Construction Locale de Sous-Graphes Couvrants Peu Denses

Bilel Derbel (LIFL/INRIA - univ. Lille 1),

Cyril Gavoille (LaBRI - univ. Bordeaux 1),

David Peleg (The Weizmann Institute, Israel),

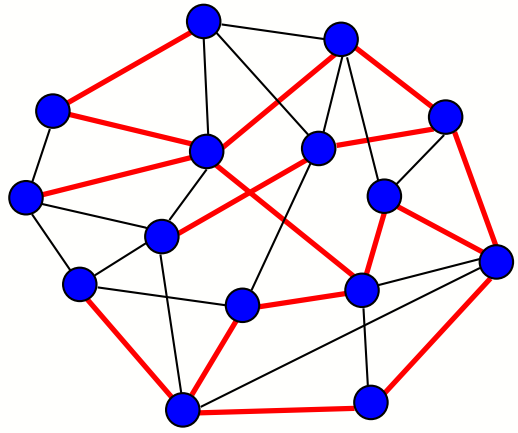
Laurent Viennot (INRIA - univ. Paris 7)

AlgoTel'08 (Saint Malo)

16 Mai 2008

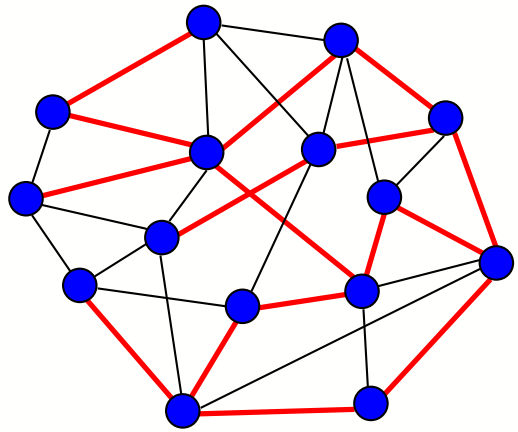
Définition du problème

- Un spanner S de G est sous graphe couvrant de G
  - e.g., un arbre, le graphe entier.



Définition du problème

- Un spanner S de G est sous graphe couvrant de G
  - e.g., un arbre, le graphe entier.



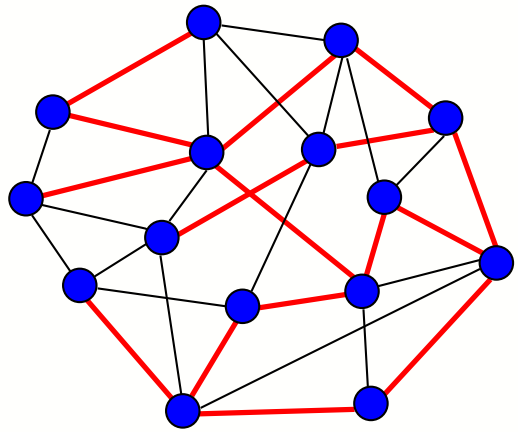
  - Le facteur d'étirement (α, β) du spanner S :

$$d_S(u,v) \leq \alpha \cdot d_G(u,v) + \beta.$$

  - La taille du spanner S = le nombre d'arêtes de S

Définition du problème

- Un spanner S de G est sous graphe couvrant de G
  - e.g., un arbre, le graphe entier.



  - Le facteur d'étirement (α, β) du spanner S :
$$d_S(u,v) \leq \alpha \cdot d_G(u,v) + \beta.$$
  - La taille du spanner S = le nombre d'arêtes de S

- Pour tout graphe, il existe un $(2k-1, 0)$-spanner avec $O(n^{1+1/k})$ arêtes
  - e.g., $(3,0)$-spanner avec $O(n^{3/2})$ arêtes.

But

- On veut construire de façon distribuée et efficace des spanners optimaux

But

- On veut construire de façon distribuée et efficace des spanners optimaux
- La localité de la construction d'un spanner ?

But

- On veut construire de façon distribuée et efficace des spanners optimaux
- La localité de la construction d'un spanner ?
  - En t-unité de temps un sommet ne peut connaître qu'une information concernant son voisinage à distance t.
  - Le modèle de Linial ($\mathcal{LOCAL}$ model) :
    - synchrone, taille des messages à priori non bornée.

But

- On veut construire de façon distribuée et efficace des spanners optimaux
- La localité de la construction d'un spanner ?
  - En t-unité de temps un sommet ne peut connaître qu'une information concernant son voisinage à distance t.
  - Le modèle de Linial ($\mathcal{LOCAL}$ model) :
    - synchrone, taille des messages à priori non bornée.

	(α, β)	taille	taille moyenne	temps
[BS'03]	$(2k-1, 0)$	$-$	$O(kn^{1+1/k})$	k
[DGP'07]	$(4k-5, 0)$	$O(kn^{1+1/k})$	$-$	$2^{O(k)} \log^{k-1} n$

But

- On veut construire de façon distribuée et efficace des spanners optimaux
- La localité de la construction d'un spanner ?
  - En t-unité de temps un sommet ne peut connaître qu'une information concernant son voisinage à distance t.
  - Le modèle de Linial ($\mathcal{LOCAL}$ model) :
    - synchrone, taille des messages à priori non bornée.

	(α, β)	taille	taille moyenne	temps
[BS'03]	$(2k-1, 0)$	–	$O(kn^{1+1/k})$	k
[DGP'07]	$(4k-5, 0)$	$O(kn^{1+1/k})$	–	$2^{O(k)} \log^{k-1} n$
Nouveau	$(2k-1, 0)$	$O(kn^{1+1/k})$	–	k

Techniques de base

- Dans les travaux passés :
  - Construire une décomposition du graphe en clusteurs
    - couvrir les arêtes de chaque clusteur
    - couvrir les arêtes connectant les clusteurs
  - Éliminer les petits cycles : Algorithmes gloutons et séquentiels en général
    - passage au distribué : construction d'une décomposition

Techniques de base

- Dans les travaux passés :
  - Construire une décomposition du graphe en clusteurs
    - couvrir les arêtes de chaque clusteur
    - couvrir les arêtes connectant les clusteurs
  - Éliminer les petits cycles : Algorithmes gloutons et séquentiels en général
    - passage au distribué : construction d'une décomposition
  - Les algorithmes les plus rapides pour construire les décompositions souhaitées sont au mieux en $\log^{O(1)} n$ temps

Techniques de base

- Dans les travaux passés :
  - Construire une décomposition du graphe en clusteurs
    - couvrir les arêtes de chaque clusteur
    - couvrir les arêtes connectant les clusteurs
  - Éliminer les petits cycles : Algorithmes gloutons et séquentiels en général
    - passage au distribué : construction d'une décomposition
  - Les algorithmes les plus rapides pour construire les décompositions souhaitées sont au mieux en $\log^{O(1)} n$ temps

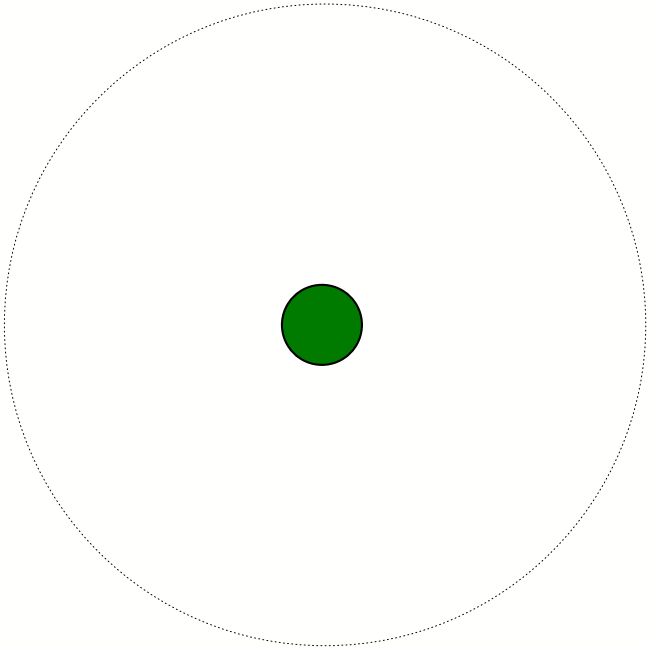
- Notre technique :
  - Construire localement des chemins qui couvrent les arêtes de chaque sommet

Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner

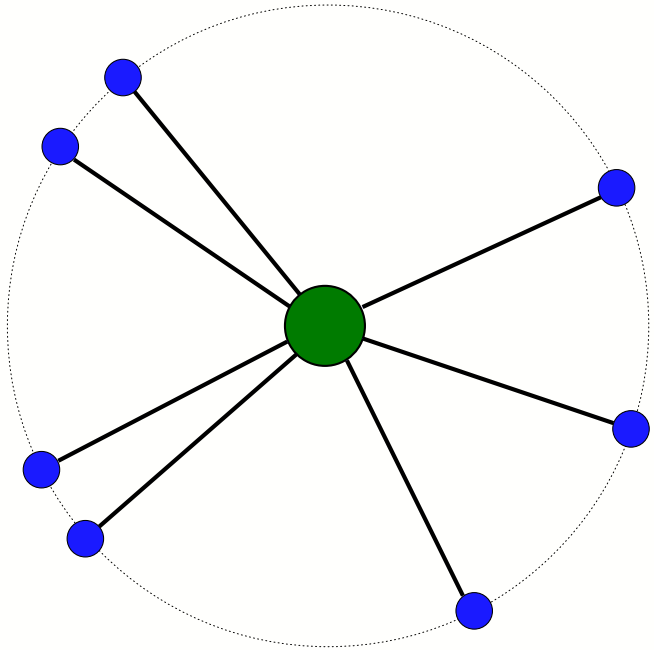
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



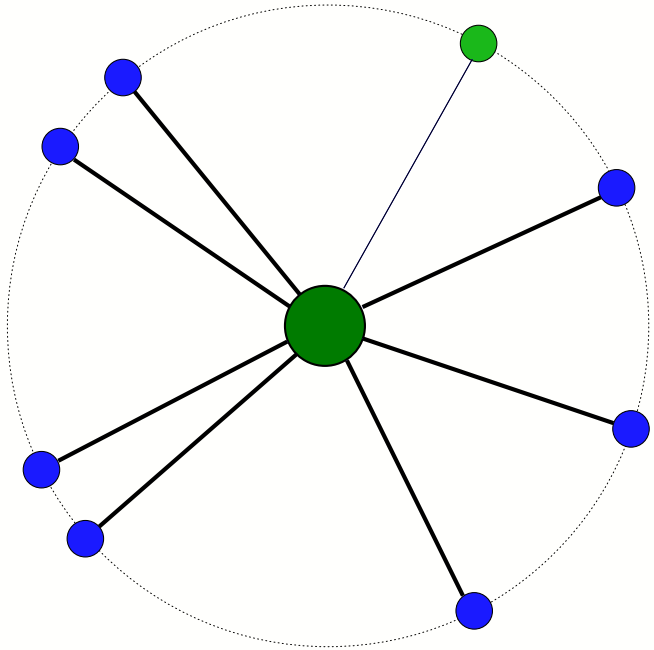
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



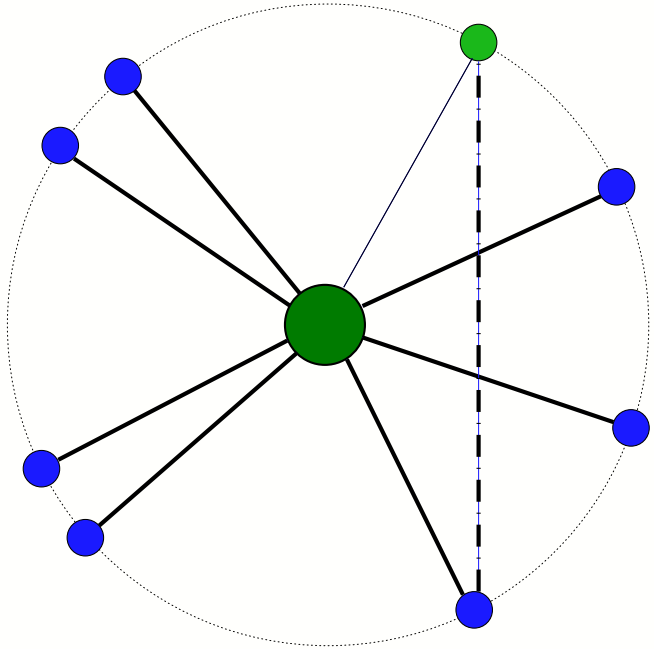
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



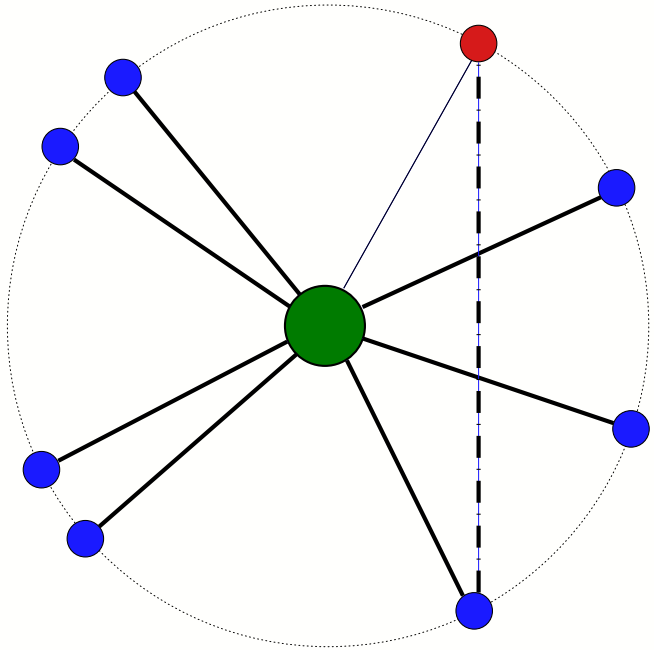
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



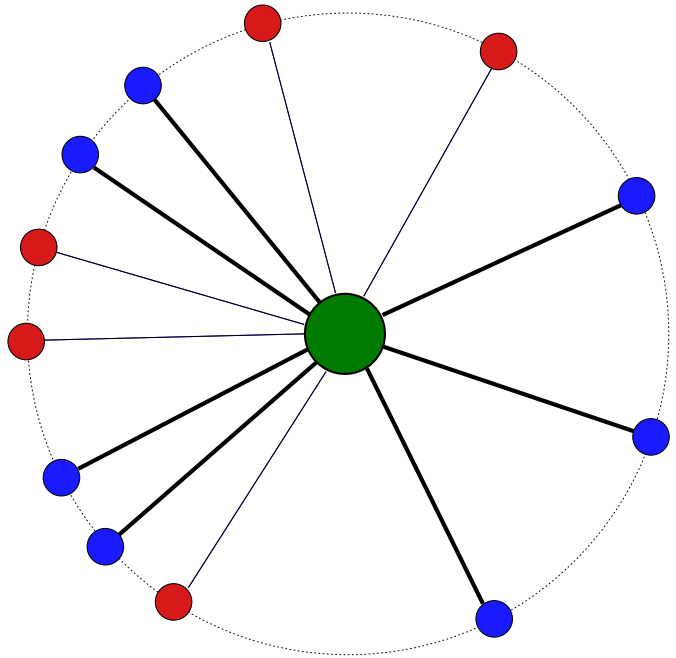
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



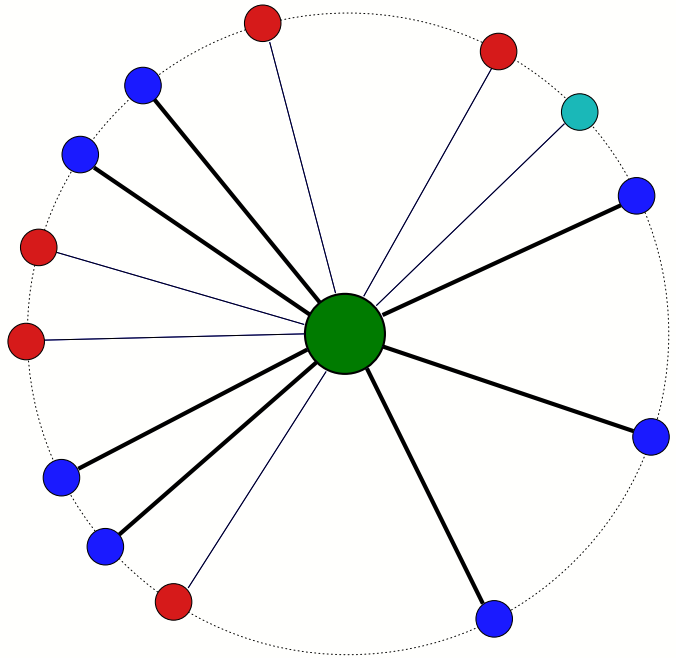
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



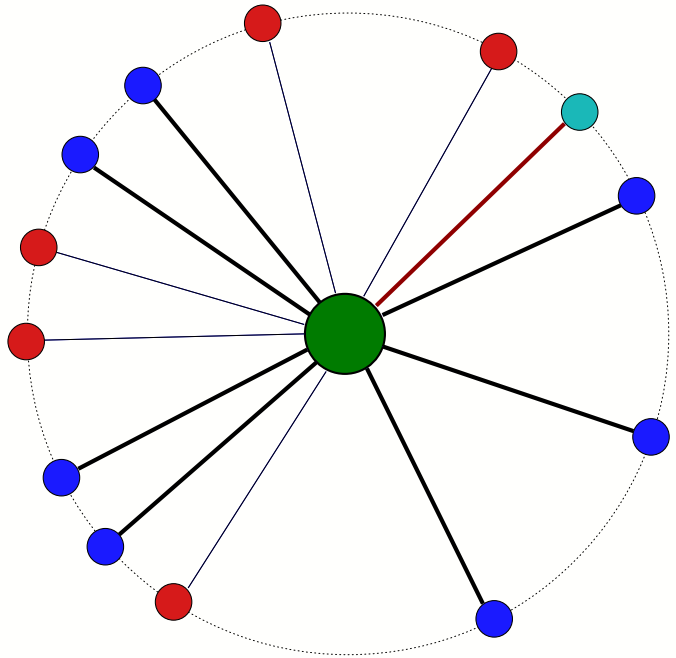
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



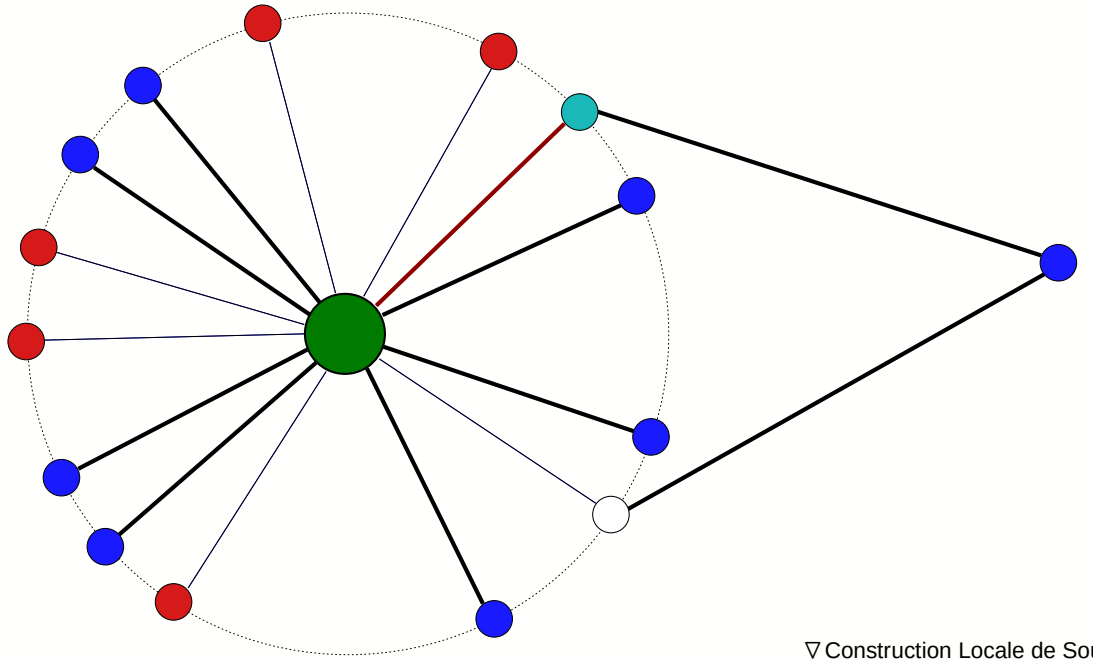
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



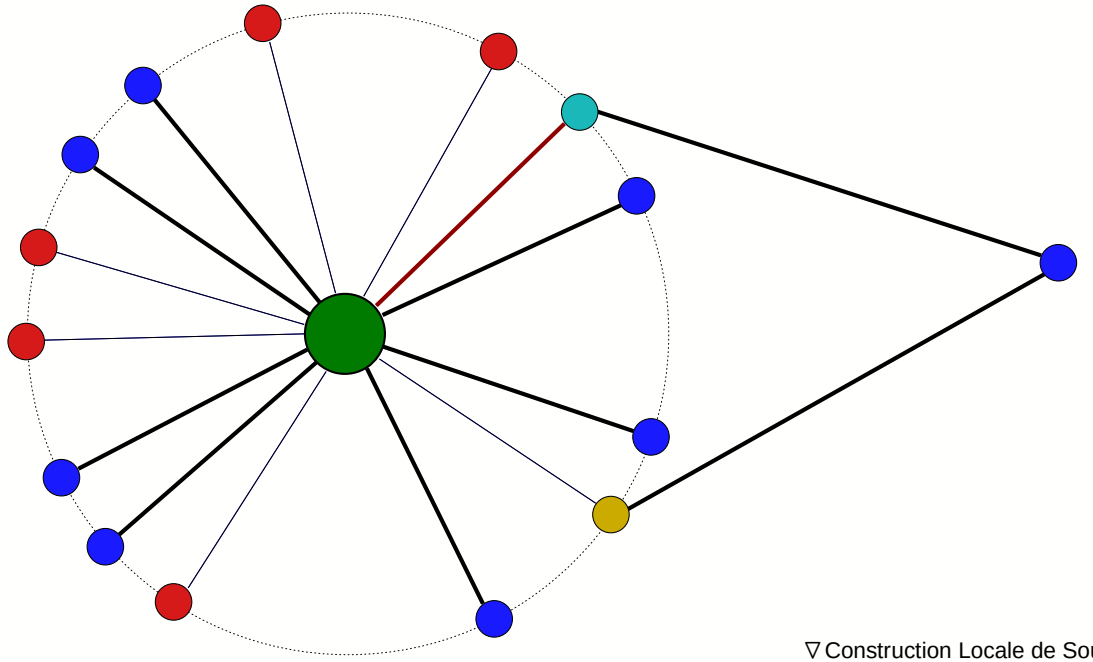
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



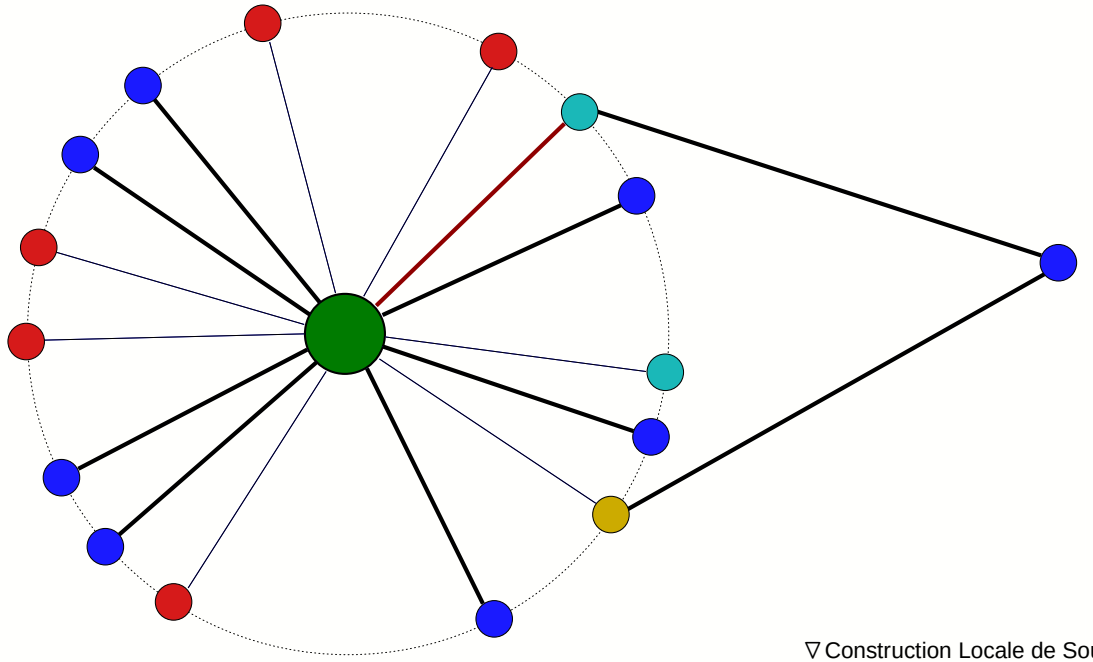
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



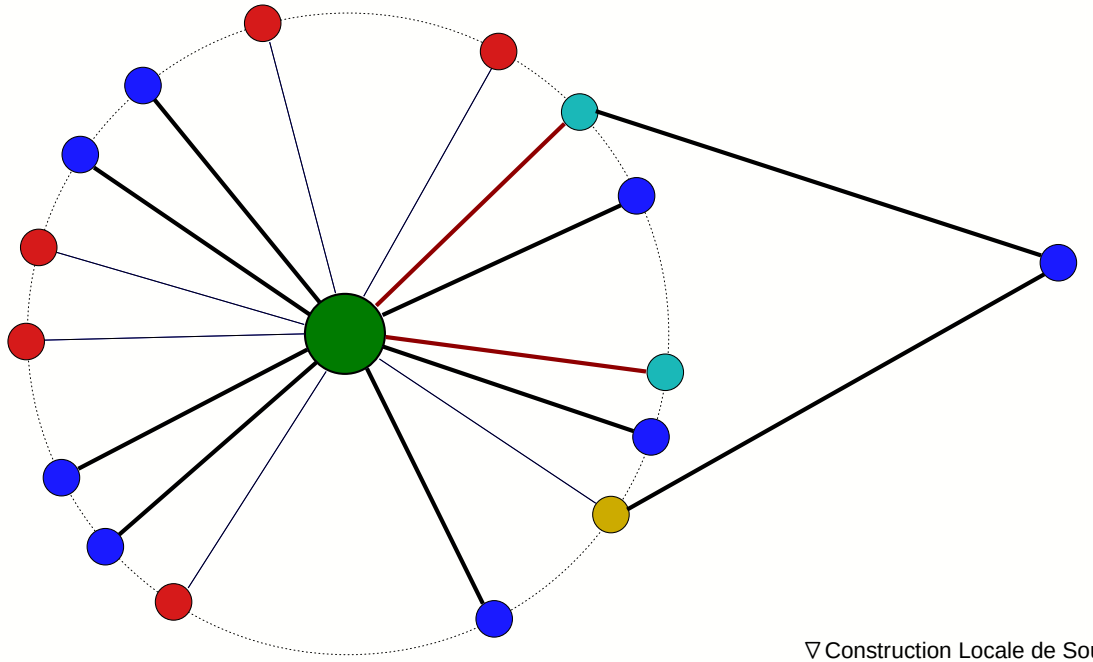
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



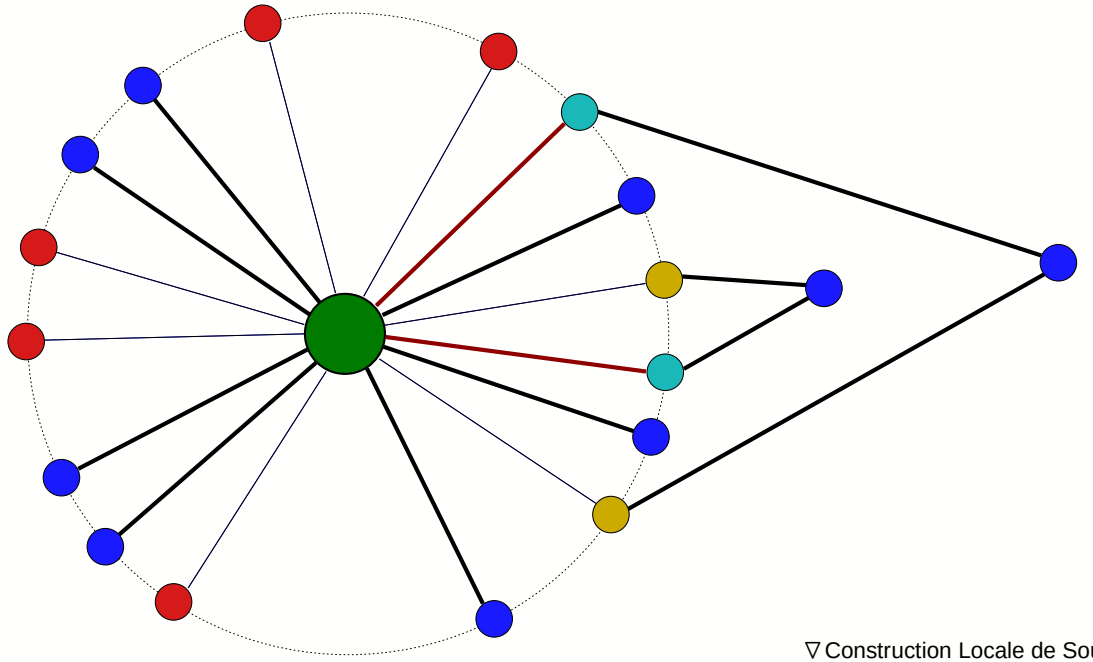
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



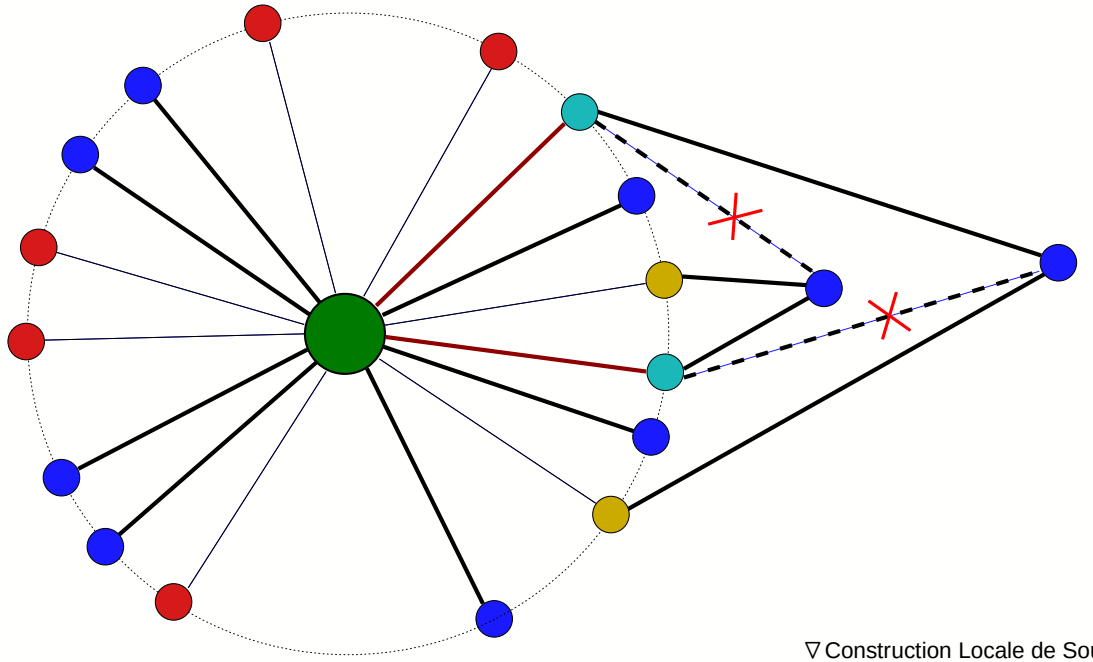
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



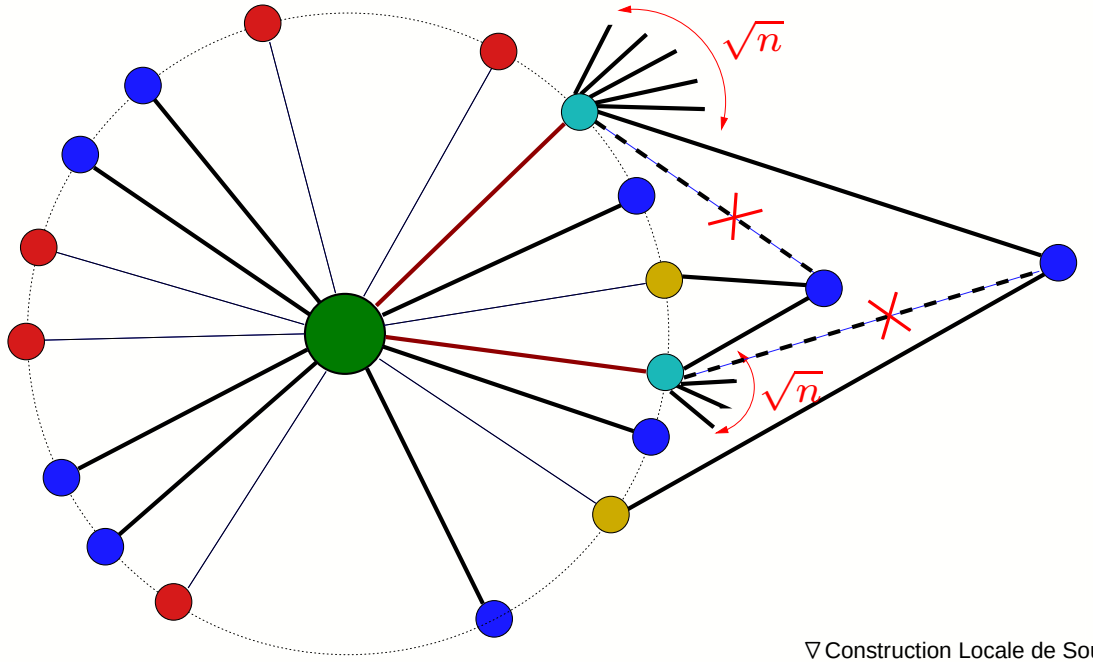
Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner



Algorithme pour $k = 2$

- Initialement, pour tout sommet u, on note W_u l'ensemble des voisins de u.
- Chaque sommet u choisit $\sqrt{n}$ voisins dans l'ensemble W_u et se connecte à eux dans le spanner.
  - On note $R(u)$ l'ensemble de ces voisins plus u.
- Pour tout sommet u, $W_u := W_u \setminus \{w \in W_u \mid R(u) \cap R(w) \neq \emptyset\}$
- Pour tout sommet u, tant que $\exists w \in W_u$ faire :
  - $W_u := W_u \setminus \{v \in W_u \mid R(v) \cap R(w) \neq \emptyset\}$
  - l'arête (u, w) est rajoutée au spanner

Théorème [k=2]

Il existe un algorithme distribué qui calcule pour tout graphe un $(3, 0)$-spanner avec $O(n^{3/2})$ arêtes en 2 unités de temps

Algorithme général ($k > 1$)

$L := C := R(u) := \{u\}$, $W := B(u, 1)$, $\sigma := |B(u, 2k-1)|^{1/k}$

pour $i := 1$ ***à*** k **faire**

- Le sommet u envoie $R(u)$ à ses voisins, et il reçoit $R(w)$ de ses voisins w
- **tant que** $\exists w \in W$ *tel que* $R(u) \cap R(w) \neq \emptyset$ **faire**
  - $W := W \setminus \{w\}$
- **tant que** $\exists w \in W$ *et* $|L| < i\sigma$ **faire**
  - $W := W \setminus \{v \in W \mid R(v) \cap R(w) \neq \emptyset\}$
  - $L := L \cup \{w\}$
  - $C := C \cup R(w)$
- $R(u) := C$

Algorithme général ($k > 1$)

$L := C := R(u) := \{u\}$, $W := B(u,1)$, $\sigma := |B(u,2k-1)|^{1/k}$

pour $i := 1$ ***à*** k **faire**

Le sommet u envoie $R(u)$ à ses voisins, et il reçoit $R(w)$ de ses voisins w

tant que $\exists w \in W$ *tel que* $R(u) \cap R(w) \neq \emptyset$ **faire**

$W := W \setminus \{w\}$

tant que $\exists w \in W$ *et* $|L| < i\sigma$ **faire**

$W := W \setminus \{v \in W \mid R(v) \cap R(w) \neq \emptyset\}$

$L := L \cup \{w\}$

$C := C \cup R(w)$

$R(u) := C$

Théorème

Il existe un algorithme distribué qui calcule pour tout graphe un $(2k-1, 0)$-spanner avec $k \cdot n^{1+1/k}$ arêtes en k unités de temps

Spanners Additifs

	(α, β)	taille	temps	temps moyen
[EP'04]	$(1, 2)$	$n^{3/2}$	$?$	$?$
[BKMP'05]	$(1, 6)$	$n^{4/3}$	$?$	$?$
[EP'04,EZ'04]*	$(1+\epsilon, \beta)$	$n^{1+\delta}$	$O(n^{1+\delta})$	$-$
[DISC'07]	$(1+\epsilon, 4)$	$n^{3/2}$	$O(\epsilon^{-1}) + o(n^{\epsilon'})$	$O(\log n + \epsilon^{-1})$
[DISC'07]	$(1+\epsilon, 8\log n)$	$n^{3/2}$	$O(\epsilon^{-1} \log n)$	$-$

$* : \beta = \beta(\delta^{-1}, \epsilon^{-1}, n)$

Spanners Additifs

	(α, β)	taille	temps	temps moyen
[EP'04]	$(1, 2)$	$n^{3/2}$	?	?
[BKMP'05]	$(1, 6)$	$n^{4/3}$	?	?
[EP'04,EZ'04]*	$(1+\epsilon, \beta)$	$n^{1+\delta}$	$O(n^{1+\delta})$	–
[DGP'07]	$(1+\epsilon, 4)$	$n^{3/2}$	$O(\epsilon^{-1}) + o(n^{\epsilon'})$	$O(\log n + \epsilon^{-1})$
[DGP'07]	$(1+\epsilon, 8\log n)$	$n^{3/2}$	$O(\epsilon^{-1} \log n)$	–
Nouveau	$(1+\epsilon, 2)$	$O(\epsilon^{-2} n^{3/2})$	$O(\epsilon^{-1})$	–

$* : \beta = \beta(\delta^{-1}, \epsilon^{-1}, n)$

Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Analyse : Complexité

$$\text{Temps} = O(r)$$

Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Analyse : La taille

- Pour un t fixé, il existe au plus $\sqrt{n}$ sommets w.
  $\Longrightarrow$ On ajoute au plus $\sqrt{n}$ chemins de longueur t
  $\Longrightarrow$ Au plus $O(r^2\sqrt{n})$ arêtes par sommet

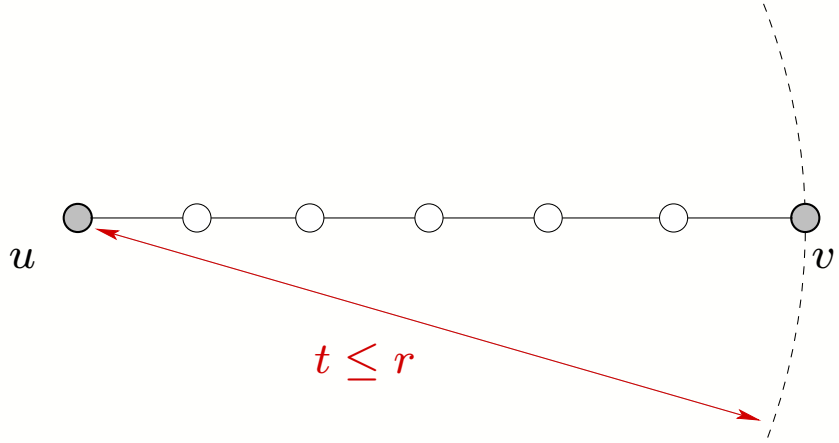
$$\text{Taille} = O(r^2 \cdot n^{3/2})$$

Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Analyse : Stretch

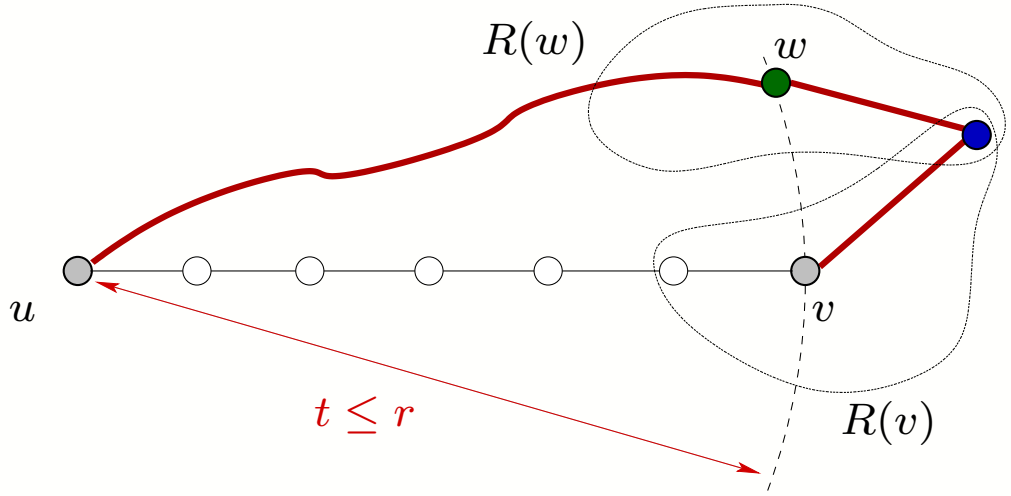


Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Analyse : Stretch



Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

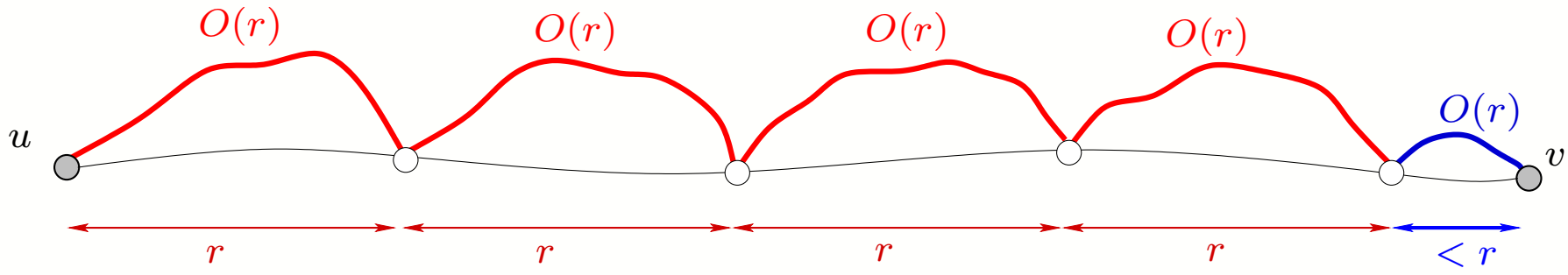
Analyse : Stretch

Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Analyse : Stretch

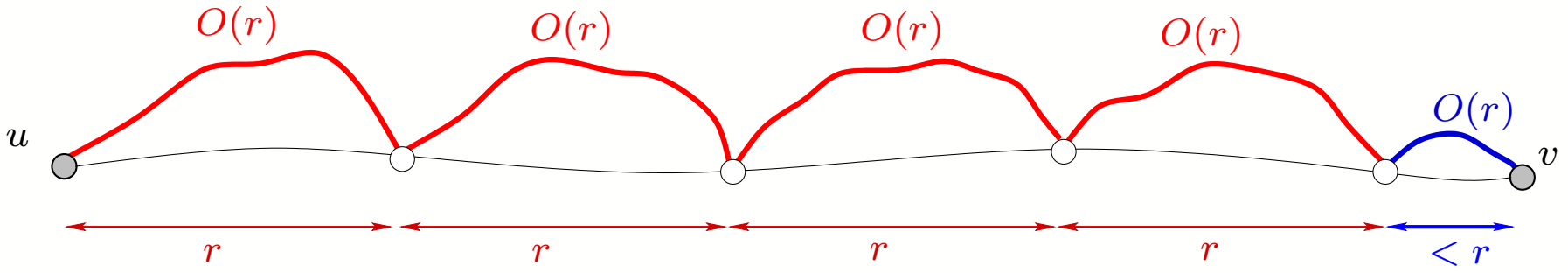


Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Analyse : Stretch



$$\text{Stretch} = \left(1 + O(1/r) \ , \ 2 \right)$$

Algorithme : Idées générales

Pour tout sommet u :

- u choisit $\sqrt{n}$ voisins et forme sa région $R(u)$
- Soit w un voisin (fort) à distance t ($t = 1, ..., \mathbf{r}$)
- u ajoute un plus court chemin entre u et w
- Les sommets v telque $R(v) \cap R(w) \neq \emptyset$ ne sont plus considérés

Pour $r = O(1/\epsilon)$, on obtient :

Stretch	Taille	Temps
$(1+\epsilon, 2)$	$O(\epsilon^{-2} n^{3/2})$	$O(\epsilon^{-1})$

Borne Inférieure : spanners additifs

Un algorithme distribué calculant un spanner de taille au plus $O(n^{1+1/k+\epsilon})$ en n^{ϵ} temps doit avoir un facteur d'étirement (purement) additif au moins $n^{\Omega(\epsilon)}$

Borne Inférieure : spanners additifs

Théorème

Soit k, tel que $1 < k \leq \log n$, et $1/\epsilon > 3k^2 - k$. Soit un algorithme distribué A qui calcule pour tout graphe à n sommets un spanner de taille plus petite que $\lambda \cdot n^{1+1/k}$ en temps t, pour $k - 3 \leq t \leq n^{\epsilon}$ et $\lambda \leq n^{\epsilon}$. Alors, il existe un graphe G et deux sommets u et v à distance $d_G(u,v) = \Omega(tn^{\epsilon})$ tel que dans le spanner H calculé par l'algorithme A, on a :

$$d_S(u,v) > \left(1 + \frac{k-1}{t+1}\right) d_G(u,v) + 2k - 3$$

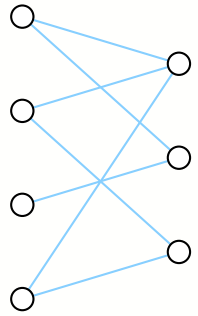
Borne Inférieure : spanners additifs

Théorème

Soit k, tel que $1 < k \leq \log n$, et $1/\epsilon > 3k^2 - k$. Soit un algorithme distribué A qui calcule pour tout graphe à n sommets un spanner de taille plus petite que $\lambda \cdot n^{1+1/k}$ en temps t, pour $k-3 \leq t \leq n^\epsilon$ et $\lambda \leq n^\epsilon$. Alors, il existe un graphe G et deux sommets u et v à distance $d_G(u,v) = \Omega(tn^\epsilon)$ tel que dans le spanner H calculé par l'algorithme A, on a :

$$d_S(u,v) > \left(1 + \frac{k-1}{t+1}\right) d_G(u,v) + 2k - 3$$

B



$$|V(B)| = p,$$

$$|E(B)| = p^{1+1/\Theta(k)},$$

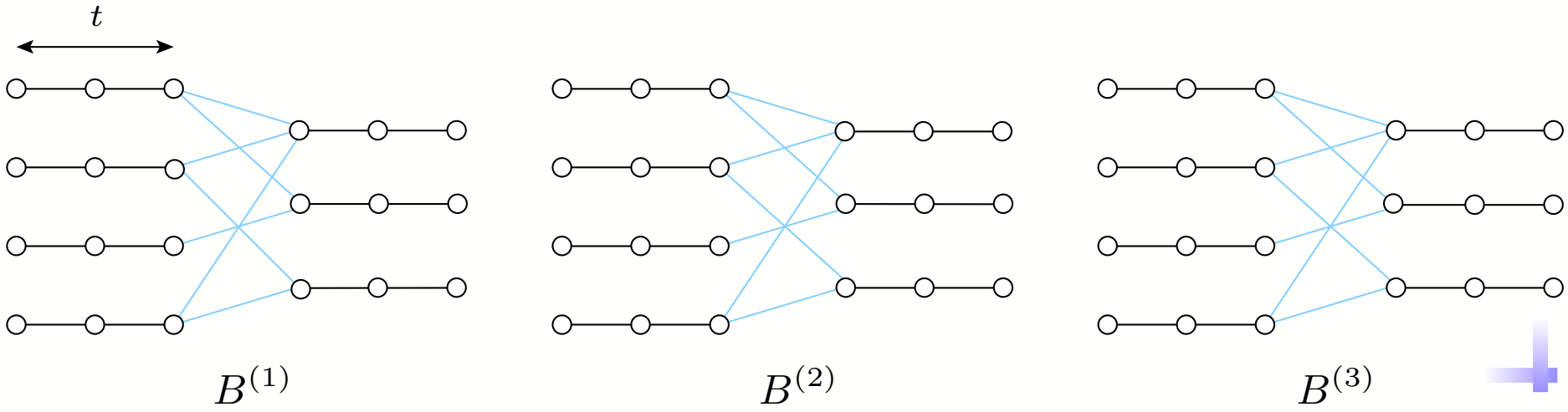
$$girth(B) = \Theta(k)$$

Borne Inférieure : spanners additifs

Théorème

Soit k, tel que $1 < k \leq \log n$, et $1/\epsilon > 3k^2 - k$. Soit un algorithme distribué A qui calcule pour tout graphe à n sommets un spanner de taille plus petite que $\lambda \cdot n^{1+1/k}$ en temps t, pour $k - 3 \leq t \leq n^\epsilon$ et $\lambda \leq n^\epsilon$. Alors, il existe un graphe G et deux sommets u et v à distance $d_G(u,v) = \Omega(tn^\epsilon)$ tel que dans le spanner H calculé par l'algorithme A, on a :

$$d_S(u,v) > \left(1 + \frac{k-1}{t+1}\right) d_G(u,v) + 2k - 3$$

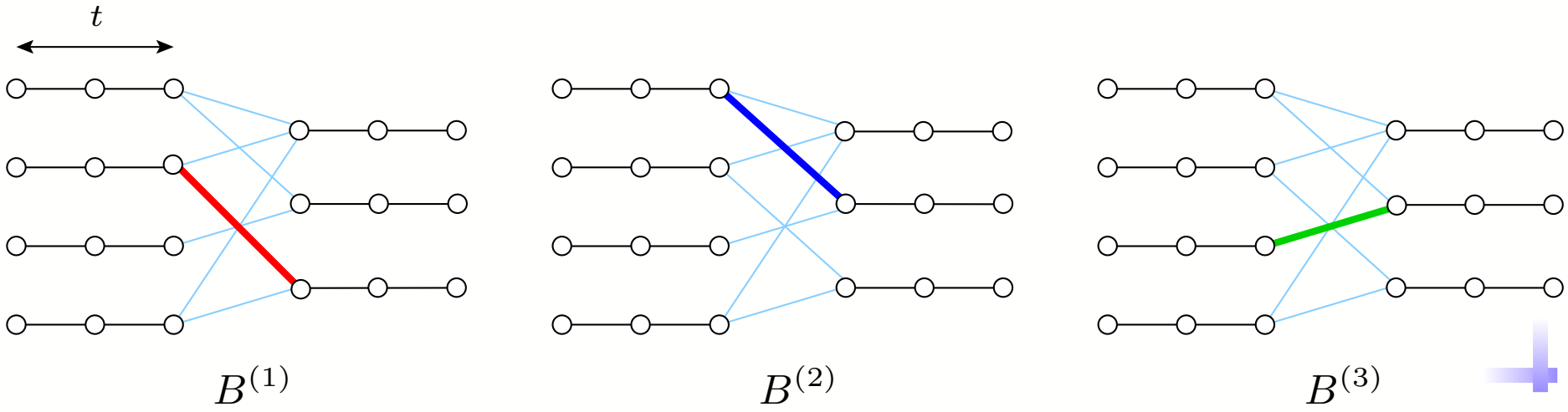


Borne Inférieure : spanners additifs

Théorème

Soit k, tel que $1 < k \leq \log n$, et $1/\epsilon > 3k^2 - k$. Soit un algorithme distribué A qui calcule pour tout graphe à n sommets un spanner de taille plus petite que $\lambda \cdot n^{1+1/k}$ en temps t, pour $k - 3 \leq t \leq n^\epsilon$ et $\lambda \leq n^\epsilon$. Alors, il existe un graphe G et deux sommets u et v à distance $d_G(u,v) = \Omega(tn^\epsilon)$ tel que dans le spanner H calculé par l'algorithme A, on a :

$$d_S(u,v) > \left(1 + \frac{k-1}{t+1}\right) d_G(u,v) + 2k - 3$$

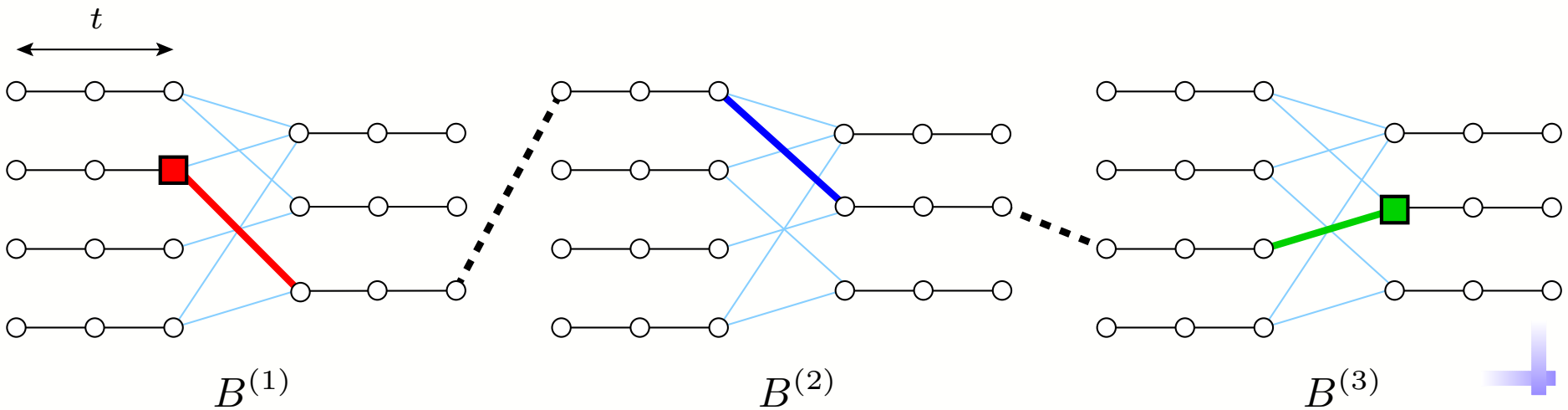


Borne Inférieure : spanners additifs

Théorème

Soit k, tel que $1 < k \leq \log n$, et $1/\epsilon > 3k^2 - k$. Soit un algorithme distribué A qui calcule pour tout graphe à n sommets un spanner de taille plus petite que $\lambda \cdot n^{1+1/k}$ en temps t, pour $k - 3 \leq t \leq n^\epsilon$ et $\lambda \leq n^\epsilon$. Alors, il existe un graphe G et deux sommets u et v à distance $d_G(u,v) = \Omega(tn^\epsilon)$ tel que dans le spanner H calculé par l'algorithme A, on a :

$$d_S(u,v) > \left(1 + \frac{k-1}{t+1}\right) d_G(u,v) + 2k - 3$$



Conclusion

Des algorithmes locaux pour construire des spanners optimaux

Problèmes ouverts :

- Utiliser des messages de taille $O(\log n)$ ($\mathcal{CONGEST}$ model).
- Trouver des $(1, f(k))$-spanners avec $O(n^{1+1/k})$ arêtes ($k > 3$) même en séquentiel.