Introduction aux différents modèles de streaming dans les graphes.

Ugo Giocanti

Université Grenoble Alpes, Laboratoire G-SCOP, France

Journées Calamar 2024

Références

- Andrew McGregor. Graph Stream Algorithms: A Survey (2014).
- Sepehr Assadi lecture's notes about Graph Streaming Algorithms and Lower Bounds (2020).

Algorithmes de streaming (flot)

Motivation: Internet, réseaux neurones.

Algorithmes de streaming (flot)

Motivation: Internet, réseaux neurones.
Entrée: Séquence d'objets S d'un domaine $\mathcal{D}$ (flot de données/data stream).
Sortie: Calculer une valeur/statistique/propriété sur la séquence globale S.

Algorithmes de streaming (flot)

Motivation: Internet, réseaux neurones.
Entrée: Séquence d'objets S d'un domaine $\mathcal{D}$ (flot de données/data stream).
Sortie: Calculer une valeur/statistique/propriété sur la séquence globale S.
But: Minimiser l'espace mémoire utilisé.

Modèles de streaming dans les graphes.

$V = \{v_1, \ldots, v_n\}$ fixé.

Modèles de streaming dans les graphes.

$V = \{v_1, \ldots, v_n\}$ fixé.
Stream graph/(graphe flux?): Séquence $S = \langle e_1, \ldots, e_m \rangle$ d'arêtes, décrivant un graphe $G = (V, \{e_1, \ldots, e_m\})$.

Modèles de streaming dans les graphes.

$V = \{v_1, \dots, v_n\}$ fixé.
Stream graph/(graphe flux?): Séquence $S = \langle e_1, \dots, e_m \rangle$ d'arêtes, décrivant un graphe $G = (V, \{e_1, \dots, e_m\})$.
Dynamic stream graph/(graphe dynamique flux?): Séquence $S = \langle (e_1, c_1), \dots, (e_d, c_d) \rangle$ où: $e_i \in \binom{n}{2}$ et $c_i \in \{\pm 1\}$ (ajout/déletion d'arête). Décrit un graphe G.

Modèles de streaming dans les graphes.

$V = \{v_1, \ldots, v_n\}$ fixé.
Stream graph/(graphe flux?): Séquence $S = \langle e_1, \ldots, e_m \rangle$ d'arêtes, décrivant un graphe $G = (V, \{e_1, \ldots, e_m\})$.
Dynamic stream graph/(graphe dynamique flux?): Séquence $S = \langle (e_1, c_1), \ldots, (e_d, c_d) \rangle$ où: $e_i \in \binom{n}{2}$ et $c_i \in \{\pm 1\}$ (ajout/déletion d'arête). Décrit un graphe G.
Sliding window /fenêtre coulissante: Séquence infinie $S = \langle e_1, e_2, \ldots \rangle$ + largeur $w \gg n$ décrivant une séquence de graphes $(G_t)_{t \geqslant 0}$ où: $G_t = (V, \{e_{t-w+1}, \ldots, e_t\})$.

Algorithmes

Definition

Un algorithme de semi-streaming est un algorithme prenant en entrée un graphe flux et fonctionnant en espace $\widetilde{O}(n) = O(n \cdot \mathrm{polylog}(n))$.

Algorithmes

Definition

Un algorithme de semi-streaming est un algorithme prenant en entrée un graphe flux et fonctionnant en espace $\widetilde{O}(n) = O(n \cdot \mathrm{polylog}(n))$.

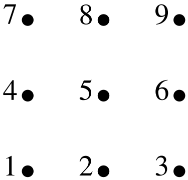
Variantes d'algorithmes considérés:

- 1 ou $O(1)$ passes.
- Déterministes ou randomisés.
- Exacts ou approx.
- Complexités spatiales $\mathrm{polylog}(n)$ ou $o(n^2)$.
- …

(k-)Connexité pour les stream graphs

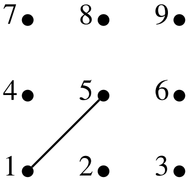
Algorithme semi-streaming pour tester connexité d'un graphe?

Algorithme semi-streaming pour tester connexité d'un graphe?

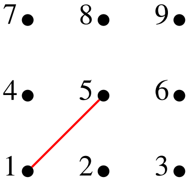


(k-)Connexité pour les stream graphs

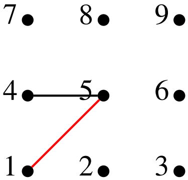
Algorithme semi-streaming pour tester connexité d'un graphe?



Algorithme semi-streaming pour tester connexité d'un graphe?

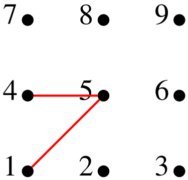


Algorithme semi-streaming pour tester connexité d'un graphe?

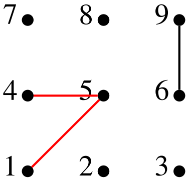


(k-)Connexité pour les stream graphs

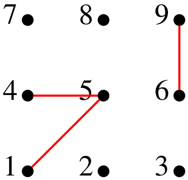
Algorithme semi-streaming pour tester connexité d'un graphe?



Algorithme semi-streaming pour tester connexité d'un graphe?

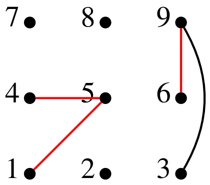


Algorithme semi-streaming pour tester connexité d'un graphe?

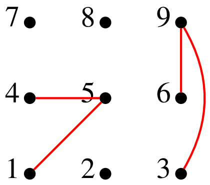


(k-)Connexité pour les stream graphs

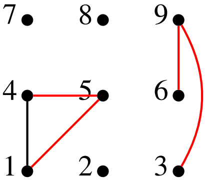
Algorithme semi-streaming pour tester connexité d'un graphe?



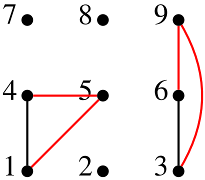
Algorithme semi-streaming pour tester connexité d'un graphe?



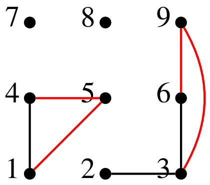
Algorithme semi-streaming pour tester connexité d'un graphe?



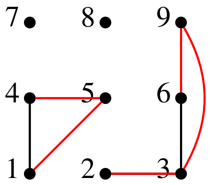
Algorithme semi-streaming pour tester connexité d'un graphe?



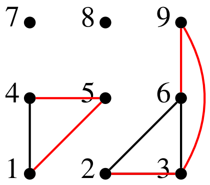
Algorithme semi-streaming pour tester connexité d'un graphe?



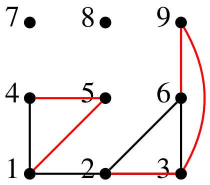
Algorithme semi-streaming pour tester connexité d'un graphe?



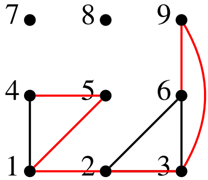
Algorithme semi-streaming pour tester connexité d'un graphe?



Algorithme semi-streaming pour tester connexité d'un graphe?

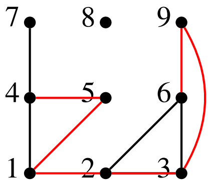


Algorithme semi-streaming pour tester connexité d'un graphe?

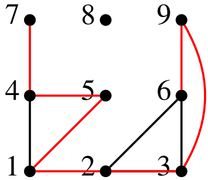


(k-)Connexité pour les stream graphs

Algorithme semi-streaming pour tester connexité d'un graphe?

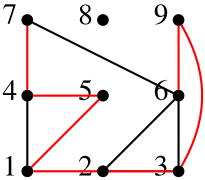


Algorithme semi-streaming pour tester connexité d'un graphe?



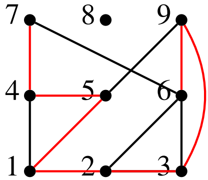
$(k$-$)$Connexité pour les stream graphs

Algorithme semi-streaming pour tester connexité d'un graphe?

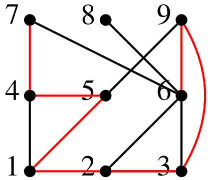


(k-)Connexité pour les stream graphs

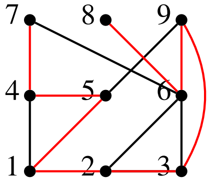
Algorithme semi-streaming pour tester connexité d'un graphe?



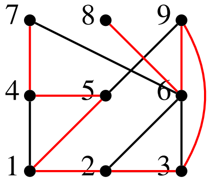
Algorithme semi-streaming pour tester connexité d'un graphe?



Algorithme semi-streaming pour tester connexité d'un graphe?



Algorithme semi-streaming pour tester connexité d'un graphe?

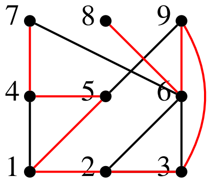


Remark

On sait maintenir une forêt couvrante dans un stream graph en utilisant au plus $2(n-1)\log(n)$ *bits de mémoire.*

(k-)Connexité pour les stream graphs

Algorithme semi-streaming pour tester connexité d'un graphe?



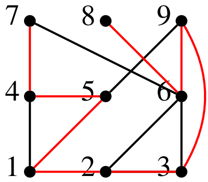
Remark

On sait maintenir une forêt couvrante dans un stream graph en utilisant au plus $2(n-1)\log(n)$ *bits de mémoire.*

Exercice: Algorithme semi-streaming pour tester la k-arête connexité (espace $O(nk\log(n))$)?

(k-)Connexité pour les stream graphs

Algorithme semi-streaming pour tester connexité d'un graphe?



Remark

On sait maintenir une forêt couvrante dans un stream graph en utilisant au plus $2(n-1)\log(n)$ bits de mémoire.

Exercice: Algorithme semi-streaming pour tester la k-arête connexité (espace $O(nk\log(n))$)? Pour biparti?

Bornes inf?

Theorem (Sun, Woodruff (2015))

$\Omega(n log n)$ *bits de mémoire nécessaires pour tester la connexité.*

Bornes inf?

Theorem (Sun, Woodruff (2015))

$\Omega(nk\log n)$ *bits de mémoire nécessaires pour tester la* k*-connexité.*

Bornes inf?

Theorem (Sun, Woodruff (2015))

$\Omega(nk \log n)$ *bits de mémoire nécessaires pour tester la* k*-connexité.*

Pour $k = 1$: Vrai même pour algos randomisés avec proba de succès constante.

Bornes inf?

Theorem (Sun, Woodruff (2015))

$\Omega(nklogn)$ *bits de mémoire nécessaires pour tester la k-connexité.*

Pour $k = 1$: Vrai même pour algos randomisés avec proba de succès constante.
Bornes inf viennent généralement des bornes inf pour des problèmes de protocoles de communication:
"Tout algorithme de streaming utilisant s bits de mémoire et p passes permet d'obtenir un protocole de communication faisant intervenir $O(mp)$ bits de communication."

Bornes inf?

Theorem (Sun, Woodruff (2015))

$\Omega(nklogn)$ bits de mémoire nécessaires pour tester la k-connexité.

Pour $k = 1$: Vrai même pour algos randomisés avec proba de succès constante.
Bornes inf viennent généralement des bornes inf pour des problèmes de protocoles de communication:
"Tout algorithme de streaming utilisant s bits de mémoire et p passes permet d'obtenir un protocole de communication faisant intervenir $O(mp)$ bits de communication."
Ici: $\Omega(n)$ pour la connexité seulement.

Protocoles de communication

$$f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$$

$$f(x_A, x_B)?$$

Alice

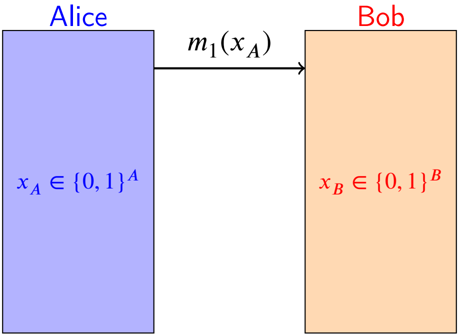
$x_A \in \{0,1\}^A$

Bob

$x_B \in \{0,1\}^B$

$$f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$$

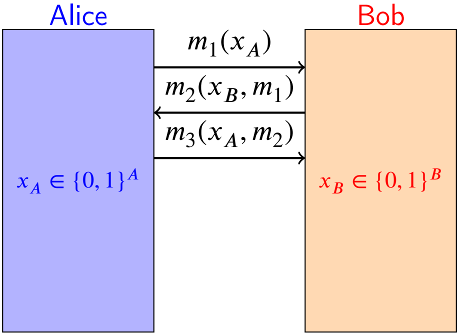
$$f(x_A, x_B)?$$



Protocoles de communication

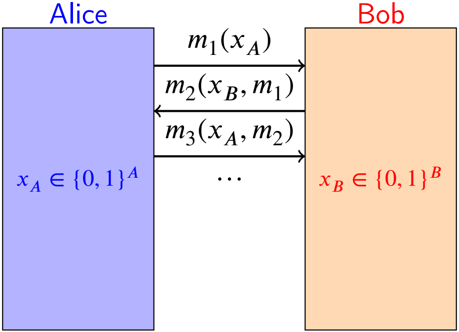
$$f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$$

$$f(x_A, x_B)?$$



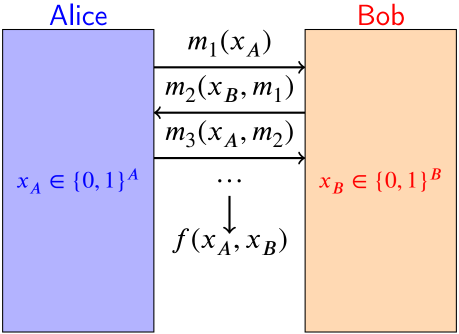
$$f\ :\ \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$$

$$f(x_A, x_B)?$$



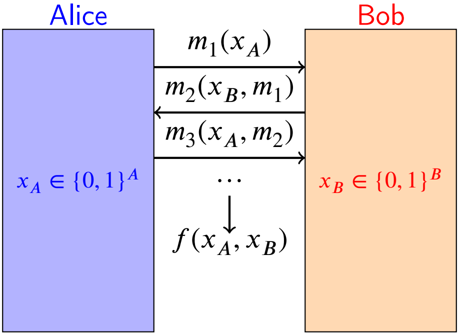
$$f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$$

$$f(x_A, x_B)?$$



$$f : \{0,1\}^A \times \{0,1\}^B \to \{0,1\}$$

$$f(x_A, x_B)?$$

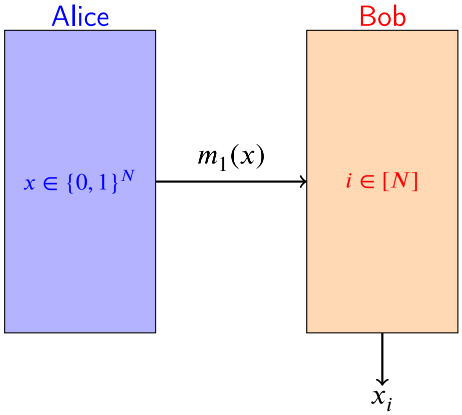


But: minimiser nombre de bits total transmis.

$D(f) :=$ coût du meilleur protocole calculant f.

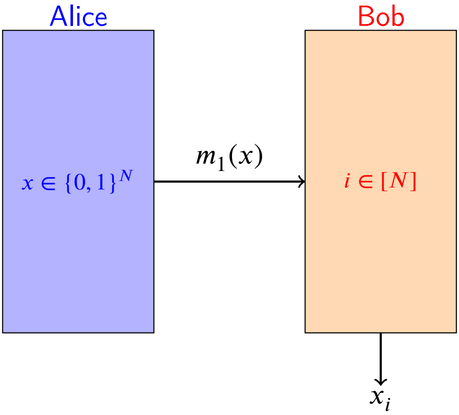
Index

$$Ind_N(x,i) := x_i$$



Index

$$Ind_N(x, i) := x_i$$



→ N bits de communication nécessaires (tiroirs!).

Connexité vers Index

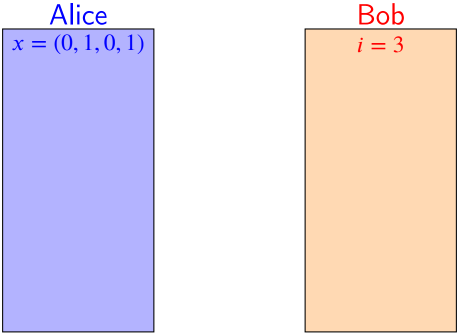
Lemma

Tout algorithme de streaming pour résoudre Conn_n *avec* $s(n)$ *bits de mémoire donne un protocole pour* Ind_{n-2} *avec* 1 *message de taille* $s(n)$.

Connexité vers Index

Lemma

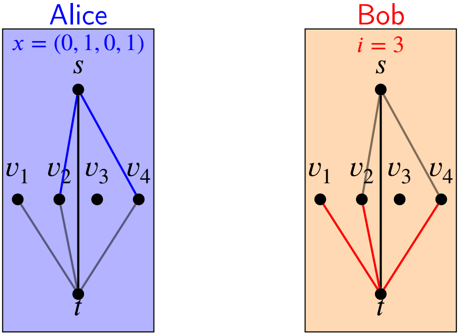
Tout algorithme de streaming pour résoudre Conn_n *avec* $s(n)$ *bits de mémoire donne un protocole pour* Ind_{n-2} *avec* 1 *message de taille* $s(n)$.



Connexité vers Index

Lemma

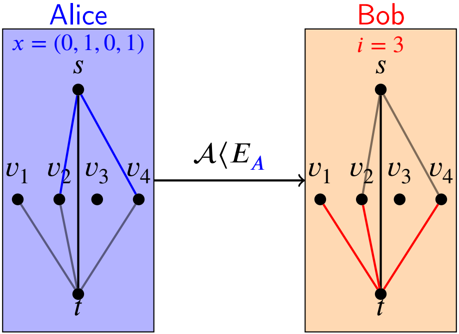
Tout algorithme de streaming pour résoudre Conn_n *avec* $s(n)$ *bits de mémoire donne un protocole pour* Ind_{n-2} *avec* 1 *message de taille* $s(n)$*.*



Connexité vers Index

Lemma

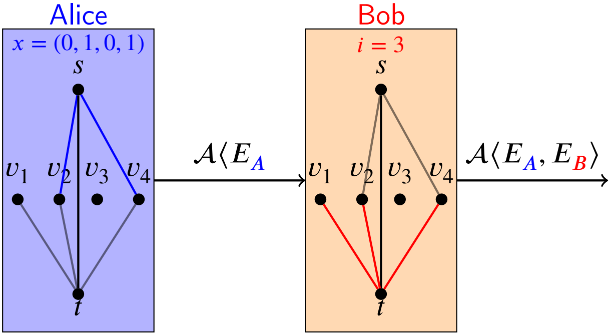
Tout algorithme de streaming pour résoudre Conn_n *avec* $s(n)$ *bits de mémoire donne un protocole pour* Ind_{n-2} *avec* 1 *message de taille* $s(n)$*.*



Connexité vers Index

Lemma

Tout algorithme de streaming pour résoudre Conn_n *avec* $s(n)$ *bits de mémoire donne un protocole pour* Ind_{n-2} *avec* 1 *message de taille* $s(n)$*.*



Connexité pour les dynamic stream graphs

Suppression d'arête autorisée.

Suppression d'arête autorisée.
Entrée: $S = \langle (e_1, \Delta_1), (e_2, \Delta_2), \ldots \rangle$, $e_i \in \binom{V}{2}$ et $\Delta_i \in \{\pm 1\}$.
Graphe considéré: $G = (V, E)$ où $e \in E(G)$ lorsque $\sum_{e_i = e} \Delta_i = 1$.

Suppression d'arête autorisée.
Entrée: $S = \langle (e_1, \Delta_1), (e_2, \Delta_2), \ldots \rangle$, $e_i \in \binom{V}{2}$ et $\Delta_i \in \{\pm 1\}$.
Graphe considéré: $G = (V, E)$ où $e \in E(G)$ lorsque $\sum_{e_i = e} \Delta_i = 1$.
En déterministe, on ne peut plus faire grand chose.

Connexité pour les dynamic stream graphs

Suppression d'arête autorisée.
Entrée: $S = \langle (e_1, \Delta_1), (e_2, \Delta_2), \ldots \rangle$, $e_i \in \binom{V}{2}$ et $\Delta_i \in \{\pm 1\}$.
Graphe considéré: $G = (V, E)$ où $e \in E(G)$ lorsque $\sum_{e_i = e} \Delta_i = 1$.
En déterministe, on ne peut plus faire grand chose.

Lemma (folklore)

Tout algo déterministe prenant en entrée un stream graph dynamique G et renvoyant une arête $e \in E(G)$ utilise au moins $\binom{n}{2}$ bits de mémoire.

Connexité pour les dynamic stream graphs

Suppression d'arête autorisée.
Entrée: $S = \langle (e_1, \Delta_1), (e_2, \Delta_2), \dots \rangle$, $e_i \in \binom{V}{2}$ et $\Delta_i \in \{\pm 1\}$.
Graphe considéré: $G = (V, E)$ où $e \in E(G)$ lorsque $\sum_{e_i = e} \Delta_i = 1$.
En déterministe, on ne peut plus faire grand chose.

Lemma (folklore)

Tout algo déterministe prenant en entrée un stream graph dynamique G et renvoyant une arête $e \in E(G)$ utilise au moins $\binom{n}{2}$ bits de mémoire.

Preuve: Exercice: Réduction à $\mathrm{Ind}_{\binom{n}{2}}$. □

l_0-sampling

Approche vecteur: $\mathbf{x} \in D^{N}$, $D \subseteq \mathbb{R}$.

l_0-sampling

Approche vecteur: $\mathbf{x} \in D^{N}$, $D \subseteq \mathbb{R}$.
Initialement $\mathbf{x} = (0, \ldots, 0)$.

l_0-sampling

Approche vecteur: $\mathbf{x} \in D^N$, $D \subseteq \mathbb{R}$.
Initialement $\mathbf{x} = (0, \dots, 0)$.
Flot de mises à jour $S = \langle (i_1, \Delta_1), (i_2, \Delta_2), \dots, (i_d, \Delta_d) \rangle$ où (i_k, Δ_k) correspond à l'action: $x_{i_k} \leftarrow x_{i_k} + \Delta_k$.
$\text{Supp}(\mathbf{x}) := \{i \in [N], x_i \neq 0\}$.

l_0-sampling

Approche vecteur: $\mathbf{x} \in D^N$, $D \subseteq \mathbb{R}$.
Initialement $\mathbf{x} = (0, \dots, 0)$.
Flot de mises à jour $S = \langle (i_1, \Delta_1), (i_2, \Delta_2), \dots, (i_d, \Delta_d) \rangle$ où (i_k, Δ_k) correspond à l'action: $x_{i_k} \leftarrow x_{i_k} + \Delta_k$.
$\mathrm{Supp}(\mathbf{x}) := \{i \in [N], x_i \neq 0\}$.

Remark

graphe $G \leftrightarrow \mathbf{x} \in \{0, 1\}^{\binom{n}{2}}$.

l_0-sampling

Approche vecteur: $\mathbf{x} \in D^N$, $D \subseteq \mathbb{R}$.
Initialement $\mathbf{x} = (0, \dots, 0)$.
Flot de mises à jour $S = \langle (i_1, \Delta_1), (i_2, \Delta_2), \dots, (i_d, \Delta_d) \rangle$ où (i_k, Δ_k) correspond à l'action: $x_{i_k} \leftarrow x_{i_k} + \Delta_k$.
$\mathrm{Supp}(\mathbf{x}) := \{i \in [N], x_i \neq 0\}$.

Remark

graphe $G \leftrightarrow \mathbf{x} \in \{0, 1\}^{\binom{n}{2}}$.

Problem (l_0-sampling)

Entrée: *flot* S *de mises à jour de* $\mathbf{x} \in \mathbb{R}^N$.
Sortie: *Simuler un tirage* $i \in Supp(\mathbf{x})$ *avec probabilité* $\frac{1}{|Supp(\mathbf{x})|}$.

$$(0, 0, 0, 0, 0)$$

$$(0, 1, 0, 0, 0)$$

$$S = \langle (2, +1)$$

$$(0, 1, 0, 1, 0)$$

$$S = \langle (2, +1), (4, +1)$$

$$(-1, 1, 0, 1, 0)$$

$$S = \langle (2, +1), (4, +1), (1, -1)$$

$$(-1, 1, 0, 1, 1)$$

$$S = \langle (2, +1), (4, +1), (1, -1), (5, +1)$$

$$(-1, 0, 0, 1, 1)$$

$$S = \langle (2, +1), (4, +1), (1, -1), (5, +1), (2, -1)$$

$$(-1, 0, 2, 1, 1)$$

$$S = \langle (2, +1), (4, +1), (1, -1), (5, +1), (2, -1), (3, +2)$$

$$(-1, 0, 2, 1, 1)$$

$$S = \langle (2, +1), (4, +1), (1, -1), (5, +1), (2, -1), (3, +2) \rangle$$

l_0-sampling sur x: simuler $\mathscr{U}(\{1, 3, 4, 5\})$.

"En s'autorisant un peu d'erreur, on peut simuler un l_0-sampling."

Theorem (Jowhari, Salgam, Tardos (2011))

Pour tout $\delta > 0$, il existe un algorithme randomisé effectuant du sampling l_0 sur un vecteur $\mathbf{x} \in \mathbb{R}^N$ prenant en entrée un flot S de mises à jour, utilisant $O(\log^2(N)\log(1/\delta))$ bits d'espace et avec probabilité d'erreur $\leqslant \delta$.

"En s'autorisant un peu d'erreur, on peut simuler un l_0-sampling."

Theorem (Jowhari, Salgam, Tardos (2011))

Pour tout $\delta > 0$, il existe un algorithme randomisé effectuant du sampling l_0 sur un vecteur $\mathbf{x} \in \mathbb{R}^N$ prenant en entrée un flot S de mises à jour, utilisant $O(\log^2(N)\log(1/\delta))$ bits d'espace et avec probabilité d'erreur $\leqslant \delta$.

Si r: graine aléatoire de l'algo, ce qui est renvoyé est une projection linéaire $M_r \cdot \mathbf{x} \in \mathbb{R}^d$ où $M_r \in \mathbb{R}^{d \times N}$ et $d = O(\log(N))$ (M_r pas calculée intégralement durant l'exécution).

l_0-sampling, esquisses linéaires

"En s'autorisant un peu d'erreur, on peut simuler un l_0-sampling."

Theorem (Jowhari, Salgam, Tardos (2011))

Pour tout $\delta > 0$, il existe un algorithme randomisé effectuant du sampling l_0 sur un vecteur $\mathbf{x} \in \mathbb{R}^N$ prenant en entrée un flot S de mises à jour, utilisant $O(\log^2(N)\log(1/\delta))$ bits d'espace et avec probabilité d'erreur $\leqslant \delta$.

Si r: graine aléatoire de l'algo, ce qui est renvoyé est une projection linéaire $M_r \cdot \mathbf{x} \in \mathbb{R}^d$ où $M_r \in \mathbb{R}^{d \times N}$ et $d = O(\log(N))$ (M_r pas calculée intégralement durant l'exécution).
Linéarité ⇒ si $M_r \cdot \mathbf{x}, M_r \cdot \mathbf{y} \in \mathbb{R}^d$ ont été renvoyés, alors on peut en déduire $M_r \cdot (\mathbf{x} + \mathbf{y}) = M_r \cdot \mathbf{x} + M_r \cdot \mathbf{y}$ et faire du l_0-sampling sur $\mathbf{x} + \mathbf{y}$.

l_0-sampling dans des coupes

Une coupe de G est une paire $(S, V \setminus S)$ pour $S \subseteq V$.
$\delta(S) := E(G[S, V \setminus S])$.

l_0-sampling dans des coupes

Une coupe de G est une paire $(S, V \setminus S)$ pour $S \subseteq V$.
$\delta(S) := E(G[S, V \setminus S])$.
Pour chaque $v_k \in V$, on définit $\mathbf{a}^k \in \{-1, 0, 1\}^{\binom{n}{2}}$:

$$\mathbf{a}^k_{i,j} := \begin{cases} -1 & \text{si } k = i \leqslant j \text{ et } v_i v_j \in E(G) \\ 1 & \text{si } k = j \geqslant i \text{ et } v_i v_j \in E(G) \\ 0 & \text{sinon.} \end{cases}$$

l_0-sampling dans des coupes

Une coupe de G est une paire $(S, V \setminus S)$ pour $S \subseteq V$.
$\delta(S) := E(G[S, V \setminus S])$.
Pour chaque $v_k \in V$, on définit $\mathbf{a}^k \in \{-1, 0, 1\}^{\binom{n}{2}}$:

$$\mathbf{a}^k_{i,j} := \begin{cases} -1 & \text{si } k = i \leqslant j \text{ et } v_i v_j \in E(G) \\ 1 & \text{si } k = j \geqslant i \text{ et } v_i v_j \in E(G) \\ 0 & \text{sinon.} \end{cases}$$

Si $S \subseteq V$, $\mathbf{a}^S := \sum_{v_k \in S} a^k \in \{-1, 0, 1\}^{\binom{n}{2}}$.

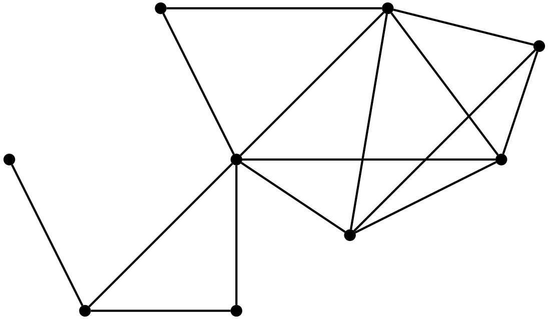
Remark

$\delta(S) \leftrightarrow \mathrm{Supp}(a^S)$

Ce que l'on va simuler:

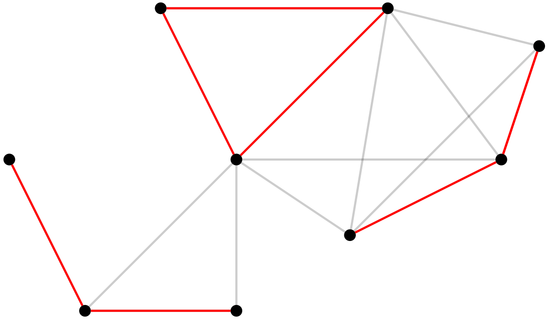
Connexité dans un stream graph dynamique

Ce que l'on va simuler:



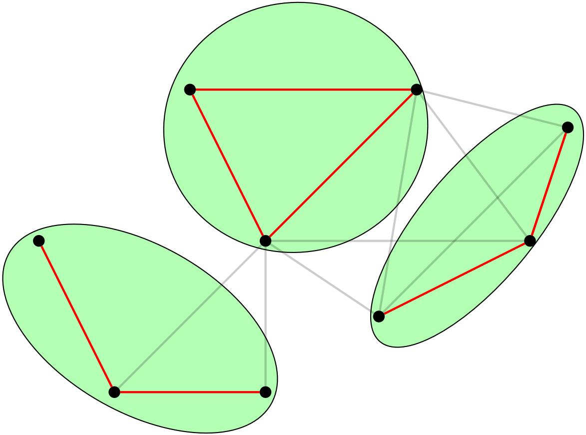
Connexité dans un stream graph dynamique

Ce que l'on va simuler:



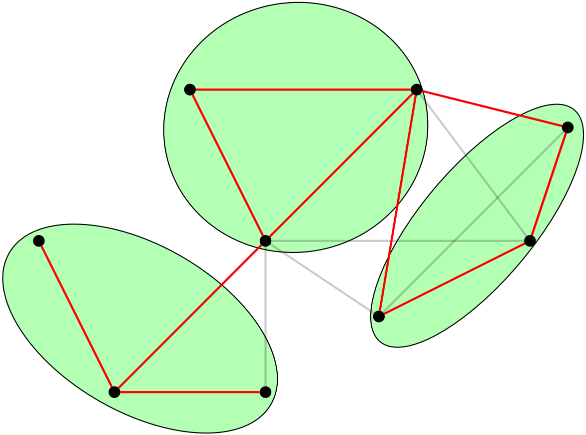
Connexité dans un stream graph dynamique

Ce que l'on va simuler:



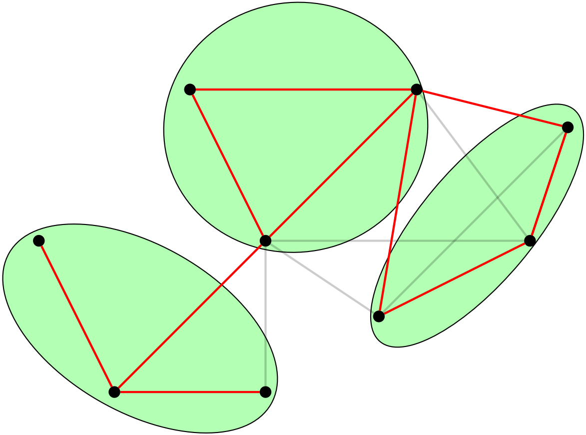
Connexité dans un stream graph dynamique

Ce que l'on va simuler:



Connexité dans un stream graph dynamique

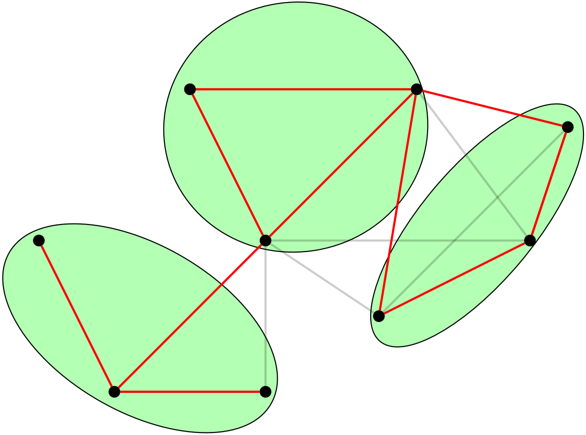
Ce que l'on va simuler:



Si on sait faire du l_0-sampling sur les a^S à chaque étape: $\log(n)$ étapes suffisent.

Connexité dans un stream graph dynamique

Ce que l'on va simuler:



Si on sait faire du l_0-sampling avec erreur $\leqslant \delta$ sur les a^S à chaque étape:
$O_\delta(\log(n))$ étapes suffisent pour terminer avec proba d'erreur $\leqslant \delta$.

Algo R:

Phase 1: On lit le flot S en entrée, et pour chaque $v_k \in V$, on fait du l_0-sampling $t = O_\delta(\log(n))$ fois sur $\mathbf{a}^k \in \{-1, 0, 1\}^{\binom{n}{2}}$. Soient $M_1 \cdot \mathbf{a}^k, \ldots, M_t \cdot \mathbf{a}^k \in \mathbb{R}^d$ les t vecteurs obtenus, $d = O_\delta(\log^2(n))$.
$\rightarrow$ On stocke $n \cdot t = O_\delta(n \log(n))$ esquisses de taille $O_\delta(\log^2(n))$.

Algo R:

Phase 1: On lit le flot S en entrée, et pour chaque $v_k \in V$, on fait du l_0-sampling $t = O_\delta(\log(n))$ fois sur $\mathbf{a}^k \in \{-1, 0, 1\}^{\binom{n}{2}}$. Soient $M_1 \cdot \mathbf{a}^k, \ldots, M_t \cdot \mathbf{a}^k \in \mathbb{R}^d$ les t vecteurs obtenus, $d = O_\delta(\log^2(n))$.

→ On stocke $n \cdot t = O_\delta(n \log(n))$ esquisses de taille $O_\delta(\log^2(n))$.

Phase 2: "Post processing": $\hat{V} := V$.

Pour $i = 1$ à t: Pour $S \in \hat{V}$, tirer une arête dans $\delta(S)$ si possible avec $M_i \cdot \mathbf{a}^S := \sum_{v \in S} M_i \cdot \mathbf{a}^v \in \mathbb{R}^d$.

Si SS' tirée: remplacer S et S' par $S \cup S'$ dans $\hat{V}$ et les sketchs $M_j \cdot \mathbf{a}^S, M_j \cdot \mathbf{a}^{S'}$ par $M_j \cdot (\mathbf{a}^S + \mathbf{a}^{S'}) \cdot \mathbf{a}^S M_j \cdot \mathbf{a}^S + M_j \cdot \mathbf{a}^{S'}$ dans la mémoire (pour tout $j \geqslant i$).

Connexité dans un stream graph dynamique

Algo R:

Phase 1: On lit le flot S en entrée, et pour chaque $v_k \in V$, on fait du l_0-sampling $t = O_\delta(\log(n))$ fois sur $\mathbf{a}^k \in \{-1, 0, 1\}^{\binom{n}{2}}$. Soient $M_1 \cdot \mathbf{a}^k, \dots, M_t \cdot \mathbf{a}^k \in \mathbb{R}^d$ les t vecteurs obtenus, $d = O_\delta(\log^2(n))$.

$\rightarrow$ On stocke $n \cdot t = O_\delta(n \log(n))$ esquisses de taille $O_\delta(\log^2(n))$.

Phase 2: "Post processing": $\hat{V} := V$.

Pour $i = 1$ à t: Pour $S \in \hat{V}$, tirer une arête dans $\delta(S)$ si possible avec $M_i \cdot \mathbf{a}^S := \sum_{v \in S} M_i \cdot \mathbf{a}^v \in \mathbb{R}^d$.

Si SS' tirée: remplacer S et S' par $S \cup S'$ dans $\hat{V}$ et les sketchs $M_j \cdot \mathbf{a}^S, M_j \cdot \mathbf{a}^{S'}$ par $M_j \cdot (\mathbf{a}^S + \mathbf{a}^{S'}) \cdot \mathbf{a}^S M_j \cdot \mathbf{a}^S + M_j \cdot \mathbf{a}^{S'}$ dans la mémoire (pour tout $j \geqslant i$).

Theorem (Ahn, Guha, McGregor (2012))

Algo R utlise $O_\delta(n \cdot \log^2(n))$ bits de mémoire et répond correctement au problème de connexité avec probabilité d'erreur $\leqslant \delta$.

Merci!