1414–1432. doi : 10.1137/S0097539796308485.

[DGJP93] F. Desprez, C. Gavoille, B. Jargot, and M. Pourzandi, *Tests des performances des communications de la machine Volvox IS-860*, La lettre du Transputer et des Calculateurs Parallèles, 19 (1993), pp. 11–35.

[EGP03] T. Eilam, C. Gavoille, and D. Peleg, *Compact routing schemes with low stretch factor*, Journal of Algorithms, 46 (2003), pp. 97–114. doi : 10.1016/S0196-6774(03)00002-6.

[FG97] P. Fraigniaud and C. Gavoille, *Universal routing schemes*, Distributed Computing, 10 (1997), pp. 65–78. doi : 10.1007/s004460050025.

[FG01] P. Fraigniaud and C. Gavoille, *Routing in trees*, in 28th International Colloquium on Automata, Languages and Programming (ICALP), F. Orejas, P. G. Spirakis, and J. van Leeuwen, eds., vol. 2076 of Lecture Notes in Computer Science, Springer, July 2001, pp. 757–772. doi : 10.1007/3-540-48224-5_62.

[FJ88] G. N. Frederickson and R. Janardan, *Designing networks with compact routing tables*, Algorithmica, 3 (1988), pp. 171–190. doi : 10.1007/BF01762113.

[FP94] E. Fleury and M. Picquendar, *Massivelly parallel machine based on T9000 and C104*, Research Report 94-14, LIP, École Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France, May 1994. (short version in Transputers'94).

[Gav00] C. Gavoille, *A survey on interval routing*, Theoretical Computer Science, 245 (2000), pp. 217–253. doi : 10.1016/S0304-3975(99)00283-2.

[GG01] C. Gavoille and M. Gengler, *Space-efficiency of routing schemes of stretch factor three*, Journal of Parallel and Distributed Computing, 61 (2001), pp. 679–687. doi : 10.1006/jpdc.2000.1705.

[GH99] C. Gavoille and N. Hanusse, *Compact routing tables for graphs of bounded genus*, in 26th International Colloquium on Automata, Languages and Programming (ICALP), J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., vol. 1644 of Lecture Notes in Computer Science, Springer, July 1999, pp. 351–360. doi : 10.1007/3-540-48523-6_32.

[GP96a] C. Gavoille and S. Pérennès, *Lower bounds for interval routing on 3-regular networks*, in 3rd International Colloquium on Structural Information & Communication Complexity (SIROCCO), N. Santoro and P. G. Spirakis, eds., Carleton University Press, June 1996, pp. 88–103.

[GP96b] C. Gavoille and S. Pérennès, *Memory requirement for routing in distributed networks*, in 15th Annual ACM Symposium on Principles of Distributed Computing (PODC), ACM Press, May 1996, pp. 125–133. doi : 10.1145/248052.248075.

[Inm91] C. Inmos, *The T9000 Transputer Products Overview Manual*, 1991.

[MT90] M. May and P. Thompson, *Transputers and routers : Components for concurrent machines*, in Transputers and Routers : Function, Performance and Applications, INMOS Ltd., April 1990, ch. 1.

[MTW93] M. May, P. Thompson, and P. H. Welch, *Networks, Routers and Transputers : Function, Perfomance, and Applications*, IOS Press, Netherlands, February 1993. isbn : 90-5199-129-0.

[Pel00] D. Peleg, *Distributed Computing : A Locality-Sensitive Approach*, SIAM Monographs on Discrete Mathematics and Applications, 2000. isbn : 0-89871-464-8. doi : 10.1137/1.9780898719772.

[SK85] N. Santoro and R. Khatib, *Labelling and implicit routing in networks*, The Computer Journal, 28 (1985), pp. 5–8. doi : 10.1093/comjnl/28.1.5.

[TZ01] M. Thorup and U. Zwick, *Compact routing schemes*, in 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), ACM Press, July 2001, pp. 1–10. doi : 10.1145/378580.378581.

[War93] R. Warlimont, *Factorisatio numerorum with constraints*, Journal of Number Theory, 45 (1993), pp. 186–199. doi : 10.1006/jnth.1993.1071.

CHAPITRE 4

Structures de données compactes et distribuées

Sommaire

Mots clés et notions abordées dans ce chapitre :

- schéma d'étiquetage, adjacence, distance, ancêtre, plus petit ancêtre commun
- dégénérescence, arboricité, graphe universel
- graphes séparables, décomposition et largeur arborescente, mineur

Un complément d'information sur certaines notions abordées dans ce chapitre peut être trouvé dans [Pel00a].

4.1 Généralités

Dans la suite, on appellera *propriété de graphe* toute fonction définie sur des paires de sommets d'un graphe. Un exemple de propriété de graphe pourrait être : $P(x, y, G) =$ nombre de chemins entre x et y dans G. C'est pour simplifier. Tout ce qui est fait dans ce chapitre pourrait être généralisé à des propriétés concernant des sous-ensembles plus grands de sommets ou/et d'arêtes.

Définition 9 (P-schéma d'étiquetage) *Soit P une propriété de graphe. Un* P-schéma d'étiquetage *pour une famille de graphes $\mathcal{F}$ est une paire (L, f) de fonctions vérifiant, pour tout graphe $G \in \mathcal{F}$,*

- *pour tout $x \in V(G)$, $L(x, G)$ est une étiquette*[1]

1. En principe, les étiquettes seront des chaînes binaires mais pas toujours ! les chaînes binaires étant bien pratiques pour mesurer leur taille.

- *pour tous* $x, y \in V(G)$, $f(L(x,G), L(y,G)) = P(x,y,G)$.

*L est appelée l'*étiquetage*, et f le* décodeur *ou* marqueur.

L'objectif est de construire, pour une propriété P donnée, des P-schémas d'étiquetage utilisant les étiquettes les plus courtes possibles. La mesure d'efficacité d'un schéma d'étiquetage est ainsi la longueur de la plus grande étiquette utilisée par le schéma. (Notons qu'on aurait aussi pu choisir la longueur *moyenne* des étiquettes utilisées par les sommets d'un graphes, plutôt que la longueur *maximum*).

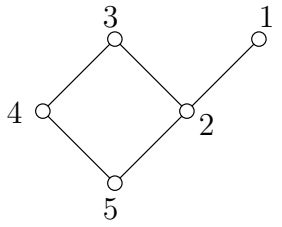
Quelques exemples d'applications

- routage à base d'adresse
- distance pour le routage économique sans table
- descendance dans les arbres pour les bases de données

4.2 Étiquetage d'adjacence

Dans cette section, la propriété P est définie par :

$$P(x,y,G) = \begin{cases} 1 & \text{si } x \text{ et } y \text{ sont adjacents dans } G \\ 0 & \text{sinon} \end{cases}$$



$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ & 0 & 1 & 0 & 1 \\ & & 0 & 1 & 0 \\ & & & 0 & 1 \\ & & & & 0 \end{pmatrix}$$

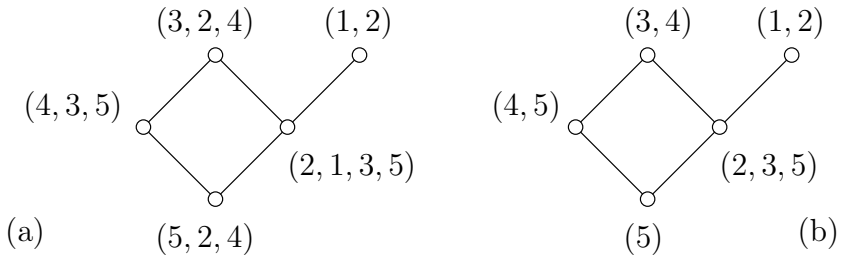


FIGURE 4.1 – Schémas d'adjacence : (a) par liste d'adjacence ; (b) par liste tronquées.

Pour cette propriété là, plusieurs schémas simples peuvent être envisagés. En voici quatre.

Schéma 1 : $L(x, G) = (x, M)$, $L(y, G) = (y, M)$.
Décodeur : `return M[x,y]`.
Taille : $n(n-1)/2 + \lceil \log n \rceil = O(n^2)$ bits.

Schéma 2 : $L(x, G) = (x, v_1, \ldots, v_d)$ où $v_1, \ldots, v_d$ sont les voisins de x.
Décodeur : on vérifie si y est dans la liste des voisins de x.
Taille : $(d+1)\lceil \log n \rceil = O(n \log n)$ bits dans le pire des cas.

Schéma 3 : $L(x, G) = (x, v'_1, \ldots, v'_k)$ avec $k \leqslant \deg(x)$ (par listes tronquées).
Décodeur : on vérifie si y est dans la liste de x ou si x est dans la liste de y.
Taille : on peut espérer au mieux diviser par deux la taille des listes, mais cela reste en $O(n \log n)$ dans le pire des cas.

Schéma 4 : $L(x, G) = (x, M_x)$ où M_x dénote la $x$ème ligne de M dans laquelle les bits de colonne $\leqslant x$ ont été supprimés, la matrice étant symétrique.
Décodeur : `return M_x[y]` si $x < y$, sinon `return M_y[x]`.
Taille : $n - x + \lceil \log n \rceil = n + O(\log n)$ bits pour les première lignes (x petits).

Proposition 11 *Tout schéma d'adjacence sur la famille des graphes à n sommets nécessite une étiquette de taille au moins $(n-1)/2$ bits.*

Preuve. ... □

On observe qu'en moyenne les étiquettes utilisées par le schéma 4 sont de taille $n/2 + \lceil \log n \rceil$ bits, atteignant ainsi la borne inférieure donnée par la proposition 11 (à un facteur additif près de $O(\log n)$). À la question de savoir s'il est possible d'atteindre $n/2$ bits dans le pire des cas, la réponse est donnée par la proposition 12.

Notons bien que le pire des cas souligné par la borne inférieure (proposition 11) n'arrive pas toujours dans la pratique, même s'il est facile de voir qu'un graphe aléatoire (n sommets et l'on tire avec probabilité $1/2$ les arêtes entre toutes paires de sommets $\{x, y\}$) nécessite précisément des étiquettes de taille $\geqslant n/2 - O(\log n)$. Si le modèle de graphe ne permet pas de capturer précisément les propriétés d'un objet particulier, il faut alors utiliser une compression standard générique : `gzip` par exemple, qui éventuellement sera tirer profit de certaine redondance. Cependant l'étape de modélisation et l'étude du pire des cas dans le modèle fixé est nécessaire, puisque en pratique il sera bien évidemment toujours plus intéressant de faire `gzip` sur une chaîne binaire de taille $n/2$ plutôt que sur n bits.

Proposition 12 *La famille des graphes à n sommets possède un schéma d'adjacence avec des étiquettes de taille au plus $\lceil (n-1)/2 \rceil + \lceil \log n \rceil$ bits.*

Preuve. Soit M la matrice d'un graphe G. Pour simplifier on supposera que les indices de la matrices vont de 0 à $n-1$, et donc que $V(G) = \{0, \ldots, n-1\}$. L'étiquette du sommet

i dans G est $L(i,G) = (i, M_i^k)$ où M_i^k est la chaîne binaire obtenue en concaténant les k bits suivant :

$$M[i,(i+1) \bmod n], M[i,(i+2) \bmod n], \ldots, M[i,(i+k) \bmod n] \ .$$

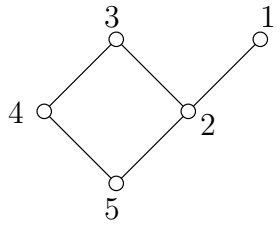
Autrement dit on prend les k bits de la ligne i de M à partir de la colonne $i+1$, éventuellement en continuant à partir de la colonne 0 si la dernière colonne est atteinte. k est un paramètre que l'on va optimiser.

Chaque sommet stocke $k + \lceil \log n \rceil$ bits, le sommet i stockant le voisinage d'exactement pour k autres sommets. Notons $[\![a,b]\!] = \bigcup_{i=a}^{i=b} (i \bmod n)$. Avec cette notation, l'ensemble des sommets dont i connaît le voisinage (le bit 0 ou 1 M), i compris, est $[\![i, i+k]\!]$ comprenant $k+1$ valeurs. On peut visualiser $[\![i, i+k]\!]$ comme un arc sur le cercle $0, \ldots, n-1$. Il est clair que si $[\![a,b]\!]$ et $[\![c,d]\!]$ s'intersectent alors soit $c \in [\![a,b]\!]$ ou bien $a \in [\![c,d]\!]$, c'est-à-dire que l'intersection concerne forcément l'un des bords gauche de l'arc de cercle. Si $2(k+1) > n$ alors, pour toute paire $\{i,j\}$, les arcs $[\![i, i+k]\!]$ et $[\![j, j+k]\!]$ s'intersectent car chacun des arcs comportes $k+1 > n/2$ éléments. Par conséquent $j \in [\![i, i+k]\!]$ ou $i \in [\![j, j+k]\!]$.

En particulier, en fixant $k = \lceil (n-1)/2 \rceil$, soit le bit concernant j est stocké dans $L(i,G)$ soit c'est le contraire, à savoir le bit concernant i est stocké dans $L(j,G)$.

Le test peut s'écrire : si $i = j$, alors renvoyer 0, sinon, si $j \in [\![i, i+k]\!]$ alors renvoyer $M_i^k[(j-i+n-1) \bmod k]$, sinon renvoyer $M_j^k[(i-j+n-1) \bmod k]$, en supposant que les indices de la chaîne M_i^k vont de 0 à $k-1$. On laisse au lecteur le soin d'écrire le test $j \in [\![i, i+k]\!]$.

Voilà ce que cela donne sur un exemple concrêt, avec $n = 5$ et $k = \lceil (n-1)/2 \rceil = 2$:



$$M = \begin{pmatrix} 0 & \underline{1} & \underline{0} & 0 & 0 \\ 1 & 0 & \underline{1} & \underline{0} & 1 \\ 0 & 1 & 0 & \underline{1} & \underline{0} \\ \underline{0} & 0 & 1 & 0 & \underline{1} \\ \underline{0} & \underline{1} & 0 & 1 & 0 \end{pmatrix} \qquad \begin{array}{l} L(1,G) = (1,10) \\ L(2,G) = (2,10) \\ L(3,G) = (3,10) \\ L(4,G) = (4,10) \\ L(5,G) = (5,01) \end{array}$$

□

Définition 10 (*k*-dégénéré) *Un graphe G est* k-dégénéré *si tout sous-graphe de G possède un sommet de degré au plus k.*

Cela revient à dire que dans toute sous-matrice de la matrice de G il existe une ligne avec $\leqslant k$ entrée à '1'.

Remarques :

- Si G est k-dégénéré, alors $k \geqslant$ au degré minimum de G.
- Les arbres sont 1-dégénérés, il existe toujours une feuille.

Proposition 13 *La famille des graphes k-dégénérés à n sommets possède un schéma d'adjacence avec des étiquettes de $(k+1)\lceil \log n \rceil$ bits.*

Preuve. On « épluche » le graphe G par un sommet de degré minimum. Pour tout $i \in \{1, \ldots, n\}$, soit v_i le sommet de degré minimum du graphe $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$. $G_1 = G$.

L'étiquette de v_i est $L(v_i, G) = (v_i, N_i)$ où N_i est l'ensemble des voisins de v_i dans G_i. Les graphes G_i sont des sous-graphes (en fait induits) de G, par conséquent, $|N_i| \leqslant k$. Les étiquettes sont représentées par des suites d'au plus k entiers $\in \{1, \ldots, n\}$, donc de taille $(k+1)\lceil \log n \rceil$ bits au plus. En fait, on peut faire $O(k \log(n/k))$ bits en codant N_i en tant qu'ensemble plutôt que suite ordonnée.

Soient v_i et v_j deux sommets dans cet épluchage. Montrons que l'on a :

$$v_i \text{ adjacent à } v_j \quad \Longleftrightarrow \quad v_i \in N_j \text{ ou } v_j \in N_i$$

ce qui donne un test immédiat pour l'adjacence entre deux sommets d'étiquettes $L(x, G)$ et $L(y, G)$.

$\Leftarrow$ Si $v_i \in N_j$ alors v_i est adjacent à v_j car N_j est un sous-ensemble de voisins de v_j. De même si $v_j \in N_i$.

$\Rightarrow$ Supposons maintenant que v_i est adjacent à v_j, et sans perte de généralité, supposons que $i < j$, c'est-à-dire que v_i a été supprimé de G avant v_j. Le graphe G_i, le sous-graphe induit de G avant la suppression de v_i, contient v_i et v_j, et donc l'arête (v_i, v_j). Par conséquent, $v_j \in N_i$, ensemble contenu dans l'étiquette de v_i. □

Il est important d'observer que tout graphe de m arêtes et n sommets possède un sommet de degré au plus $2m/n$. En effet, en utilisant un argument de moyenne on a : $S = \sum_{x \in G} \deg(x) = 2m$. Donc il existe un sommet x_0 avec $\deg(x_0) \leqslant S/n = 2m/n$, sinon la somme S serait $> 2m$.

Remarques :

- Pour un arbre, $m = n - 1$ et donc on retrouve $k \leqslant \lfloor 2(n-1)/n \rfloor = 1$.
- Pour un graphe planaire, $m \leqslant 3n - 6$ et donc $k \leqslant \lfloor 6 - 12/n \rfloor = 5$.
- Pour les graphes planaires extérieurs, $m \leqslant 2n - 3$, donc $k \leqslant \lfloor 4 - 6/n \rfloor = 3$. En fait, on peut montrer que les planaires extérieurs sont 2-dégénérés (voir la proposition 14).
- De manière générale, pour les graphes dessinés sur des surfaces de genre g (pour les graphes planaires $g = 0$), $m \leqslant 3n + g - 6$, donc $k \leqslant \lfloor 6 - (2g+12)/n \rfloor = 5$ si $2g + 12 < n$, c'est-à-dire si $g < n/2 - 6$. Ceci explique pourquoi les matrices utilisées pour les triangulations de surface bornée sont creuses.

Pour les arbres, la proposition 13 donne des étiquettes d'adjacence de $2\lceil \log n \rceil$ bits (voir exemple figure 4.2).

Proposition 14 *Les graphes planaires extérieures sont 2-dégénérés.*

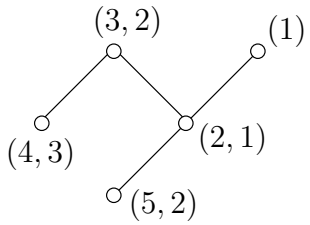


FIGURE 4.2 – Proposition 13 dans le cas d'un arbre.

Preuve. *(Preuve basée uniquement sur la définition « géométrique » des graphes planaires extérieures)* On montre que le degré minimum de tout planaire extérieur est au plus deux. On dessine le graphe sur le plan avec tous les sommets sur une face. On suppose qu'il est maximal pour le nombre d'arête, cela ne pouvant diminuer son degré minimum. On observe alors qu'entre deux sommets x et y, si l'on tourne dans le sens trigonométrique en suivant la face extérieure, les chemins P de x à y et Q de y à x sont disjoints (à part en x et en y) et passent par tous les sommets, c'est-à-dire $P \cup Q$ induit un cycle hamiltonien. En effet, sinon il existerai un sommet $z \notin \{x, y\}$ appartenant à P et Q. Dans ce cas il est facile d'ajouter une arête entre le précédesseur de z sur P et son successeur (sur P) ce qui mène à une contradiction.

Une arête est une *corde* si elle n'appartient pas au cycle hamiltonien. On observe alors que tout sommet de degré 3 ou plus possède au moins une corde. Dit autrement, tout sommet non incident à une corde est de degré au plus deux. Notons que l'énoncé de la proposition est vrai s'il y a $n < 3$ sommets. On montre alors par induction que dans tout graphe planaire extérieur d'au moins $n \geqslant 3$ sommets possédant un cycle hamiltonien C avec deux sommets x, y distingués adjacents dans C, il existe un sommet $z \notin \{x, y\}$ de degré deux. En effet, x a deux voisins sur le cycle : y et disons x'. Comme $n \geqslant 3$ et comme il n'y a pas d'arête multiples, $x' \neq y$. Si x' n'a pas de corde, alors il est de degré deux, et c'est fini $z = x'$. Sinon, posons y' le voisin de x' sur cette corde. On observe que la corde $x'y'$ coupe deux graphes dont l'un est planaire extérieur strictement plus petit (sans le sommet x) possédant un cycle hamiltonien et deux sommets marqués x' et y' voisins dans ce cycle. Par induction il existe un sommet $z \notin \{x', y'\}$ de degré deux. □

Théorème 10 (Alstrup-Rauhe [AR02]) *La famille des arbres à n sommets possède un schéma d'adjacence avec des étiquettes de $\log n + O(\log^* n)$ bits, où $\log^* n = \min\{i \mid \log^{(i)} n \leqslant 1\}$.*

Notons que pour $n = 2^{65\,536} = 2^{2^{16}} = 2^{2^{2^4}} = \cdots = 2^{2^{2^{2^{2^1}}}}$, soit un nombre de 19 729 chiffres, $\log^* n = 5$ alors qu'il n'y que 2^{400} particules élémentaires dans l'univers. La fonction $\log^* n$, bien que non bornée, est donc très très petite. En pratique, pour des valeurs de n

raisonables, les constantes cachées derrières la notation O sont bien plus grandes que la valeur $\log^* n$ elle-même.

En fait, en utilisant un autre concept, « l'arborescence », on obtient toujours de meilleurs résultats qu'avec la « dégénérescence ».

Définition 11 (t-arboré) *Un graphe est* t-arboré *s'il existe une partition de ses arêtes en* t *ensembles, chaque ensemble induisant une forêt.*

FIGURE 4.3 – Partition en trois et deux forêts de K_4.

Proposition 15 *Soit* G *un graphe et soient* t *et* k *les valeurs minimum telles que* G *est* t*-arboré et* k*-dégénéré. Alors* $t \leqslant k \leqslant 2t-1$.

Preuve. Supposons que G soit t-arboré. Alors tout sous-graphe H de G a au plus $t \cdot (|V(H)|-1)$ arêtes. Donc H possède un sommet de degré $k \leqslant 2t \cdot (|V(H)|-1)/|V(H)| < 2t$. Donc $k \leqslant 2t-1$.

Supposons que G soit k-dégénéré, et soit v_i le sommet de degré minimum dans le graphe $G_i = G \setminus \{v_1, \ldots, v_{i-1}\}$, $i \in \{1, \ldots, n\}$ où $n = |V(G)|$. On a que G_n, graphe réduit à un seul sommet, est 1-arboré et donc k-arboré car $k \geqslant 1$ (on suppose qu'il y a au moins une arête dans G). Par induction, supposons que G_i est k-arboré, et considérons l'ajoût du sommet v_{i+1} pour former G_{i+1}. Il suffit alors de mettre chacune des arêtes de v_{i+1} dans G_{i+1} dans une partition différentes (il y a au plus k arêtes) évitant la formation de tout cycle avec les forêts de G_i. Donc G_{i+1} est k-arboré, ce qui montre que $G_1 = G$ est t-arboré avec $t \leqslant k$. □

Remarques :

- Les arbres sont évidemment 1-arborés.
- Les grilles $2D$ sont 2-arborées, car 2-dégénérées et $t \neq 1$.
- Les graphes planaires extérieurs sont 2-arborés (car $k = 2$ et $t \neq 1$)
- Les graphes planaires sont 3-arborés (non trivial).
- L'arboricité d'un graphe peut être calculée en temps polynomial.

En fait, la forme générale du théorème 10 est plutôt celle-ci.

Théorème 11 (Alstrup-Rauhe [AR02]) *La famille des graphes t-arborés à n sommets possède un étiquetage d'adjacence avec des étiquettes de taille $t \log n + O(\log^* n)$ bits.*

[Notons que ce n'est pas $O(t \log^* n)$. En effet, l'étiquette pour un arbre/forêt est composée d'un identifiant $\in [1, n]$ pouvant être extrait de l'étiquette. Donc on stocke une fois l'étiquette complète d'un arbre, puis l'identifiant du parent des $t-1$ autres arbres/forêts.]

En pratique la longueur des étiquettes est la somme des longueurs des étiquettes dans chacune des forêts, celles-ci ne couvrant pas forcément tous les sommets. Ceci dit, pour gagner un facteur deux sur le terme $\log n$, il faut des forêts couvrant seulement $\sqrt{n}$ sommets, et donc cela signifie que k est très grand (ici $\geqslant \sqrt{n}$). Donc on peut espérer gagner seulement si k est très grand, pour k « petit » on ne gagnera pas grand chose.

Le théorème 11 donne toujours de meilleurs résultats que la proposition 13, puisque l'arboricité est $\leqslant$ à la dégénérescence par la proposition 15. Par exemple, pour les graphes planaires, la proposition 13 donne des tailles d'étiquettes de $6 \log n$, alors que le théorème 11 implique des tailles de $3 \log n + O(\log^* n)$ bits. Pour les arbres de degré interne maximum $\hat{d}$, c'est-à-dire le degré maximum de l'arbre obtenu en supprimant tous les sommets de degré 1, il existe un schéma de $\log n + O(\log \hat{d})$ bits [BGL06]. En remarquant que les arêtes d'un graphe planaire peuvent être partionnées en trois forêts, dont l'une de degré bornée (et donc avec un schéma en $\log n + O(1)$), on peut améliorer le schéma des planaires en $3 \log n + O(1)$. Récemment, la borne pour les planaires, et de manière beaucoup plus générale pour les graphes excluant un mineur fixé, a été améliorée en $2 \log n + O(\log \log n)$ [GL07].

Dans les graphes planaires extérieurs, et de manière plus générale pour les graphes de largeur arborescente bornée, la borne de $\log n + O(\log \log n)$ bits peut être atteinte [GL07]. Notons que même pour les graphes planaires, la meilleure borne inférieure connue est $\log n + \Omega(1)$, en utilisant le fait que le nombre de graphes planaires étiquetés à n sommets est assymptotiquement est $n! 2^{\Theta(n)}$. La suite des n étiquettes peut servir à coder chaque graphes étiquetés, donc il existe une étiquette de taille $\geqslant \frac{1}{n} \log(n! 2^{\Theta(n)}) \approx \log n + O(1)$.

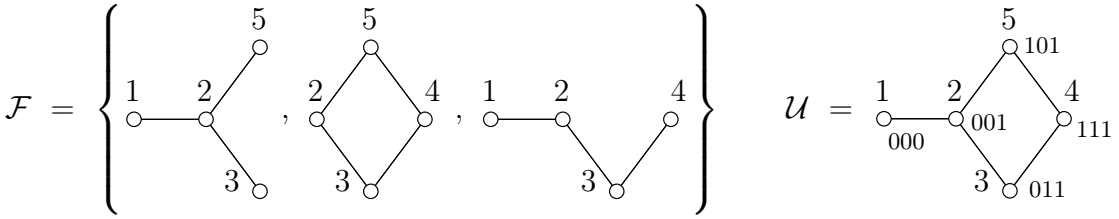
Il pourrait donc se produire que pour les graphes planaires, et en particulier pour les planaires extérieurs et les arbres, des étiquettes de taille $\log n + O(1)$ sont possibles. On pourrait même conjecturer que toute famille héréditaire [2] de graphes à au plus n sommets et ayant $n! 2^{O(n)}$ éléments possède un étiquetage d'adjacence de taille $\log n + O(1)$. On a besoin de l'« héréditarité », puisque sinon on peut toujours considérer un graphe à n sommets avec une partie quelconque de $\sqrt{n}$ où des étiquettes de taille $\sqrt{n}/2$ sont nécessaires par la proposition 11. Or il y a au plus $n! 2^{\sqrt{n}^2} = n! 2^n$ tels graphes étiquetés ... Cependant, en considérant la famille des graphes obtenus à partir d'arbres enracinés et en ajoutant les arêtes entre frères, on peut montrer [ABR05] que des étiquettes de $\log n + \Theta(\log \log n)$ sont nécessaires et suffisantes. Or cette classe est close par sous-graphe induit. Il faut donc une réstriction supplémentaire.

2. Une famille de graphe $\mathcal{F}$ est *héréditaire* si elle est close par sous-graphe induit.

Problème ouvert : démontrer ou infirmer que toute famille héréditaire de graphes à au plus n sommets, $O(n)$ arêtes et $n!2^{O(n)}$ graphes étiquetés possède un étiquetage d'adjacence de taille $\log n + O(1)$.

Définition 12 *$\mathcal{U}$ est un graphe* universel *pour une famille de graphe $\mathcal{F}$ si tout graphe de $\mathcal{F}$ est isomorphe à un sous-graphe induit de $\mathcal{U}$.*

Exemple :



On peut aussi numéroter les sommets de $\mathcal{U}$ avec 3 bits avec l'algorithme d'adjacence suivant : retourner 1 ssi les étiquettes diffèrent exactement d'un bit. Cela marche aussi pour la distance (retourner la distance de Hamming entre deux étiquettes).

But : minimiser le nombre de sommet de $\mathcal{U}$.

Motivation : circuits intégrés universels.

Ici on supposera que tout étiquetage doit produire des étiquettes différentes pour chacun des sommets du graphe. On pourrait imaginer qu'il faut une seule étiquette pour un graphe complet par exemple.

Proposition 16 *Si une famille $\mathcal{F}$ possède un schéma d'adjacence avec des étiquettes différentes d'au plus k bits, alors $\mathcal{F}$ possède un graphe universel à $2^{k+1} - 1$ sommets. Inversement, si $\mathcal{F}$ possède un graphe universel à N sommets, alors $\mathcal{F}$ possède un schéma d'adjacence avec des étiquettes d'au plus $\lceil \log N \rceil$ bits.*

Preuve. Soit (L, f) un étiquetage d'adjacence pour la famille $\mathcal{F}$. On construit le graphe $\mathcal{U}$ de la manière suivante : $V(\mathcal{U})$ est l'ensemble des chaînes binaires d'au plus k bits. $E(\mathcal{U})$ est l'ensemble des paires $\{\ell, \ell'\}$ telles qu'il existe un graphe $G \in \mathcal{F}$ et deux sommets x, y de G tels que $\ell = L(x, G)$, $\ell' = L(y, G)$ et $f(\ell, \ell') = 1$.

Il est clair que $|V(\mathcal{U})| = 2^{k+1} - 1$ soit le nombre de chaînes binaires de longueur au plus k. Montrons que $\mathcal{U}$ est un graphe universel pour $\mathcal{F}$. Soit G un graphe arbitraire de $\mathcal{F}$. Si x et y sont deux sommets de G, alors $L(x, G)$ et $L(y, G)$ sont deux sommets (distincts) de $\mathcal{U}$. Si $\{x, y\}$ est une arête de G alors $L(x, G)$ et $L(y, G)$ sont adjacents dans $\mathcal{U}$ par construction. Si x et y ne sont pas adjacents dans G, il ne peut exister de graphe H avec deux sommets adjacents u, v tels que $L(u, H) = L(x, G)$ et $L(v, H) = L(y, G)$, car sinon on aurait $f(L(u, H), L(v, H)) \neq f(L(x, G), L(y, G))$ ce qui est absurde puisque la fonction f est appliquée sur les mêmes étiquettes.

Inversement, supposons que $\mathcal{U}$ est un graphe universel pour $\mathcal{F}$ de N sommets. Donc pour tout $G \in \mathcal{F}$, G est isomorphe à un sous-graphe induit de $\mathcal{U}$, disons H. Soit $\ell_G : V(G) \to V(H)$ l'isomorphisme de G sur H. On définit $L(x, G) = \ell(x)$. $\ell(x)$ est un sommet de $\mathcal{U}$ qui peut être représenté par un entier entre 1 et N ou encore une chaîne binaire de longueur $\lceil \log n \rceil$. La fonction f est définit par : $f(\ell(x), \ell(y)) = 1$ si et seulement si $\ell(x)$ et $\ell(y)$ sont adjacent dans $\mathcal{U}$. Ceci termine la preuve. □

Remarquons que le schéma (L, f) construit dans la preuve de la proposition 16 à partir du graphe universel $\mathcal{U}$ ne dépend que de la famille $\mathcal{F}$. Donc, avec la seule donnée $\mathcal{F}$, il est possible de construire un graphe universel pour $\mathcal{F}$ à N sommets, s'il existe bien sur. Il faut prendre le plus petit possible, éventuellement y ajouter des sommets pour en avoir N exactement. Cette construction de schéma d'étiquetage à partir de $\mathcal{F}$ est évidemment inefficace en pratique, puisque pour la plupart des familles de graphes $\mathcal{F}$ à n sommets intéressantes possèdent un nombre exponentiel de graphes. Donc trouver le graphe universel le plus petit, bien que calculable, est une tâche très couteuse. Le calcul des étiquettes et le test d'adjacence sont donc particulière long par cette méthode, très générale, qui a le seul mérite de montrer que la minimisation des étiquettes et du nombre de sommets d'un graphe universelle sont un même problème.

4.3 Étiquetage de distance

Dans cette section, la propriété P est définie par $P(x, y, G) = d_G(x, y)$ pour tous sommets x, y de G. C'est donc une généralisation de l'adjacence.

4.3.1 Les arbres

Théorème 12 (Peleg [Pel00b]) *Les arbres à n sommets possèdent un étiquetage de distance avec des étiquettes de $O(\log^2 n)$ bits (et $\log^2 n$ est la meilleure complexité possible).*

Preuve. On ne démontre ici que la borne supérieure : l'existence d'un schéma avec étiquette en $O(\log^2 n)$ bits. Pour la borne inférieure se référer à [GPPR04]. En gros, il y est construit une famille d'arbres « quasi-binaires complets » de hauteur h et dont les arêtes sont valuées par des poids entiers pris dans l'intervalle $[1, M]$. Il est prouvé que tout schéma de distance sur cette famille (donc avec tous les poids possibles pour chacune les arêtes) nécessite au moins $M^{h/2}$ étiquettes différentes. Pour obtenir une famille d'arbres d'au plus n sommets on choisit $h = \lfloor \log \sqrt{n} \rfloor - 1$, $M = \lfloor \sqrt{n} \rfloor$, et on remplace chaque arête valuée $x \in [1, \lfloor \sqrt{n} \rfloor]$ par x arêtes (le nombre de sommets étant alors au plus de $(2^{h+1} - 1) \cdot M \leqslant n$). Pour un étiquetage utilisant le plus petit nombre d'étiquettes possible, chacune des étiquettes est utilisée sur un sommet et dans un arbre au moins, et donc un sommet se voit affecter d'une étiquette de taille au moins $\frac{1}{2} h \log M \approx \frac{1}{8} \log^2 n$ bits.

Borne supérieure. Soit T un arbre. À chaque sommet x on associe une suite de sommets $c(x) = (c_1, \ldots, c_k)$ et définie comme suit :

- c_1 est le centroïde de T ;
- c_2 est le centroïde de la composante connexe de $T \setminus \{c_1\}$ contenant x ;
- c_3 est le centroïde de la composante connexe de $T \setminus \{c_1, c_2\}$ contenant x ;
- ...
- $c_k = x$ si x est le centroïde de $T \setminus \{c_1, \ldots, c_{k-1}\}$.

Dans le même temps on définit la suite $d(x)$ des distances dans T de x à tous les sommets de $c(x)$:

$$d(x) \;=\; (d_T(x, c_1), \ldots, d_T(x, c_k)) \;.$$

Notons que la dernière valeur $d_T(x, c_k) = 0$. L'étiquette de x est tout simplement

$$\text{label}(x) \;=\; (c(x), d(x)) \;.$$

Calcule de la distance entre x et y. On suppose que sont données les étiquettes $\text{label}(x) = (c(x), d(x))$ et $\text{label}(y) = (c(y), d(y))$.

1. Calculer la longueur i du plus long préfixe commun entre $c(x)$ et $c(y)$.
2. Renvoyer $d(x)[i] + d(y)[i]$.

Le calcul est correct car si i est la longueur du plus long préfixe commun entre $c(x)$ et $c(y)$, alors c'est que x, y appartiennent à des composantes connexes différentes de $T \setminus \{c_1, \ldots, c_i\}$ et qu'ils appartiennent à la même composante connexe de $T \setminus \{c_1, \ldots, c_{i-1}\}$. Donc c_i est sur le (l'unique) chemin entre x et y dans T.

Taille des étiquettes. On remarque (par récurrence) que, pour chaque $i \in \{1, \ldots, k-1\}$, x appartient à une composante connexe de $T \setminus \{c_1, \ldots, c_i\}$ de taille $\leqslant n/2^i$. Comme pour $i = k-1$ la composante C comporte encore au moins un sommet, x, on a donc que

$$1 \;\leqslant\; |C| \;\leqslant\; n/2^{k-1}$$

ce qui implique $k \leqslant \log n + 1$.

Il suit que $c(x)$ et $d(x)$ comportent au plus $\log n + 1$ valeurs chacune codée sur $\lceil \log n \rceil$ bits, ce qui fait une taille d'étiquette d'au plus $2 \log^2 n + O(\log n)$. □

Un schéma d'étiquetage de distance (L, f) est *approximé à un facteur s près* pour le graphe G, si pour tous $x, y \in V(G)$,

$$d_G(x, y) \;\leqslant\; f(L(x, G), L(y, G)) \;\leqslant\; s \cdot d_G(x, y) \;.$$

Théorème 13 *Les arbres à n sommets possèdent un étiquetage de distance approximé à un facteur $1 + O(1/\log n) = 1 + o(1)$ près avec des étiquettes de $O(\log n \cdot \log\log n)$ bits (et $\log n \cdot \log\log n$ est la meilleure complexité possible).*

Preuve. On ne démontre ici que la borne supérieure : l'existence d'un schéma avec des étiquettes de $O(\log n \cdot \log \log n)$ bits. La borne inférieure est prouvée dans [GKK+01].

...

□

4.3.2 Graphes avec petits séparateurs

Il s'agit de généraliser le schéma des arbres.

Définition 13 (demi-séparateur) *Soit G un graphe à n sommets et $S \subseteq V(G)$. S est un* demi-séparateur *de G si toute composante connexe de $G \setminus S$ a au plus $n/2$ sommets.*

Définition 14 (k-séparable) *Un graphe est* k-séparable *si tous ses sous-graphes possèdent un demi-séparteur de taille au plus k.*

On peut vérifier que les arbres sont 1-séparables : nous avons en effet déjà montrer que tout sous-arbre, donc un arbre, possède un centroïde qui n'est rien de plus qu'un demi-séparateur.

Le problème de calculer le plus petit k tel que G est k-séparable est NP-complet, mais il existe un algorithme de complexité $O(2^{O(k^3)}n)$, donc exponentiel en k et linéaire en n, qui calcule cette valeur [Bod96]. Le meilleure algorithme d'approximation (polynomial) connu donne un facteur $O(\sqrt{\log k}\,)$ [FHL05]. Cependant les constantes cachées dans la notation « $O(.)$ » sont > 1000 et l'algorithme d'approximation reste très complexe.

Avant de décrire des familles de graphes k-séparables pour k « petit », montrons la construction générale en gardant à l'esprit que pour $k = 1$ il s'agit quasiment du schéma des arbres vu précédemment (théorème 12).

Théorème 14 *La famille des graphes k-séparables à n sommets possède un étiquetage de distance avec des étiquettes de $O(k \log^2 n)$ bits.*

Preuve. On utilise une décomposition en arbre du graphe de hauteur $O(\log n)$.

...

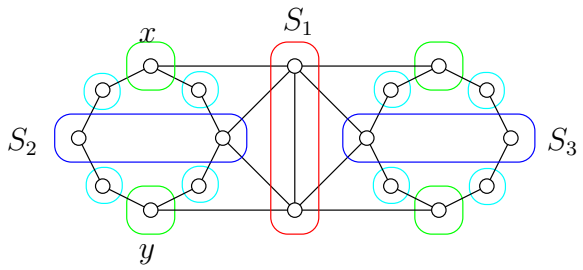
Les ensembles S_i partitionne les sommets du graphe. Soient $x \in S_i$ et $y \in S_j$.

Attention : le chemin de x à y peut passer par un séparateur ancêtre à S_i et S_j, mais qui n'est pas forcément le plus petit ancêtre commun (différent du schéma des arbres).

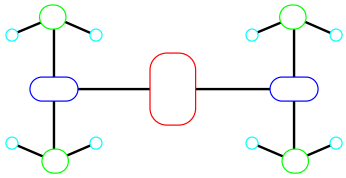
...

□

En fait, si $k = k(n)$ est une fonction de n suffisamment croissante (typiquement $k(n) = \Omega(n^\varepsilon)$ avec $\varepsilon \in]0, 1]$ constante), alors la taille des étiquettes est de l'ordre de $\sum_{i=0}^{\log n} k(n/2^i) \log n = O(k(n) \log n)$ au lieu de $k \log^2 n$.



(a) Partition des sommets en séparateurs



(b) Arbre de décomposition

FIGURE 4.4 – Le plus court chemin de x à y évite S_2 dans G, pourtant S_2 sépare x et y dans $G \setminus S_1$.

- Les graphes planaires sont $O(\sqrt{n}\,)$-séparables (Lipton-Tarjan [LT79]). Intuitivement, un graphe dessiné sur le plan a une surface de n, et donc, par analogie possède un diamètre (une coupe) en $O(\sqrt{n}\,)$. Notons que $\sqrt{n}$ est la meilleure complexité possible puisque tout demi-séparateur d'une grilles carrées à n sommets à une taille $\geqslant \sqrt{n}$.
- Les graphes de genre g sont $O(\sqrt{gn}\,)$-séparables [GHT84].

Définition 15 (mineur) *H est un* mineur *de G si H est un sous-graphe d'un graphe obtenu par contraction d'arêtes de G.*

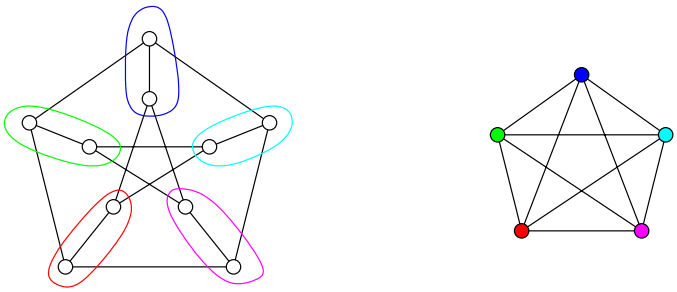


FIGURE 4.5 – Le graphe de Petersen possède K_5 comme mineur.

- Il faut un temps $O(|V(G)|^3)$ pour déterminer si H est un mineur de G [RS95].
- Les graphes qui excluent K_r (la clique à r sommets) sont $O(r^{3/2}\sqrt{n}\,)$-séparables, ce qui généralise le résultat de séparation des graphes planaires ($r = 5$). Ces graphes sont aussi $O(r\sqrt{n \log n}\,)$-séparables, et il est conjecturé qu'ils sont $O(r\sqrt{n}\,)$-séparables [AST90].
- Un graphe est planaire si et seulement s'il ne possède ni de K_5 ni de $K_{3,3}$ comme mineur. En fait, c'est équivalent à dire ni K_5 ni $K_{3,3}$ comme mineur *topologique* : H

est un mineur topologique de G s'il existe une subdivision des arêtes de H qui est un sous-graphe de G. Donc, si H est un mineur topologique de G, H est un mineur de G, mais le contraire n'est pas forément vrai. En tout cas, la famille des graphes sans K_5 comme mineur inclut tous les graphes planaires.

- Les arbres n'ont pas K_3 comme mineur.
- Les graphes planaires extérieurs n'ont ni K_4, ni $K_{3,2}$ comme mineurs (on remarque que la planarité extérieure est une propriété close par mineure, et ni K_4 ni $K_{3,2}$ ne sont planaires extérieures). En fait c'est une équivalence. Les graphes série-parallèles sont les graphes sans K_4 comme mineur.
- Les graphes de genre g n'ont pas $K_{r(g)}$ comme mineur, où $r(g) = \lfloor (7+\sqrt{48g+1})/2 \rfloor + 1 = O(\sqrt{g}\,)$.
- Le *Théorème des Mineurs de Graphes* de Robertson et Seymour [RS04] établit que dans toute famille infinie de graphes, il existe toujours deux graphes dont l'un est mineur de l'autre.

  En fait, toute famille $\mathcal{F}$ close par mineur peut être caractérisée par une suite finie de mineurs exclus de taille finie. Donc toute famille $\mathcal{F}$ close par mineur exclue une clique de taille finie (K_r pour un certain r ne dépendant que de $\mathcal{F}$, et pas de n, la taille du graphe).

Définition 16 (décomposition & largeur arborescente) *Une* décomposition arborescente *d'un graphe G est un arbre T dont les sommets, appelés* sacs*, sont des sous-ensembles de sommets de G, tel que :*

1. *Tout sommet est contenu dans au moins un sac ;*
2. *Toute arête a ses deux extrémité dans un même sac ;*
3. *L'ensemble des sacs contenant un sommet donné de G induit un arbre dans T.*

La largeur de T *est le nombre de sommet -1 du plus gros sac de T et la* largeur arborescente *de G est la plus petite largeur de ses décompositions arborescentes.*

Il y a bien sur, beaucoup de décompositions possibles pour un graphe, la plus triviale étant l'arbre réduit à un sac contenant tous les sommets du graphe. La définition précédente ne dit pas comment trouver une décomposition non-triviale, de petite largeur par exemple. C'est malheureusement NP-complet de déterminer si G possède un décomposition arborescente de largeur r.

- Les graphes de largeur arborescente r excluent K_{r+2}.
- Les graphes qui exclus un graphe planaire H sont de largeur arborescente au plus $f(H)$ [RS86]. Donc les graphes planaires extérieurs et séries-parallèles sont de largeur arborescente bornée. La meilleure borne connues est $f(H) \leqslant 20^{2(2|V(H)|+4|E(H)|)^5}$. Ici, cela donnerai $f(K_4) \leqslant 20^{2(2\cdot 4+4\cdot 6)^5} = 20^{2^{26}} = 2^{67\,108\,864}$, mais en fait la valeur optimale est $f(K_4) = 2$. Dans [RST94], il est conjecturé que la vraie borne est $f(H) = O(|V(H)|^2 \log |V(H)|)$.

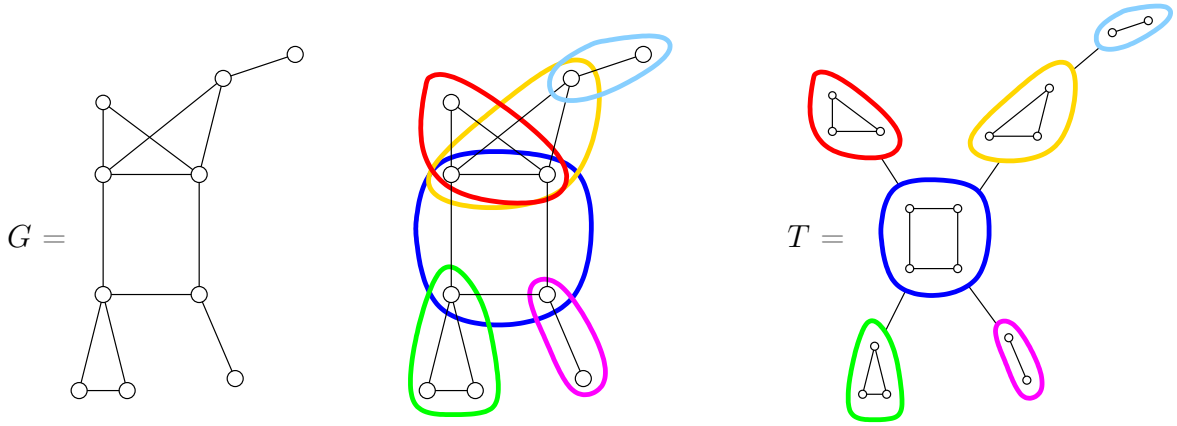


FIGURE 4.6 – Une décomposition arborescente de largeur 3.

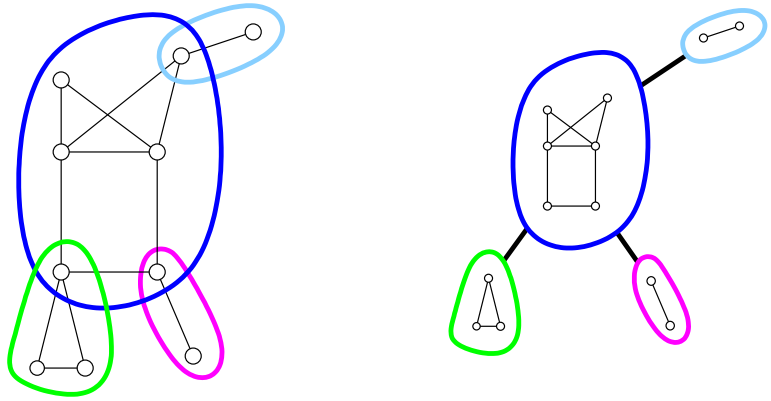


FIGURE 4.7 – Décomposition en composantes 2-connexes, de largeur 5.

Proposition 17 *Tout graphe de largeur arborescente r est $(r+1)$-séparable.*

Preuve. Soit H un sous-graphe quelconque d'un graphe G et T une décomposition arborescente de G de largeur $\leqslant r$. Soit T_H le graphe obtenu à partir de T en remplaçant tout sac de S par $S \cap V(H)$ et en suprimant les sacs qui deviennent ainsi vides. On peut vérifier que T_H est une forêt, H n'étant pas forcément connexe. On transforme T_H en un arbre T'_H en liant arbitrairement les sous-arbres de T_H par de nouvelles arêtes. L'arbre T'_H est une décomposition arborescente de H de largeur $\leqslant r$.

On va maintenant se servir de la structure de T'_H pour « couper » H en deux, c'est-à-dire trouver le demi-séparateur. On suppose que T'_H est enraciné en R, et on pose $p(X)$ le père du sac X dans T'_H. Par convention, $p(R) = \varnothing$. Soit $T'_H(X)$ le sous-arbre de T'_H enraciné en X.

On définit alors le *poids* d'un sac X de T'_H, noté $w(X)$, comme le nombre de sommets

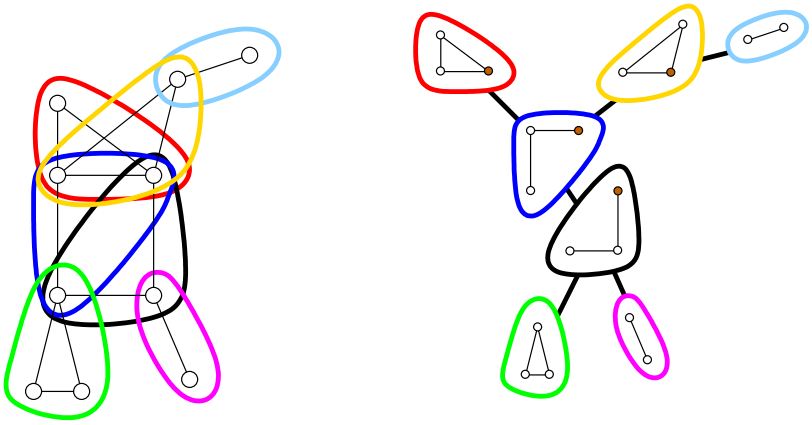


FIGURE 4.8 – Décomposition de largeur 2.

du graphe induit par $\cup_{Y \in T'_H(X)} Y \setminus p(X)$. Supposons que H a n sommets. Alors, $w(R) = n$. On effectue une marche de R vers le « centroïde valué » de T'_H. A savoir, on suit la règle suivante : on cherche un sac S de T'_H tel que $w(S) > n/2$ et $w(S) \leqslant n/2$ pour tout fils de S.

Il faut alors remarquer (démontrer) que, si S existe, alors toutes les composantes connexes de $H \setminus S$ ont au plus $n/2$ sommets, c'est-à-dire S est le demi-séparateur recherché, $|S| \leqslant r + 1$. L'autre point est que S existe toujours. En effet, la somme des poids des fils d'un sac X est $\leqslant w(X)$ (les fils induisent des sous graphes disjoints). Donc, il ne peut avoir qu'un seul fils de poids $w(F) > n/2$. Comme le poids d'un fils est au plus celui du père (car la somme des fils est $\leqslant$), et que l'arbre est fini, la marche s'arêtera en un sommet S vérifiant la propriété. □

Remarques :

- La borne « $r + 1$ » de la proposition 17 est la meilleure possible, pour chaque $r \geqslant 2$, à cause du graphe suivant, noté G_r (voir la figure 4.9) : (1) on part d'une clique à $r + 1$ sommets ; puis (2) pour chaque sommet x_i de la clique, on ajoute un sommet y_i connecté à tous les sommets de la clique sauf x_i ; et (3) à chaque sommet y_i on connecte un sommet z_i de degré 1. Le graphe G_r a $n = 3(r + 1)$ sommets. Sa largeur arborescente est r : il possède K_{r+1} et on peut construire facilement, d'après sa définition, un arbre de hauteur 3 dont les sacs sont de taille $r + 1$ ou 2 (pour les feuilles). Cependant, G_r n'est pas r-séparable. En effet la suppression d'un sous-ensemble S de r sommets permet au mieux de déconnecter une seule « branche » parmi les $r+1$ de G_r. En effet, dans la clique privée de S, il reste au moins un sommet appartenant au voisinage de r branches (et donc qui les maintient ensemble). Il restera donc toujours une composante connexe avec au moins $3(r + 1) - (r + 2) = 2r + 1$ sommets. Or $2r + 1 > n/2 = 3(r + 1)/2$ pour tout $r > 1$.

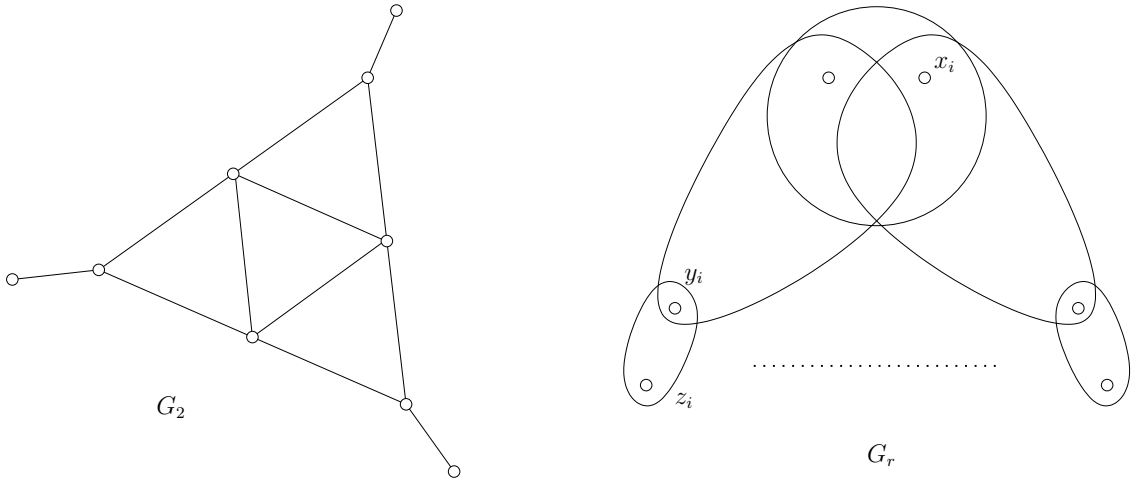


FIGURE 4.9 – Un graphe G_r de largeur arborescente r mais qui n'est pas r-séparable.

- Notons que la clique K_{r+1} de largeur arborescente r est $\lceil (r+1)/2 \rceil$-séparable. Les cliques ne sont donc pas des graphes « extrêmaux » pour la proposition 17. Notons que $K_{r,r}$, de largeur arborescente r, n'est pas $(r-1)$-séparable.
- On peut en fait démontrer que tout graphe r-séparable possède une décomposition arborescente de largeur au plus $3r+2$, et de hauteur $\leqslant \log n$.

4.3.3 Cas général

On ne peut pas faire mieux que $(n-1)/2$ bits, et clairement on peut le faire avec $O(n \log n)$ bits en codant la table de distance du sommet vers tous les autres.

Le résultat suivant est une amélioration de la technique développée initialement dans [GPPR04] où la taille des étiquettes était de $11n+o(n)$ bits. Dans le théorème 16 nous verrons encore une amélioration du résultat ci-dessous, mais la méthode reste la même.

Théorème 15 (Gavoille *et al.* [GPPR04]) *Les graphes à n sommets possèdent un étiquetage de distance avec des étiquettes de $28n/9 + o(n) \approx 3.11n$ bits. Le temps de calcul des distances est $O(\log \log n)$.*

On commence par prouver le résultat suivant :

Proposition 18 *Dans tout graphe connexe à n sommets et pour tout entier k, $0 \leqslant k \leqslant n-1$, il existe un ensemble k-dominant de taille au plus $n/(k+1)$.*

Preuve. On choisit un sommet arbitraire x_0, puis pour tout entier $i \in \{0, \ldots, k\}$, on définit L_i comme l'ensemble des sommets y tels que $d_T(x_0, y) \bmod (k+1) = i$, où T est un arbre

couvrant arbitraire enraciné en x_0 (il suffit en fait de calculer un ensemble k-dominant pour T, il le sera aussi pour G). On vérifie que chaque ensemble $L_i \cup \{x_0\}$ est un ensemble k-dominant. [A FINIR]

Comme $(L_i)_{i=0}^{k}$ forme une partition des n sommets, il est clair qu'un des ensembles est de cardinalité au plus $n/(k+1)$ $+1$?. □

Preuve. (Théorème 15) On considère une suite d'ensembles de sommets $S_0, S_1, \ldots, S_t$ et d'entiers naturels $k_0, k_1, \ldots, k_t$ tels que S_i est un ensemble k_i-dominant pour tout $i \in \{0, \ldots, t\}$. On déterminera les valeurs des k_i plus tard. On impose cependant les restrictions suivantes aux deux suites :

- $k_0 = 0$;
- $|S_t| = o(n/\log n)$;
- $t = o(n/\log n)$; et
- $k_i \in [0, n-1]$ sont entiers,

si bien que grâce à la proposition précédente, on supposera que $|S_i| \leqslant n/(k_i+1)$. Notons que $S_0 = V(G)$ car $k_0 = 0$.

Soit $i \in \{0, \ldots, t\}$. À chaque sommet x on associe $x \mapsto p_i(x)$ un sommet de S_i le plus proche de x. La remarque fondamentale est qu'on peut calculer la distance entre deux sommets $x, y \in S_{i-1}$ en utilisant S_i par :

$$d(x,y) \;=\; d(p_i(x), p_i(y)) + e_i(x,y)$$

où $e_i(x,y) \in [-2k_i, +2k_i]$ est un terme correctif dépendant de x et y. En effet, en utilisant l'inégalité triangulaire entre x et y, on obtient :

$$\begin{aligned} d(x,y) &\leqslant d(x,p_i(x)) + d(p_i(x),p_i(y)) + d(p_i(y),y) \\ d(x,y) - d(p_i(x),p_i(y)) &\leqslant 2k_i \ . \end{aligned}$$

D'autre part, en utilisant l'inégalité triangulaire entre $p_i(x)$ et $p_i(y)$, on obtient :

$$\begin{aligned} d(p_i(x),p_i(y)) &\leqslant d(p_i(x),x) + d(x,y) + d(y,p_i(y)) \\ d(p_i(x),p_i(y)) - d(x,y) &\leqslant 2k_i \\ -2k_i &\leqslant d(x,y) - d(p_i(x),p_i(y)) \ . \end{aligned}$$

Il faut alors remarquer que $d(p_i(x), p_i(y))$ peut se calculer récursivement à l'aide de l'ensemble dominant S_{i+1}. Donc pour tous sommets $x, y \in S_0 = V(G)$, on peut calculer la distance entre x et y par :

$$\begin{aligned} d(x,y) &= e_1(x,y) + d(p_1(x),p_1(y)) \\ &= e_1(x,y) + e_2(p_1(x),p_1(y)) + d(p_2(p_1(x)),p_2(p_1(y))) \\ &= \cdots \end{aligned}$$

En notant $x_i = p_i(p_{i-1}(\cdots p_1(p_0(x))\cdots))$, on a alors (remarquons que $x_0 = p_0(x) = x$)

$$d(x,y) \;=\; \left(\sum_{i=1}^{t} e_i(x_{i-1}, y_{i-1})\right) + d(x_t, y_t)$$

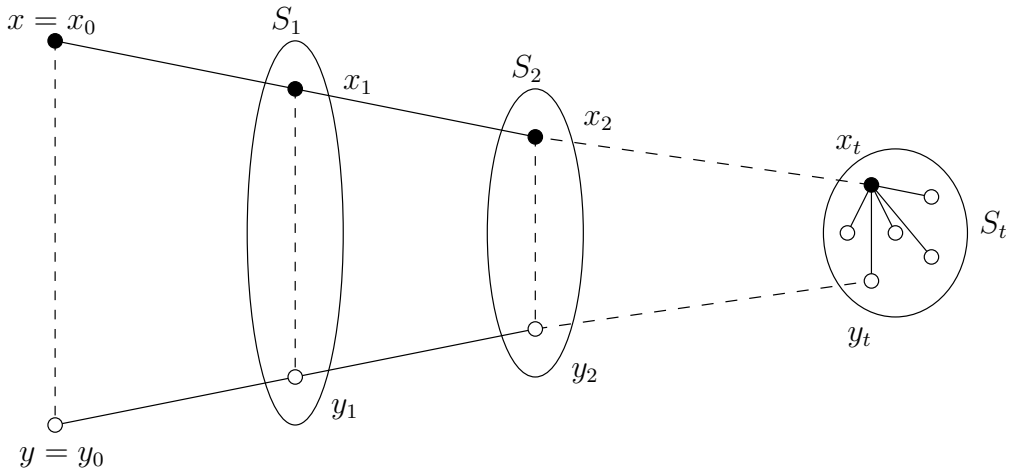


FIGURE 4.10 – Calcul de distance à l'aide d'ensembles S_i k_i-dominants (pas forcément disjoints).

À chaque sommet $u \in S_i$, on note $r_i(u)$ son « rang » dans S_i, c'est-à-dire le nombre de sommets plus petit à u dans S_i, en supposant un ordre total sur les sommets. Donc $r_i(u) \in \{0, \ldots, |S_i| - 1\}$.

On pose $E_i(x)$ le tableau des $e_i(x,y)$ pour tous les sommets $y \in S_{i-1}$, $i > 0$. Plus précisément, si $r(y) = j$, alors $E_i(x)[j] = e_i(x,y)$.

On note $D(x)$ le tableau des distances entre x et tous les $y \in S_t$. Plus précisément, si $r_t(y) = j$, alors $D(x)[j] = d(x,y)$. On remarque que $D(x)$ se code avec $|S_t| \cdot \lceil \log n \rceil = o(n)$ bits puisque $|S_t| = o(n/\log n)$.

L'étiquette de x :

$$\begin{aligned} L(x,G) \;= (\;& r_0(x_0), r_1(x_1), \ldots, r_{t-1}(x_{t-1}), r_t(x_t), \\ & E_1(x_0), E_2(x_1), \ldots, E_t(x_t), \\ & D(x_t)\;) \end{aligned}$$

Décodeur de distance :

$$f(L(x,G), L(y,G)) \;=\; \left(\sum_{i=1}^{t} E_i(x_{i-1})[r_{i-1}(y_{i-1})]\right) + D(x_t)[r_t(y_t)]$$

ce qui peut se calculer en temps $O(t)$.

Taille des étiquettes : Il reste à évaluer la taille des étiquettes, et à fixer les k_i. Remarquons qu'il y a $4k_i + 1$ valeurs possibles pour chaque entrée $e_i(x, y) \in [-2k_i, 2k_i]$ du tableau $E_i(x)$. Donc $\lceil \log(4k_i + 1) \rceil$ bits par entrée suffisent. Les valeurs $r_0(x_0), \ldots, r_t(x_t)$ peuvent se coder sur $O(t \log n) = o(n)$ car $t = o(n/\log n)$. On a déjà établit que $D(x_t)$ se code sur $o(n)$ bits.

On observe que la distance étant symétrique, $E_i(x)[r_i(y)] = E_i(y)[r_i(x)]$. Donc on peut appliquer la même astuce que pour l'adjacence en $\lceil (n-1)/2 \rceil + \lceil \log n \rceil$ bits, à savoir, la distance $d(x, y)$ est stockée en x seulement pour les sommets y de rang juste supérieur à celui de x (modulo $|S_i|$). Ces sommets y sont au nombre de $\lceil (|S_i| - 1)/2 \rceil = \lfloor |S_i|/2 \rfloor$. En comparant les rangs de x et de y, on peut savoir si l'information doit être extraite du tableau de x ou de y.

Soit T la taille maximale des étiquettes pour un graphe à n sommets (rappelons que $|S_i| \leqslant n/(k_i + 1)$).

$$\begin{aligned} T &\leqslant \left(\sum_{i=1}^{t} \left\lfloor \frac{1}{2} |S_{i-1}| \right\rfloor \cdot \lceil \log(4k_i + 1) \rceil \right) + o(n) \\ &\leqslant n \cdot \underbrace{\left(\sum_{i=1}^{t} \frac{1}{2} \cdot \frac{\lceil \log(4k_i + 1) \rceil}{k_{i-1} + 1} \right)}_{S} + o(n) \end{aligned}$$

En posant $k_i = 4^i - 1$, nous avons $k_0 = 0$, et en remarquant que pour tout $i \geqslant 1$, $\lceil \log(4k_i + 1) \rceil = \lceil \log(4^{i+1} - 3) \rceil = (i + 1) \log 4 = 2i + 2$,

$$S = \sum_{i=1}^{t} \frac{1}{2} \cdot \frac{\lceil \log(4^{i+1} - 3) \rceil}{4^{i-1}} < \sum_{i=1}^{+\infty} \frac{i+1}{4^{i-1}} = \frac{2}{1} + \frac{3}{4} + \frac{4}{16} + \frac{5}{64} + \cdots = \frac{28}{9}$$

En fait, la somme se calcule exactement en utilisant la formule suivante :

$$\sum_{i=0}^{+\infty} \frac{ai + b}{c^i} = \frac{c(a - b + bc)}{(c-1)^2}$$

Avec $a = 1$, $b = 2$, et $c = 4$ on obtient,

$$S < \sum_{i=1}^{+\infty} \frac{i+1}{4^{i-1}} = \sum_{i \geqslant 0} \frac{i+2}{4^i} = \frac{4(1 - 2 + 8)}{3^2} = \frac{28}{9} = 3.111\ldots$$

Finalement, nous avons montré que $T \leqslant 28n/9 + o(n)$. Il nous reste à déterminer t et à vérifier que $t = o(n/\log n)$. On pose $t = \lceil \log \log n \rceil$ si bien que

$$\begin{aligned} |S_t| &\leqslant \frac{n}{4^t + 1} \\ &\leqslant O(n/2^{2 \log \log n}) = O(n/2^{\log(\log^2 n)}) \\ &\leqslant O(n/\log^2 n) = o(n/\log n)\ . \end{aligned}$$

□

Remarque :

- Winkler a établit dans [Win83] une borne de $1.58n$ bits ($\approx \log 3$), mais avec un temps $O(n)$ pour le décodage de la distance.
- La meilleure borne connue [KP02] pour l'étiquetage de distance est $1.35n$ bits avec un temps $O(\log n)$ pour le décodage. La borne se réduit à $1.16n$ bits ($1.16 \approx \frac{1}{2}\log 5$) si le graphe est 2-connexe (toujours un temps de $O(\log n)$ pour le décodage). L'idée générale de ces résultats est d'ordonner les sommets selon un cycle Hamiltonien de G^3 (une arête est dans G^3 si x et y sont à distance au plus trois dans G). En fait il est montré que le cube d'un arbre est toujours Hamiltonien. Ainsi les distances $d(x,u_i)$ et $d(x,u_{i+1})$ où u_i et u_{i+1} sont consécutifs selon ce cycle, sont proches : $d(x,u_i) - d(x,u_{i+1}) \in [-3,+3]$ ce qui fait 7 valeurs possibles.
- En combinant ces deux résultats, il est possible (je pense) de faire $1.21n$ bits ($\approx \frac{1}{2}\log 7$) pour tout graphe, et peut être même en temps constant.

En fait, un rafinement de la technique précédente (en particulier un choix différent des k_i) permet d'obtenir des étiquettes de taille $2.95n$ avec un temps de décodage encore plus cours $O(\log^* n)$.

Théorème 16 *Les graphes à n sommets possèdent un étiquetage de distance avec des étiquettes de $2.95n + o(n)$ bits. Le temps de calcul des distances est $O(\log^* n)$.*

Preuve. Rappelons que les k_i et t peuvent être choisis de façon à minimiser le temps t et la somme S suivante :

$$S = \sum_{i=1}^{t} \frac{1}{2} \cdot \frac{\lceil \log(4k_i+1) \rceil}{k_{i-1}+1}$$

avec les conditions suivantes :

- $k_0 = 0$;
- $k_i \in [0, n-1]$ sont entiers ;
- $k_t = \omega(\log n)$, c'est-à-dire [3] $\log n = o(k_t)$.

En fait une première optimisation consiste à se débaraser des parties entières $\lceil \cdots \rceil$ tout en chosissant mieux les k_i, ...

[A FINIR]

...

Le problème est difficile, car c'est une optimisation en nombre entier. En fait on choisit les premières valeurs de $k_0, \ldots, k_4$ comme suit : $(k_i)_{i=0}^4 = (0, 2, 15, 145, 940)$. Pour les valeurs plus grandes, on choisit une autre fonction dont la somme $\sum_{i\geqslant 5}^{t}$ peut être facilement bornée

3. On rappelle que $f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$.

par une assez petite valeur, et surtout qui converge plus vite (c'est-à-dire dont le dernier terme est plus petit que $1/\log n$, en $t = O(\log^* n)$ étapes.

Par exemple, en prenant

$$k_i \;=\; \frac{1}{4}\left(2^{(k_{i-1}+1)/c^i} - 1\right)$$

on obtient que

$$\sum_{i=5}^{t} \frac{1}{2} \cdot \frac{\lceil \log(4k_i + 1) \rceil}{k_{i-1} + 1} \;<\; \frac{1}{2} \sum_{i=5}^{+\infty} \frac{1}{c^i} \;<\; \frac{1}{2c^4(c-1)} \,.$$

Bien sûr il faut $k_i > k_{i-1}$ d'où la restriction que $c^i < k_{i-1}$. Par exemple $c = 3$ donne $c^5 = 243 < k_4 = 940$, ce qui fait une erreur en $\frac{1}{2c^4(c-1)} < 1/324$.

[A FINIR]

...

Une optimisation supplémentaire possible est d'essayer de se débarasser du premier terme en $\frac{1}{2}\log 5 \approx 1.16\ldots$ qui vient du fait que $k_0 = 0$ et on utilise ensuite un ensemble 1-dominant (si on utilise $k_i > 1$ le premier terme est encore plus grand, et si on utilise $k_1 = 0$, le problème se pose avec k_2). Pour cela, on pourrait (?) utiliser l'adjacence que l'on sait résoudre avec $\frac{1}{2}n$ bits seulement ... □

4.4 Ancêtre et plus petit ancêtre commun

4.4.1 Ancêtre

4.4.2 Plus petit ancêtre commun

Bibliographie

[ABR05] S. Alstrup, P. Bille, and T. Rauhe, *Labeling schemes for small distances in trees*, SIAM Journal on Discrete Mathematics, 19 (2005), pp. 448–462. doi : 10.1137/S0895480103433409.

[AR02] S. Alstrup and T. Rauhe, *Small induced-universal graphs and compact implicit graph representations*, in 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society Press, November 2002, pp. 53–62. doi : 10.1109/SFCS.2002.1181882.

[AST90] N. Alon, P. D. Seymour, and R. Thomas, *A separator theorem for graphs with an excluded minor and its applications*, in 22nd Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 1990, pp. 293–299. doi : 10.1145/100216.100254.

[BGL06] N. Bonichon, C. Gavoille, and A. Labourel, *Short labels by traversal and jumping*, in 13th International Colloquium on Structural Information & Communication Complexity (SIROCCO), vol. 4056 of Lecture Notes in Computer Science, Springer, July 2006, pp. 143–156. doi : 10.1007/11780823_12.

[Bod96] H. L. Bodlaender, *A linear time algorithm for finding tree-decompositions of small treewidth*, SIAM Journal on Computing, 25 (1996), pp. 1305–1317. doi : 10.1137/S0097539793251219.

[FHL05] U. Feige, M. Hajiaghayi, and J. R. Lee, *Improved approximation algorithms for minimum-weight vertex separators*, in 37th Annual ACM Symposium on Theory of Computing (STOC), ACM Press, May 2005, pp. 563–572. doi : 10.1145/1060590.1060674.

[GHT84] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan, *A separation theorem for graphs of bounded genus*, Journal of Algorithms, 5 (1984), pp. 391–407. doi : 10.1016/0196-6774(84)90019-1.

[GKK+01] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg, *Approximate distance labeling schemes*, in 9th Annual European Symposium on Algorithms (ESA), F. M. auf der Heide, ed., vol. 2161 of Lecture Notes in Computer Science, Springer, August 2001, pp. 476–487. doi : 10.1007/3-540-44676-1_40.

[GL07] C. Gavoille and A. Labourel, *Shorter implicit representation for planar graphs and bounded treewidth graphs*, in 15th Annual European Symposium on Algorithms (ESA), L. Arge and E. Welzl, eds., vol. 4698 of Lecture Notes in Computer Science, Springer, October 2007, pp. 582–593. doi : 10.1007/978-3-540-75520-3_52.

[GPPR04] C. Gavoille, D. Peleg, S. Pérennès, and R. Raz, *Distance labeling in graphs*, Journal of Algorithms, 53 (2004), pp. 85–112. doi : 10.1016/j.jalgor.2004.05.002.

[KP02] S. Kogan and H. Petersen, *Distance labeling in general graphs*, in Workshop Für Komplexitätstheorie, Datenstrukturen und Effiziente Algorithmen (Theorietag), Wilhelm-Schickard-institut für Informatik, 2002, p. 9.

[LT79] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM Journal on Applied Mathematics, 36 (1979), pp. 177–189.

[Pel00a] D. Peleg, *Distributed Computing : A Locality-Sensitive Approach*, SIAM Monographs on Discrete Mathematics and Applications, 2000. isbn : 0-89871-464-8. doi : 10.1137/1.9780898719772.

[Pel00b] D. Peleg, *Proximity-preserving labeling schemes*, Journal of Graph Theory, 33 (2000), pp. 167–176. doi : 10.1002/(SICI)1097-0118(200003)33:3<>1.0.CO;2-Y.

[RS86] N. Robertson and P. D. Seymour, *Graph minors. V. Excluding a planar graph*, Journal of Combinatorial Theory, Series B, 41 (1986), pp. 92–114. doi : 10.1016/0095-8956(86)90030-4.

[RS95] N. Robertson and P. D. Seymour, *Graph minors. XIII. The disjoint paths problem*, Journal of Combinatorial Theory, Series B, 63 (1995), pp. 65–110. DOI : 10.1006/jctb.1995.1006.

[RS04] N. Robertson and P. D. Seymour, *Graph minors. XX. Wagner's conjecture*, Journal of Combinatorial Theory, Series B, 92 (2004), pp. 325–357. DOI : 10.1016/j.jctb.2004.08.001.

[RST94] N. Robertson, P. D. Seymour, and R. Thomas, *Quickly excluding a planar graph*, Journal of Combinatorial Theory, Series B, 62 (1994), pp. 323–348. DOI : 10.1006/jctb.1994.1073.

[Win83] P. Winkler, *Proof of the squashed cube conjecture*, Combinatorica, 3 (1983), pp. 135–139. DOI : 10.1007/BF02579350.