Algorithmique Distribuée

Réseaux Asynchrones

Shantanu Das

http://pageperso.lif.univ-mrs.fr/~shantanu.das/M1algodist/

Aix-Marseille Université

6/7 Mars 2018

Dernier Cours :

- Modèles Différents de Calcul Distribué
- Complexité de la communication
- Problèmes fondamentaux : Election/Consensus
- Problème de l'Election dans l'anneau (Impossiblité, Algorithme LCR, Algorithme Phases)
- Algorithme d'innondation
- Consensus (avec défaillances), impossiblité

Cette Semaine :

Réseaux Asynchrones

- Calcul dans les arbres (*Saturation*)
- Construction d'arbre couvrant
- Election : Borne inférieure
- Optimal Algorithme pour l'élection (GHS)
- Autres algorithmes (KKM, YoYo)
- Election dans les topologies spécifiques
- *Synchronizers*

Réseaux Asynchrones

Suppositions

- Chaque message qui a été envoyé est finalement reçu à la destination.
- Il n'y a pas de limites sur les délais de transmission..

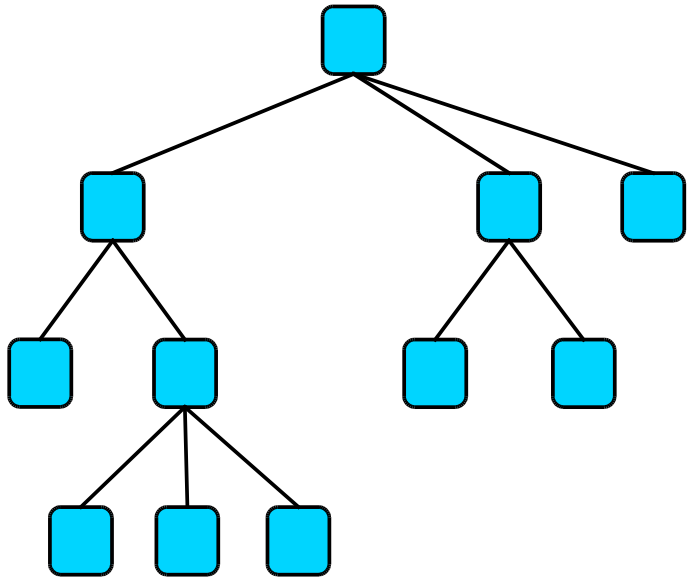
Réseaux Asynchrones

Suppositions

- Chaque message qui a été envoyé est finalement reçu à la destination.
- Il n'y a pas de limites sur les délais de transmission..

- Par conséquence ce n'est pas possible de détecter des fautes. (On suppose il n'y a pas de fautes !)
- Le but : Trouver les meilleurs algorithmes (Complexité en termes de nombre de messages transmis.)

Calcul dans les arbres



Calcul dans les arbres

C'est plus facile de faire les calculs distribué dans un arbre que dans un réseau arbitraire.

Technique de *Saturation* ("Convergecast")

- Feuille : Envoyer un message au seul voisin.
- Autres : Attendre de recevoir les messages de tous les voisins, sauf une et ensuite envoyer à lui.
- Il existe un processes qui reçois des messages de tous les voisins. ["Saturé"]

Calcul dans les arbres

C'est plus facile de faire les calculs distribué dans un arbre que dans un réseau arbitraire.

Technique de *Saturation* ("Convergecast")

- Feuille : Envoyer un message au seul voisin.
- Autres : Attendre de recevoir les messages de tous les voisins, sauf une et ensuite envoyer à lui.
- Il existe un processes qui reçois des messages de tous les voisins. ["Saturé"]

- Il y a exactement un ou deux processus qui va être *saturé* (recevoir les messages de tout son voisins).
- S'il y a deux processus saturé, ils sont voisins !

Applications de *Saturation*

(1) ELECTION dans une arbre :

- Effectuer Saturation
- Si il y a un seul processus qui est Saturé, il est élu.
- Sinon, les deux processus saturé communiquent entre eux, et celui qui a la plus grande UID est élu.

Complexité $= n + 2$ messages.

Applications de *Saturation*

(2) Le Problème de GOSSIP dans une arbre :

- Chaque processus effectuer Saturation avec leurs donnée (et les info reçus).
- Les processus qui sont Saturés, recevoir tout les informations.
- Les processus saturés effectuer un *Broadcast*

Complexité $= 2n - 1$ messages.

Applications de Saturation

(3) Trouver le centre d'une arbre :

(Centre= Le sommet qui minimise la distance maximal à tous)

- Effectuer Saturation avec le valeur de hauteur (distance à partir de feuilles).
- Les feuilles envoient "0" ; Les autres envoient 1+ (le max reçu) ;
- Les processus qui sont Saturés, effectuer un résolution.
  - Si ($Max - 2emeMax \leq 1$) devenir Centre
  - Sinon envois $(1 + 2emeMax)$ à celui de qui Max a été reçu.

Complexité $= 1,5 \times n$ messages.

Calculs dans les graphes arbitraires

Dans un réseau arbitraire ...

Comment construire un Arbre Couvrant (Spanning Tree) ?

Rappel :

On peut utiliser un arbre couvrant pour réduire le quantité de communication dans le réseau.

Calculs dans les graphes arbitraires

Construction d'un Arbre Couvrant [SPT]

Calculs dans les graphes arbitraires

Construction d'un Arbre Couvrant [SPT]

- Effectuer un algorithme d'élection
- Le Elu initie un Diffusion (Broadcast)
- Si un processus reçoit des messages de plusieurs processus, il nomme l'un d'eux comme son parent.
- Chaque processus envoie des messages à {Voisinage - envoyeurs}.

(Cette algorithme crée un arbre couvrant enraciné.)

Calculabilité

- Est-il toujours possible de construire un arbre couvrant (SPT) ?

Calculabilité

- Est-il toujours possible de construire un arbre couvrant (SPT) ?
- On sait : ELECTION => SPT
- Rappel : Election n'est pas possible dans un anneau anonyme !

Théorème

La construction d'un arbre couvrant n'est pas possible dans les graphes arbitraires anonymes.

Attention : Il existe des graphes où SPT est résoluble mais pas ELECTION.

ELECTION
dans les
graphes arbitraires

ELECTION dans les graphes arbitraires

Supposition :
Chaque processus a un identifier unique (UID)

- Avec cette supposition, Il est toujours possible de effectuer l'ELECTION.
- On est intéressé à la complexité du problème d'ELECTION.

Complexité d'Election

Y at-il une borne inférieure sur le nombre de messages doit être transmis à résoudre le problème d'ELECTION ?

Borne inférieure pour ELECTION

Théorème

Election dans les graphes arbitraires nécessite $\Omega(m + n \log n)$ *messages dans le pire des cas.*

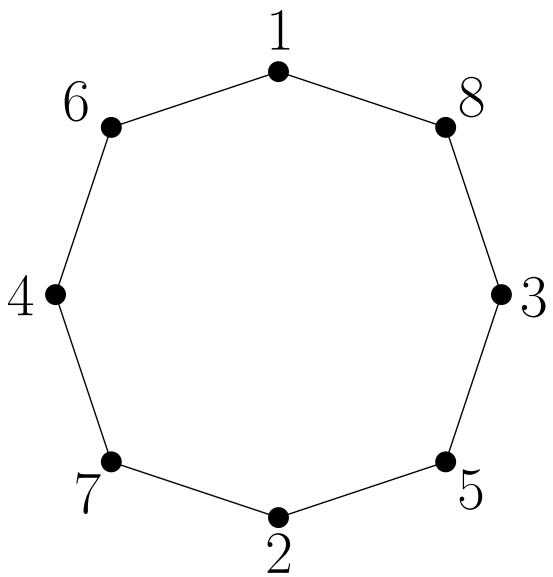
$m = \#$arêtes

$n = \#$sommets

Borne inférieure pour ELECTION

Lemme ([Burns 1980, Bodlaender 1987])

Election dans un anneau nécessite le transmission de $\Omega(n \log n)$ *messages.*



Borne inférieure

Lemme

Election dans un graphe arbitraire nécessite le transmission de $\Omega(m)$ *messages.*

Borne inférieure

Lemme

Election dans un graphe arbitraire nécessite le transmission de $\Omega(m)$ messages.

- Il faut envoyer au moins un message sur chaque arête !
- Sinon, supposons il existe un algorithme A qui effectue ELECTION dans un graphe G sans envoyer des messages sur un arête $e = (u, v)$.
- On prends le graphe $\{G - e\}$ et une deuxième copie de la meme. On ajoute des arêtes entre noeud u dans le premiere graphe et noeud v dans l'autre copie. On appel cette graphe $G2$.
- Si nous exécutons l'algorithme A sur $G2$, deux noeuds seront élus $=>$ Algorithme A n'est pas correct ! Par contradiction, le lemme est vrai

Optimal Algorithme

Théorème

Election dans les graphes arbitraires nécessite $\Omega(m + n \log n)$ *messages dans le pire des cas.*

Y a-t-il un algorithme qui utilise nombre optimal de messages ?

Optimal Algorithme

Théorème

Election dans les graphes arbitraires nécessite $\Omega(m + n \log n)$ *messages dans le pire des cas.*

Y a-t-il un algorithme qui utilise nombre optimal de messages ?

GHS algorithme [Gallager, Humblet, Spira 1983]

Construction d'un arbre couvrant minimal [MST] dans les graphes arbitraires, avec la complexité de $O(m + n \log n)$ messages.

L'algorithme de MST peut être utilisé aussi pour effectuer l'ELECTION.

Construction d'un arbre couvrant minimal

Définition (MST)

Etant doné un graphe G où chaque arête a un poids, l'arbre couvrant minimal T est un arbre couvrant de G avec le poids total minimum.

Si chaque arête de G a un poids unique, il existe un unique MST.

Construction d'un arbre couvrant minimal

Définition (MST)

Etant doné un graphe G où chaque arête a un poids, l'arbre couvrant minimal T est un arbre couvrant de G avec le poids total minimum.

Si chaque arête de G a un poids unique, il existe un unique MST.

Algorithme distribué pour MST :

- On commence par V(G) ; C'est une forêt où chaque sommet est un arbre.
- Fusionner 2 arbres avec l'arête de poids minimum entre eux. (Encore une forêt, #arbres diminue)
- Continuer jusqu'à ce qu'il n'y a qu'un seul arbre.

Algorithme GHS

- Chaque arbre possède un nom (UID de la racine) et un niveau (initialement 1).
- Chaque arbre choisit l'arête minimum sortant. Comment ?
- Il envoie un message sur l'arête choisi $e = (x, y)$, avec son nom et niveau.
  [Fusion] : Si Niveau($T(x)$) = Niveau($T(y)$) et les deux choisit (x, y), les deux arbres sont fusionnés ;
  Nom du nouvel arbre = MAX($UID(x)$, $UID(y)$).
  Niveau = niveau(T(x))+1 ;
  [Absorption] : Si Niveau($T(x)$) <Niveau($T(y)$),
  l'arbre T(x) est absorbé dans T(y). On garde le nom et le niveau de $T(y)$.
- Dans tous les autres cas, l'envoyeur d'un message doit attendre.

Correction d' Algorithme GHS

- Un processus peut attends seulement pour un autre qui a la même niveau.
- Un de processus qui a la même niveau va augmenter sa niveau.
- Il n' y a pas d'attendre en cycle => Pas de blocages !
- Il y a toujours un forêt et le nombre d'arbres diminue de façon monotone.

Complexité d' Algorithme GHS

- Chaque arbre à niveau i a au moins 2^i processus.
- Niveau$(T) \leq \log n$, pour chaque arbre T.
- Tout processus à niveau i ensemble envoient $O(n)$ messages.
- Il y a au plus $2(m - n + 1)$ messages sur les arêtes qui n'est pas partie du MST.

Complexité $= O(m + n \log n)$.

Comment effectuer ELECTION ?

- La racine du MST est Elu.
- Il faut donner des poids uniques aux arêtes de G. Chaque arête e = (x, y) est attribuée le poid (a, b) où $a = MAX(UID(x), UID(y))$ et $b = MIN(UID(x), UID(y))$.
- Comment comparer les poids ?
  $(a1, a2) < (b1, b2)$ ssi $a1 < a2$ ou, $a1 = b1$ et $a2 < b2$.
- Pour savoir le valeurs de poids, chaque processus doit communiquer avec ses voisins. On ajoute $2m$ à la complexité.

Le algorithme GHS est optimal pour ELECTION, en termes de nombre de messages.

Autres Algorithmes

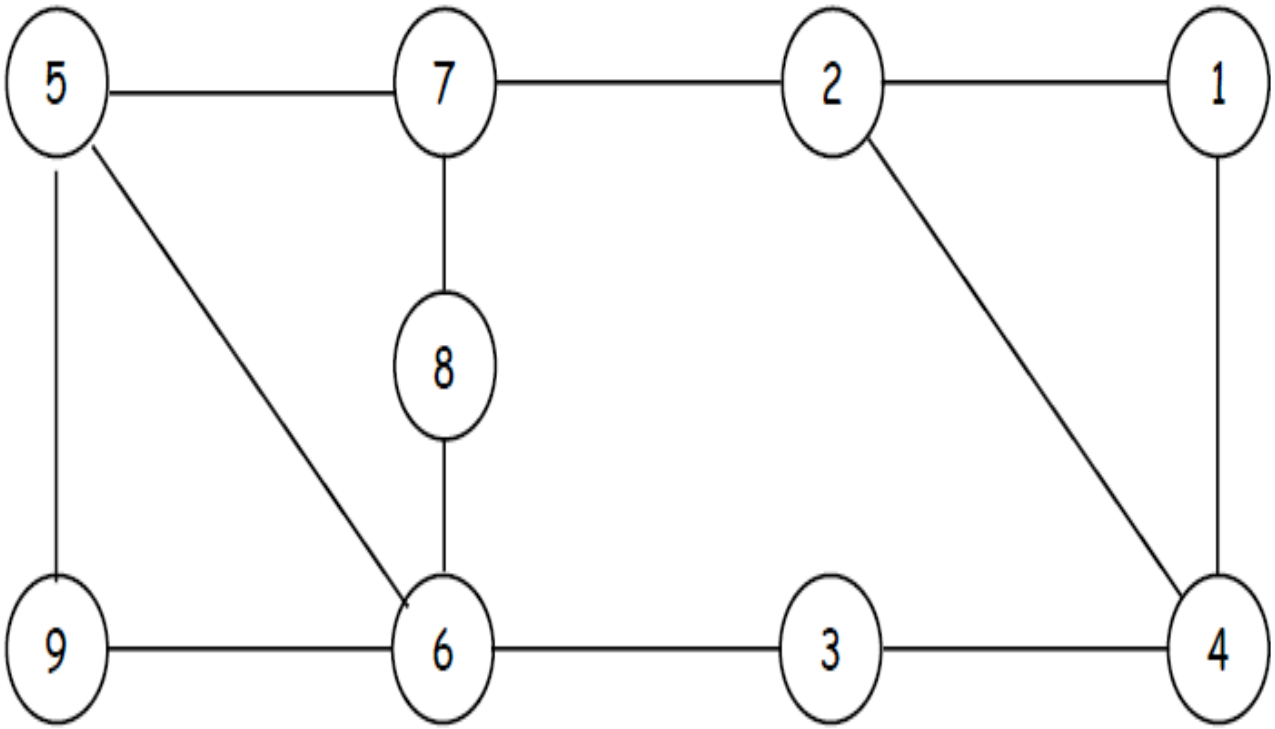
Algorithme YO-YO [Santoro 2007]

- Donner un orientation à chaque arête (min $->$ max)
- Un ou plusieurs noeuds deviennent des puits.
- Un ou plusieurs noeuds deviennent des sources.
- Le reste sont des noeuds intermédiaires.

Le graphe est devenu un DAG (Acyclique Graphe Dirigé)

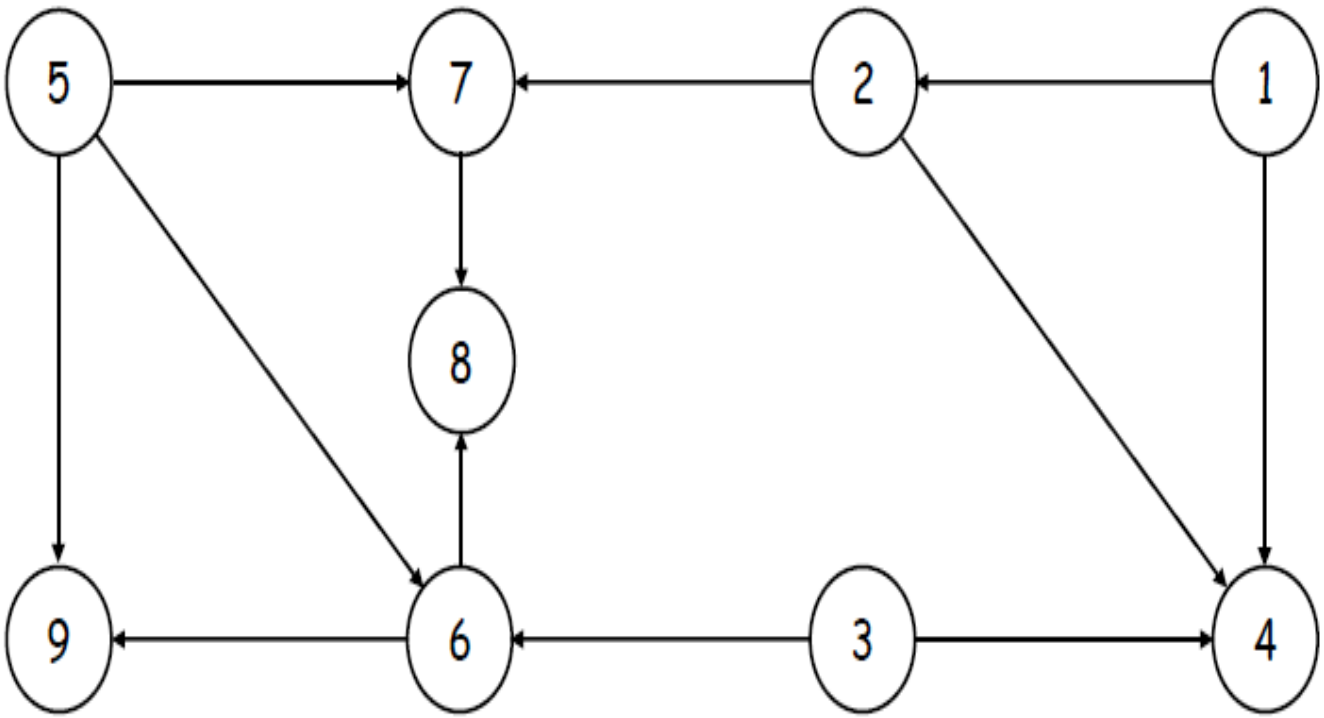
Algorithme YO-YO

Un graphe arbitraire (bidirectionnel)



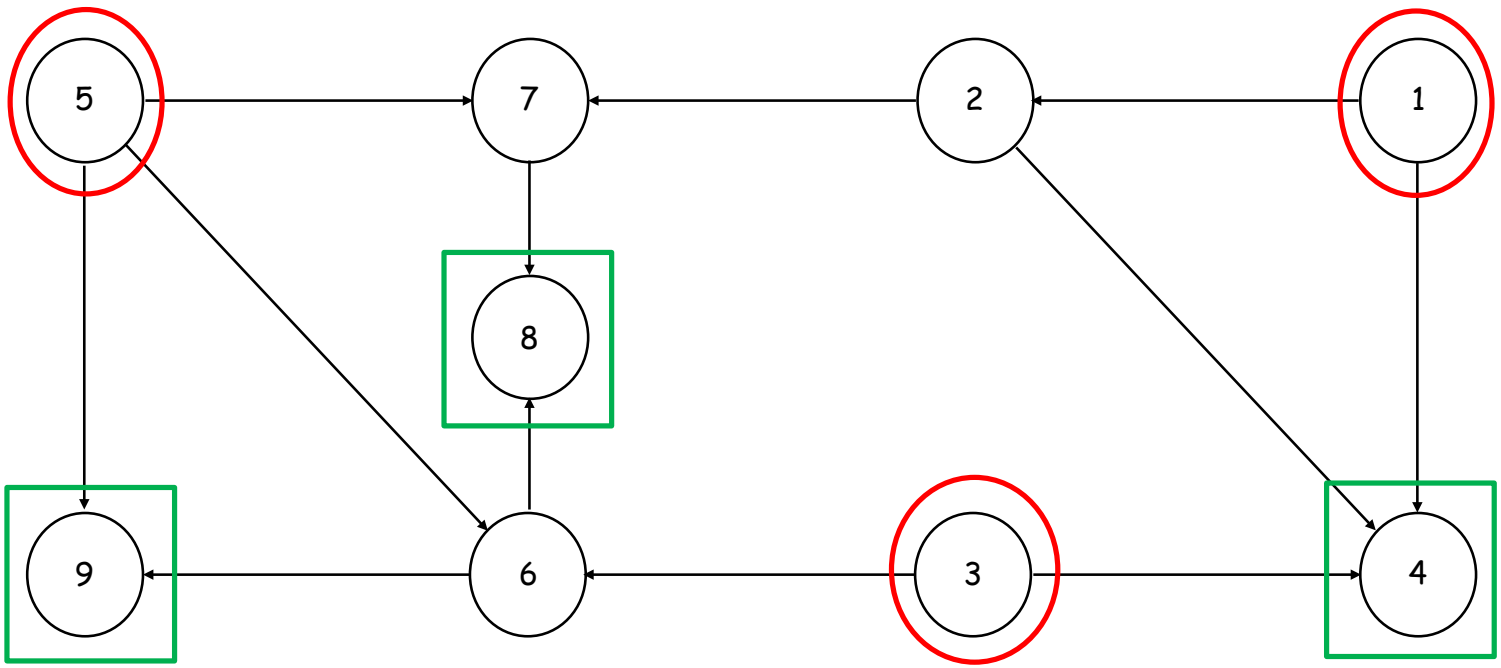
Algorithme YO-YO

Donner un orientation sur chaque arête !



Algorithme YO-YO

Les Sources et les Puits

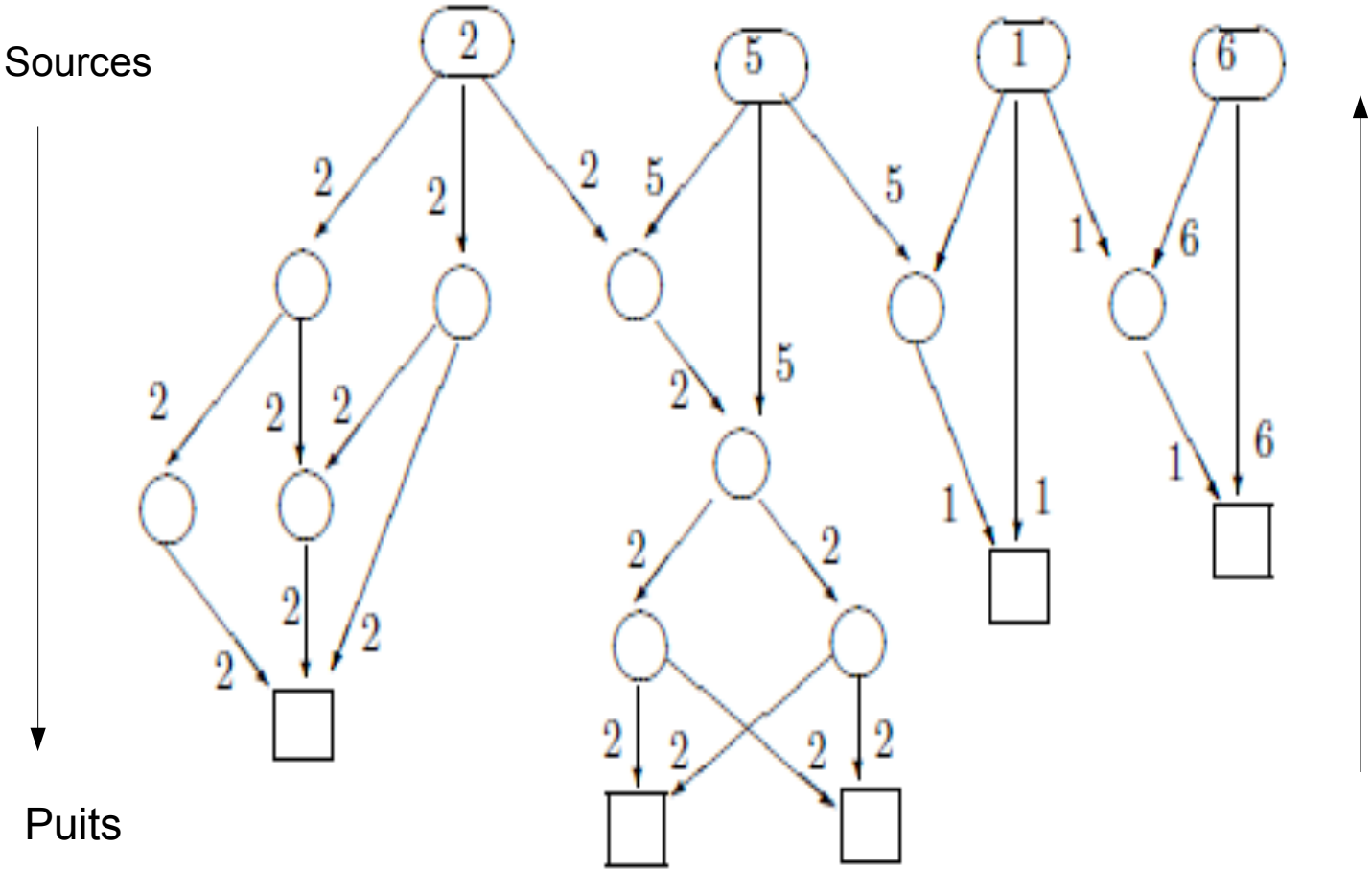


Algorithme YO-YO

- Les sources sont les candidates ; ils envoient leurs UID.
- Les intermédiaires envoient le minimum de valeurs reçus.
- Les puits dites OUI si le valeur reçu est le plus petit (et dites NON sinon).
- Les sources qui ont reçu OUI de tous, sont candidates pour la prochaine étape.
- On change l'orientation des arêtes sur laquelle NON a été envoyé.

Le graphe est toujours un DAG mais le nombre de sources diminue à chaque étape.

Algorithme YO-YO



Algorithme YO-YO

- Le nombre de sources diminue à chaque étape.
- Finalement il y un seule source => le Elu
- Comment détecter le terminaison ?

Algorithme YO-YO

- Le nombre de sources diminue à chaque étape.
- Finalement il y un seule source => le Elu
- Comment détecter le terminaison ?
- Tailler la partie du graphe qui n'est pas utile (à chaque étape)
- A la fin, il reste un sommet, le Elu.

Algorithme YO-YO

- Le nombre de sources diminue à chaque étape.
- Finalement il y un seule source => le Elu
- Comment détecter le terminaison ?
- Tailler la partie du graphe qui n'est pas utile (à chaque étape)
- A la fin, il reste un sommet, le Elu.

Complexité de l'algorithme : $O(m \log n)$ messages
(il y au plus $\log n$ étapes de l'algorithme)

Famille de Graphes Specifiques

Doit-on toujours besoin de $O(m + n \log n)$ messages pour l'ELECTION ?

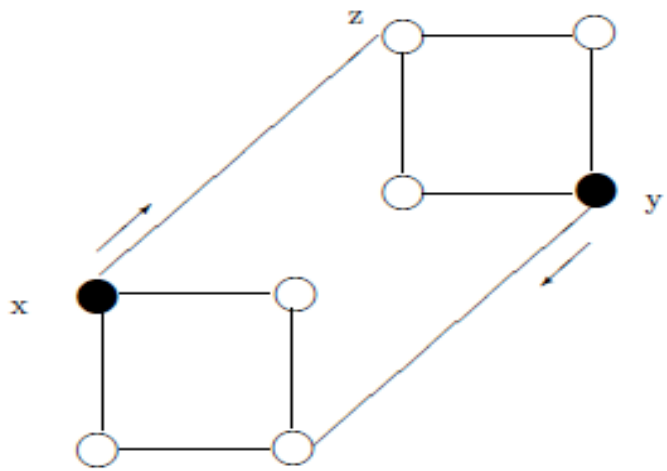
Famille de Graphes Specifiques

Doit-on toujours besoin de $O(m + n \log n)$ messages pour l'ELECTION ?

Pour les famille de graphes spécifiques, c'est possible de trouver plus efficace algorithmes pour ELECTION.
Quelques Exemples :

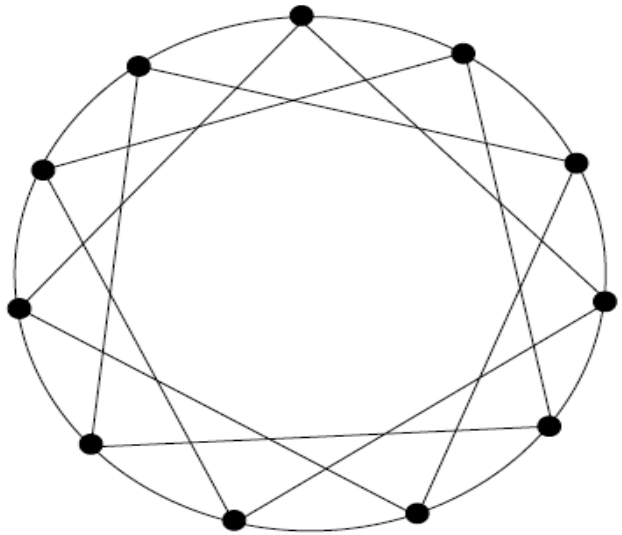
- La Famille d'Arbres
- Les Hypercubes orientés.
- Les graphes cordales.

Election dans les Hypercubes Orientés



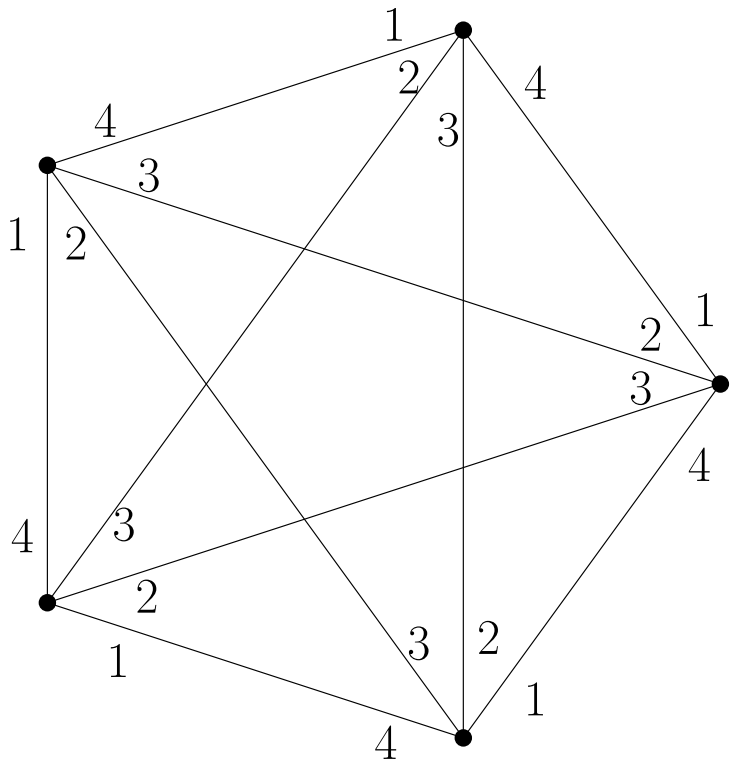
- A étape i, faire un compétition sur le dimension i.
- Complexité : $O(n)$ messages.
  - $n/2^{i-1}$ candidats en étape i ; chaque candidat envois 1 message sur arête 'i'

Election dans les Anneaux Cordales



- **Anneau Cordal :** Il y a des cordes entre les sommets à distance $\leq t$, sur l'anneau.
- Algorithme de *Phases*, avec une modification.
- Utiliser les chordes pour éviter les sommets inactifs.

Election dans les Anneaux Cordales



- Complexité : $O(n + n/t \log n/t)$ messages.
- Pour un graphe complet avec étiquetage cordal : $(t = n/2) => O(n)$ messages.

Election Efficace

Algorithmes d'Election pour

- Anneaux : $O(n \log n)$ messages
- Toutes Arbres : $O(n)$ messages
- Les Hypercubes orientés : $O(n)$ messages
- Les graphes cordales : $O(n + n/t \log n/t)$ messsages

Election Efficace

Algorithmes d'Election pour

- Anneaux : $O(n \log n)$ messages
- Toutes Arbres : $O(n)$ messages
- Les Hypercubes orientés : $O(n)$ messages
- Les graphes cordales : $O(n + n/t \log n/t)$ messsages

Quels propriétés des graphes permet un algorithme efficace pour l'Election ?

Technique Modulaire pour ELECTION

Technique Modulaire pour l'ELECTION

- ELECTION dans les graphes arbitraires.
- Utiliser un exploration efficace pour effectuer un ELECTION efficace.
- **Algorithme KKM**
  [*Korach, Kutten, et Moran 1990*]
- Complexité : $(n + f(n))(\log n + 1)$

$f(n)$: Complexité de l'exploration (visiter tous les sommets)

KKM Algorithme

- Chaque initiateur envoie un jeton avec son UID (et niveau= 1).
- Le jeton fais un trajet du graphe.
- Si deux jetons se rencontrent, ils sont fusionnés et le niveau est augmenté par 1.
- Si le jeton arrive à un sommet marqué par un jeton de plus haute niveau ou plus grand UID, il est perdu.
- Un seul jeton réussit à traverser le graphe entièrement et rentrer chez lui. Ce sommet est Elu.

KKM Algorithme

- Règles pour les jetons :
  - Un jeton est soit *Annexing*(A), *Chasing*(C), ou *Waiting*(W)
  - Chaque jeton écrit sur chaque sommet qu'il visite.
  - Un jeton est tué quand il arrive à un sommet de plus haut niveau.
  - Un jeton-A devient C (W) s'il arrive à somme de même niveau, mais marquée par plus petit (grand) UID.
  - Un jeton-C devient W s'il arrive à sommet marqué par un autre jeton-C.
  - S'il y a deux jetons de même niveau à un sommet, ils sont fusionnés et le niveau est augmenté.

Correction et Complexité (KKM)

Correction :

- S'il existe $x > 1$ jetons à niveau i, il existe au moins 1 et au plus $x/2$ jetons à niveau $(i + 1)$
- S'il y a un seul jeton à niveau i, il va réussir à visiter tous les sommets.

Correction et Complexité (KKM)

Correction :

- S'il existe $x > 1$ jetons à niveau i, il existe au moins 1 et au plus $x/2$ jetons à niveau $(i+1)$
- S'il y a un seul jeton à niveau i, il va réussir à visiter tous les sommets.

Complexité :

- Pour k initiateurs, il y a $k/2^i$ jetons à niveau i.
- Nombre de messages à chaque niveau est $f(n) + n$.
- Complexité $= (n + f(n))(\log k + 1)$ messages.

"Synchronizers"

Synchronisation

- On a vu les algorithmes for les réseaux asynchrone quelconque.
- C'est plus facile (efficace) les algorithmes pour les réseaux synchrone.
- Comment utiliser les algorithmes synchrones dans les réseaux asynchrones ?

Synchronisation

- On a vu les algorithmes for les réseaux asynchrone quelconque.
- C'est plus facile (efficace) les algorithmes pour les réseaux synchrone.
- Comment utiliser les algorithmes synchrones dans les réseaux asynchrones ?

Définition (Synchonizer)

Un algorithme qui simule un réseau synchrone sur un réseau asynchrone.

Algorithmes pour Synchonisers

- Synchronizer α
  - Après ronde i envoyer OK à chaque voisin.
  - Commencer ronde $i+1$ après avoir reçu OK de chaque voisin.

Algorithmes pour Synchonisers

- Synchronizer α
  - Après ronde i envoyer OK à chaque voisin.
  - Commencer ronde $i+1$ après avoir reçu OK de chaque voisin.

- Complexité :
  - Messages = $O(m)$ par ronde.
  - Temps = $O(1)$ par ronde.

Algorithmes pour Synchonisers

- Synchronizer β
  Construire un arbre couvrant T (enraciné)
  - Après ronde i envoyer OK à parent.
  - Racine : Recevoir OK de tout enfants ; Diffuser message OK sur T.
  - Commencer ronde $i + 1$ après avoir reçu OK de parent.

Algorithmes pour Synchonisers

- Synchronizer β
  Construire un arbre couvrant T (enraciné)
  - Après ronde i envoyer OK à parent.
  - Racine : Recevoir OK de tout enfants ; Diffuser message OK sur T.
  - Commencer ronde $i+1$ après avoir reçu OK de parent.

- Complexité :
  - Messages $= O(n)$ par ronde.
  - Temps $= O(n)$ par ronde.

Algorithmes pour Synchonisers

- Synchronizer γ
  - Construire un regroupement (avec un arête préféré entre chaque deux groupes).
  - Effectuer Synchroniser β dans chaque groupe.
  - Effectuer Synchroniser α sur les arêtes préférés pour synchroniser entre les groupes.
- Complexité :
  - Messages = $O(k.n)$ par ronde.
  - Temps = $O(\log n / \log k)$ par ronde.

  Pour $1 < k < n$, avec regroupement [Awerbuch'85]
- Sans regroupement
  - Si chaque sommet est un groupe => Synchronizer α
  - Si tout le graphe est un groupe => Synchronizer β

Synchonisers

- Synchronizer α.
  - Messages = $O(m)$ par ronde.
  - Temps = $O(1)$ par ronde.
- Synchronizer β.
  - Messages = $O(n)$ par ronde.
  - Temps = $O(n)$ par ronde.
- Synchronizer γ.
  - Une mélange de les deux.
  - Le coût dépend de la méthode de regroupement.