Algorithmes d'approximation à mémoire limitée pour le traitement de grands graphes

Le problème du Vertex Cover

Romain Campigotto

Encadrants : Eric Angel et Christian Laforest

Laboratoire IBISC, EA 4526 – Université d'Evry-Val d'Essonne

Mardi 6 décembre 2011

Plan

Plan

Contexte et motivations

Traitement classique d'un problème d'optimisation sur un graphe
$\longrightarrow$ disponible **dans sa totalité** sur l'unité de traitement :

- modifications, mises à jour possibles ;
- conservation de la solution construite en mémoire.

Contexte et motivations

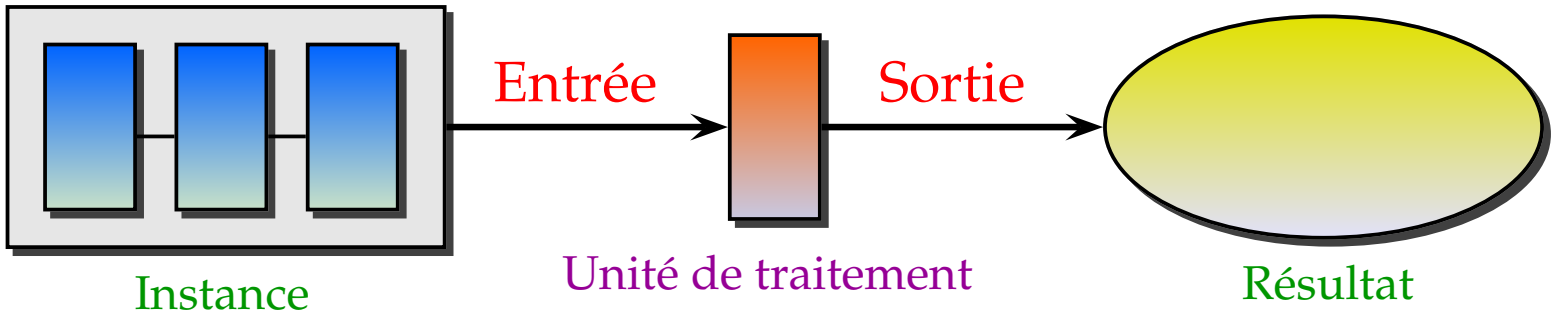
Traitement classique d'un problème d'optimisation sur un graphe
$\longrightarrow$ disponible **dans sa totalité** sur l'unité de traitement :

- modifications, mises à jour possibles ;
- conservation de la solution construite en mémoire.

Quantités de données à traiter de plus en plus importantes.
$\longrightarrow$ *Exemples : biologie, météorologie, finance, etc.*

Un modèle naturel de traitement

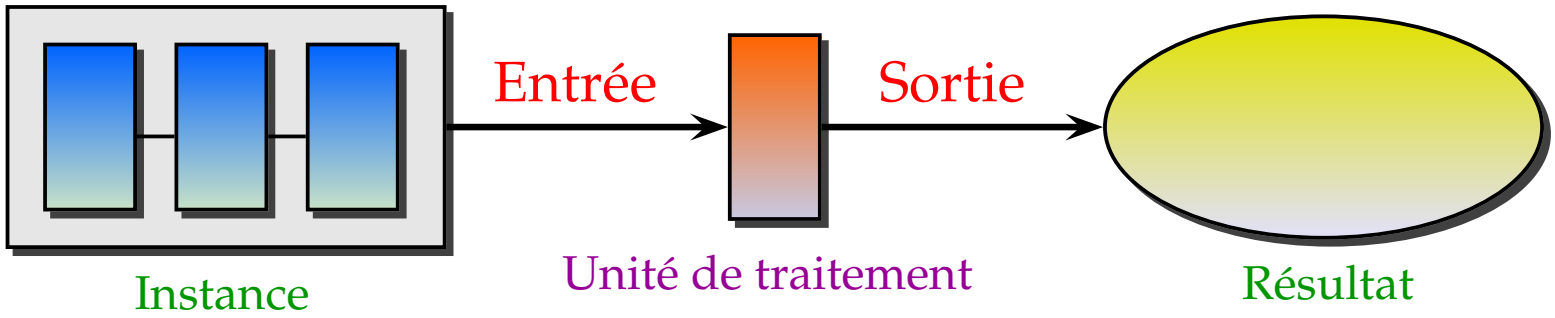
Modèle strict



C_1. Pas de modification des données fournies en entrée : *intégrité* de l'instance.

Un modèle naturel de traitement

Modèle strict

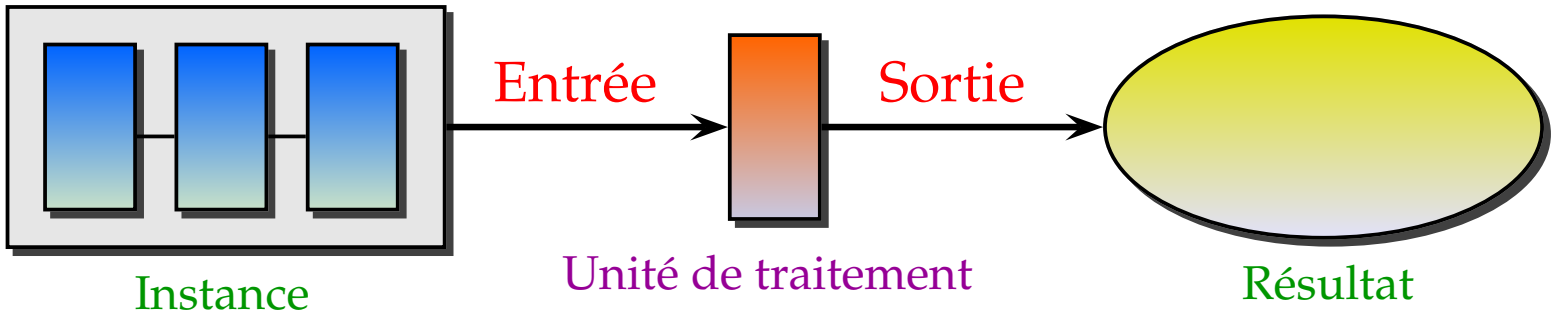


C_1. Pas de modification des données fournies en entrée : *intégrité* de l'instance.

C_2. Très peu de mémoire sur l'unité de traitement (nombre constant de variables).

Un modèle naturel de traitement

Modèle strict



C_1. Pas de modification des données fournies en entrée : *intégrité* de l'instance.

C_2. Très peu de mémoire sur l'unité de traitement (nombre constant de variables).

C_3. Envoi de la solution « à la volée » dans « Résultat ».

Autres approches liées aux grandes instances

Existance d'autres modèles :

$\longrightarrow$ quelques différences avec le notre...

Les algorithmes *en ligne*

Dévoilement de l'instance au fur et à mesure :

- ils ne disposent pas de la totalité des données ;
- ils ne maîtrisent pas l'ordre dans lequel elles arrivent ;

$\longrightarrow$ ils ne connaissent pas le futur.

Les algorithmes *en ligne*

Dévoilement de l'instance au fur et à mesure :

- ils ne disposent pas de la totalité des données ;
- ils ne maîtrisent pas l'ordre dans lequel elles arrivent ;

$\longrightarrow$ ils ne connaissent pas le futur.

Maintenir à tout moment une solution *réalisable* :

- ils doivent construire leurs solutions « à la volée » ;

$\longrightarrow$ ils doivent faire des choix **irrévocables**.

Les algorithmes *en ligne*

Dévoilement de l'instance au fur et à mesure :

- ils ne disposent pas de la totalité des données ;
- ils ne maîtrisent pas l'ordre dans lequel elles arrivent ;

$\longrightarrow$ ils ne connaissent pas le futur.

Maintenir à tout moment une solution *réalisable* :

- ils doivent construire leurs solutions « à la volée » ;

$\longrightarrow$ ils doivent faire des choix **irrévocables**.

$\longrightarrow$ Pas de contraintes sur la quantité de mémoire disponible !

Le modèle *streaming*

Arrivée du flux de données de façon arbitraire et continue :

- les algorithmes peuvent effectuer plusieurs *passes* sur le flux ;
- ils ne peuvent pas le modifier (accès en lecture seule).

Le modèle *streaming*

Arrivée du flux de données de façon arbitraire et continue :

- les algorithmes peuvent effectuer plusieurs *passes* sur le flux ;
- ils ne peuvent pas le modifier (accès en lecture seule).

Espace mémoire restreint (en $\mathcal{O}(n \log n)$ bits).

Le modèle *streaming*

Arrivée du flux de données de façon arbitraire et continue :

- les algorithmes peuvent effectuer plusieurs *passes* sur le flux ;
- ils ne peuvent pas le modifier (accès en lecture seule).

Espace mémoire restreint (en $\mathcal{O}(n \log n)$ bits).

$\longrightarrow$ Contraintes mémoire de nos modèles plus strictes !

L'approche *I/O-efficient*

Minimiser le nombre d'E/S effectuées par les algorithmes.

L'approche *I/O-efficient*

Minimiser le nombre d'E/S effectuées par les algorithmes.

Algorithmes exécutés par un processeur qui dispose :

- d'une mémoire interne (de petite taille) ;
- de disques de grande taille :
  - qui contiennent l'instance à traiter ;
  - qui stockeront la solution (à construire).

$\longrightarrow$ Le processeur met plus de temps à communiquer avec les disques qu'avec la mémoire interne.

L'approche *I/O-efficient*

Minimiser le nombre d'E/S effectuées par les algorithmes.

Algorithmes exécutés par un processeur qui dispose :

- d'une mémoire interne (de petite taille) ;
- de disques de grande taille :
  - qui contiennent l'instance à traiter ;
  - qui stockeront la solution (à construire).

$\longrightarrow$ Le processeur met plus de temps à communiquer avec les disques qu'avec la mémoire interne.

$\longrightarrow$ Disques accessibles à la fois en lecture et en écriture !

$\longrightarrow$ Utilisés comme extension de la mémoire interne !

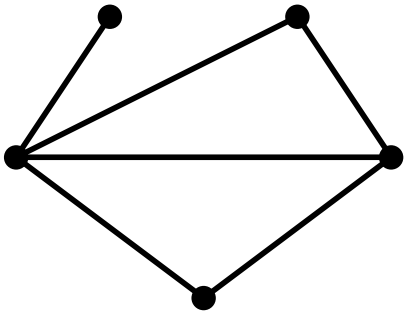
Plan

Le Vertex Cover : présentation

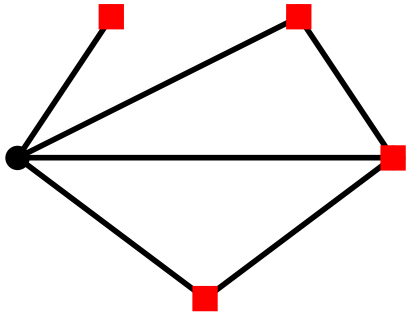
Définition (Couverture)

Soit $G = (V, E)$ un graphe simple. C est une couverture, si $C \subseteq V$ et $\forall e = uv \in E$, $u \in C$ ou $v \in C$ (ou les 2).

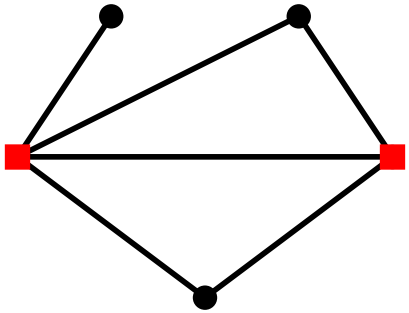
→ Le problème du Vertex Cover (dans sa version *problème d'optimisation*) : trouver une *couverture* de taille minimale.



Un graphe



Une couverture



Une couverture optimale

Etudier le Vertex Cover : pour quelles raisons ?

Un problème **NP**-complet bien connu (*R. Karp*, 1971).

Il intervient dans de nombreux problèmes appliqués :

- surveillance des réseaux (*network monitoring*),
- *résolution de conflits SNP* (alignement de séquences d'ADN),
- *etc.*

où les instances à traiter peuvent être très grandes.

Le Vertex Cover : quel(s) algorithme(s) choisir ?

Il existe beaucoup d'algorithmes approchés pour le Vertex Cover.

(Inapproximabilité en dessous de $10\sqrt{5} - 21 \approx 1.3606$)

- Meilleur rapport connu : $2 - \Theta\left(\frac{1}{\sqrt{\log n}}\right)$ (*G. Karakostas*, 2005) ;
- 2-approximation : `ED`, `DFS`, *programmation linéaire* ;
- $H(\Delta)$-approximation : *Maximum Degree Greedy* ;
- Δ-approximation : `LR`.

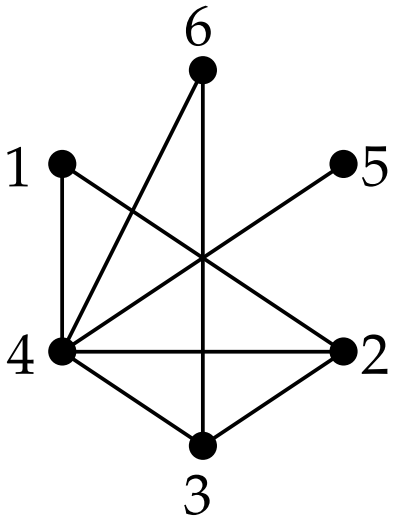
$\longrightarrow$ Ils ne sont pas adaptés à notre modèle strict !

Plan

Présentation de LL

Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.

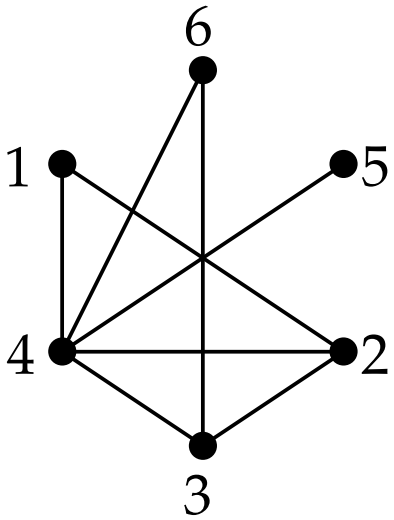


Etiquetage des sommets de 1 à n.

Présentation de LL

Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.



$2 \longrightarrow 4, 1, 3$
$4 \longrightarrow 3, 5, 2, 6, 1$
$1 \longrightarrow 2, 4$
$6 \longrightarrow 4, 3$
$3 \longrightarrow 2, 6, 4$
$5 \longrightarrow 4$

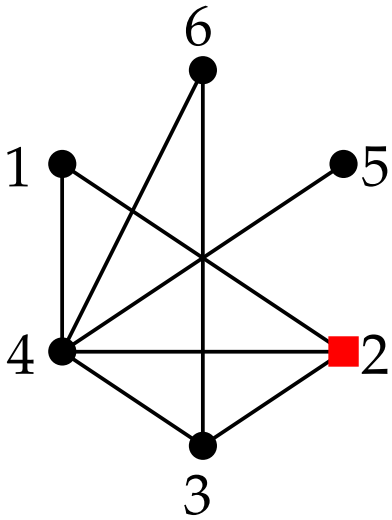
Etiquetage des sommets de 1 à n.
Stockage sous forme de *liste d'adjacence* :

- pour chaque sommet, accès aux *voisins*, récupérables au fur et à mesure.

Présentation de LL

Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.



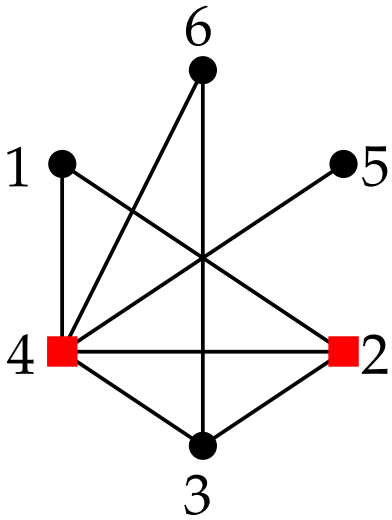
$2 \longrightarrow 4, 1, 3$
$4 \longrightarrow 3, 5, 2, 6, 1$
$1 \longrightarrow 2, 4$
$6 \longrightarrow 4, 3$
$3 \longrightarrow 2, 6, 4$
$5 \longrightarrow 4$

Etiquetage des sommets de 1 à n.
Stockage sous forme de *liste d'adjacence* :

- pour chaque sommet, accès aux *voisins*, récupérables au fur et à mesure.

Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.



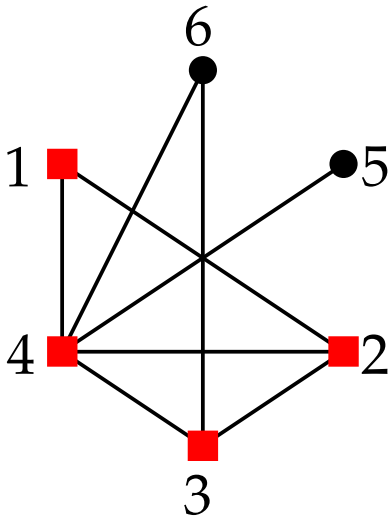
$2 \longrightarrow 4, 1, 3$
$4 \longrightarrow 3, 5, 2, 6, 1$
$1 \longrightarrow 2, 4$
$6 \longrightarrow 4, 3$
$3 \longrightarrow 2, 6, 4$
$5 \longrightarrow 4$

Etiquetage des sommets de 1 à n.
Stockage sous forme de *liste d'adjacence* :

- pour chaque sommet, accès aux *voisins*, récupérables au fur et à mesure.

Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.



$2 \longrightarrow 4, 1, 3$
$4 \longrightarrow 3, 5, 2, 6, 1$
$1 \longrightarrow 2, 4$
$6 \longrightarrow 4, 3$
$3 \longrightarrow 2, 6, 4$
$5 \longrightarrow 4$

Etiquetage des sommets de 1 à n.
Stockage sous forme de *liste d'adjacence* :

- pour chaque sommet, accès aux *voisins*, récupérables au fur et à mesure.

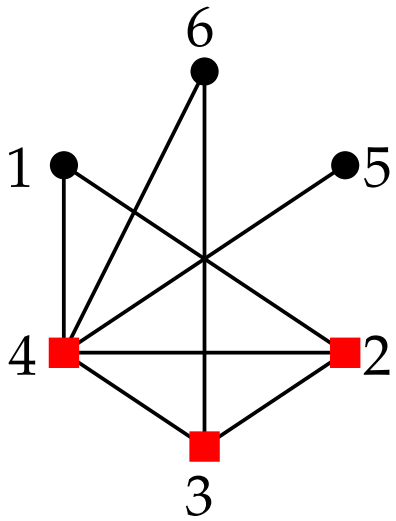
$\longrightarrow$ Résultat indépendant de l'ordre de récupération des sommets !

Présentation de SLL (*Sorted-LL*)

Algorithme (SLL)

Soit $G = (V, L, E)$ un graphe étiqueté. $\forall u \in V$, $u \in C$ ssi u a au moins un voisin de *degré inférieur* **ou** un *voisin droit de même degré*.

$\rightarrow$ Accès aux degrés des voisins.



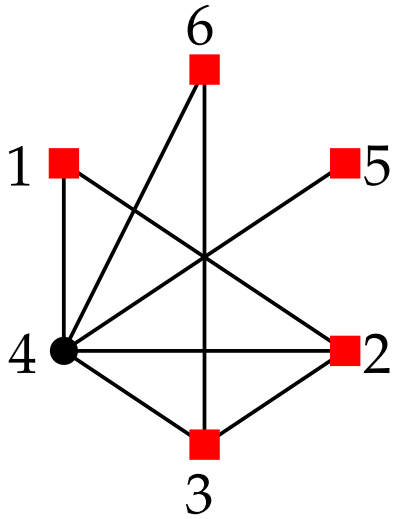
$2 \longrightarrow 4, 1, 3$
$4 \longrightarrow 3, 5, 2, 6, 1$
$1 \longrightarrow 2, 4$
$6 \longrightarrow 4, 3$
$3 \longrightarrow 2, 6, 4$
$5 \longrightarrow 4$

Présentation de ASLL (*Anti Sorted-LL*)

Algorithme (ASLL)

Soit $G = (V, L, E)$ un graphe étiqueté. $\forall u \in V$, $u \in C$ ssi u possède au moins un voisin de *degré supérieur* **ou** un *voisin gauche de même degré*.

$\rightarrow$ Accès aux degrés des voisins.



$2 \rightarrow 4, 1, 3$
$4 \rightarrow 3, 5, 2, 6, 1$
$1 \rightarrow 2, 4$
$6 \rightarrow 4, 3$
$3 \rightarrow 2, 6, 4$
$5 \rightarrow 4$

Des algorithmes adaptés

Inspirés par *"A list heuristic for Vertex Cover"* de *D. Avis et T. Imamura* (*ORL* 2006).

Parcours du graphe *sommet par sommet*. $\forall u \in V(G)$:

- récupération de ses voisins 1 par 1 ;
- comparaison du sommet u avec chacun de ses voisins.

$\longrightarrow$ A la fin, on prend une décision **irrévocable**.

Des algorithmes adaptés

Inspirés par *"A list heuristic for Vertex Cover"* de *D. Avis et T. Imamura* (*ORL* 2006).

Parcours du graphe *sommet par sommet*. $\forall u \in V(G)$:

- récupération de ses voisins 1 par 1 ;
- comparaison du sommet u avec chacun de ses voisins.

$\longrightarrow$ A la fin, on prend une décision **irrévocable**.

1. Le graphe n'est pas modifié.
2. On stocke un nombre constant de mots mémoire.

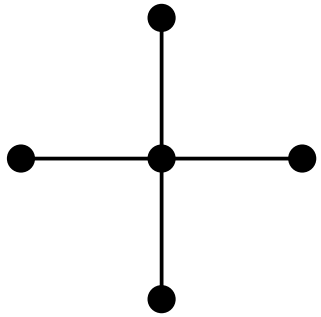
$\longrightarrow$ $\mathcal{O}(\log n)$ bits mémoire.

3. La solution est produite « à la volée ».

Etude en pire cas

Rapports d'approximation

LL et ASLL : au moins Δ-approché (borne atteinte sur les étoiles).



SLL : $\left(\frac{\sqrt{\Delta}}{2} + \frac{3}{2}\right)$-approché (*D. Avis et T. Imamura*).

$\rightarrow$ *A priori*, SLL est meilleur que LL et ASLL.

Vers une analyse plus fine

Ces trois algorithmes (exécutés sur des graphes étiquetés) sont déterministes.

$\longrightarrow$ Peu importe l'ordre de traitement des sommets.

Vers une analyse plus fine

Ces trois algorithmes (exécutés sur des graphes étiquetés) sont déterministes.
$\longrightarrow$ Peu importe l'ordre de traitement des sommets.

$\longrightarrow$ Mais il existe $n!$ manières d'étiqueter un graphe !

Comparaison en moyenne, en considérant *de manière équiprobable* les $n!$ étiquetages possibles.

Taille moyenne des solutions retournées

Théorème

Soit $G = (V, E)$ un graphe quelconque.

$$\mathbb{E}\big[\mathtt{LL}(G)\big] = n - \sum_{u \in V} \frac{1}{d(u)+1}$$

$$\mathbb{E}\big[\mathtt{SLL}(G)\big] = n - \sum_{u \in S} \frac{1}{\sigma(u)+1}$$

$$\mathbb{E}\big[\mathtt{ASLL}(G)\big] = n - \sum_{u \in \bar{S}} \frac{1}{\sigma(u)+1}$$

- $\sigma(u)$: nombre de *voisins de même degré* que u
- S : sommets qui n'ont pas de *voisin de degré inférieur*
- $\bar{S}$: sommets qui n'ont pas de *voisin de degré supérieur*

Taille moyenne des solutions retournées

Preuve pour LL

Démonstration.

Nombre d'étiquetages tels que $u \notin C_{\mathrm{LL}}$:

$$\binom{n}{d(u)+1} \cdot d(u)! \times (n-(d(u)+1))! \quad = \quad \frac{n! \times d(u)!}{(d(u)+1)!} \qquad (1)$$

En divisant (1) par $n!$, on obtient bien

$$\mathbb{P}\left[u \notin C_{\mathrm{LL}}\right] \quad = \quad \frac{1}{d(u)+1}$$

□

Application sur des familles de graphes

- Les chemins P_n :

$$\mathbb{E}\big[\mathtt{SLL}(P_n)\big] \;<\; \mathbb{E}\big[\mathtt{LL}(P_n)\big] \;<\; \mathbb{E}\big[\mathtt{ASLL}(P_n)\big]$$

$$\mathbb{E}\big[\mathtt{SLL}(P_n)\big] \;=\; \frac{2n}{3} - \frac{2}{3}$$

$$\mathbb{E}\big[\mathtt{LL}(P_n)\big] \;=\; \frac{2n}{3} - \frac{1}{3}$$

$$\mathbb{E}\big[\mathtt{ASLL}(P_n)\big] \;=\; \frac{2n}{3} + \frac{1}{3}$$

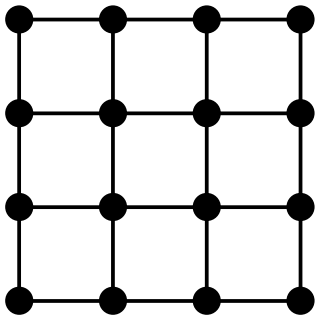
Application sur des familles de graphes

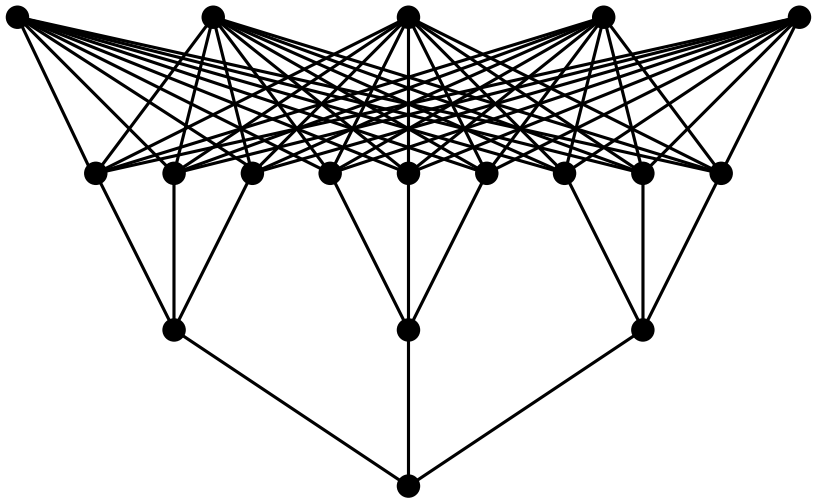
- Les grilles $GR_{p\times q}$:

$$\mathbb{E}\left[\mathtt{LL}(GR_{p\times q})\right] \;<\; \mathbb{E}\left[\mathtt{SLL}(GR_{p\times q})\right] \;<\; \mathbb{E}\left[\mathtt{ASLL}(GR_{p\times q})\right]$$

- les graphes d'*Avis-Imamura* étendus AI_a^+ :

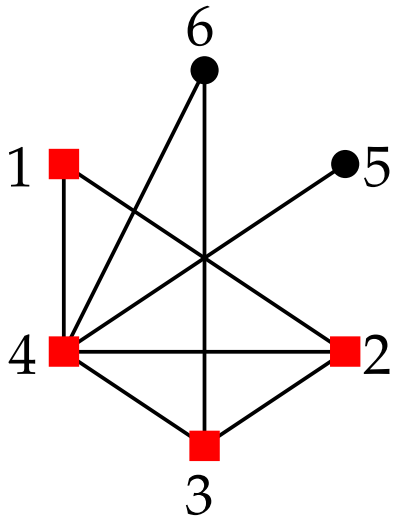
$$\mathbb{E}\left[\mathtt{ASLL}(AI_a^+)\right] \;<\; \mathbb{E}\left[\mathtt{LL}(AI_a^+)\right] \;<\; \mathbb{E}\left[\mathtt{SLL}(AI_a^+)\right]$$





Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.



$2 \longrightarrow \mathbf{4}, 1, 3$

$4 \longrightarrow 3, \mathbf{5}, 2, 6, 1$

$1 \longrightarrow \mathbf{2}, 4$

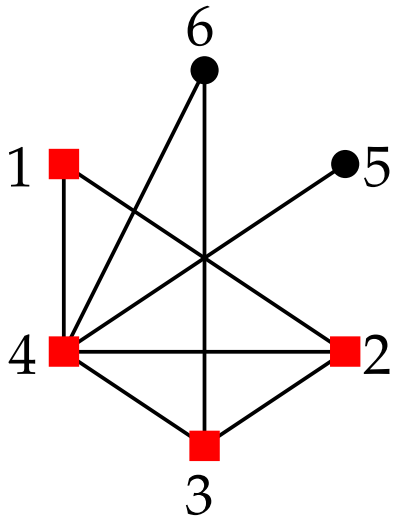
$6 \longrightarrow 4, 3$

$3 \longrightarrow 2, \mathbf{6}, 4$

$5 \longrightarrow 4$

Algorithme (LL)

Soit $G = (V, L, E)$ un *graphe étiqueté*. $\forall u \in V$, $u \in C$ ssi u possède au moins un *voisin droit*.



$2 \longrightarrow \mathbf{4}, 1, 3$

$4 \longrightarrow 3, \mathbf{5}, 2, 6, 1$

$1 \longrightarrow \mathbf{2}, 4$

$6 \longrightarrow 4, 3$

$3 \longrightarrow 2, \mathbf{6}, 4$

$5 \longrightarrow 4$

Nombre de requêtes effectuées : 9

Une *requête* : récupération d'un voisin d'un sommet.

Complexité en pire cas

On considère *le pire ordre possible* sur **tous les étiquetages**.

Théorème

Le nombre maximum de requêtes effectuées par l'algorithme $\mathtt{A}$ *sur le graphe* G *est*

$$m + |C_{\mathtt{A}}^{\max}|$$

avec $|C_{\mathtt{A}}^{\max}|$ *: taille max d'une couverture retournée par* $\mathtt{A}$ *sur* G*.*

($\mathtt{A}$ désigne à la fois LL, SLL et ASLL)

Complexité en pire cas

On considère *le pire ordre possible* sur **tous les étiquetages**.

Théorème

Le nombre maximum de requêtes effectuées par l'algorithme $\mathtt{A}$ *sur le graphe* G *est*

$$m + |C_{\mathtt{A}}^{\max}|$$

avec $|C_{\mathtt{A}}^{\max}|$ *: taille max d'une couverture retournée par* $\mathtt{A}$ *sur* G*.*

($\mathtt{A}$ désigne à la fois LL, SLL et ASLL)

$\longrightarrow$ Borne max (pour les 3 algorithmes) : $m + n - 1$

Complexité en moyenne

On considère tous les ordres **sur un étiquetage donné**.

Hypothèse d'équiprobabilité des $\prod_{u \in V} d(u)!$ ordres possibles.

Complexité en moyenne

On considère tous les ordres **sur un étiquetage donné**.

Hypothèse d'équiprobabilité des $\prod_{u \in V} d(u)!$ ordres possibles.

Lemme

Soit $G = (V, L, E)$ un graphe étiqueté quelconque.

$$\mathbb{E}\left[\mathrm{Q}_{\mathrm{LL}}(G, L)\right] = \sum_{u \notin C_{\mathrm{LL}}} d(u) + \sum_{u \in C_{\mathrm{LL}}} \frac{d(u)+1}{d^+(u)+1}$$

- $d^+(u)$: nombre de voisins droits de u

(Nombre moyen de tirages sans remise dans un sac)

(Formules équivalentes pour les algorithmes `SLL` et `ASLL`)

Complexité en moyenne

Hypothèse d'équiprobabilité des $n!$ étiquetages possibles.

Théorème

Soit $G = (V, E)$ un graphe quelconque.

$$\mathbb{E}\left[\mathrm{Q}_{\mathrm{LL}}(G)\right] = \sum_{u \in V} H(d(u))$$

où $H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ et $H(0) = 0$ (nombre harmonique).

Complexité en moyenne

Preuve pour LL

Soit u un sommet quelconque de G.

Complexité en moyenne

Preuve pour LL

Soit u un sommet quelconque de G.

$\beta_k(u)$: proportion d'étiquetages $L \in \mathbb{L}(G)$ tels que $d^+(u) = k$.

Complexité en moyenne

Preuve pour LL

Soit u un sommet quelconque de G.

$\beta_k(u)$: proportion d'étiquetages $L \in \mathbb{L}(G)$ tels que $d^+(u) = k$.

Contribution de u dans $\mathbb{E}\left[\mathrm{Q}_{\mathrm{LL}}(G)\right]$:

$$\beta_0(u) \cdot d(u) + \sum_{k=1}^{d(u)} \beta_k(u) \cdot \frac{d(u)+1}{k+1}$$

Complexité en moyenne

Preuve pour LL

Soit u un sommet quelconque de G.

$\beta_k(u)$: proportion d'étiquetages $L \in \mathbb{L}(G)$ tels que $d^+(u) = k$.

Contribution de u dans $\mathbb{E}\left[\mathrm{Q}_{\mathrm{LL}}(G)\right]$:

$$\beta_0(u) \cdot d(u) + \sum_{k=1}^{d(u)} \beta_k(u) \cdot \frac{d(u)+1}{k+1}$$

Il y a exactement

$$\binom{n}{d(u)+1} \cdot d(u)! \times (n-(d(u)+1))!$$

étiquetages pour lesquels $d^+(u) = k$.

Complexité en moyenne

Preuve pour LL (suite)

Donc $\forall k \in \{0, \ldots, d(u)\}$, $\beta_k(u) = \dfrac{1}{d(u)+1}$

Complexité en moyenne

Preuve pour LL (suite)

Donc $\forall k \in \{0, \ldots, d(u)\}$, $\beta_k(u) = \dfrac{1}{d(u)+1}$

On a donc

$$\begin{aligned}
\beta_0(u) \cdot d(u) + \sum_{k=1}^{d(u)} \beta_k(u) \cdot \frac{d(u)+1}{k+1} &= \frac{d(u)}{d(u)+1} + \sum_{k=1}^{d(u)} \frac{1}{k+1} \\
&= \frac{d(u)}{d(u)+1} + \sum_{k=2}^{d(u)+1} \frac{1}{k} \\
&= 1 + \sum_{k=2}^{d(u)} \frac{1}{k} = H(d(u))
\end{aligned}$$

Application sur des familles de graphes

- Les étoiles S_n :

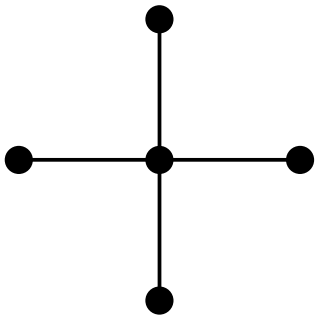
$$\mathbb{E}\big[\mathtt{SLL}(S_n)\big] \;<\; \mathbb{E}\big[\mathtt{LL}(S_n)\big] \;<\; \mathbb{E}\big[\mathtt{ASLL}(S_n)\big]$$

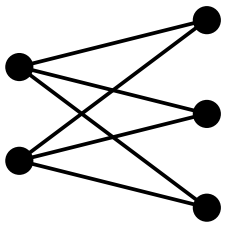
- Les graphes bipartis complets $K_{a,b}$:

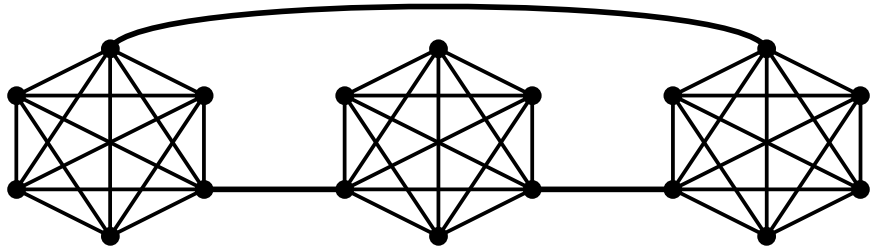
$$\mathbb{E}\big[\mathtt{LL}(K_{a,b})\big] \;<\; \mathbb{E}\big[\mathtt{SLL}(K_{a,b})\big] \;<\; \mathbb{E}\big[\mathtt{ASLL}(K_{a,b})\big]$$

- les colliers $CK_{l,w}$:

$$\mathbb{E}\big[\mathtt{ASLL}(CK_{l,w})\big] \;<\; \mathbb{E}\big[\mathtt{SLL}(CK_{l,w})\big] \;<\; \mathbb{E}\big[\mathtt{LL}(CK_{l,w})\big]$$



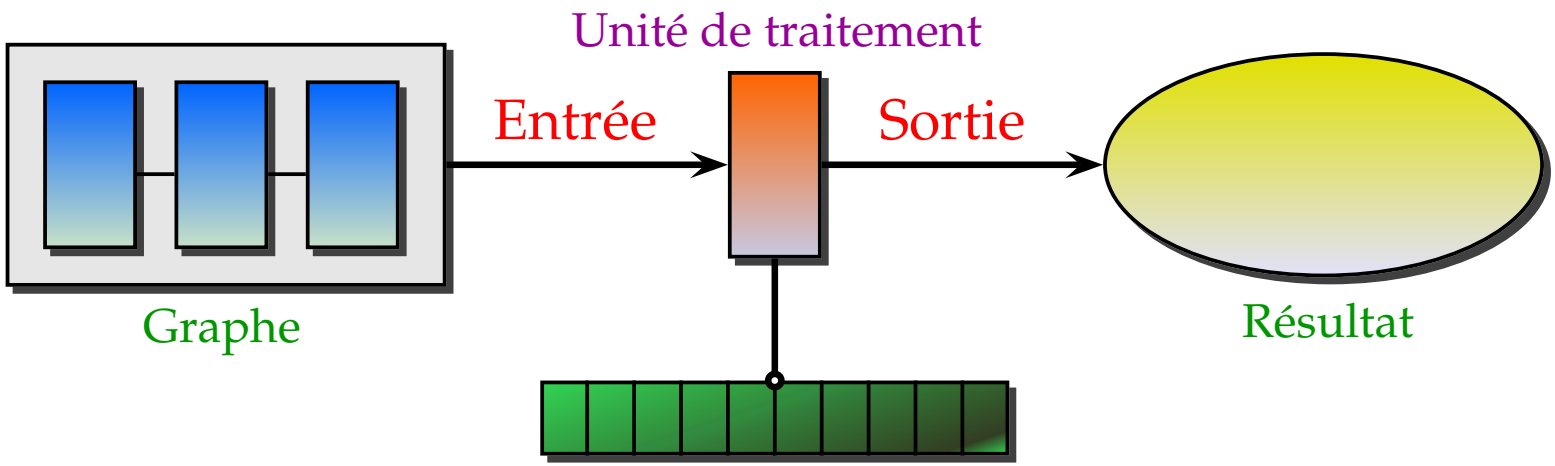




Plan

Avec un peu plus de mémoire...

Modèle relâché



C_1. Pas de modification des données fournies en entrée : *intégrité* de l'instance.

C_2'. L'unité de traitement dispose d'une mémoire d'au plus n bits.

C_3. Envoi de la solution « à la volée » dans « Résultat ».

Adaptation de l'algorithme LR

Algorithme (LR)

Soit $G = (V, E)$ un graphe.

- $C \leftarrow \emptyset$

Pour chaque sommet $u \in V$:

- si $u \notin C$, alors $C \leftarrow C \cup N(u)$.

- $N(u)$: *voisins* de u

Adaptation de l'algorithme LR

Algorithme (LR)

Soit $G = (V, E)$ un graphe.

- $C \leftarrow \emptyset$

Pour chaque sommet $u \in V$:

- si $u \notin C$, alors $C \leftarrow C \cup N(u)$.

- $N(u)$: *voisins* de u

Nombre de requêtes effectuées par LR sur G pour retourner C_{LR} :

$$\sum_{u \notin C_{\mathrm{LR}}} d(u)$$

L'algorithme Pitt

Variante *à mémoire limitée* de l'algorithme probabiliste de *L. Pitt* (Yale, 1985).

Algorithme (Pitt)

Soit $\mathfrak{M}$ la mémoire de taille k.

Traiter les arêtes de $G = (V, E)$ dans un ordre arbitraire.

Pour chaque arête $uv \in E$, si $\{u, v\} \cap \mathfrak{M} = \emptyset$:

- mettre u (resp. v) dans la solution avec probabilité $\frac{1}{2}$;
- l'ajouter à la mémoire (si $|\mathfrak{M}| < k$).

L'algorithme Pitt

Variante *à mémoire limitée* de l'algorithme probabiliste de *L. Pitt* (Yale, 1985).

Algorithme (Pitt)

Soit $\mathfrak{M}$ la mémoire de taille k.

Traiter les arêtes de $G = (V, E)$ dans un ordre arbitraire.

Pour chaque arête $uv \in E$, si $\{u, v\} \cap \mathfrak{M} = \emptyset$:

- mettre u (resp. v) dans la solution avec probabilité $\frac{1}{2}$;
- l'ajouter à la mémoire (si $|\mathfrak{M}| < k$).

- Si $k = n$, $\mathbb{E}\big[\texttt{Pitt}(G)\big] \leq 2 \cdot OPT(G)$ (démontré par *L. Pitt*)
- Si $k = 0$, $\mathbb{E}\big[\texttt{Pitt}(G)\big] = n - \sum_{u \in V} \frac{1}{2^{d(u)}}$

L'algorithme Pitt

Si $k = \frac{n}{2}$, il peut être très mauvais *en moyenne*.

L'algorithme `Pitt`

Si $k = \frac{n}{2}$, il peut être très mauvais *en moyenne*.

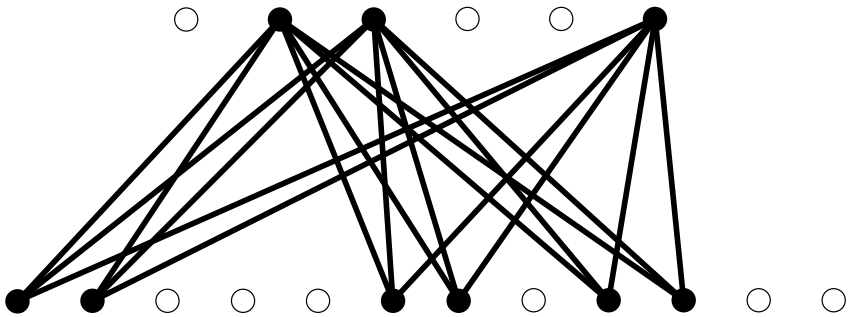
Lorsque la mémoire est pleine, `Pitt` travaille « en aveugle » sur les arêtes restantes.

L'algorithme Pitt

Si $k = \frac{n}{2}$, il peut être très mauvais *en moyenne*.

Lorsque la mémoire est pleine, Pitt travaille « en aveugle » sur les arêtes restantes.

→ Si $G - \mathfrak{M}$ a un degré minimum de 3, on retourne en moyenne $\frac{7}{8}$ des sommets de $V \setminus \mathfrak{M}$.



Graphe induit $G - \mathfrak{M}$

L'algorithme S-Pitt

Apparition des arêtes $uv \in E$ dans un ordre *lexicographique* :

- rangées par ordre croissant des sommets u ;
- $v > u$.

L'algorithme S-Pitt

Apparition des arêtes $uv \in E$ dans un ordre *lexicographique* :

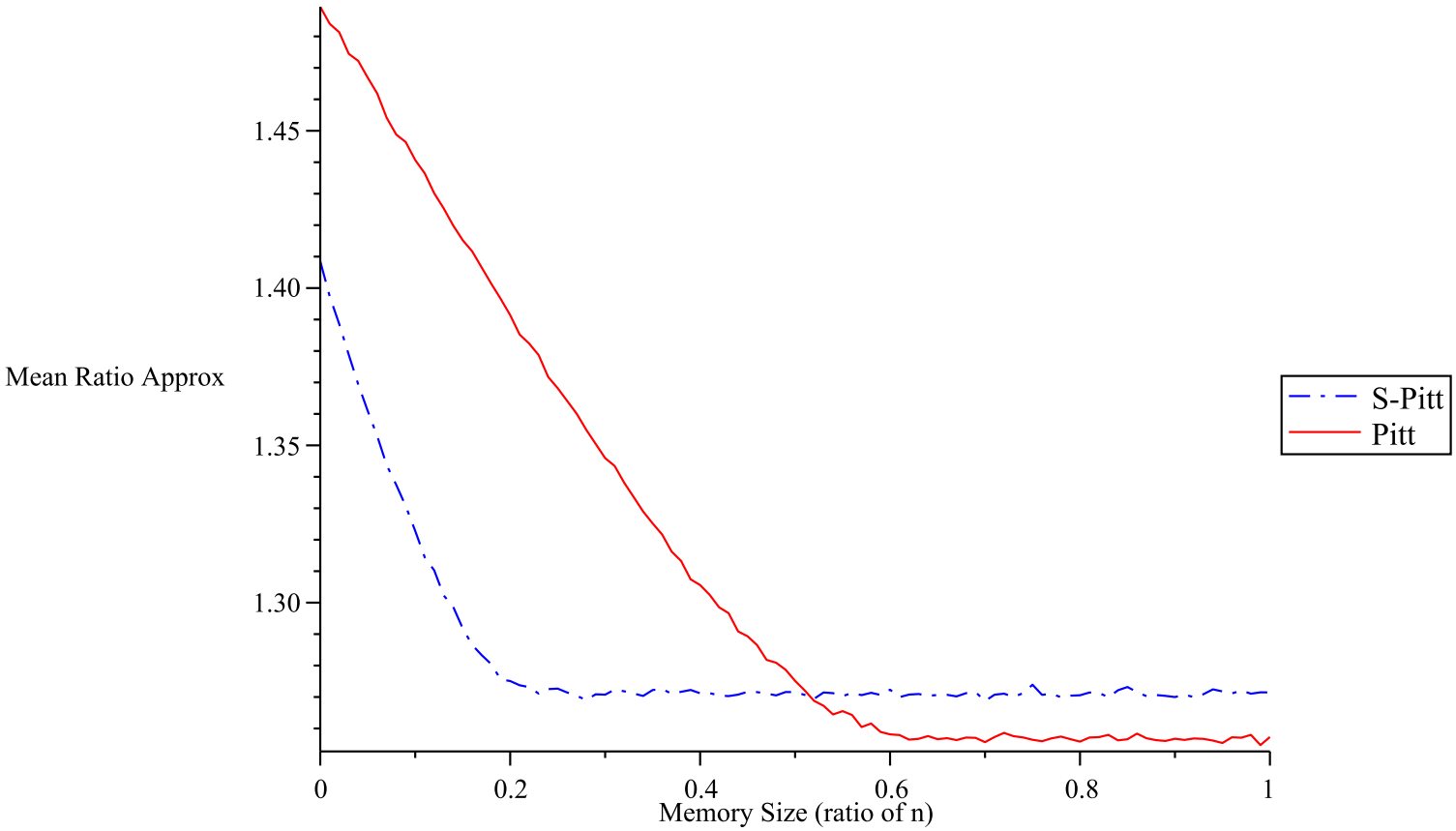
- rangées par ordre croissant des sommets u ;
- $v > u$.

- Pas besoin de stocker tous les sommets de la solution en mémoire.

$\longrightarrow$ Meilleure gestion de la mémoire !

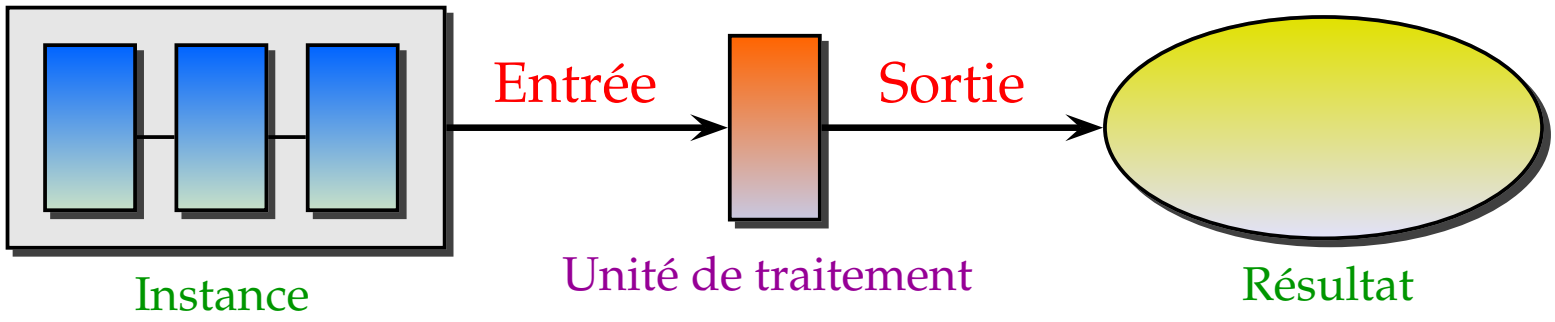
Inconvénient

Dégradation possible de la qualité des solutions (par rapport à `Pitt`) :



(Expérimentations réalisées sur des chemins de taille 100)

Plan



- **Instance :** stockée sur un disque dur externe.
- **Unité de traitement :** une machine standard (cet ordinateur !).
- **Résultat :** écrit « à la volée » sur un disque dur externe.

Implémentation de six algorithmes

`LR`, `ED`, `S-Pitt`, `LL`, `SLL` et `ASLL`.

Implémentation de six algorithmes

`LR`, `ED`, `S-Pitt`, `LL`, `SLL` et `ASLL`.

`LR`, `ED` et `S-Pitt` fonctionnent avec n bits mémoire.

Implémentation de six algorithmes

`LR`, `ED`, `S-Pitt`, `LL`, `SLL` et `ASLL`.

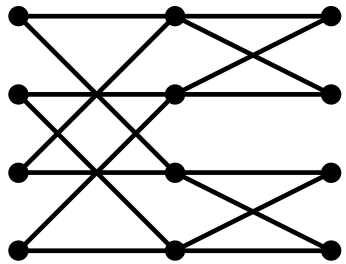
`LR`, `ED` et `S-Pitt` fonctionnent avec n bits mémoire.

`SLL` et `ASLL` ont besoin de calculer les degrés des voisins.

Graphes générés

Des graphes peu denses (où $n \in \mathcal{O}(m)$) :

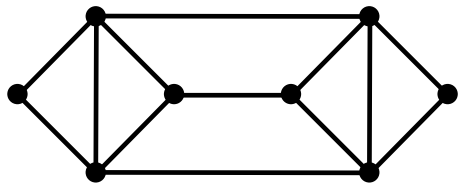
- des *ButterFly* ;
- des graphes de *de Bruijn* ;
- des grilles.



ButterFly de dimension 2

Des graphes denses (où $n \in \Omega(\sqrt{m})$) :

- des hypercubes ;
- des graphes bipartis complets ;
- des *split graphs* complets.

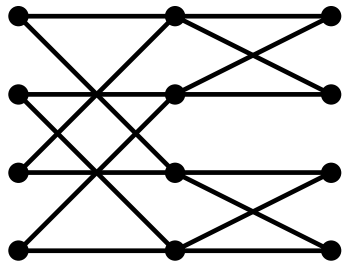


de Bruijn de dimension 3

Graphes générés

Des graphes peu denses (où $n \in \mathcal{O}(m)$) :

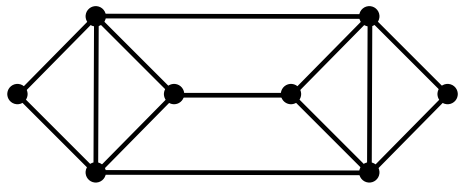
- des *ButterFly* ;
- des graphes de *de Bruijn* ;
- des grilles.



ButterFly de dimension 2

Des graphes denses (où $n \in \Omega(\sqrt{m})$) :

- des hypercubes ;
- des graphes bipartis complets ;
- des *split graphs* complets.



de Bruijn de dimension 3

Pour quelles raisons avons-nous choisi ces graphes ?

1. Ils peuvent facilement être fabriqués « à la volée ».
2. Souvent, $OPT(G) \leq \frac{n}{2}$

Résultats obtenus sur des graphes peu denses

Exemple avec l'instance `butterfly-28`

n	m	OPT
7 784 628 224	15 032 385 540	3 758 096 384

Taille sur le disque : 282 Go

	Qualité ($\times OPT$)	Complexité ($\times m$)	Durées
`LR`	**1**	0,99	$\approx$ 1H12
`ED`	2	**0,61**	$\approx$ 1H16
`S-Pitt`	1,63	0,91	$\approx$ 1H19
`LL`	1,93	1,03	$\approx$ 1H21
`SLL`	1,92	1,01	$\approx$ 5H11
`ASLL`	2	0,72	$\approx$ 3H39

Résultats obtenus sur des graphes denses

Exemple avec l'instance `compbip-35000.500000`

n	m	OPT
535 000	17 500 000 000	35 000

Taille sur le disque : 261 Go

	Qualité ($\times OPT$)	Complexité ($\times m$)	Durées
LR	14,28	1	$\approx$ 24 min
ED	2	0,93	$\approx$ 23 min
S-Pitt	2,01	**0,92**	$\approx$ 22 min
LL	**1**	1,001	$\approx$ 23 min
SLL	**1**	1,001	$\approx$ 6H12
ASLL	14,28	1,002	$\approx$ 6H12

Observations générales

Qualité de solution :

1. `LR` est souvent le meilleur ;
2. suivi de près par `SLL` ;
3. `S-Pitt` et `LL` sont moyens :

→ « variabilité » de `LL` plus forte que `S-Pitt`.

4. `ED` et `ASLL` sont mauvais.

Observations générales

Qualité de solution :

1. `LR` est souvent le meilleur ;
2. suivi de près par `SLL` ;
3. `S-Pitt` et `LL` sont moyens :

→ « variabilité » de `LL` plus forte que `S-Pitt`.

4. `ED` et `ASLL` sont mauvais.

Complexité :

1. `ED` est souvent le meilleur ;
2. `LR` atteint souvent m requêtes,

→ mais il ne peut pas faire pire.

→ `LL`, `SLL` et `ASLL` peuvent atteindre leurs bornes max ($> m$).

Observations générales

Temps d'exécution

SLL et ASLL peuvent être beaucoup plus longs.

$\longrightarrow$ Calcul des degrés des voisins.

Observations générales

Temps d'exécution

`SLL` et `ASLL` peuvent être beaucoup plus longs.
$\longrightarrow$ Calcul des degrés des voisins.

Sur les graphes peu denses (où $n \in \mathcal{O}(m)$), influencés en partie par :

- la qualité des solutions écrites ;
- le nombre de requêtes effectuées.

Sur les graphes denses, influencés surtout par le nombre de requêtes.

Observations générales

Temps d'exécution

`SLL` et `ASLL` peuvent être beaucoup plus longs.
$\longrightarrow$ Calcul des degrés des voisins.

Sur les graphes peu denses (où $n \in \mathcal{O}(m)$), influencés en partie par :

- la qualité des solutions écrites ;
- le nombre de requêtes effectuées.

Sur les graphes denses, influencés surtout par le nombre de requêtes.

Autres aspects à prendre en compte :

- temps d'accès disque, gestion des tampons, *etc.*

$\longrightarrow$ caractéristiques mises en avant dans le modèle *I/O-efficient*.

Limites rencontrées

Sur l'instance `compbip-250000.380000`, de taille :

- $n = 630\ 000$ et $m = 95\ 000\ 000\ 000$ (1,41 To sur disque),

on peut exécuter (presque) tous les algorithmes.

Limites rencontrées

Sur l'instance `compbip-250000.380000`, de taille :

- $n = 630\ 000$ et $m = 95\ 000\ 000\ 000$ (1,41 To sur disque),

on peut exécuter (presque) tous les algorithmes.

Mais, sur l'instance `butterfly-30` :

n	m
33 285 996 544	64 424 509 440

Taille sur le disque : 1,17 To

$\longrightarrow$ seul `LL` peut être exécuté sur notre machine (4 Go RAM) !

(Temps d'exécution $\approx$ 5H48)

Synthèse générale

En dépit de ses performances, l'algorithme *le mieux adapté* est `LL` :

- pas besoin de calculer les degrés des voisins ;
- pas besoin d'allouer n bits en mémoire.

Synthèse générale

En dépit de ses performances, l'algorithme *le mieux adapté* est `LL` :

- pas besoin de calculer les degrés des voisins ;
- pas besoin d'allouer n bits en mémoire.

De plus, il peut facilement être parallélisé :

- amélioration de ses temps d'exécution ;

On pourrait alors :

- l'exécuter plusieurs fois (avec des labels différents) ;

$\longrightarrow$ améliorer la qualité de ses résultats.

Plan

Bilan

Analyse d'algorithmes en $\mathcal{O}(\log n)$ bits mémoire (`LL`, `SLL` et `ASLL`) :

- qualité des solutions produites ;
- complexité en nombre de requêtes.

$\longrightarrow$ Aucun algorithme ne domine les autres !

Bilan

Analyse d'algorithmes en $\mathcal{O}(\log n)$ bits mémoire (`LL`, `SLL` et `ASLL`) :

- qualité des solutions produites ;
- complexité en nombre de requêtes.

$\longrightarrow$ Aucun algorithme ne domine les autres !

Analyse d'algorithmes en n bits mémoire :

- mauvais si espace mémoire $< n$ (`Pitt`) ;
- meilleure gestion de la mémoire avec un ordre *lexicographique* ;

$\longrightarrow$ mais perte de qualité possible.

Bilan

Mise en œuvre pratique sur de très gros graphes :

- taille max $\approx$ 100 milliards (1,4 To).

$\rightarrow$ Temps d'exécution *raisonnables* (moins d'une journée) ;

Meilleure qualité de solution : `LR`.

$\rightarrow$ Inutilisable si n est trop grand.

Bilan

Mise en œuvre pratique sur de très gros graphes :

- taille max $\approx$ 100 milliards (1,4 To).

$\rightarrow$ Temps d'exécution *raisonnables* (moins d'une journée) ;

Meilleure qualité de solution : `LR`.

$\rightarrow$ Inutilisable si n est trop grand.

`LL` est le mieux adapté :

- exécutable sur toutes nos instances ;
- parallélisable, donc possibilité d'améliorer :
  - ses temps d'exécution ;

  $\rightarrow$ la qualité de ses solutions.

Perspectives

Poursuite des travaux théoriques :

- calcul de la *variance* de `LL` (et de la *skewness*) ;
- rapports d'approximation *moyens* des algorithmes ;
- lier la quantité de mémoire disponible et la qualité.

Perspectives

Poursuite des travaux théoriques :

- calcul de la *variance* de `LL` (et de la *skewness*) ;
- rapports d'approximation *moyens* des algorithmes ;
- lier la quantité de mémoire disponible et la qualité.

Expérimentations plus poussées avec `LL` :

- générer d'autres types de graphes ;
- utiliser plusieurs machines ;
- appliquer des *permutations* sur les labels.

Perspectives

Poursuite des travaux théoriques :

- calcul de la *variance* de `LL` (et de la *skewness*) ;
- rapports d'approximation *moyens* des algorithmes ;
- lier la quantité de mémoire disponible et la qualité.

Expérimentations plus poussées avec `LL` :

- générer d'autres types de graphes ;
- utiliser plusieurs machines ;
- appliquer des *permutations* sur les labels.

A plus long terme :

- étudier d'autres variantes du Vertex Cover ;
- étudier d'autres problèmes d'optimisation.

Merci de votre attention !

Avez-vous des questions ?

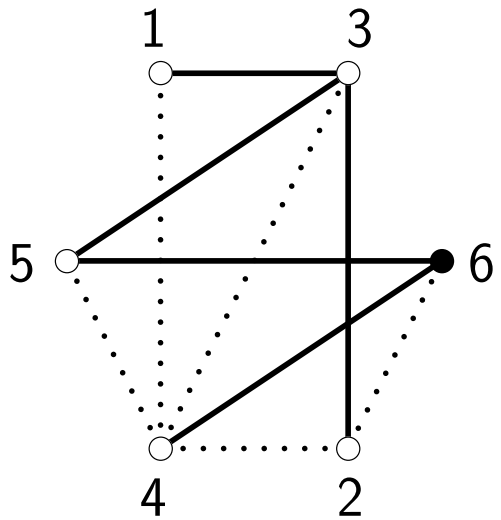
Propriétés spécifiques de LL

Théorème

Pour tout graphe G, il existe toujours un étiquetage L^* tel que LL *retourne une solution optimale sur $G = (V, L^*, E)$.*

Théorème

Pour tout graphe connexe G, il existe toujours un étiquetage L_w tel que LL *retourne une solution de taille $n - 1$ sur $G = (V, L_w, E)$.*



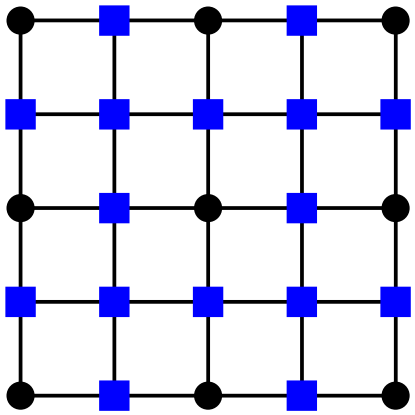
Application sur des grilles $GR_{p\times q}$

$$\mathbb{E}\left[\mathtt{LL}(GR_{p\times q})\right] = \frac{4n}{5} - \frac{p+q}{10} - \frac{2}{15}$$

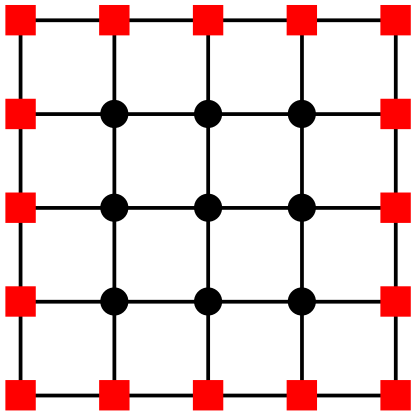
$$\mathbb{E}\left[\mathtt{SLL}(GR_{p\times q})\right] = \frac{4n}{5} + \frac{2(p+q)}{15} - \frac{28}{15}$$

$$\mathbb{E}\left[\mathtt{ASLL}(GR_{p\times q})\right] = \frac{4n}{5} + \frac{3(p+q)}{10} - \frac{8}{15}$$

$\rightarrow \forall p > 2 \quad \text{et} \quad \forall q > 2.$



SLL



ASLL

■ : Sommets $\notin S$ ■ : Sommets $\notin \bar{S}$

Preuve de *M. Glaymann*

Extraite de « *Où le premier n'est pas toujours premier... Pièce probabiliste en trois actes pour des enfants de 10 ans* » (Educational Studies in Mathematics, 1976).

- *Combien faut-il tirer en moyenne de boules pour rencontrer la première boule noire ?*

Démonstration.

Un sac avec p boules noires et $n-p$ boules blanches : les boules noires partagent l'ensemble des boules blanches en $p+1$ parties.
Chaque partie contient en moyenne $\frac{n-p}{p+1}$ boules blanches.
Il faut donc en tirer en moyenne

$$\frac{n-p}{p+1}+1 = \frac{n-p+p+1}{p+1} = \frac{n+1}{p+1}$$

pour obtenir la première boule noire ! □

Complexité moyenne

Formules des espérances pour SLL et ASLL

$$\mathbb{E}\left[\mathrm{Q}_{\mathtt{SLL}}(G)\right] = \sum_{u \in V} \frac{d(u)+1}{\sigma(u)+1}\left(H(d_{\inf}(u)+\sigma(u)+1) - H(d_{\inf}(u))\right) - \sum_{u | d_{\inf}(u)=0} \frac{1}{\sigma(u)+1}$$

$$\mathbb{E}\left[\mathrm{Q}_{\mathtt{ASLL}}(G)\right] = \sum_{u \in V} \frac{d(u)+1}{\sigma(u)+1}\left(H(d_{\sup}(u)+\sigma(u)+1) - H(d_{\sup}(u))\right) - \sum_{u | d_{\sup}(u)=0} \frac{1}{\sigma(u)+1}$$

Application sur des colliers $CK_{l,w}$

En admettant que $l > 1$ et $w > 4$, on a

$$\mathbb{E}\left[\mathrm{Q}_{\mathtt{ASLL}}(CK_{l,w})\right] = n \cdot H(w) - \frac{5n}{18} + \frac{5l}{9}$$

$$\mathbb{E}\left[\mathrm{Q}_{\mathtt{SLL}}(CK_{l,w})\right] = n \cdot H(w-2) + \frac{2l}{3} \cdot \frac{3w^2-1}{w(w-1)} - l$$

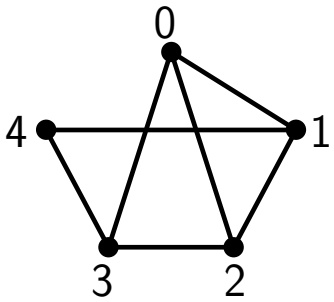
$$\mathbb{E}\left[\mathrm{Q}_{\mathtt{LL}}(CK_{l,w})\right] = n \cdot H(w-1) + \frac{2l}{w}$$

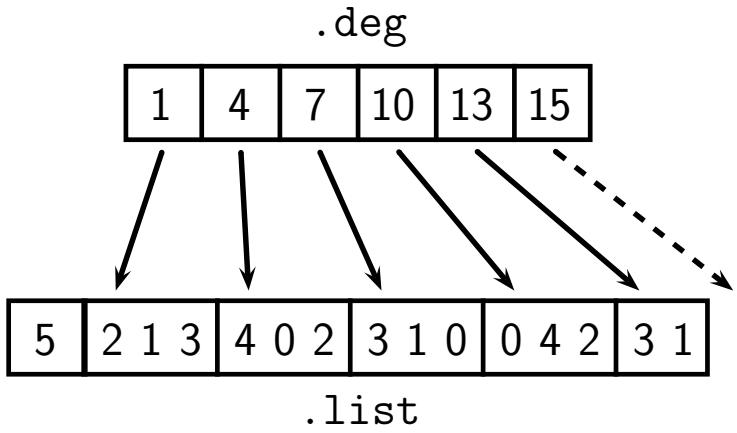
Stockage et lecture d'un graphe

Stockage à l'aide de 2 fichiers :

`.list` : succession des voisins des sommets (stockage de la valeur n au début) $\longrightarrow 2m+1$ valeurs en tout ;

`.deg` : pointeurs délimitant les voisins de chaque sommet (calcul des degrés) $\longrightarrow n+1$ valeurs en tout.





- parcours du fichier `.list` à l'aide du fichier `.deg`
- possibilité d'« enjamber » des voisins dans le fichier `.list`